Lehrstuhl für Informatik II

(Programmier- und Dialogsprachen sowie Compiler)

Leiter:

Prof. Dr. Hans Jürgen Schneider

Mitarbeiter:

Behnsen, Wolfgang, Dipl.-Math.	(Programmierer)	bis 30. 06. 96
Dormeyer, Ricarda, Dipl.-Inf.	(wiss. Mitarb.)	ab 01. 10. 96
Fischer, Ingrid, Dipl.-Inf.	(wiss. Mitarb.)	
Hodek, Roman, Dipl.-Inf.	(wiss. Mitarb.)	
Jacob, Christian, Dr.-Ing.	(wiss. Mitarb.)	beurl. ab 01. 11. 96
Keil, Martina, M.A.	(wiss. Mitarb.)	bis 31. 03. 96
Lührs, Erni	(Sekretärin - halbtags)	
Minas, Mark, Dr.-Ing.	(Akad. Rat)	
Nilson, Jörg, Dipl.-Inf.	(wiss. Mitarb.)	
Schörmal, Elfriede	(Sekretärin - halbtags)	
Uebler, Manfred	(Programmierer)	
Wilke, Peter, Dr.-Ing.	(Akad. Rat)	

Gastwissenschaftler:

Billing, Gunnar, Dipl.-Inf.	(Bay. Kultusministerium)	ab 16. 05. 96
Heidenreich, Georg, Dr.-Ing.	(BMFT)	bis 31. 05. 96
Kókai, Gabriella	(Bay. Kultusministerium)	bis 31. 08. 96
Schoberth, Andreas, M. Sc.	(FAST e.V.)	ab 01. 08. 96

Lehrbeauftragte:

Feder-Andres, Christiane, Dr.-Ing.	(Fa. sd&m, München)
Hindel, Bernd, Dr.-Ing.	(Fa. 3SOFT/QM, Erlangen)
Kips, Detlef, Dr.-Ing.	(Fa. BASYS, Erlangen)
Schorr, Ruth, Dr.-Ing.	(Deutsche Bank, Frankfurt/Main)

Gäste:

Berns, Karsten, Dr. rer. nat.	(Univ. Karlsruhe)
Grabska, Ewa, Prof. Dr.	(Univ. Krakau (Polen))
Rauch Henzinger, Monika, Prof. Ph. D.	(Cornell University (USA))

1 Graphtransformationssysteme

Graphen zur Darstellung von statischen Strukturen und dynamischen Vorgängen haben ihren festen Platz in der theoretischen und angewandten Informatik, aber auch in vielen anderen Wissenschaften, gefunden. Seit Beginn der siebziger Jahre beschäftigt sich der Lehrstuhl intensiv mit Graphtransformationssystemen, das sind Regelsysteme, die die Gewinnung neuer Graphen aus bereits bekannten beschreiben. Die Beschreibung dynamischer Veränderungen ist damit unmittelbar einsichtig. Diese Ideen haben weltweite Resonanz gefunden; an verschiedenen Stellen wurden Weiterentwicklungen vorgenommen und Alternativen untersucht.

Ein für theoretische Untersuchungen sehr fruchtbarer Ansatz wurde von uns 1973 vorgestellt. Er basiert auf Mitteln der Kategorientheorie und benutzt eine Pushout-Konstruktion, um den Begriff der Konkatenation von Zeichenreihen zu verallgemeinern. Die Ableitbarkeit wird dann mit einem Doppelpushout beschrieben, indem sowohl die linke als auch die rechte Seite einer Ersetzungsregel längs eines Klebegraphen mit dem verbleibenden Kontext verbunden wird. Dies ist eine geradlinige Verallgemeinerung des Chomskyschen Ableitbarkeitsbegriffes.

Im Berichtszeitraum wurde nun eine Idee von R. Banach aufgegriffen. Er hatte in Anlehnung an die Vorgehensweise bei der Termersetzung die Pfeilrichtung der beiden Pushoutdiagramme umgedreht. Anschaulich bedeutet das, daß zunächst der einzusetzende Teil mit dem Ausgangsgraphen verklebt und dann der zu ersetzende Teil entfernt wird. Banach konnte die Äquivalenz beider Vorgehensweisen für den speziellen Fall von Graphen mit injektiver Einbettung des Klebeobjekts in die linke bzw. rechte Seite zeigen. Wir haben sein Ergebnis in zwei Richtungen erweitert. Zum einen läßt sich die Äquivalenz in rein kategorieller Terminologie ohne Bezug zu dem Spezialfall der Graphen beweisen. (Dies ist insbesondere für Anwendungen nützlich, bei denen durch die Markierung Gesichtspunkte hineinkommen, die mit üblichen Graphen nicht so einfach behandelt werden können.) Zum andern lieferte der Beweis automatisch eine Abschwächung der Voraussetzung: Die Äquivalenz gilt bereits, wenn eine der beiden Seiten der Produktion injektiv ist.

Schließlich konnte gezeigt werden, daß der Banachsche Ansatz in engem Zusammenhang mit den von S. M. Kaplan eingeführten Delta-Grammatiken steht: Er kann als theoretische Fundierung dieser Grammatiken verwandt werden und liefert so neue Einsichten. Insbesondere könnten auch Transformationen von Graphen, deren Knoten (oder Kanten) ihrerseits mit Graphen markiert sind, in der von Kaplan verwendeten Weise anschaulich dargestellt werden. (Schneider)

2 Soft-Computing - Prinzip, Simulation und Anwendung

2.1 Prinzip des Soft-Computing

Die Arbeit befaßt sich mit den grundlegenden Prinzipien des Soft-Computing, der Entwicklung und Simulation von Soft-Computing-Anwendungen und den dafür notwendigen Werkzeugen. Soft-Computing (abgekürzt SC) faßt die Techniken zusammen, die sich mit numerischen Berechnungen zur Ermittlung von Näherungslösungen befassen. Dazu gehören Fuzzy-Logik (FL), Neuro-Computing (NC) und probabilistisches Schließen (engl.: Probabilistic Reasoning PR), was u. a. evolutionäre Algorithmen (EA) einschließt. Der Begriff Neuro-Computing umfaßt die technische Realisierung und die zugrundeliegende Theorie der neuronalen Netze.

Neuronale Netze sind Systeme hochgradig verbundener Neuronen. Durch parallele Informationsverarbeitung können mit neuronalen Netzen komplexe Aufgaben gelöst werden. Dabei wird, ausgehend von einer zufällig gewählten Anfangskonfiguration, durch einen Lernprozeß Wissen im neuronalen Netz gespeichert, indem die Gewichtungen der Informationsweiterleitung zwischen den Neuronen verändert werden. Der Lernprozeß wird beendet, wenn entweder das neuronale Netz korrekt auf Testanfragen antwortet oder eine Höchstzahl an Lernschritten erfolgt ist.

Fuzzy-Systeme bestehen aus Wenn-Dann-Regeln, die das Systemverhalten beschreiben. Bei der Formulierung der Regeln können Variablen verwendet werden, die einen unscharfen Wert als Zugehörigkeit zu Kategorien beschreiben. Diese Transformation wird als Fuzzyfizierung bezeichnet. Die Bedingungen können partiell erfüllt sein. Die Aktionen werden dann mit dem Erfüllungsgrad der Bedingung ausgeführt. Man spricht hier von approximativem Schließen. Die Systemantwort wird durch Überlagerung (Inferenz) der partiellen Aktionen ermittelt und als scharfer Ausgangswert dargestellt. Diese Transformation wird als Defuzzyfizierung bezeichnet. Das Fuzzy-System wird an das Problem durch Veränderung der Regelbasis und der Zugehörigkeitsfunktionen angepaßt, bis eine Lösung gefunden oder die Konstruktion als erfolglos abgebrochen wird.

Evolutionäre Algorithmen basieren auf der Evolutionsstrategie der Natur und stellen ein massiv paralleles Optimierungsverfahren dar. Die Individuen werden in einer Darstellung repräsentiert, die für die Optimierung geeignet ist. Ausgehend von einer Anfangspopulation werden die an das Problem am besten angepaßten Individuen ermittelt. Dies geschieht durch eine Bewertungsfunktion (Fitnessfunktion). Die nächste Population wird mit genetischen Operationen (Selektion, Mutation, Rekombination) aus der aktuellen Population generiert. Dieser Zyklus wird wiederholt, bis entweder eine Lösung gefunden oder eine Höchstzahl an Populationen erzeugt wird.

Ist für eine Problemstellung ein exaktes Lösungsverfahren bekannt und liefert es zu akzeptablen Kosten eine Lösung, so sollte dieses Verfahren im allgemeinen eingesetzt werden. Ist dies nicht möglich, so sollte geprüft werden, ob die Lösung mit den Methoden des Soft-Computing erzielt werden kann. Grob (!) läßt sich das

Problem dabei wie folgt klassifizieren:

- Läßt sich das Problem anhand von Regeln, Gesetzmäßigkeiten etc. wenigstens ungefähr beschreiben, so bietet sich eine Lösung mit einem Fuzzy-Logik-System an. Dabei werden die das Problem beschreibenden Regeln als unscharfe Regeln formuliert, und ein Inferenz-Mechanismus kombiniert diese Regeln so, daß das beschriebene System modelliert wird. Diese Modellierung stellt dann die Problemlösung dar.
- Ist über das Problem wenig oder nichts bekannt, existieren aber Beispiele, anhand derer korrekte und inkorrekte Problemlösungen unterschieden werden können, so kann eine Modellierung mit künstlichen neuronalen Netzen erfolgreich sein. Das Problemwissen wird dabei in einer verteilten Darstellung gespeichert und in einem Lernprozeß mit Hilfe der Beispiele solange verändert, bis das Problem ausreichend exakt modelliert wurde, d. h. eine Lösung gefunden ist.
- Stehen für die Problembeschreibung weder Regeln noch Beispiele zur Verfügung, kann die Problemlösung mittels evolutionärer Algorithmen systematisch gesucht werden. Das Problem wird in einer codierten Form dargestellt und eine gewisse Anzahl von initialen Lösungskandidaten ausgewählt. Diese werden decodiert, und ihre Güte wird bewertet. Die Lösungskandidaten werden systematisch verändert und miteinander kombiniert, wobei die Kandidaten untereinander in Konkurrenz treten, um neue Lösungen zu bilden. Dieses Verfahren wird iterativ angewendet, wobei der beste Lösungskandidat der letzten Iteration die Problemlösung darstellt.
- Ist keine der oben beschriebenen Methoden direkt anwendbar, so kommt noch die Kombination von zwei oder drei der Soft-Computing-Methoden oder konventioneller Methoden in Frage. Man spricht dann von hybriden Systemen. Denkbar sind zum Beispiel Neuro-Fuzzy-Systeme, in denen ein neuronales Netz die Regeln eines Fuzzy-Systems lernt, oder die Suche nach einem optimalen neuronalen Netz mittels eines evolutionären Algorithmus.

Die einzelnen Paradigmen haben unterschiedliche Vor- und Nachteile. Durch die Kombination der Soft-Computing-Methoden miteinander und mit konventionellen Verfahren lassen sich diese Nachteile mindern oder aufheben. Die drei Paradigmen des Soft-Computing ergänzen sich also, und in Zukunft werden hybride Systeme immer mehr an Bedeutung gewinnen. Auch wird sich die Grenze zwischen traditioneller Künstlicher Intelligenz und Soft-Computing immer mehr verwischen.

Es wird ein Konzept vorgestellt, das die drei bisher getrennten Paradigmen des Soft-Computing in einem einheitlichen Modell zusammenfaßt. Damit ist es erstmals möglich, sowohl homogene wie auch hybride Soft-Computing-Systeme auf eine einheitliche und effiziente Art zu beschreiben. Für die Beschreibung wurde ein objekt-orientierter Mechanismus entwickelt, der zugleich die Basis einer effizienten Implementierung ist.

2.2 Simulation von Soft-Computing-Systemen

Soft-Computing-Systeme lassen sich hierarchisch top-down beschreiben, wobei die

Einund Ausgabe von der internen Darstellung getrennt wird. Der Grund liegt in der Notwendigkeit, die Ein- und Ausgabedaten zu puffern. Die Ein- und Ausgabedaten eines SoftComputing-Systems liegen in Form von Vektoren reeller Zahlen vor, die in den Ein- bzw. Ausgabepuffern des Systems während der gesamten Berechnung zur Verfügung stehen. Verbindungen zwischen den Ein- und Ausgabepuffern und den Komponenten des SoftComputing-Systems regeln den Datenfluß im Gesamtsystem.

Die Komponenten eines Soft-Computing-Systems können selbst Soft-Computing-Systeme oder Multi-Knoten und -Verbindungen sein, wobei die Multi-Knoten durch Ordnungsstrukturen (Schichten, Gruppen) zusammengefaßt werden können. Ein Multi-Knoten ist eine Zusammenfassung mehrerer Knoten gleicher Funktionalität. Zwischen den Multi-Knoten verlaufen Multi-Verbindungen, die die vollständige Verbindung jedes Knotens des QuellMulti-Knotens mit denen des Ziel-Multi-Knotens herstellen. Damit kann für Visualisierung die Information über die Topologie reduziert und konzentriert werden (wegen der hohen Verbindungsdichte bei neuronalen Netzen), und die Effizienz der internen Repräsentierung kann durch die Verwendung von Laufschleifen gesteigert werden.

Den Knoten, Verbindungen und SC-Systemen können eine oder mehrere Variablen zugeordnet sein, die ihren aktuellen Berechnungszustand charakterisieren. Die aktuell auszuführende Berechnung des SC-Systems wird durch den Modus (Init, Learn, usw.) festgelegt. Die Knoten und Verbindungen verfügen über Funktionen, mit denen sie ihren Zustand, abhängig vom aktuellen Modus, neu bestimmen.

Die Beschreibung eines SC-Systems, bestehend aus den Berechnungsvorschriften der Modi und aus der Beschreibung der Teilkomponenten, wird als Modell bezeichnet.

2.3 Die NeuroGraph-Entwicklungsumgebung

NeuroGraph ist eine vollständige, integrierte Entwicklungs- und Simulationsumgebung für Soft-Computing-Systeme, dazu zählen künstliche neuronale Netze, evolutionäre Algorithmen und Fuzzy-Logik, wobei diese Paradigmen allein oder in Kombination verwendet werden können. So lassen sich z. B. Neuro-Fuzzy-Systeme konstruieren oder neuronale Netze mit genetischen Algorithmen optimieren.

NeuroGraph zeichnet sich durch eine einfache und intuitive Bedienung aus. Alle graphischen Elemente der Komponenten entsprechen den OSF/Motif- bzw. MS-Windows-Stilrichtlinien. NeuroGraph ist streng objekt-orientiert entwickelt und komplett in C++ implementiert worden.

Als Hard- und Software-Plattformen stehen alle wichtigen Betriebssystem- und HardwarePlattformen zur Verfügung. Alle Versionen sind kompatibel, so daß die entwickelten Anwendungen auf allen Plattformen ausführbar sind. Die automatisierte Parallelisierung und Codeerzeugung unterstützt Spezial-Rechner (SIMD und MIMD).

Das NeuroGraph-Kernsystem besteht aus den Komponenten zur Simulation und Visualisierung von neuronalen Netzen, genetischen Algorithmen, Fuzzy-Logik und einer Bibliothek effizienter Implementierungen der wichtigsten Modelle. Der Benutzer

kann mit einem graphischen Werkzeug neue Modelle entwickeln oder die vorhandenen Modelle modifizieren.

Die Implementierung der Anwendungen wird durch die Werkzeuge Codeerzeugung, 2Dund 3D-Editoren, Parallelisierung und DOS-Toolkit unterstützt. Zur Ausgabe der Ergebnisse stehen Protokollierungs- und Visualisierungskomponenten zur Verfügung.

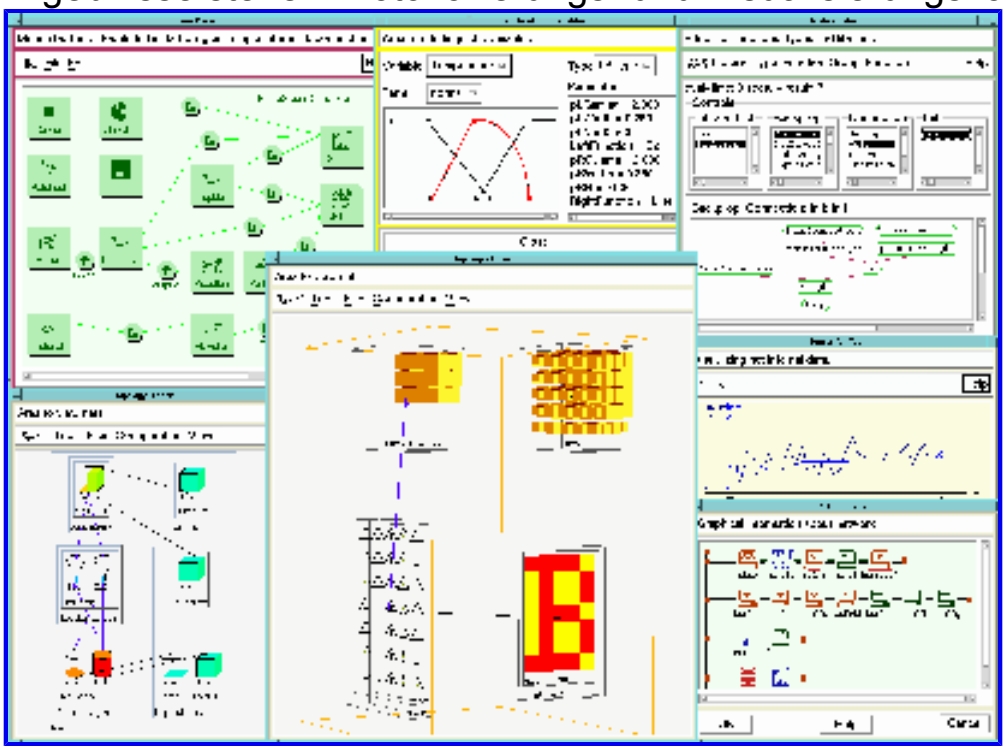

Abb.: Ein Bildschirmschnappschuß mit den Komponenten (beginnend links oben, im Uhrzeigersinn): Projekt-Manager, Editor für Fuzzy-Zugehörigkeitsfunktionen, Modell-Editor, Diagramm-Werkzeug, Parallelisierungskomponente, 3D- und 2D-Darstellung eines Netzes im Topologie-Editor

NeuroGraph unterstützt alle Paradigmen des Soft-Computing. Diese können einzeln oder in beliebiger Kombination verwendet werden.

NeuroGraph ermöglicht die Konstruktion, den Test und die Ausführung beliebiger neuronaler Netze. Dafür stehen vor- und benutzerdefinierte Netzwerkmodelle zur Verfügung. Die Anzahl der Komponenten eines Netzwerkes (Schichten, Neuronen, Verbindungen etc.) ist dabei nur durch den zur Verfügung stehenden Speicherplatz beschränkt. Es können sowohl einzelne Netze wie auch Zusammenschaltungen von Netzen konstruiert werden, wobei jede Komponente einzeln und in Verbindung mit den anderen trainiert und ausgeführt werden kann. Weiterhin können hybride Systeme, z. B. Neuro-Fuzzy-Systeme oder durch genetische Algorithmen optimierte neuronale Netze entwickelt werden.

Für evolutionäre und genetische Algorithmen stellt NeuroGraph alle Standardoperatoren wie Mutation, Selektion und Rekombination zur Verfügung. Die Beschreibung der Bewertungs- und Fitness-Funktionen erfolgt durch den Benutzer selbst, da diese Funktionen problemabhängig sind. NeuroGraph stellt dafür ein graphisches Werkzeug zur Verfügung, so daß keinerlei Programmierkenntnisse notwendig sind.

Fuzzy-Logik-Systeme werden in NeuroGraph durch die Angabe der linguistischen

Variablen, Zugehörigkeitsfunktionen, Regeln und Defuzzyfizierungsmethoden beschrieben. Die wichtigsten Methoden werden von NeuroGraph standardmäßig zur Verfügung gestellt. Die Regeln und Zugehörigkeitsfunktionen werden mit einem graphischen Werkzeug interaktiv beschrieben.

2.4 Anwendungen

Zur Demonstration der Fähigkeiten des Soft-Computing werden einige klassische Experimente des Soft-Computing vorgestellt. Dies sind:

- eine neuronale Datenbank,
- Feinmotorik (Balancieren eines Balls auf einer Wippe) und
- ein System, das englische Sprache laut vorlesen kann (Sprachsynthese).

Soft-Computing wird unter drei Aspekten dargestellt:

- Prinzip Soft-Computing stellt einen vielversprechenden Ansatz dar, einige bisher nicht handhabbare Probleme zu lösen. Die Paradigmen beruhen auf Prinzipien, die bereits in der Natur ihre Leistungsfähigkeit unter Beweis gestellt haben. Durch Anpassung an die technischen Erfordernisse ist es möglich, Soft-Computing im Rahmen industrieller Software-Produktion einzusetzen. Dazu ist einerseits ein fundiertes Wissen auf dem Gebiet des Soft-Computing notwendig, andererseits stehen die Software- und Hardwareressourcen zur Verfügung, um auch anspruchsvolle Probleme in Angriff zu nehmen.
- Simulation Für die Entwicklung von Soft-Computing-Anwendungen ist die Simulation fast unabdingbare Voraussetzung. Die Simulation hat sich als ökonomisch und ökologisch sinnvolle Alternative zu anderen Methoden etabliert. Gerade im Bereich Soft-Computing sind viele Experimente und Entwicklungen erst bzw. nur durch Simulation möglich. Für den Entwickler steht eine Vielzahl von Entwicklungswerkzeugen zur Verfügung. Die Bandbreite reicht von einfachsten Programmen bis zu kompletten Entwicklungsumgebungen und vom PC bis zum Multi-Prozessor-System.
- Anwendung Auf den Gebieten neuronale Netze und Fuzzy-Logik hat das Soft-Computing die Labors bereits verlassen und in Form von Produkten Einzug in unser Leben gehalten. Bei evolutionären Algorithmen ist der Ressourcenbedarf momentan noch so hoch, daß Produkte noch nicht angekündigt sind. Allerdings existieren schon Produkte im Handel, die unter Verwendung von evolutionären Algorithmen entwickelt wurden.

Soft-Computing hat sich in den letzten Jahren rasant entwickelt. Es ist zu hoffen, daß sich diese Entwicklung in Zukunft fortsetzt und viele nützliche Produkte hervorbringt. (Wilke)

3 Die Verarbeitung verbaler Phraseologismen

Im Bereich der maschinellen Verarbeitung natürlicher Sprache standen lange Zeit Idiome wie kick the bucket und ins Gras beißen im Mittelpunkt. Sie werden als nichtkompositionell bezeichnet, da ihre Komponenten weder mit ihrer literalen noch mit einer ihrer metaphorischen Bedeutungen in die Gesamtbedeutung der Idiome, nämlich "sterben", eingehen. Solche Idiome können als ganzes im Lexikon eingetragen werden und sind so bei der Verarbeitung unproblematisch.

Bei Idiomen wie jmdm. einen Bären aufbinden, einen Bock schießen, Staub aufwirbeln, Blech reden geht ebenfalls keine der in den Idiomen auftretenden Komponenten in ihrem wörtlichen Sinne in die idiomatische Gesamtbedeutung ein. Sie lassen aber dennoch stilistisch unmarkierte Modifikationen und Umformungen zu, wie in den folgenden Beispielen gezeigt wird:

- Jede Menge Staub wirbelte die Rede des Gewerkschaftssprechers auf.
- Er hat in unserer Firma so manchen großen Bock geschossen.
- Was für einen Bären hat sie dir diesmal aufgebunden?
- In welches Fettnäpfchen ist er nun schon wieder getreten?
- Den Bock, den er auf der Sitzung geschossen hat, wird man ihm nicht so schnell verzeihen.

Es ist möglich, die Gesamtbedeutung dieser Idiome zu modifizieren, indem die Bedeutung einer Komponente des Idioms modifiziert bzw. eine Komponente des Idioms durch Anwendung einer Umformungsoperation fokussiert wird. Die Komponente selbst trägt also eine herausfilterbare Teilbedeutung, die in die idiomatische Gesamtbedeutung eingeht. Die idiomatische Gesamtbedeutung kommt demnach auf einer figurativen Ebene kompositionell zustande. Bei einer Bedeutungsangabe eines solchen übertragen-kompositionellen Idioms über eine komponentennahe Paraphrase wird versucht, zunächst die einzelnen Teilbedeutungen der idiomatischen Komponenten zu identifizieren und diese dann als wörtliche Komponenten in der Paraphrase abzubilden. Es erfolgt sozusagen eine Übersetzung der idiomatischen Komponenten in den wörtlichen Sprachgebrauch, d. h. eine Übersetzung in die "eigentlich gemeinte Bedeutung". Zwei Beispiele werden in der folgenden Tabelle gezeigt:

einen	Bock	schießen
einen	Fehler	machen
einen	Bären	aufbinden
eine	Lügengeschichte	erzählen

Die identifizierbaren Teilbedeutungen von Idiomen werden auch als semantisch autonom bezeichnet.

Daraus folgt, daß man ein übertragen-kompositionelles Idiom wie einen Bock schießen insofern als zerlegbar bzw. dekomponierbar betrachten kann, als daß es dazu verwendet wird, auf ein komplexes Ereignis, nämlich "Einen-Fehler-machen", zu

referieren, wobei einzelne Rollen dieses Ereignisses identifiziert werden können: der "Fehlermachende" und der "Fehler". Dieses Ereignis kann demnach als eine offene Relation Rxb betrachtet werden, wobei die Akkusativnominalphrase des Idioms auf b, den Fehler, referiert und das Verb auf R, während x die offene Position des Referenten der Subjektnominalphrase repräsentiert.

Auf der Grundlage dieser Ideen ist es möglich, auch kompositionelle Idiome einfach zu parsen. Mit Hilfe eines am IMMD VIII entstandenen Parsers wurde ein Verfahren entwickelt, wie die internen Modifikationen bzw. die Umformungen solcher Idiome dargestellt werden können. Als semantischer Formalismus wurde der von M. Pinkal, Universität des Saarlandes, entwickelte Formalismus der lambda-DRT verwendet. Dort dienen Diskursrepräsentationsstrukturen als Datenstrukturen, die beim Parsing kompositionell mit Hilfe der funktionalen Komposition verbunden werden.

Jeder Komponente des Idioms wird nun im semantischen Lexikon sein Teil der Paraphrase zugeodnet. Im syntaktischen Lexikon enthalten die einzelnen Einträge eines Idioms Hinweise auf noch fehlende Bestandteile. Zusätzliche Grammatikregeln garantieren, daß Idiome richtig zusammengesetzt werden. Wird nun ein Idiom intern modifiziert, kann diese Modifikation mit Hilfe der normalen Grammatikregeln durchgeführt werden. Die kompositionelle Bildung neuer Diskursrepräsentationsstrukturen garantiert, daß der richtige Teil der Paraphrase mit der Modifikation verbunden wird. (Fischer, Keil)

4 Ein generisches Organisationsschema der Konfigurationsverwaltung im Software-Engineering

4.1 Einführung

In den letzten Jahren wurden die Grenzen üblicher Verfahren des Software-Engineering, zum Teil auch aufgrund der Größe der Projekte, immer deutlicher an steigenden Kosten und abnehmender Planbarkeit von Software-Projekten sichtbar. Oft ähneln sich die Probleme verschiedener Projekte; Planung und Wirklichkeit liegen weit auseinander, die Lücke zwischen den verschiedenen Abstraktionsgraden von Lastenheft und Spezifikation wird nicht durch Zwischenstufen ausgefüllt, der Projektfortschritt läßt sich an den Arbeitsergebnissen nicht ablesen, Vorgehensweisen und Aufgabenverteilungen sind unklar, projektbezogene Pläne liegen gleichzeitig in verschiedenen Versionen vor. Dies alles sind Situationen, die nicht nur den Projektfortschritt, sondern auch dessen Transparenz einschränken. Es wird immer deutlicher, daß bei der Entwicklung von Software nicht die einzelne Entwurfstechnik, sondern eine neue, von Aufgabenstellungen und Methoden unabhängige, übergreifende Denk-, Organisations- und Arbeitsweise erforderlich ist, um die allein schon durch die Komplexität heutiger technischer Systeme verursachten Probleme in den Griff zu bekommen.

Etwa seit Beginn der neunziger Jahre werden Verfahren der Qualitätssicherung auch bei der Entwicklung von Software angewendet. Allgemeine Qualitätssicherungspläne geben allerdings nur eine Orientierungshilfe bei der Planung der Projektverwaltung und Kontrolle in einem konkreten Software-Entwicklungsprojekt. Zuviele Details hängen von der Aufgabenstellung, den gewählten Entwurfsmethoden, Programmiersprachen und der jeweiligen bestehenden Unternehmensorganisation ab. Die in der Norm ISO 9000, Teil 3 [2] beschriebenen qualitätssichernden Maßnahmen betonen analytische Verfahren, setzen aber als Grundlage die immer wieder an die jeweilige Unternehmung angepaßte Projektplanung und Konfigurationsverwaltung voraus. Dokumentenlenkung, Konfigurationsverwaltung, Werkzeugintegration, Projektplanung und -dokumentation sind Aspekte, die sich von den konkreten Methoden und Inhalten eines Projektes trennen, sich als organisatorische Grundlage festlegen und zudem formalisieren lassen. Da Software heute auf dem Rechner und mit rechnerbasierten Verfahren erstellt wird und technische Entwürfe, Dokumente und Korrespondenz im Umfeld technischer Unternehmungen sowieso schon auf Rechnern erstellt und mit Rechnern weitergegeben und verwaltet werden, bietet sich die weitgehende Umsetzung der automatisierbaren organisatorischen Maßnahmen der Qualitätssicherung (QS) nach der Norm ISO 9000, Teil 3, auf ein rechnergestütztes Verfahren an.

Ausgehend von der Überzeugung, daß Rechner im Software-Engineering einerseits zu oft mit zu phantastischen Erwartungen eingesetzt werden und daß andererseits noch vieles im Entwicklungsprozeß durch Rechner praktisch unterstützt werden kann, werden in diesem Projekt Verfahren entwickelt, die ein Team von Entwicklern unterstützen und lediglich allgemeine Voraussetzungen machen, die in der Praxis

leicht einzuhalten sind. Als Ergebnis des Projekts wird ein methoden- und projektunabhängiges formales System zur Unterstützung der Konfigurationsverwaltung und qualitätssichernder Maßnahmen vorgestellt. Das dazu gehörende Verfahren der Konsistenzkontrolle benutzt eine interessante Analogie zwischen dem Software-Entwicklungsprozeß und der Konstruktion globaler, konsistenter Zustände in verteilten Berechnungen.

Die Bedeutung des menschlichen Urteils in den Ergebnissen von Reviews soll durch rechnergestützte Qualitätssicherungsverfahren keineswegs gemindert werden. Rechnergestützte Verfahren erlauben vielmehr die Zuordnung von formalen Dokumenten zu den zugehörigen Kommentaren und Bewertungen sowie umgekehrt das Auffinden von formalen Dokumenten mittels informeller Hinweise. Das in diesem Projektbericht beschriebene Modell soll nicht die menschliche Vernunft in irgendeiner Situation des SoftwareEngineering ersetzen, sondern deren Einfluß auf die weiteren Projektaktivitäten erhöhen. Das Chaos der Versionen und die Unklarheit über den Projektfortschritt soll durch eine Konfigurationsverwaltung sowie Übersichten über den Projektzustand ersetzt werden.

4.2 Abgrenzung der Forschungsarbeiten

Die Unterstützung von Methoden durch Werkzeuge sowie die Verwendung von Übersetzern für höhere Programmiersprachen erhöhen die Qualität von Software ganz wesentlich. Die Bedeutung höherer Programmiersprachen für Effizienz und Erfolg des Software-Engineering kann als bekannt vorausgesetzt werden. Entscheidend ist hier lediglich, daß Notationen auf abstrakten Ebenen (wie z. B. graphische Darstellungen) mit Generatoren in einer korrekten und reproduzierbaren Weise auf maschinennahe Notationen abgebildet werden können.

Ausgangspunkt unserer Forschung ist die sogenannte "analytische" Vorgehensweise, die den "top-down"-Ansatz unterstützt. Bei Verwendung der "top-down"-Methode werden die aus der Sicht der Softwaretechnik abstrakten, aber anwendungsnahen Formulierungen des Lastenheftes im Laufe von sogenannten Phasen durch Einfließenlassen von Implementierungsentscheidungen konkreter, bis schließlich eine softwaretechnisch konkrete Formulierung, d. h. Programmcode, entstanden ist. Eine Anordnung solcher Phasen mit der Angabe der Abhängigkeiten von Phasenergebnissen und Vorgaben heißt Lebenszyklusmodell, wobei die Reihenfolge der Phasen durch die Abhängigkeiten bestimmt wird, nicht aber den Informationsfluß oder die Arbeitsweise des Software-Entwicklers festlegt, der ja gelegentlich zu einer im Lebenszyklusmodell "früheren" Phase zurückkehren und seine bisherige Arbeit modifizieren will.

Software-Entwicklungsprojekte umfassen Dokumente, von denen verschiedene konkrete Ausprägungen unterschieden werden müssen, die wir Revisionen nennen. Die Revisionen werden mit Transformatoren und Editoren (allgemein: Werkzeuge) bearbeitet, und zwar von Projektteilnehmern, deren Rollen im Projekt ebenfalls unterschieden werden müssen. Das Projektergebnis besteht schließlich nicht nur aus der ausführbaren Programmdatei, sondern aus allen zu deren Erstellung benötigten Projektdokumente mit den jeweils verwendeten Revisionen. Diese Revisionsmenge ist die Konfiguration. Die sogenannte Konfigurationsverwaltung identifiziert und

dokumentiert jede Änderung der Dokumente, zudem verwaltet sie die Überprüfung und Freigabe von Produktversionen. Die Rekonstruierbarkeit von Software wird sichergestellt, indem die verschiedenen Revisionen einzeln identifiziert und einer Konfiguration zugeordnet werden. Somit ist der Aufbaustatus einer Konfiguration ermittelbar, und die Bearbeitung von Dokumenten an verschiedenen Standorten und durch verschiedene Bearbeiter wird koordiniert. Die Dokumentenlenkung macht die Dokumente, die zum Aufbau der jeweiligen Konfiguration benötigt werden, in der aktuellen Ausprägung an den Stellen verfügbar, die diese Dokumente be- bzw. verarbeiten. Zu diesen Stellen gehören Projektteilnehmer ebenso wie Werkzeuge, die Dokumente verarbeiten und daraus andere erstellen. Änderungen an Dokumenten sind nur den erstausgebenden Stellen möglich und führen zu einer neuen Revision des jeweiligen Dokuments. Eine Feststellung des laufenden Änderungszustands schließt die Verwendung nicht anzuwendender ("abgelaufener") Revisionen von Projektdokumenten aus. Alle Dokumente werden mit ihren Revisionen in einer Projektdatenbank verwaltet, die sowohl ein Lebenszyklusmodell als auch die Möglichkeit der Dokumentenlenkung unterstützt.

4.3 Ergebnisse

Unser Ziel war es, diese automatisierbaren konstruktiven Maßnahmen der Qualitätssicherung in einem generischen Meta-Werkzeug der Software-Entwicklung zu integrieren. Ein wichtiger Meilenstein bei unseren Arbeiten war daher die Definition eines generischen Lebenszyklusmodells, das neben den Phasen auch die beteiligten Dokumente, Werkzeuge, Projektbearbeiter (Rollen im Projekt) und deren Beziehungen untereinander darstellen kann und das darauf aufbauend die Aufgaben von Konfigurationsverwaltung und Dokumentenlenkung übernimmt [7, 8, 9].

Eine Analogie zwischen der Entwicklung technischer Produkte sowie der Ermittlung konsistenter globaler Zustände in verteilten Berechnungen ist die Grundlage eines von uns entwickelten Markierungsmodells über der Projektdatenbank. Mit diesen Markierungen können auf elegante Weise viele Aufgaben der Koordination und Kontrolle realisiert werden. Wir haben eine Liste elementarer Aufgaben von Konfigurationsverwaltungen und Qualitätssicherungssystemen formuliert und gezeigt, wie sie unter Verwendung unseres Markierungsmodells elegant gelöst werden können [3, 5].

Die Beobachtung, daß die Entwicklung keinesfalls ein total geordneter Vorgang ist, sondern immer wieder Verzweigungen und Vereinigungen in der Geschichte der Revisionen je eines Dokuments vorkommen können, hat uns veranlaßt, das Markierungsverfahren zu verallgemeinern, so daß nunmehr partiell geordnete Markierungen zur Klassifikation und Selektion von Revisionen verwendet werden. Bei der Erzeugung einer Revision kann der Entwickler insbesondere klassifizierende Merkmale durch die zugehörige Revisionsmarkierung ausdrücken. Ein solches Merkmal kann beispielsweise die Eignung der erstellten Revision für eine bestimmte Datenbank oder Bedienoberfläche beschreiben. Anhand vorgegebener Markierungen kann später eine Konfiguration spezifiziert werden, wobei ein Selektionsverfahren die zu den vorgegebenen Merkmalen passend markierten Revisionen auswählt und der Konfiguration zuordnet. Bei der Integration eines derartigen Selektionsverfahrens mit

den Dependenzen, die durch das generische Prozeßmodell beschrieben werden, entsteht eine Art intensionales Konstruktionsverfahren für Konfigurationen, das genau die gleichen Aufgaben wahrnehmen kann wie das oben vorgestellte Markierungsverfahren, jedoch mit sogenannten sprechenden Markierungen, die (intensional) Merkmale der jeweils ausgewählten Revisionen und der konstruierten Konfiguration beschreiben können [6]. Die Datenstruktur der Merkmale für Revisionen und Konfigurationen bildet einen Verband, auf den die vorher für total geordnete Markierungen formulierten Gesetzmäßigkeiten sinngemäß übertragen wurden.

Abschließend haben wir eine Projektdatenbank entworfen, mit der das oben vorgestellte generische Lebenszyklusmodell sowie das dazugehörige Markierungsverfahren realisiert werden können [4]. Die oben eingeführten Markierungen für Dokumente, Revisionen und Konfigurationen sind die Primärschlüssel entsprechender Tabellen. Neben einem relationalen Datenbankschema haben wir auch alle notwendigen elementaren Transaktionen zur Beschreibung der Projektaktivitäten definiert. Die elementaren Transaktionen dienen dabei zum Aufbau "langer Transaktionen", mit denen die Auswirkungen von Änderungen über alle Phasen des Projektes verfolgt und gesteuert werden können. Die elementaren Transaktionen sind gewissermaßen der Befehlssatz, über dem die Entwickler (als Bediener dieser Projektdatenbank) "lange Transaktionen" formulieren können. Während die elementaren Transaktionen atomar, isoliert und mit einer begrenzten Menge von Daten (hier: Revisionen) arbeiten, stellen die "langen Transaktionen" die Konsistenz und die Dauerhaftigkeit der bearbeiteten Daten (nämlich: Konfigurationen) her. Die Arbeitsdaten jeder "langen Transaktion" sind dabei als Konfiguration "im Aufbau" zu interpretieren. Die "langen Transaktionen" ermöglichen die "dokumentenorientierte" Software-Entwicklung, wie sie von Denert motiviert wird, weil sie die Weitergabe von Änderungen auf alle abhängigen Dokumente sicherstellen [1]. Die Revisionen werden in unserer Projektdatenbank niemals modifiziert oder aus ihrer Konfiguration entfernt oder vertauscht, so daß die zentrale Anforderung an Qualitätssicherungssysteme nach Rekonstruierbarkeit strikt eingehalten wird.

Literatur:

[1] Denert, E.: "Dokumentenorientierte Software-Entwicklung". - Informatik-Spektrum, 16, 1993, S. 159 - 164

[2] Dt. Institut für Normen: "DIN EN ISO 9000". (Teil 3), Berlin, 1993

[3] Heidenreich, G./Kips, D.: "A general model for engineering databases". - In: Soc. for Design and Process Science, Proc. 1st World Conf. on Integrated Design and Process Technology, Ertas, Atila (Hrsg), Dallas, TX, 1995, S. 347 - 352

[4] Heidenreich, G./Kips, D./Volle, V.: "A database for configuration management". - In: Proc. 7th Intern. Conf. and Workshop on Database and Expert System Applications (DEXA 96), Wagner et al. (Hrsg.), Los Alamitos, CA, 1996. IEEE, S. 243 - 248

[5] Heidenreich, G./Minas, M./Kips, D.: "A new approach to consistency control in software engineering". - In: Proc. 18th Intern. Conf. on Software Engineering, Rombach, D. (Hrsg.), Los Alamitos, CA, 1996. IEEE, S. 289 - 297

[6] Heidenreich, G./Minas, M./Landauer, J.: "Interval construction within partially ordered object versions". - In : Proc. Conf. on Interval Methods and Computer

Aided Proofs (INTERVAL 96), Wolff von Gudenberg et al. (Hrsg.), Kluwer, 1996
[7] Heidenreich, G.: "Ein generisches Organisationsschema der Konfigurationsverwaltung im Software-Engineering", Dissertation. - Arbeitsber. des IMMD der Univ. Erlangen-Nürnberg, Band 29, Nr. 13, 1996
[8] Kips, D./Heidenreich, G.: "Projektflußgraphen - Ein Meta-Modell zur Qualitätssicherung im Software-Engineering". - In: Proc. QUALITY 94, Kongreßgesellschaft mbH, Stuttgarter Messe (Hrsg.), Ulm, Mai 1994. Müller Adreß- und Neue Medien GmbH, S. 282 - 287
[9] Kips, D./Heidenreich, G.: "Project flow graphs - a meta model to support quality assurance in software engineering". - In: Proc. Int. Conf. on Industrial Engineering and Production Management '95, Band 1, FUCaM (Hrsg.), Mons, Belgium, April 1995. S. 347 - 357

(Heidenreich)

Dissertation:

Heidenreich, G.:

Ein generisches Organisationsschema der Konfigurationsverwaltung im Software-Engineering

Diplomarbeiten:

Balzer, P.:
: Konzeption, Realisierung und Integration eines objektorientierten Domänenframeworks

Behrend, H.:
: Implementierung und Vergleich von Algorithmen zur Simulation von Soft-Computing-Systemen auf verteilten Rechnersystemen

Gronemeyer, S.:
: Gestaltbildung und Koevolution modelliert mit L-Systemen

Haake, B.:
: Entwurf, Implementierung und Analyse eines 'Profilers' für Split-C

Hauth, A.:
: Integration effizienter Mechanismen für Kommunikation und Multithreading in eine parallele objektorientierte Programmiersprache

Janaen, O.:
: Klassendesign zur Berechnung von Getriebelagerungen

Lehmkuhl, R.:
: Konzeption und Realisierung von Visualisierungselementen für ein grafisches Projektierungssystem

Löser, A.:
: Entwurf und Realisierung einer Graphik-Klassenbibliothek in der Programmiersprache Java

Plack, J.:
: Reimplementation einer Bedienoberfläche mit Integration einer Hilfekomponente

Spielvogel, R.:
: Ein Vorgehensmodell zur Erfassung und Interpretation von Software-Metriken

Studt, R.:
: Visualisierung von sich referenzierenden Objekten

Tsitlaidis, L.:
: Bewertung der Integrationsfähigkeit von Softwarekomponenten für industrielle Rechnernetze in höhere Programmierumgebungen

Volle, K. V.:
: Eine prozeßbezogene Konfigurationsverwaltung

Studienarbeiten:

Abt, J.:
: Entwicklung einer virtuellen Maschine für den Scheme-Interpreter 'Guile'

Dauerlein, M.:
: Spezifikation verteilter Algorithmen mit Graphersetzungssystemen

Deseive, S.:
: Die Modellierung von Aktorsystemen durch Graphgrammatiken

Goebel, H.:
: Portierung von Oberon auf Alpha AXP

Gottschall, J.:
: Die Darstellung von Petri-Netzen mittels Graphgrammatiken

Gröbner, M.:
: Entwurf und Implementierung eines Berechnungsschemas für die Simulation biologischer neuronaler Netze

Guth, G.:
: Erweiterung der Sprache Guile um generische Funktionen

Köhn, J.:
: Implementierung eines E/A-Editors für ein Simulatorsystem für Soft-Computing (NeuroGraph)

Metin, M.:
: Vergleich von Simulatoren für Soft-Computing auf PC's, Workstations und verteilten Rechnersystemen

Mohr, V.:
: Entwurf und prototypische Realisierung grundlegender Funktionen der Konfigurationsverwaltung

Stenzel, W.:
: Entwurf und Implementierung einer Modellbeschreibung für selbstorganisierende neuronale Netze

Stingl, B.:
: Die Verwendung von Graphtransformationssystemen zur Beschreibung paralleler Logikprogrammierung

Wetzel, C.:
: Erstellung einer Morphologie für Italienisch in der Programmierumgebung Malaga

Woppmann, H.:
: Modellierungen zur Evolution natürlicher Genome

Zobel, M.:
: Optimierung einer Produktionsplanung mit genetischen Algorithmen

Veröffentlichungen:

Feldmann, K./ Götz, K./ Wilke, P.:
"Prozeßbegleitende Qualitätssicherung in der Elektronikproduktion", Tagungsband Symposium Anwendung von Fuzzy-Technologien und Neuronalen Netzen, Erlangen, Transferstelle Mikroelektronik/Fuzzy Technologien (Hrsg.), Aachen, 1996, S. 41 - 48

Feldmann, K./ Götz, K./ Wilke, P.:
"Fehlerklassifikation auf Röntgenbildbasis", Tagungsband Symposium Anwendung von Fuzzy-Technologien und Neuronalen Netzen, Transferstelle Mikroelektronik/Fuzzy-Technologien, (Hrsg.), Aachen, 1996, S. 27 - 32

Fischer, I./ Geistert, B./ Görz, G.:
"Incremental Anaphora Resolution in a Chart-based Semantics Construction Framework using lambda-DRT". - In: Proc. of the Discourse Anaphora and Anaphor Resolution Colloquium (DAARC96), Botley, S./ Glass, J. (Hrsg.), Lancaster, Großbritannien, Juli, 1996, S. 235 - 244

Fischer, I./ Keil, M.:
"Representing Phraseologisms with Discourse Representation Theory". In: Dialogue'96 - Computational Linguistics and its Application, A.S. Narin'yani (Hrsg.), Puschino, Rußland, Mai 1996, S. 182 - 286

Fischer, I./ Keil, M.:
"Parsing idioms". - In: Proc. of the 16th Intern. Conf. on Computational Linguistics, Kopenhagen, Dänemark, August 1996, S. 388 393

Fischer, I./ Keil, M.:
"Von großen Böcken und unglaublichen Bären - Zur Verarbeitung verbaler Phraseologismen". - In: Natural Language Processing and Speech Technology, Results of the 3rd Konvens Conference in Bielefeld, Dafydd Gibbon (Hrsg.), Mouton de Gruyter, Berlin, Oktober 1996, S. 200 - 211

Heidenreich, G./ Kips, D./ Volle, V.:
"A database for configuration management". - In: Proc. 7th Intern. Conf. and Workshop on Database and Expert System Applications (DEXA 96), Wagner et al. (Hrsg.), Los Alamitos, CA, 1996. IEEE, S. 243 - 248

Heidenreich, G./ Minas, M./ Kips, D.:
"A new approach to consistency control in software engineering". - In: Proc. 18th Intern. Conf. on Software Engineering, Rombach, D. (Hrsg.), Los Alamitos, CA, 1996. IEEE, S. 289 - 297

Heidenreich, G./ Minas, M./ Landauer, J.:
"Interval construction within partially ordered object versions". - In : Proc. Conf. on Interval Methods and Computer Aided Proofs (INTERVAL 96), Wolff von Gudenberg et al. (Hrsg.), Kluwer, 1996

Heidenreich, G.:
"Ein generisches Organisationsschema der Konfigurationsverwaltung im Software-Engineering", Dissertation. - Arbeitsber. des IMMD der Univ. Erlangen-Nürnberg, Band 29, Nr. 13, 1996

Jacob, C.:
"Evolving Evolution Programs". - In: Proc. First Annual Conference on Genetic Programming, Stanford, USA (1996), MIT Press, S. 107 - 115

Jacob, C.:
"Evolution Programs Evolved". In: Proc. PPSN-IV, Parallel Problem Solving from Nature IV, H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Hrsg.), Lecture Notes in Comp. Sc. 1141, Berlin, (1996), Springer, S. 42 - 51

Kókai, G./ Alexin, Z./ Gyimóthy, T.:
"Classifying ECG Waveforms in Prolog". - In: Proc. of the Fourth International Conference on the Practical Application of Prolog London, Großbritannien, 23. - 25. April 1996, S. 173 - 201

Kókai, G./ Alexin, Z./ Gyimóthy, T.:
"Analyzing and Learning ECG Waveforms". - In: Proc. of The Sixth International Workshop on Inductive Logic Programming (ILP'96) Stockholm, Schweden, 28. - 30. August, 1996, S. 152 - 171

Kókai, G./ Alexin, Z./ Gyimóthy, T.:
"Learning Biomedical Patterns". - In: INAP-96 The 9th Exhibition and Symposium on Industrial Applications of Prolog, 16. - 19. Oktober 1996, Hino, Tokio, Japan, S. 159 - 168

Minas, M./ Shklar, L.:
"A high-level visual language for generating web structures". - In: Proc. IEEE Symposium on Visual Languages (VL'96), Boulder, Colorado, IEEE Computer Society Press, Sept. 1996, S. 284 - 285

Minas, M./ Shklar, L.:
"Visual definition of virtual documents for the world-wide web". - In: Preliminary Proc. of the Third International Workshop on Principles of Document Processing (PODP'96), Springer-Verlag, S. 193 - 204

Minas, M./ Shklar, L.:
"Visualizing information repositories on the world-wide web". - In: Herausforderungen an die Informationswirtschaft, J. Krause, M. Herfurth, and J. Marx (Hrsg.), (Proc. 5th Int. Symp. for Information Science, ISI'96, Berlin), S. 297 - 309

Schneider, H. J.:
"On outward and inward productions in the categorical graph-grammar approach and Delta-grammars". Math. Struct. in Comp. Science 6 (1996), S. 527 - 543

Wilke, P.:
"Soft-Computing - Prinzip, Simulation und Anwendung", Habilitationsschrift, Erlangen 1996, 440 Seiten

Vorträge:

Berns, K.:
"Lernende, sich an die Umwelt anpassende Steuerungsansätze am Beispiel der sechsbeinigen Laufmaschine LAURON" Informatik-Kolloquium der Univ. Erlangen-Nürnberg 11. 12. 1996

Fischer, I.:
"Representing Phraseologisms with Discourse" Representation Theory Dialoge 1996, Puschino, Rußland 07. 05. 1996

Fischer, I.:
"Parsing idioms" Coling 1996, Kopenhagen, Dänemark 09. 08. 1996

Fischer, I.:
"Java - Die Programmiersprache des Jahres 2000?" Betriebsseminar der Firma Basys GmbH, Erlangen, Schwend/Opf. 15. 09. 1996

Fischer, I.:
"Another Approach to Model Actor Systems with Graph Transformations" Dagstuhl-Seminar "Graph-Transformations in Computer Science". Dagstuhl 17. 09. 1996

Grabska, E.:
"Creative Design Aided by Composite Representation" Informatik-Kolloquium der Univ. Erlangen-Nürnberg 04. 09. 1996

Jacob, C.:
"Mathematica 3.0 - Konzepte der neuen Mathematica-Version" CeBIT, Wolfram Research, Hannover 16. - 19. 03. 1996

Jacob, C.:
"Mit der Evolution rechnen - Einblicke in die Welt der evolutionären Programmierung" Fraunhofer Institut für Integrierte Schaltungen, Erlangen-Tennenlohe 16. 04. 1996

Jacob, C.:
"Simulierte Evolution von Entwicklungsprogrammen der Natur" Tag der Erlanger Informatik 26. 04. 1996

Jacob, C.:
"Wenn Darwin programmieren würde ..." Siemens AG, Anlagentechnik, Erlangen 03. 06. 1996

Jacob, C.:
"Evolvica - Simulated Evolution of Development Programs in Nature" Mathematical Sciences Department, Universität Calgary, Kanada 25. 07. 1996

Jacob, C.:
"Evolving Evolution Programs: Genetic Programming and L-Systems" Genetic Programming Conference 1996, Stanford, CA, USA 30. 07. 1996

Jacob, C.:
"Evolution Programs Evolved" (Posterpräsentation) PPSN-IV Conference, Parallel Problem Solving from Nature, Berlin 23. 09. 1996

Kókai, G.:
"Classifying ECG Waveforms in Prolog" Intern. Conf. on the Practical Application of Prolog London, Großbritannien 24. 04. 1996

Kókai, G.:

"Analyzing and Learning ECG Waveforms" Sixth International Workshop on Inductive Logic Programming (ILP'96) Stockholm, Schweden 30. 08. 1996

Minas, M.:
"Automatische Generierung graphischer Editoren" Informatik-Kolloquium der Univ. Stuttgart 30. 04. 1996

Minas, M.:
"Automatische Generierung graphischer Editoren" Fa. INA, Herzogenaurach 09. 05. 1996

Minas, M.:
"Java - die Programmiersprache für das Internet" Fa. 3SOFT, Erlangen-Tennenlohe 24. 05. 1996

Minas, M.:
"A High-Level Visual Language for Generating Web Structures" (Posterpräsentation) VL '96 (Boulder, CO) 03. - 06. 09.1996

Minas, M.:
"Visualizing Information Repositories on the World-Wide Web" ISI'96, Berlin 17. 10. 1996

Minas, M.:
"Spezifikation und Generierung graphischer Editoren" TFS der TU Berlin 28. 11. 1996

Rauch Henzinger, M.:
"A Survey of Dynamic Graph Algorithms" Informatik-Kolloquium der Univ. Erlangen-Nürnberg 11. 07. 1996

Wilke, P.:
"Soft-Computing - Prinzip, Simulation und Anwendung" Informatik-Kolloquium RWTH Aachen, Aachen 31. 05. 1996

Wilke, P.:
"Hard- und Software für Soft-Computing" Carl-Cranz-Gesellschaft CCG, Oberpfaffenhofen 03. 07. 1996

Wilke, P.:
"Simulation von Soft-Computing-Anwendungen" Carl-Cranz-Gesellschaft CCG, Oberpfaffenhofen 4./5. Juli 1996