Werkzeugerstellung für die Migration von Anwendungen in verteilte Umgebungen

Katja Cremer
Lehrstuhl für Informatik III
RWTH Aachen, D-52074 Aachen
katja@i3.informatik.rwth-aachen.de

1 Motivation

In einem Kooperationsprojekt des Lehrstuhls für Informatik III an der RWTH Aachen, der Aachener Münchener Versicherung (AM) und der Gebühreneinzugszentrale (GEZ) wurde die Fragestellung untersucht, wie sich die **Migration** von existierenden betriebswirtschaftlichen Softwareanwendungen **in eine verteilte Umgebung** durch Methoden und Werkzeuge unterstützen läßt. Hierbei wurden die anstehenden Probleme in zwei Bereiche unterteilt: (1) Vorbereitende Maßnahmen zur Verteilung und (2) Realisierung der Verteilung mit konkreten Verteilungsplattformen (CORBA, DCOM). Die Ergebnisse aus dem ersten Bereich des Projekts werden im folgenden vorgestellt.

In Abschnitt 2 wird auf die erarbeitete Vorgehensweise zur Vorbereitung der Verteilung eingegangen. Des weiteren sind Werkzeuge entstanden, die die festgelegte Methodik unterstützen. Die Funktionalität dieser Werkzeuge wird überblicksartig in Abschnitt 3 vorgestellt. Es war eine wesentliche Zielsetzung, bei der Realisierung der Werkzeugunterstützung spezifische Techniken einzusetzen, die eine Parametrisierung und Wiederverwendung des Erstellungsprozesses erlauben. Auf diesen Aspekt wird im Detail in Abschnitt 4 eingegangen. Die entstandenen Werkzeuge werden in Abschnitt 5 exemplarisch vorgestellt.

2 Vorgehensweise

Die Methodik, die im Rahmen dieses Projekts festgelegt wurde, ist darauf ausgerichtet, Anwendungen so zu verändern, daß eine Verteilung von bestehenden Anwendungsbestandteilen möglich wird. Hierfür sind im wesentlichen **zwei Eigenschaften** notwendig: (1) Eine Anwendung muß in logisch unabhängige Teile aufteilbar sein und (2) die einzelnen Teile einer Anwendung müssen über entsprechende Schnittstellen verfügen, über die sie von anderen

Teilen angesprochen werden können. Abb. 1 veranschaulicht die Vorgehensweise, die zur Erreichung dieser Eigenschaften festgelegt wurde.

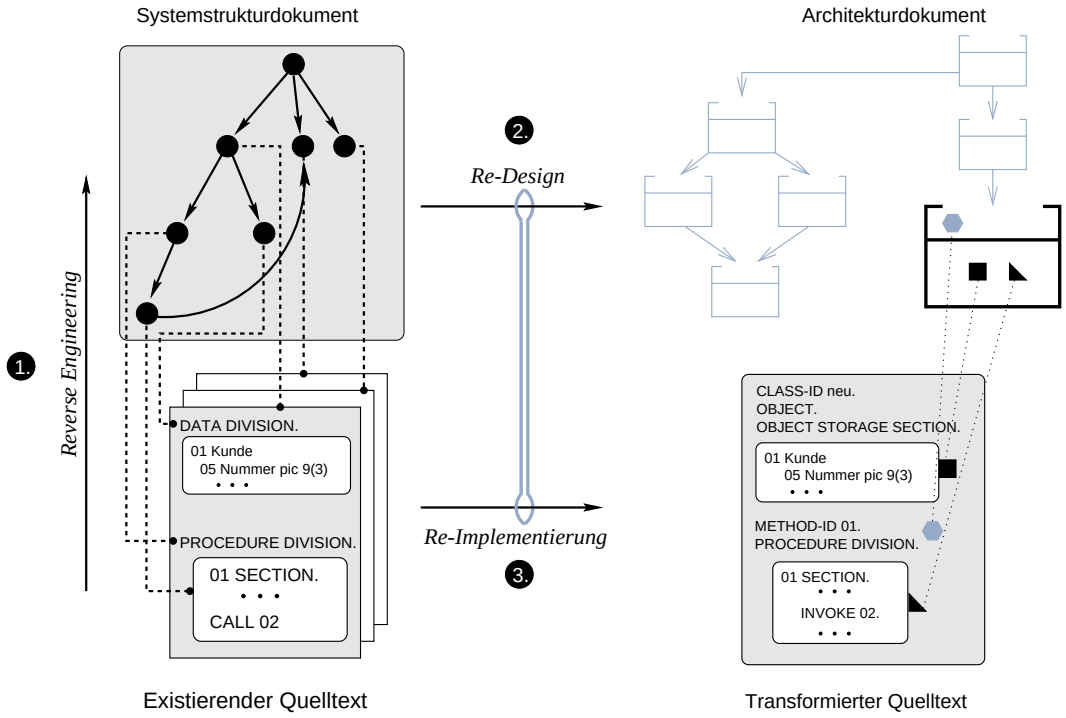

Abbildung 1: Methodik des Migrations-Ansatzes

Ausgangspunkt der Vorgehensweise ist der vollständige Quelltext der zu verteilenden Anwendung. Die vorhandenen Quelltexte sind die Grundlage für sämtliche Reverse Engineering-Aktivitäten. Das Ergebnis dieser Aktivitäten ist ein Systemstrukturdokument, welches Informationen darüber liefert, aus welchen Komponenten sich ein Anwendungssystem zusammensetzt und welche Beziehungen zwischen den einzelnen Komponenten existieren. Die aus dem Quelltext extrahierten Komponenten und Beziehungen werden in Form eines Graphen abgelegt, d.h. der zugrundeliegende Dokumenttyp des Systemstrukturdokuments ist ein Graph. Das Systemstrukturdokument bringt den aktuellen Zustand einer Anwendung zum Ausdruck.

Als nächster Schritt folgt ein Re-Design mit dem Ziel, eine Zuordnung von Komponenten und Beziehungen des Systemstrukturdokuments zu objektbasierten Architekturelementen zu treffen. Dieser Schritt schließt eine Auflösung von Abhängigkeiten und die Erzeugung von fehlenden Schnittstellen ein. Sowohl das Systemstruktur- als auch das Architekturdokument stellen Abstraktionen von konkreten Quelltexten dar.

Ein wesentliches Ziel der Vorgehensweise ist es, die Veränderungen auf der Designebene auf die Quelltextebene zu übertragen, wobei die Entscheidungen für die Vorbereitung zur Verteilung auf der Designebene getroffen werden und den Anstoß für Veränderungen auf der Implementationsebene geben. Natürlich bietet nicht jede Programmiersprache Konzepte an, um die Architekturfestlegungen der Design-Ebene unmittelbar im Quelltext umzusetzen. Je-

doch wurde bereits in [Nag90] erläutert, daß die Übertragbarkeit in eine Programmiersprache nie prinzipiell unmöglich ist. Die Übertragung ist aber je nach Alter der Programmiersprache mehr oder minder schwierig und aufwendig. Ziel dieses Schritts ist es, zum einen die Architekturvorgaben der Designebene in eine Implementation umzusetzen. Zum anderen werden vorhandene Quelltexte wiederverwendet und in die neu entstandene Struktur eingebettet. Es wird nicht möglich sein, sämtliche Veränderungen auf der Designebene automatisch auf die Quelltextebene zu übertragen, so daß bei diesem Schritt eine manuelle Nachbearbeitung notwendig wird.

3 Werkzeugunterstützung

Aufgrund der Größe realer Anwendungen ist es offensichtlich, daß die beschriebene Vorgehensweise nicht **ohne eine geeignete Werkzeugunterstützung** durchgeführt werden kann. Im Rahmen des Projekts sind vier Werkzeuge entstanden, die die einzelnen Schritte der Methodik unterstützen. Die Funktionalität der realisierten Werkzeuge wird im folgenden kurz beschrieben:

Analyse-Werkzeug: Dieses Werkzeug erzeugt aus gegebenen Quelltexten **Beschreibungen auf einem höheren Abstraktionsniveau**. Dazu durchsucht das Werkzeug einen gegebenen Quelltext nach bestimmten Konstrukten, die Aufschluß über die Struktur einer Anwendung geben. Das Vorkommen solcher Konstrukte wird in Form von Fakten festgehalten. Als Ergebnis liefert dieses Werkzeug eine textuelle Faktenbasis, welche die Informationen über aufgefundene Komponenten und Beziehungen enthält und die Grundlage für das Systemstrukturdokument darstellt. Die aktuelle Implementierung unterstützt die Analyse von Programmen in COBOL 85.

Visualisierungs-Werkzeug: Dieses Werkzeug übernimmt die **graphische Darstellung** des Systemstrukturdokuments. Dazu werden die vom Analysewerkzeug ermittelten textuellen Informationen in Form eines Graphen dargestellt. Neben der Umsetzung der Fakten in einen Graphen handelt es sich bei diesem Werkzeug um einen komfortablen **Graphbrowser**, der die Einstellung unterschiedlicher Sichten durch das Aus- und Einblenden bestimmter Arten von Knoten erlaubt.

Re-Design-Werkzeug: Dieses Werkzeug bietet Operationen an, um auf der Designebene **Veränderungen an der Struktur** der Anwendung vorzunehmen. Das Ziel der Veränderungen ist die Zusammenfassung von Elementen eines Systemstrukturdokuments und deren Zuordnung zu Architektureinheiten. Dieses Werkzeug bietet eine Reihe von Transformationen an, die Zuordnungen zwischen Bestandteilen des Systemstrukturdokuments und Achitektureinheiten vornehmen. In der aktuellen Implementierung wird eine am Lehrstuhl entwickelte Architekturbeschreibungssprache verwendet. Des weiteren bietet dieses Werkzeug vier Verfahren an, die eine vollständige Zuordnung eines Systemstrukturdokuments zu Architekturelementen vornehmen. Jedes Verfahren realisiert dabei je nach Strukturmerkmalen der Ausgangsanwendung eine andere Art der Zuordnung.

Quelltexttransformationswerkzeug: Dieses Werkzeug übernimmt die **Umsetzung der strukturellen Vorgaben** des entstandenen Architekturdokuments auf die Quelltextebene. Dabei werden Quelltexte der Altanwendung wiederverwendet und in syntaktische Konstrukte eingebettet, welche die strukturellen Vorgaben der Designebene realisieren. In der aktuellen Implementierung wird die Transformation von COBOL 85 nach COBOL 9x unterstützt. COBOL 9x unterstützt objektbasierte/orientierte Konzepte.

4 Werkzeugerstellung

Die Realisierung der Werkzeuge basiert weitgehend nicht auf Handprogrammierung, sondern auf der Nutzung von **spezifischen Realisierungstechniken**. Zur Realisierung des Analyse- und Quelltexttransformations-Werkzeugs wird die Spezifikationssprache **TXL** [CCH95] (Tree Transformation Language) eingesetzt. Diese Sprache ist auf die Unterstützung textueller Transformationsvorgänge spezialisiert. Im Rahmen der Realisierung des Analysewerkzeugs wird die Syntax der Programmiersprache COBOL mit Hilfe von entsprechenden TXL-Konstrukten festgelegt. Die von TXL zur Verfügung gestellten Textsuchmuster werden dazu eingesetzt, bestimmte Konstrukte aufzufinden, die für die Struktur einer Anwendung relevant sind. Für die aufgefundenen Quelltextartefakte wird eine Graphbeschreibung in textueller Form erstellt. Dabei abstrahieren die Graphknoten von Quelltextkomponenten, während die Kanten Beziehungen im Quelltext repräsentieren. Bei der Realisierung des Quelltexttransformationswerkzeugs wird TXL dazu genutzt, Programme, die in COBOL 85 vorliegen, so umzugestalten, daß sie Konstrukte der neuen COBOL Version 9x verwenden.

Bei der Darstellung der Vorgehensweise wurde angedeutet, daß die Systemstruktur- und Architekturdokumente intern nicht als einfache Texte abgelegt sind, sondern in Form von Graphen gespeichert werden. Der methodische Ansatz sowie die Realisierung der Transformationen des Re-Design-Werkzeugs basieren im wesentlichen auf dem Einsatz von Graphentechnik. Eine ausführliche Einführung in den Graphentechnikansatz findet sich in [Nag96]. Die VHL-Sprache (Very High Level) **PROGRES** [SWZ95] unterstützt die Erstellung grafischer Werkzeuge, die auf dem Graphentechnikansatz basieren. Aus diesem Grund wurde PROGRES zur Spezifikation des Visualisierungs- und Re-Designwerkzeugs eingesetzt.

Aufgrund der Nutzung von TXL und PROGRES kann der **Erstellungsprozeß** für Reengineering-Werkzeuge wiederverwendet werden, die die gleiche Methodik unterstützen, aber zur Bearbeitung anderer Programmiersprachen oder Veränderungsziele zum Einsatz kommen sollen. Nur wenige Festlegungen für die Werkzeugerstellung sind hart in die Werkzeuge codiert. Statt dessen sind sie in Form eigener Dokumente vorhanden, die den von TXL und PROGRES zur Verfügung gestellten Generatoren als Eingabe dienen. Aufgrund dieser Parametrisierungsmöglichkeiten kann die Ausprägung von Einzelwerkzeugen explizit beeinflußt werden. In [EKP98] werden solche Techniken als **MetaCARE-Umgebung** bezeichnet, wobei der Begriff CARE für Computer Aided REengineering steht und in Analogie zum Begriff CASE (Computer Aided Software Engineering) verwendet wird. Auf diesen Aspekt wird im folgenden eingegangen.

Die Motivation für die Nutzung von Parametrisierungsmöglichkeiten und Generatorentechnik ist darin begründet, daß oftmals Anpassungen an den Werkzeugen notwendig werden. Die Notwendigkeit von Anpassungen entsteht aufgrund der Betrachtung neuer Programmiersprachen oder anderer Versionen von Programmiersprachen im Rahmen des Reengineerings. Weiterhin können sich die Zielsetzungen beim Reengineering ändern, was eine Änderung der Transformationen auf Design- und Quelltextebene nach sich zieht. Um eine größtmögliche Flexibilität bei der Anpassung von Werkzeugen zu haben, bietet sich der Einsatz von Generatoren an, die in der Lage sind, aus entsprechenden Vorgaben Werkzeuganteile zu generieren. Die folgende Abb. 2 zeigt die in Abschnitt 3 vorgestellten Werkzeuge und ihre Generierungsvorgaben.

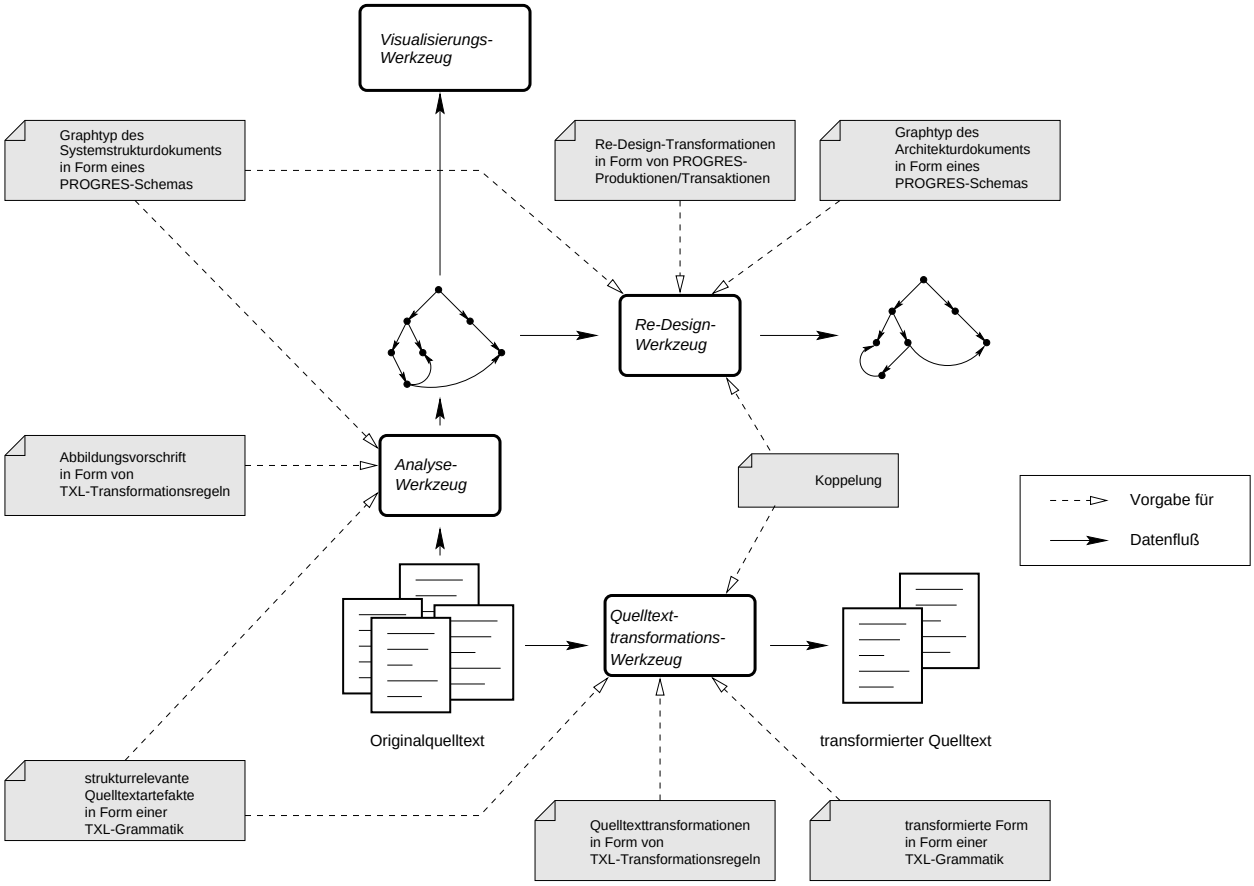

Abbildung 2: Generierte Werkzeuge und ihre Vorgaben

In dieser Abbildung sind zum einen die mit TXL und PROGRES generierten Einzelwerkzeuge des Reengineering-Prototypen dick umrahmt dargestellt. Weiterhin sind die Dokumenttypen skizziert, die von den jeweiligen Werkzeugen bearbeitet werden. Die grau hinterlegten Kästen stellen in dieser Abbildung die Dokumente dar, in denen sich die Festlegungen bezüglich der Werkzeugausprägung befinden. Diese Dokumente bilden die Eingaben für die Generatoren von TXL und PROGRES.

Diese Abbildung deutet an, daß zum einen die Analyse der Quelltexte parametrisierbar ist.

Es kann angegeben werden, welche Quelltextartefakte auf welche Graphknoten und -kanten abgebildet werden. Aus diesen Vorgaben wird ein grapherzeugender Parser generiert. Desweiteren sind sowohl der Graphtyp des Systemstrukturdokuments als auch der Graphtyp des Architekturdokuments frei wählbar. Weiterhin wird dem Werkzeugersteller die Möglichkeit angeboten, eine Auswahl an Design- und Quelltexttransformationen festzulegen. Diese Festlegungen beeinflussen die Generierung des Re-Design-Werkzeugs. Die Berücksichtigung aller Festlegungsmöglichkeiten führt zu folgender **Werkzeugerstellungsmethodik**:

1. Festlegung der Quelltextkonstrukte, die in den Originalquelltexten und in der transformierten Form relevant sind, in Form einer TXL-Grammatik

2. Festlegung der Graphtypen des Systemstruktur- und des Architekturdokuments unter Berücksichtigung der Quelltextartefakte, von welchen diese Dokumente abstrahieren, in Form von PROGRES-Graphschemata

3. Definition der Abbildungsvorschrift zwischen den strukturrelevanten Originalquelltextkonstrukten und dem Graphtyp des Systemstrukturdokuments in Form von TXL-Transformationsregeln

4. Festlegung der Re-Design-Transformationen in Form von Tripelgraphregeln in PROGRES-Notation

5. Festlegung der Quelltexttransformationen in Form von TXL-Transformationsregeln

6. Festlegung der Koppelung zwischen diesen beiden Transformationsarten in Form einer PROGRES-Spezifikation

Aus Platzgründen wird im folgenden nur auf die Festlegung der Re-Design-Transformationen eingegangen.

Im Rahmen dieses Projekts wird sowohl für das Systemstrukturdokument, welches Informationen über die Komponenten und Beziehungen einer bestehenden Anwendung aufnimmt, als auch für das Architekturdokument, welches die vorbereiteten Einheiten für die Verteilung enthält, ein eigener Graphtyp festgelegt. Re-Design bedeutet konkret, den **Übergang zwischen den unterschiedlichen Graphtypen** zu realisieren. Hierbei ergibt sich das Problem, daß nicht Operationen auf einem Graphen, sondern auf verschiedenen Graphen zu spezifizieren sind, und daß Beziehungen zwischen diesen Graphen hergestellt werden müssen. Um mit diesem Sachverhalt adäquat umzugehen, wird zur Spezifikation der Re-Design-Transformationen der Ansatz der Tripelgraphgrammatiken eingesetzt.

Die Idee der Tripelgraphgrammatiken [Sch94] basiert auf den von T. W. Pratt 1971 vorgestellten Paargrammatiken [Pra71]. Bei Paargrammatiken wird eine Transformation zwischen Dokumenten dadurch spezifiziert, daß für jedes der beteiligten Dokumente eine kontextfreie (Graph-)Grammatik erstellt wird. Diese Grammatiken werden gekoppelt, indem man zwischen ihren Regeln und zwischen Knoten dieser Regeln bijektive Abbildungen definiert. Die Grundidee des Pratt´schen Ansatzes wurde bei den Tripelgraphgrammatiken übernommen. D.h. es werden zwei Graphgrammatiken für die beteiligten Dokumente erstellt, zwischen deren Regeln dann Korrespondenzen definiert werden. Im Gegensatz zu Pratt

erlauben Tripelgraphregeln **m:n-Beziehungen** sowohl zwischen Regeln als auch zwischen Knoten und Kanten in den Regeln.

Die bei diesem Ansatz erlaubten m:n-Beziehungen müssen adäquat notiert werden. In [Sch94] geschieht dies durch einen zusätzlichen Graphen, einen sogenannten **Korrespondenzgraphen**. Für diesen Graphen kann wieder eine Graphgrammatik angegeben werden. Somit wird der Sachverhalt durch ein Tripel von Graphgrammatiken für Quell-, Korrespondenz-, und Zielgraph ausgedrückt. Daher auch der Name **Tripel**graphgrammatik. Bei dem Quellgraph handelt es sich im Rahmen dieser Arbeit um das Systemstrukturdokument, den Zielgraphen stellt das Architekturdokument dar. Zusätzlich wurde ein Korrespondenzgraph eingeführt, auf den im folgenden nicht näher eingegangen wird.

Der Übergang vom Systemstruktur- zum Architekturdokument wird wesentlich **durch Zuordnungen zwischen Objekten** und **Beziehungen** dieser beiden Dokumenttypen bestimmt. Die folgende Abb. 3 zeigt als Beispiel eine Tripelgraphregel, die Zuordnungen zwischen Knoten des Systemstrukturgraphen und Knoten des Architekturgraphen herstellt.

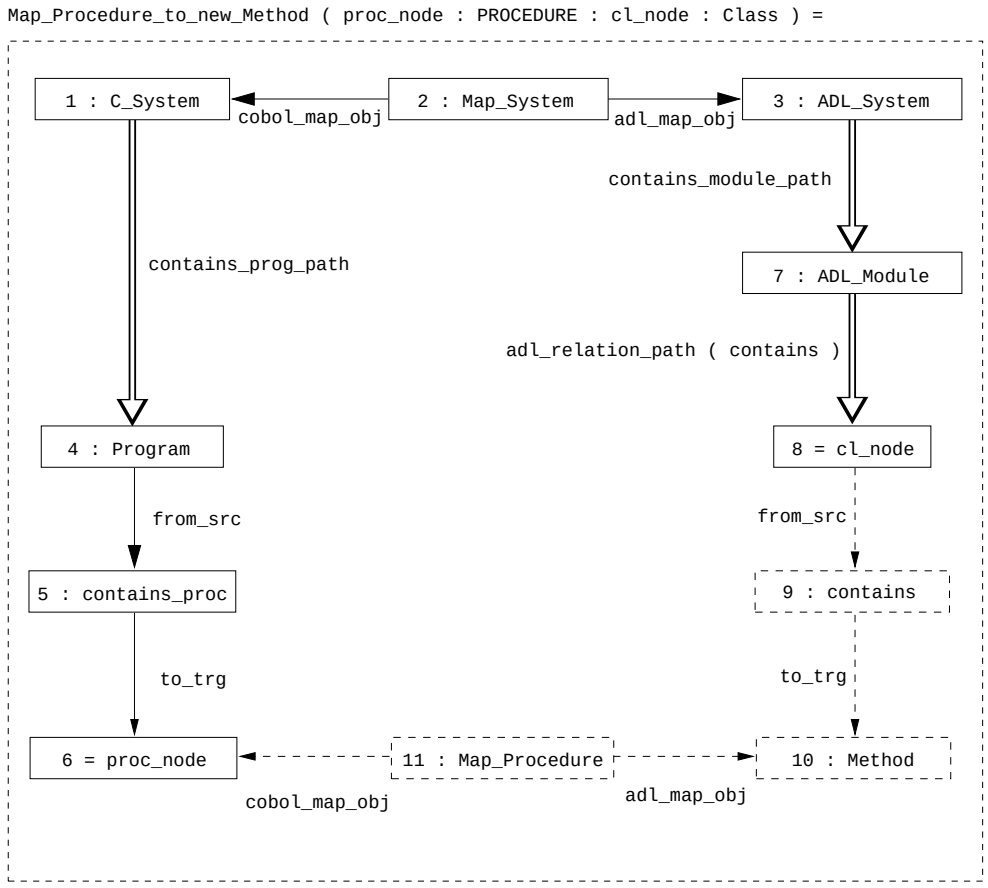

Abbildung 3: Transformation zur Zuordnung von Knoten der beteiligten Graphen

Auf der linken Seite dieser Regel finden sich die Anteile des Systemstrukturgraphen, die an dieser Regel beteiligt sind. Der mittlere Teil der Regel, zeigt die beteiligten Graphanteile des Korrespondenzdokuments. Rechts finden sich die beteiligten Knoten und Kanten aus dem Architekturdokument. Die Ausgangssituation, dargestellt durch die durchgezogenen Anteile, stellt sich folgendermaßen dar: Die Wurzelknoten der beiden Graphen, die dem Systemstruktur- und Architekturdokument zugrundeliegen, sind bereits einander zugeordnet worden. Diese Zuordnung wird durch den `Map_System`-Knoten im Korrespondenzgraphen sowie die beiden Kanten `cobol_map_obj` und `adl_map_obj` zum Ausdruck gebracht. Für die Anwendung dieser Regel wird weiterhin vorausgesetzt, daß ein `Procedure`-Knoten als Parameter übergeben wird. `Procedure`-Knoten abstrahieren von den COBOL-Konstrukten Section und Paragraph. Als weiteren Parameter erhält die Regel einen `Class`-Knoten des Architekturgraphen. Die als Parameter übergebenen Knoten sind über Pfade mit einem Graphen ihres jeweiligen Typs (Systemstruktur- bzw. Architekturgraph) verbunden. Falls ein solches Graphmuster existiert, wird an den angegebenen `Class`-Knoten mit Hilfe eines `has_method`-Konstrukts der `Method`-Knoten `10` angefügt. Zusätzlich wird durch das Einfügen des `Map_Procedure`-Knoten `11` mit den dazugehörigen Kanten eine Zuordnung zwischen dem vorhandenen `Procedure`-Knoten und dem neu erzeugten `Method` hergestellt.

Mit Hilfe solcher Regeln wird das Verhalten des Re-Design-Werkzeugs spezifiziert. Für das konkret entstandene Re-Design Werkzeug sind siebzehn Basis-Transformationsregeln festgelegt worden. Mit diesen Basistransformationen sind vier Re-Design-Verfahren realisiert worden, die den Prozeß des Re-Designs halbautomatisch vorgeben. Mit Hilfe des Generators der PROGRES-Umgebung werden das spezifizierte Werkzeugverhalten in eine Werkzeug-Implementierung umgesetzt.

5 Entstandene Werkzeuge

Das Visualisierungs-, das Re-Design- sowie das Quelltexttransformations-Werkzeug sind unter einer gemeinsamen Oberfläche im REforDI-Prototypen (REengineering for DIstribution) zusammengefügt. Die folgende Abb. 4 zeigt die Oberfläche des REforDI-Prototyps.

Der dargestellte Graph visualisiert sämtliche Programme, die Bestandteil einer Beispielanwendung sind, durch einzelne Knoten. Die Aufrufbeziehungen, die zwischen den einzelnen Programmen existieren, werden auf Kanten abgebildet. Auf Kantenbezeichner wurde aus Gründen der Übersichtlichkeit verzichtet. Anhand einer solchen Darstellung kann sich ein Werkzeugbenutzer einen Überblick über die an einer Anwendung beteiligten Programme und ihr Zusammenspiel machen. An jedem Knoten sind Informationen vorhanden, die angeben, von welchen konkreten Quelltextstücken der betreffende Knoten abstrahiert. Aufgrund dieser Informationen ist ein Zugriff auf die Quelltexte, die einem Knoten zugrundeliegen, stets möglich. Der Zugriff auf den Quelltext eines Knoten erfolgt durch die Auswahl des Menüpunkts `View_Sourcecode`. Dieser stößt den Aufruf eines Texteditors an, in dem der zugehörige Quelltext dargestellt ist.

Im unteren Teil der Abbildung ist das Einstellmenü zur Auswahl der anzuzeigenden Knoten-

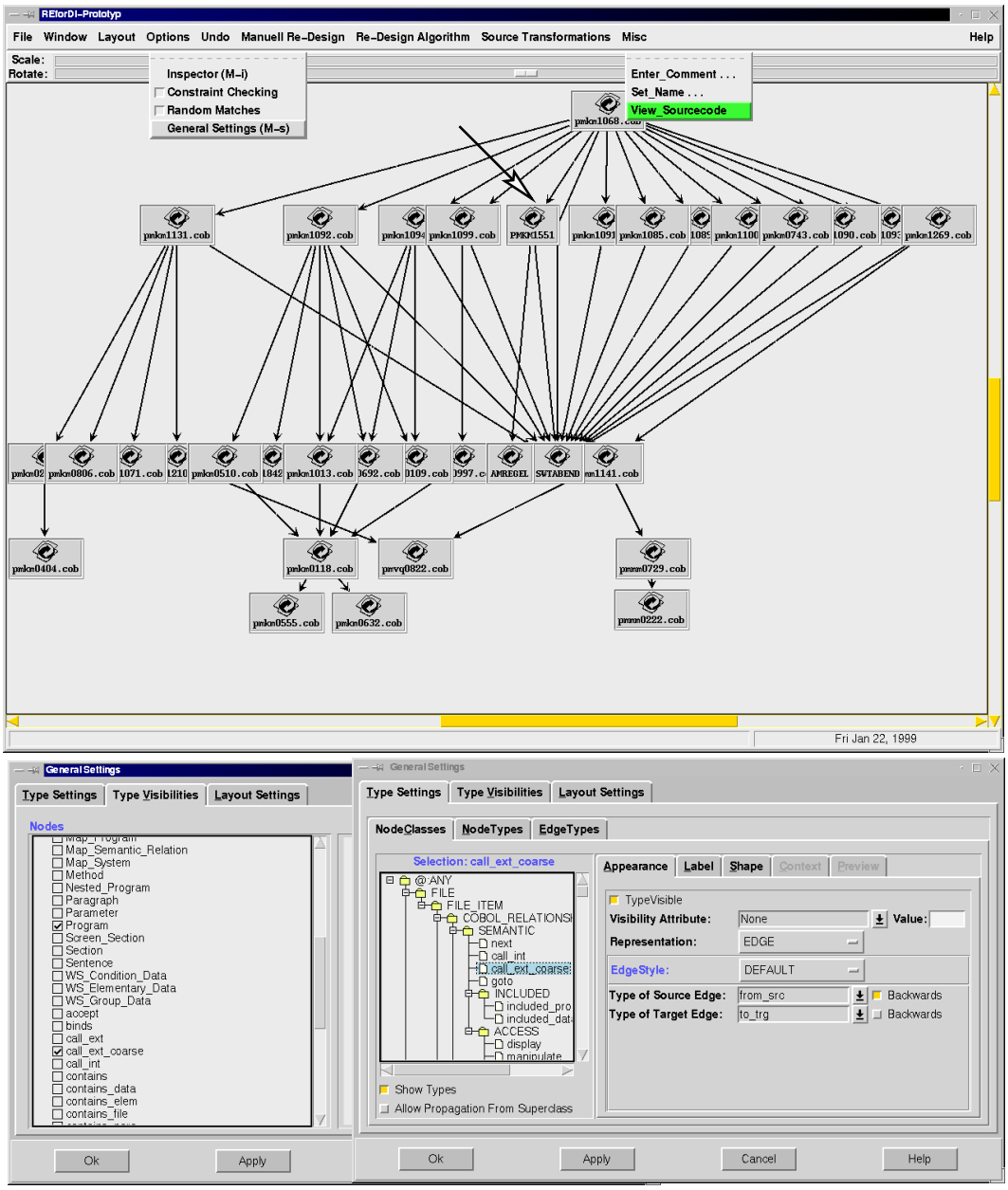

Abbildung 4: Aufrufgraph der Beispielanwendung

typen dargestellt. Hier kann durch Anklicken festgelegt werden, ob ein bestimmter Knotentyp bei der Graphdarstellung angezeigt werden soll. Für die Darstellung des Aufrufgraphen sind hier nur Knoten vom Typ `Program` ausgewählt worden. Mit Hilfe dieser Konfigurationsmöglichkeiten können **Einstellungen** vorgenommen und **Sichten** erzeugt werden, die einen Reverse Engineering Vorgang maßgeblich unterstützen.

Im Menü `Manual_Re-Design` finden sich die Menüpunkte zur Durchführung eines manuellen Re-Design unter Nutzung der Basistransformationen. Im Menü `Re-Design_Algorithm` stehen die vier Re-Design-Verfahren zur Verfügung. Nach Abschluß des Re-Design-Vorgangs

werden die erzeugten Architektureinheiten so weit wie möglich automatisch auf die Quelltextebene umgesetzt. Das Quelltexttransformations-Werkzeug arbeitet batchartig. Zum Aufruf dieses Werkzeugs wird eine Architektureinheit des Architekturgraphen ausgewählt. Der eigentliche Transformationsvorgang des Quelltextes für die ausgewählte Einheit wird durch den Menüpunkt `Transform_Code_for_ADL_Object` angestoßen.

6 Ausblick

Neben den bereits vorgestellten Werkzeugen sind eine Reihe anderer Werkzeuge in einem weiteren Teilprojekt entstanden, die die konkrete Verteilung mit einer Middleware wie z.B. CORBA unterstützen. Diese Werkzeuge kommen unmittelbar nach den hier beschriebenen vorbereitenden Maßnahmen zur Verteilung zum Einsatz.

Literatur

[CCH95] J.R. Cordy, I.H. Carmichael und R. Halliday: *The TXL Programming Language - Version 8*. Software Technology Laboratory, Department of Computing and Information Science, Queen's University, 1995.

[EKP98] Jürgen Ebert, Bernt Kullbach und Andreas Panse: *The Extract-Transform-Rewrite Cycle - A Step towards MetaCARE*. In: Paolo Nesi und Franz Lehner (Herausgeber): *Proceedings of the Second Euromicro Conference on Software Meintenance and Reengineering (CSMR)*, Seiten 165–170. IEEE Computer Society, IEEE Computer Society Press, März 1998.

[Nag90] Manfred Nagl: *Softwaretechnik - Methodisches Programmieren im Großen*. Springer Verlag, 1990.

[Nag96] Manfred Nagl (Herausgeber): *Building Tightly Integrated Software Development Environments: The IPSEN Approach*. Springer Verlag, 1996.

[Pra71] Terrence W. Pratt: *Pair Grammars, Graph Languages and String-to-Graph Translations*. Journal of Computer and System Sciences, 5(6):560–595, Dezember 1971.

[Sch94] Andy Schürr: *Specification of Graph Translators with Triple Graph Grammars*. In: *Proceedings of WG 94, Int. Workshop on Graph- Theoretic Concepts in Computer Science*, Band 903 der Reihe *Lecture Notes in Computer Science*, Seiten 151–163. Springer Verlag, 1994.

[SWZ95] Andy Schürr, Andreas Winter und Albert Zündorf: *Graph Grammar Engineering with PROGRES*. In: W. Schäfer und P. Botella (Herausgeber): *Proceedings of the 5th European Software Engineering Conference (ESEC'95)*, Seiten 219–234. Springer Verlag, 1995.