Deduction-Based Software Component Retrieval

Bernd Fischer

Universität Passau und
USRA/RIACS, NASA Ames Research Center
fisch@email.arc.nasa.gov

Deduktionsbasiertes Softwarekomponenten-Retrieval ist eine formale, semantik-basierte Methode zur Unterstützung der Software-Wiederverwendung. Dabei werden formale Spezifikationen als Komponentendeskriptoren und als Suchanfragen verwendet und passende Komponenten mit Hilfe automatischer Theorembeweiser ermittelt. Die direkte Realisierung dieses Konzepts scheitert allerdings an der Anzahl und am Profil der entstehenden Beweisbedingungen. Die Dissertation [Fis01] stellt einen alternativen Ansatz vor, der auf einer inkrementellen Filterpipeline basiert und so die Theorembeweiser entlastet. Der vorliegende Beitrag beschreibt verschiedene Wiederverwendungsmodelle, die mit Hilfe des deduktionsbasierten Softwarekomponenten-Retrievals realisiert werden können, erläutert das NORA/HAMMR-System, in dem dieser Ansatz implementiert ist, und fasst die durchgeführte ausführliche experimentelle Auswertung zusammen.

1 Einführung

Die Wiederverwendung ausgereifter Komponenten ist eines der charakteristischen Konzepte der klassischen Ingenieursdisziplinen. Im Software Engineering hat sich dieses Konzept allerdings als sehr viel schwieriger zu realisieren herausgestellt als anfänglich angenommen. Insbesondere das Ariane5-Unglück von 1996 hat die Probleme und Risiken der Software-Wiederverwendung verdeutlicht: bei der abschließenden Untersuchung [Lio96] hat sich ergeben, dass das Unglück von einer ohne Änderungen aus der Ariane4 übernommenen Softwarekomponente verursacht wurde, die unter den geänderten Anforderungen nicht mehr korrekt funktionierte.

Jézéquel und Meyer [JM97] haben daher argumentiert, dass eine erfolgreiche Software-Wiederverwendung der Unterstützung durch formale, semantik-basierte Methoden bedarf. Insbesondere haben sie darauf hingewiesen, dass die tiefere Ursache des Ariane5-Unglücks das Versäumnis war, die genauen Anforderungen der wiederverwendeten Komponente in der Form eines *Kontrakts* (d.h. formales Verhaltensmodell) explizit zu machen:

> "*reuse without a contract is a sheer folly.* ... The Ariane 5 blunder shows clearly that naïve hopes are doomed to produce results *far worse* than a traditional, reuse-less software process. To attempt to reuse software without assertions is to invite failures of potentially disastrous consequences."

An genau dieser Stelle setzt das deduktionsbasierte Softwarekomponenten-Retrieval an. Die dahinter liegende Idee ist relativ einfach: *(i)* Die wiederverwendbaren Komponenten in einer Bibliothek werden mit Hilfe von Kontrakten formal spezifiziert. *(ii)* Suchanfragen an die Bibliothek werden ebenfalls als Kontrakte formuliert. *(iii)* Aus den Kontrakten der Anfrage und der jeweiligen Bibliothekskomponente wird eine Beweisbedingung konstruiert, die den intendierten Wiederverwendbarkeitsbegriff exakt formalisiert. *(iv)* Ein automatischer Theorembeweiser wird verwendet, um die Bedingungen zu überprüfen; bewiesene Bedingungen korrespondieren zu passenden Komponenten. Da deduktionsbasierte Retrievalverfahren nur beweisbar passende Komponenten zurückliefern und so einen Rahmen schaffen, in dem jeder Wiederverwendungsschritt formal gerechtfertig ist, ist das Konzept schon verschiedentlich vorgeschlagen und untersucht worden [CJ92, PBA95, MW97b, MMM97]. Allerdings sind diese Untersuchungen meist sehr oberflächlich und eine Reihe

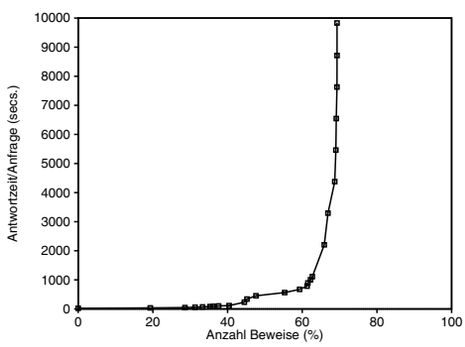

Abbildung 1: Retrievalergebnisse und Antwortzeiten

von wichtigen Fragen blieb unvollständig beantwortet: *(i)* Was ist der genaue Zusammenhang zwischen der Form der Beweisbedingungen und der Wiederverwendbarkeit der damit gefundenen Komponenten? *(ii)* Wie lässt sich das Konzept in ein benutzbares Werkzeug umsetzen? *(iii)* Lässt sich das Konzept technisch überhaupt realisieren? Sind existierende Beweiser in der Lage in akzeptabler Zeit eine ausreichende Anzahl der entstehenden Bedingungen zu beweisen?

Die vorliegende Arbeit wurde dabei insbesondere von der dritten Frage motiviert. Die direkte Umsetzung des weiter oben skizzierten "generate-and-prove"-Ansatzes ist unpraktikabel. Abb. 1 zeigt, dass die Antwortzeiten eines solch naiven Retrievalwerkzeugs selbst bei Verwendung des besten verfügbaren Theorembeweisers schon für kleinere Bibliotheken explosionsartig anwachsen, bevor eine ausreichende Anzahl von Beweisen (und damit Komponenten) gefunden werden kann. Die Ursache für dieses Verhalten liegt nicht in der durchaus nicht außergewöhnlichen Komplexität der entstehenden Beweisbedingungen, sondern in ihrem Profil: die weitaus meisten Bedingungen sind unbeweisbar weil sie zu unpassenden Komponenten korrespondieren. Ein praktisch benutzbares Werkzeug muß dieses Profil von Anfang an in Betracht ziehen; NORA/HAMMR verwendet daher eine inkrementelle Filterpipeline, in der dedizierte Widerlegungsfilter versuchen, möglichst schnell möglichst viele unbeweisbare Bedingungen zu identifizieren und so die nachfolgenden Beweiser-basierten Bestätigungsfilter zu entlasten.

Dieser Beitrag gibt einen Überblick über die Ergebnisse der Dissertation [Fis01]. Er konzentriert sich dabei auf die oben genannten Fragen, die in den folgenden drei Abschnitten näher behandelt werden.

2 Beweisbedingungen und Wiederverwendbarkeit

Ziel des Komponentenretrievals ist es, die auf eine Anfrage "passenden" Komponenten zu identifizieren; wann eine Komponente als passend betrachtet wird hängt dabei vom intendierten Wiederverwendbarkeitsbegriff ab. Im deduktionsbasierten Ansatz lässt sich dieser Begriff mit Hilfe der Kontrakte exakt—allerdings nicht eindeutig—formalisieren. Die Formalisierung hängt im Allgemeinen vom zugrundeliegenden Komponentenmodell und von der verwendeten Kontraktsprache ab. In dieser Arbeit wurde das traditionelle Modell von Funktionen als Komponenten verwendet, die mit Hilfe axiomatischer Spezifikationen (d.h. Vor- und Nachbedingungen) formalisiert wurden; diese Formalisierung muß ggf. auf andere Komponentenmodelle (z.B. Java-Beans) und Kontraktsprachen (z.B. algebraische Spezifikationen) angepasst werden. Im folgenden kennzeichnen x bzw. y jeweils eine Menge von Eingabe- bzw. Ausgabeparametern und $\mathit{pre}(x)$ bzw. $\mathit{post}(x, y)$ Vor- bzw. Nachbedingungen; die Subskripte q bzw. c kennzeichnen Anfragen (oder *queries*) bzw. Komponenten. Vor- und Nachbedingungen sind hier Formeln in Prädikatenlogik erster Ordnung; andere Logiken könnten auch verwendet werden.

Der strikteste Wiederverwendbarkeitsbegriff ist *Austauschbarkeit*: eine Komponente kann wiederverwendet werden, wenn sie von außen gesehen verhaltensgleich ist. Dies lässt sich als Äquivalenz der Vor- und Nachbedingungen formalisieren; dabei kann die Äquivalenz der Nachbedingungen auf den durch die Anfrage gegebenen Bereich beschränkt werden:

$$\forall x_q, x_c, y_q, y_c \cdot \big(\mathit{pre}_q(x_q) \Leftrightarrow \mathit{pre}_c(x_c)\big) \wedge \big(\mathit{pre}_q(x_q) \Rightarrow (\mathit{post}_q(x_q, y_q) \Leftrightarrow \mathit{post}_c(x_c, y_c))\big)$$

Austauschbarkeit unterstützt *black-box*-Wiederverwendung im größtmöglichen Ausmaß: da passende Komponenten den exakt gleichen Wertebereich haben müssen wie die Anfrage, und auf diesem Wertebereich exakt das spezifizierte Verhalten aufweisen müssen, können sie das Gesamtsystemverhalten nicht durch unbeabsichtigte Nebeneffekte ändern. Austauschbarkeit ist daher besonders geeignet zur Unterstützung der Komponentenwiederverwendung im Re-Engineering und für Wartungsaufgaben, z.B. Migration zwischen Plattformen, oder Ersetzen eines Prototyps durch eine funktional äquivalente aber effizientere Implementierung.

In vielen Fällen ist Austauschbarkeit aber zu strikt. Der häufigste Wiederverwendbarkeitsbegriff ist daher *Einsetzbarkeit*; dabei werden die Äquivalenzen durch Implikationen ersetzt:

$$\forall x_q, x_c, y_q, y_c \cdot \big(\mathit{pre}_q(x_q) \Rightarrow \mathit{pre}_c(x_c)\big) \wedge \big((\mathit{pre}_c(x_c) \wedge \mathit{post}_c(x_c, y_c)) \Rightarrow \mathit{post}_q(x_q, y_q)\big)$$

Unter dem Begriff der Einsetzbarkeit kann eine Komponente also wiederverwendet werden, wenn sie für einen größeren Wertebereich arbeitet als in der Anfrage verlangt, und

dort gleichzeitig spezifischere Ergebnisse garantiert. Einsetzbarkeit zielt auf die maximale aber sichere Ausnutzung einer Bibliothek: viele wiederverwendbare Komponenten würden unter dem strikteren Begriff der Austauschbarkeit nicht als passend identifiziert werden, da sie sehr allgemein aber gleichzeitig sehr defensiv implementiert sind und z.B. auch auf erweiterten Wertebereichen noch sinnvoll arbeiten.

Deduktionsbasiertes Retrieval kann aber auch zur Unterstützung der sogenannten *white-box*-Wiederverwendung eingesetzt werden. Dabei werden Komponenten bereits als passend betrachtet, wenn sie eine Teillösung des in der Anfrage spezifizierten Problems bereitstellen. Dies kann z.B. durch den Wiederverwendbarkeitsbegriff der *Teilüberdeckung* formalisiert werden:

$$\forall x_q, x_c, y_q, y_c \cdot (pre_q(x_q) \wedge pre_c(x_c) \wedge post_c(x_c, y_c)) \Rightarrow post_q(x_q, y_q)$$

Offensichtlich muß hier zur sicheren Wiederverwendung der gefundenen Komponenten der Aufrufkontext modifiziert werden. Der entscheidende Unterschied zu einem informellen Wiederverwendungsprozess ist aber, dass die Spezifikationen beschreiben, welche Teilaspekte bereits gelöst wurden. Dies kann nun ausgenutzt werden, um Nachfolgeanfragen für weitere Teillösungen zu stellen. Die Form der Nachfolgeanfragen hängt dabei von der Art und Weise ab, in der die Teillösungen kombiniert werden; für sequentielle Komposition erhält man $(pre_q(x_q), pre_c(x_c))$ (d.h. die Komponentenvorbedingung pre_c dient als neue Anfragenachbedingung), für alternative Komposition entsprechend $(pre_q(x_q) \wedge \neg pre_c(x_c), post_q(x_q, y_q))$.

3 Das NORA/HAMMR-System

Deduktionsbasiertes Softwarekomponenten-Retrieval benötigt Werkzeugunterstützung, um überhaupt praktisch anwendbar zu sein. Im Rahmen dieser Dissertation wurde daher das NORA/HAMMR-System entwickelt. Ein Hauptentwicklungsziel war dabei, den Einsatz (automatischer) Theorembeweiser vollständig unsichtbar zu machen und so ein Werkzeug zu implementieren, das Benutzern ohne Vorkenntnisse auf dem Gebiet des Theorembeweisens zugänglich ist.

Das Ziel der vollständigen Automatisierung schließt offensichtlich den Einsatz der im Software Engineering üblicherweise verwendeten interaktiven Beweiser aus. Darüberhinaus hat es aber noch eine Reihe subtilerer Auswirkungen. Insbesondere müssen die Beweisbedingungen vollautomatisch von der Kontraktsprache (NORA/HAMMR verwendet VDM-SL) in die Beweisernotation übersetzt werden. Dieser Schritt umfasst nicht nur relativ einfache Syntaxkonvertierungen, sondern auch Logiktransformationen (hier von einer dreiwertigen Logik in die klassische zweiwertige Prädikatenlogik) und Vorvereinfachungen sowie die Auswahl von Axiomen bzw. Lemmas und das Setzen von Beweiserkontrollparametern; die letzten Schritte bleiben üblicherweise dem “erfahrenen Anwender” überlassen.

NORA/HAMMR verbirgt diese Aspekte hinter der in Abb. 2 gezeigten Benutzeroberfläche, deren zentrales Element die Filterpipeline ist. Die einzelnen Filter übernehmen die

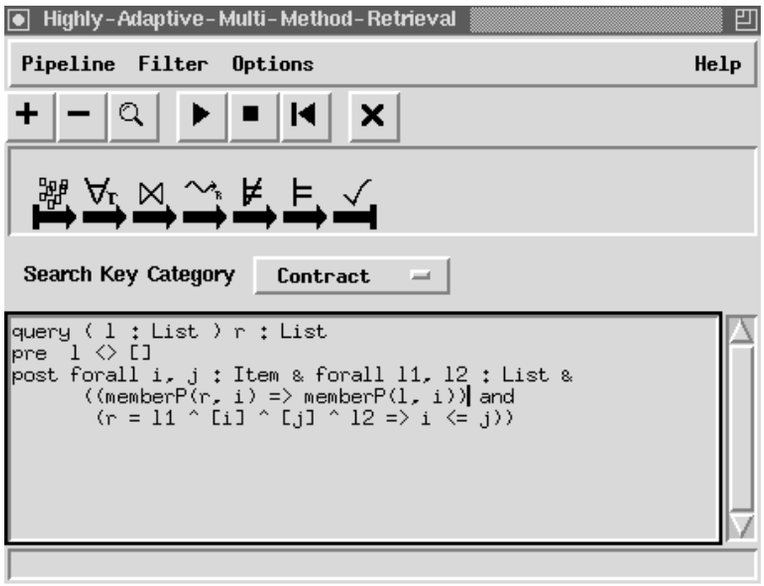

Abbildung 2: NORA/HAMMR – Benutzeroberfläche

verschiedenen Aufgaben (Bibliothekszugriff, Konstruktion der Beweisbedingungen, Vorvereinfachung usw.) und können vom Benutzer spezifisch instantiiert werden (verwendete Bibliothek, Art der Beweisbedingungen, Vorvereinfachungsmethode usw.); darüberhinaus kann der Benutzer die Anordnung der Filter in der Pipeline in gewissem Rahmen selbst festlegen. Die Steuerung des Retrievalprozesses erfolgt über einfache Start/Stop-Tasten. Insbesondere kann der Prozess jederzeit angehalten und wieder fortgesetzt werden; da jeder Filter den Zugriff auf die an ihm vorliegenden Komponenten erlaubt, kann der Benutzer so rasch erkennen, ob die gestellte Anfrage noch zum Erfolg führen kann. Die Suchanfragen werden in VDM-SL formuliert, unabhängig von den in der Pipeline spezifizierten Beweisern; spezielle Logik- oder Beweiserkenntnisse sind nicht nötig.

4 Experimentelle Auswertung

In der Literatur ist deduktionsbasiertes Retrieval vor allem unter theoretischen Aspekten betrachtet worden. Keine der oben zitierten Arbeiten enthält eine aussagekräftige experimentelle Auswertung, die daher ein wesentlicher Schwerpunkt der Dissertation war. Ziel der Experimente war festzustellen, ob das Retrieval-Konzept im Allgemeinen und das Pipelineverfahren im Speziellen technologisch realisierbar sind, d.h., ob (i) die Widerlegungsfilter ausreichen, um genügend unbeweisbare Bedingungen zu identifizieren und (ii) die verfügbaren Theorembeweiser genügend Bedingungen beweisen können. Darüberhinaus wurde untersucht, ob (iii) verschiedene Beweiser gleich gut geeignet sind, und (iv) welche Effekte verschiedene Vorverarbeitungsschritte auf das Retrievalergebnis haben.

4.1 Aufbau

Aufgrund dieser Ziele muß der experimentelle Aufbau vor allem eine große Anzahl von Beweisbedingungen produzieren und gleichzeitig eine einfache Wiederholbarkeit gewährleisten. Daher wurden keine manuellen Anfragen formuliert, sondern die Komponentenspezifikation auch als Anfragen verwendet. Die verwendete Testbibliothek enthält 119 in VDM-SL spezifizierte Listenfunktionen, so dass sich etwas mehr als 14.000 Beweisbedingungen ergeben. Als Wiederverwendbarkeitsbegriff wurde Einsetzbarkeit gewählt; damit sind 1838 oder 13.0% der Bedingungen gültig. Alle Experimente wurden auf einer unbelasteten Sun ULTRA1 Workstation mit 170MHz durchgeführt.

4.2 Kennzahlen

Retrievalsysteme werden üblicherweise mit Hilfe der *recall/precision/fallout*-Metriken ausgewertet [SM83]. Diese Metriken werden aus der Größe der Bibliothek $\mathcal{L}$ sowie der Mengen der relevanten (d.h. den Wiederverwendbarkeitsbegriff erfüllenden) Komponenten *REL* und der tatsächlich gefundenen Komponenten *RET* berechnet:

$$r = \frac{\mid REL \cap RET \mid}{\mid REL \mid} \qquad p = \frac{\mid REL \cap RET \mid}{\mid RET \mid} \qquad f = \frac{\mid RET \backslash REL \mid}{\mid \mathcal{L} \backslash REL \mid}$$

Der Wiedergewinnungsfaktor oder *recall* r misst die Fähigkeit des Systems, relevante Komponenten zu finden, während die Präzision p den relativen Anteil der relevanten Komponenten in der Antwortmenge misst. Idealerweise sollten beide Metriken einen Wert von eins haben; dann würde das System alle passenden Komponenten finden, aber keine unpassenden. In der Realität streben die beiden Werte jedoch häufig auseinander. Der Ausfallfaktor oder *fallout* f misst schließlich den Filtereffekt, d.h. die relative Anzahl der unpassenden Komponenten, die ein Filter durchlässt.

4.3 Widerlegungsfilter

Theorembeweiser sind sorgfältig optimierte, komplexe Suchprozeduren, deren Ziel es ist, einen Beweis einer (beweisbaren) Bedingung zu finden. Sie sind ungeeignet nachzuweisen, dass eine Bedingung unbeweisbar ist; üblicherweise terminieren sie in diesem Fall nicht einmal. Aufgabe der Widerlegungsfilter ist es daher, mit möglichst geringem Rechenaufwand möglichst viele unbeweisbare Bedingungen zu identifizieren und so die Theorembeweiser zu entlasten. Eine Beweisbedingung F kann aus zwei verschiedenen Gründen unbeweisbar sein: (i) ihre Negation ist allgemeingültig, d.h. $\models \neg F$ oder (ii) es gibt ein Modell $\mathcal{M}$, in dem F den Wahrheitswert falsch hat, d.h. $\mathcal{M} \not\models F$.

Termersetzungssysteme (TRS) sind abstrakte Berechnungsmodelle für gleichungsdefinierte Theorien; im Gegensatz zu Beweisern benötigen sie keine Suche. Sie können daher direkt zur schnellen Erkennung unbeweisbarer Bedingungen des Typs (i) verwendet werden:

TRS	t (sek./Anfrage)	r (%)	p (%)	f (%)
$\mathcal{K}$	8.0	100.0	13.0	100.0
$\mathcal{D}$	7.4	100.0	17.5	69.3
$\mathcal{Q}$	175.6	100.0	29.0	28.3
$\mathcal{M}$	63.6	100.0	35.2	24.8

Abbildung 3: Ergebnisse der Widerlegungsfilter

falls F mit Hilfe eines korrekten TRS $\mathcal{R}$ zu *false* reduziert werden kann (d.h. $F \rightarrow_{\mathcal{R}} \mathit{false}$), folgt auch dass $\models \neg F$ gilt. Allerdings stellt sich die Frage welches TRS verwendet werden soll. In der Dissertation wurden verschiedene Alternativen untersucht. Die einfachste Variante $\mathcal{K}$ verwendet nur die Kernregeln der Aussagenlogik, d.h. Assoziativität und Kommutativität von Konjunktion und Disjunktion, DeMorgans Gesetze usw. Der Vorteil von $\mathcal{K}$ ist, dass es unabhängig von der Domäne der Bibliothek ist und daher fest in das Retrievalsystem eingebaut werden kann. Die Variante $\mathcal{D}$ enthält zusätzliche domänenspezifische Regeln, die die Eigenschaften der zur Spezifikation der Bibliothekskomponenten verwendeten Symbole widerspiegelt. $\mathcal{D}$ muss daher zusammen mit der Bibliothek entwickelt werden. Für abstrakte Datentypen wie z.B. Listen lassen sich die Datentypgeneratoren verwenden, um Quantoren schrittweise zu eliminieren:

$$\forall l \cdot F[l] \rightarrow_{\mathcal{Q}} F[\mathit{nil}] \wedge \forall i, l' \cdot F[\mathrm{cons}(i, l')]$$

Diese Quantorenauffaltung $\mathcal{Q}$ simuliert die übliche manuelle Analyse einer Beweisbedingung: F ist gültig wenn es sowohl für die leere Liste *nil* als auch für alle mindestens einelementigen Listen der Form $\mathit{cons}(i, l)$ gilt. Offensichtlich werden die Quantoren nicht wirklich eliminiert sondern nur nach innen verschoben. $\mathcal{Q}$ terminiert daher im Allgemeinen nicht und muss (ähnlich wie ein Beweiser) mit einer Zeitschranke versehen werden; in den Experimenten wurden 5.0 *sek.* je Bedingung verwendet.

Abb. 3 zeigt die Ergebnisse der auf den verschiedenen TRS basierenden Widerlegungsfilter. Es zeigt sich, dass das domänenunabhängige TRS $\mathcal{K}$ als Widerlegungsfilter (zumindest für die Testbibliothek) völlig ungeeignet ist—der *fallout* beträgt 100%. Durch die domänenspezifische Regeln kann der Filtereffekt verbessert werden, aber erst durch das wiederholte Auffalten der Quantoren können die in den Bedingungen enthaltenen Widersprüche aufgedeckt werden. Insgesamt können so in relativ kurzer Zeit mehr als 70% der unpassenden Komponenten identifiziert werden, ohne dass passende Kandidaten verloren gehen.

Unbeweisbare Bedingungen des Typs (ii) können ebenfalls mit TRS erkannt werden. Dazu muss das intendierte Gegenmodell durch zusätzliche Regeln formalisiert werden. Diese Formalisierung muss nicht unbedingt exakt sein: durch die Verwendung aggressiver Approximationen lässt sich eventuell der Filtereffekt steigern, allerdings können dadurch Komponenten verloren gehen. Das TRS $\mathcal{M}$ ist eine solche Approximation, in der alle Listenelemente miteinander identifiziert werden. Die Experimente zeigen, dass damit der Filtereffekt gegenüber $\mathcal{Q}$ nochmals geringfügig gesteigert wird ohne dass der *recall* leidet.

4.4 Bestätigungsfilter

Zur Implementierung der Bestätigungsfilter wurden vier verschiedene Beweiser für die Prädikatenlogik erster Ordnung mit Gleichheit verwendet: GANDALF [Tam97], OTTER [MW97a], SETHEO [MIL$^+$97], und SPASS [Wei97]. Die Beweiser verwenden intern verschiedene Kalküle, werden hier aber nur als *black boxes* betrachtet, die entweder innerhalb der erlaubten Zeit einen Beweis finden oder mit einem Timeout terminieren. Alle verwendeten Beweiser haben in den letzten Jahren an der Spitze des jährlichen CASC-Beweiserwettbewerbs (siehe `http://www.math.miami.edu/~tptp/CASC`) gelegen und repräsentieren daher den *state-of-the-art*.

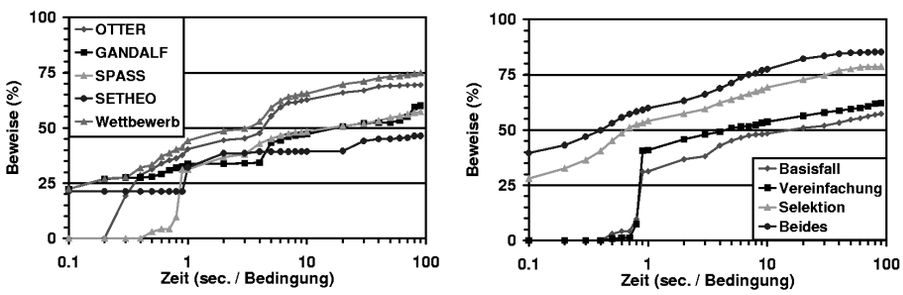

Abbildung 4: Ergebnisse der Bestätigungsfilter (Prozentsatz der insgesamt gefundenen Beweise in Abhängigkeit von der Zeitschranke je Bedingung): *(links)* Vergleich verschiedener Beweiser für Originalbedingungen *(rechts)* Effekt der Vorverarbeitungsschritte auf SPASS.

In einer ersten Reihe von Experimenten wurde untersucht, inwieweit die Theorembeweiser als *off-the-shelf*-Komponenten für diese Anwendung geeignet sind, und inwieweit sie sich in ihren Leistungen unterscheiden. Der linke Graph in Abb. 4 fasst die Ergebnisse dieser Experimente zusammen. Es zeigt sich, dass die Beweiser an der Komplexität der automatisch generierten Beweisbedingungen mehr oder weniger scheitern. Die meisten Beweiser brauchen ein Zeitlimit von mehr als 20 Sekunden je Bedingung, bevor sie die Hälfte der gültigen Bedingungen zeigen können, und keiner der Beweiser erreicht mehr als 70%. Aufgrund der ungleichmässigen Verteilung der relevanten Komponenten ergeben sich daher über alle Anfragen nur durchschnittliche *recall*-Werte von 29.4%–57.2%.

Der Graph zeigt auch, dass die verschiedenen Beweiser deutliche Unterschiede in ihren Leistungen aufweisen: der beste Beweiser (OTTER) findet in der gleichen Zeit beinahe 50% mehr Beweise als der schlechteste (SETHEO). Diese Unterschiede lassen sich ausnutzen, indem die Beweiser auf verschiedenen Prozessoren wettbewerbsparallel gelaufen lassen werden; der erste erfolgreiche Beweiser beendet dann die anderen. Dieser Wettbewerb löst insgesamt 74.8% der Beweise und liefert damit einen durchschnittlichen *recall*-Wert von 68.8%.

Eine weitere Verbesserung der Ergebnisse lässt sich erzielen, wenn die Beweisbedingungen vorverarbeitet werden, bevor sie an die Beweiser weitergereicht werden. In der Dissertation wurden zwei verschiedenen Methoden untersucht, Vorvereinfachung und Lemmaauswahl. Der rechte Graph in Abb. 4 zeigt die Resultate dieser Vorverarbeitungsschritte

exemplarisch für den Beweiser SPASS; für die anderen Beweiser wurden ähnliche Verbesserungen erzielt. Es zeigt sich, dass insbesondere die automatische Lemmaauswahl einen großen Einfluss auf die Ergebnisse hat; darüberhinaus sind beide Methoden in gewissem Maße komplementär und ihre Kombination ergibt einen durchschnittlichen *recall*-Wert von 79.4%.

5 Ausblick

In der vorliegenden Arbeit wurde das Konzept des deduktionsbasierten Komponenten-Retrieval theoretisch und erstmalig auch experimentell ausführlich untersucht. Die umfangreichen Experimente haben gezeigt, dass das Konzept technologisch realisierbar ist: dedizierte Widerlegungsfilter können in kurzer Zeit genügend unbeweisbare Bedingungen identifizieren, Bestätigungsfilter auf der Basis existierender automatischer Theorembeweiser können genügend Beweise finden. Der Berechnungsaufwand bleibt sehr hoch, kann aber durch die sorgfältige Integration der Beweiser in eine geeignete Architektur in akzeptablen Grenzen gehalten werden. Der Spezifikationsaufwand dagegen bleibt der gegenwärtig limitierende Faktor für einen praktischen Einsatz des Konzeptes. Das Problem ist dabei weniger der initiale Aufwand, existierende Bibliotheken zu formalisieren und so für das Retrieval zu erschliessen, als der wiederholte Aufwand entsprechend formalisierte Anfragen zu stellen. Anwendungen werden sich daher wohl vorläufig nur in Bereichen mit sehr hohen Zuverlässigkeitsanforderungen (z.B. in der Raumfahrt) finden. Dort allerdings liefert das deduktionsbasierten Retrieval eine formal gerechtfertigte Wiederverwendungsmethodologie, in der mit den eigentlichen Softwarekomponenten auch der Verifikationsaufwand wiederverwendet und damit amortisiert werden kann.

Zukünftige Arbeiten sollten sich vor allem mit der Reduktion des Spezifikationsaufwandes befassen. Eine Möglichkeit ist die Integration des Retrievals in andere formale Softwareentwicklungsmethoden (z.B. Programmsynthese), so dass Anfragen automatisch aus dem Entwicklungsprozess abgeleitet werden können. Eine andere Möglichkeit ist der Übergang zu anderen, grobkörnigeren Komponentenmodellen wie Java-Beans; damit ist die Hoffnung verbunden, dass der Wiederverwendungseffekt für eine einzelne Anfrage deutlich ansteigt. Darüberhinaus lässt sich dieser Ansatz gut mit formalen Software-Architekturen verbinden, so dass Anfragen wiederum automatisch generiert werden können.

Die beim deduktionsbasierten Komponenten-Retrieval enstehenden Beweisbedingungen sind relativ typisch für viele Anwendungen im Software Eingineering (z.B. Programmverifikation, Programmsynthese, oder Protokollanalyse). Die experimentellen Ergebnisse lassen sich daher auch auf diese Gebiete übertragen; insbesondere lassen sie den Schluss zu, dass der Beweiserbau in den letzten Jahren genügend Fortschritte gemacht hat und so die Anwendung von automatischen Theorembeweisern auf Probleme des Software Engineering wieder interessant geworden ist.

Literaturverzeichnis

[CJ92] B.H.C. Cheng and J.-J. Jeng. Using Automated Reasoning to Determine Software Reuse. *Intl. J. Software Engineering and Knowledge Engineering*, 2(4):523–546, 1992.

[Fis01] B. Fischer. *Deduction-Based Software Component Retrieval*. Dissertation, Universität Passau, 2001. Erhältlich über http://elib.ub.uni-passau.de/opus/volltexte/2002/23/.

[JM97] J.-M. Jézéquel and B. Meyer. Design by Contract: The Lessons of Ariane. *IEEE Computer*, 30(1):129–130, 1997.

[Lio96] J. L. Lions et al. Ariane 5 Flight 501 Failure Report, 1996.

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *J. Automated Reasoning*, 18:237–246, 1997.

[MMM97] A. Mili, R. Mili, and R. Mittermeir. Storing and Retrieving Software Components: A Refinement-Based System. *IEEE Trans. Software Engineering*, 23(7):445–460, 1997.

[MW97a] W. McCune and L. Wos. Otter—The CADE-13 Competition Incarnations. *J. Automated Reasoning*, 18(2):211–220, 1997.

[MW97b] A. Moorman Zaremski and J.M. Wing. Specification Matching of Software Components. *ACM Trans. Software Engineering and Methodology*, 6(4):333–369, 1997.

[PBA95] J. Penix, P. Baraona, and P. Alexander. Classification and Retrieval of Reusable Components Using Semantic Features. In D. Setliff (Hrsg.), *Proc. 10th Knowledge-Based Software Engineering Conf.*, pp. 131–138, 1995. IEEE Comp. Soc. Press.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[Tam97] T. Tammet. Gandalf. *J. Automated Reasoning*, 18(2):199–204, 1997.

[Wei97] C. Weidenbach. SPASS—Version 0.49. *J. Automated Reasoning*, 18(2):247–252, 1997.

Bernd Fischer war nach Abschluss seines Diploms in Informatik 1990 als wissenschaftlicher Mitarbeiter an der TU Braunschweig beschäftigt, erst in der Abt. Programmiersprachen, dann von 1991 bis 1998 in der Abt. Softwaretechnik. Dort beschäftigte er sich im Rahmen des DFG-Schwerpunktprogramms “Deduktion” mit der Anwendung von automatischen Theorembeweisern im Software Engineering; seine Dissertation ist im wesentlichen im Rahmen dieser Arbeiten entstanden. Seit 1998 arbeitet er für USRA/RIACS am NASA Ames Research Center in den USA. Sein Forschungsschwerpunkt liegt im Gebiet der automatischen Programmsynthese, insbesondere für Anwendungen in der statistischen Datenanalyse und in der Satellitennavigation sowie für Softwarezuverlässigkeitsschätzungen. Dr. Fischer hat mehr als 25 Publikationen zur Automatisierung des Software Engineerings veröffentlicht und ist regelmäßiges Mitglied der Programmkommittees verschiedener Workshops und Konferenzen auf diesem Gebiet.