Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik

Diplomarbeit

"SolutionBroker"
Konzept und Realisierung eines Systems zur verteilten Problemlösung

Robert Sösemann

Aufgabensteller:	Dr. Andreas Arning (IBM), Andreas Schmitz (IBM)
Betreuer:	Prof. Dr. W. Spruth
Abgabetermin:	13.11.2003

Erklärung

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Tübingen, den 13.11.2003

_____

Danksagungen

Der *Firma IBM* für die Möglichkeit, an diesem interessanten Projekt arbeiten zu können.

Meinen beiden Betreuern bei IBM vor Ort *Dr. Andreas Arning und Andreas Schmitz* für die hervorragende fachliche Betreuung und ihre stete Diskussionsbereitschaft. Besonders *Andreas Arning* möchte ich für sein außergewöhnliches Engagement und die wertvollen Gespräche danken, in denen ich viel von ihm lernen konnte.

Prof. Dr. Spruth für seinen unschätzbaren Kontakt zu IBM, die Bereitschaft, die Betreuung meiner Diplomarbeit zu übernehmen, und zahlreiche wertvolle Anregungen in effizienten Besprechungen.

Den Mitarbeitern im IBM Entwicklungslabor Böblingen, für die vielen aufschlussreichen Diskussionen, besonders meinem hilfsbereiten Zimmerkollegen *Andreas Schön* und dem Datenbankteam *Antje Dengler* und *Frank Neumann*.

Herzlichen Dank auch an

Amar Subramanian, für die Bereitstellung seines immensen technischen Know-hows auf den Fahrten zur Arbeit und per ICQ.

Florian Pahl, für das Word Template seiner Diplomarbeit. Ihm verdanke ich zahlreiche gute Gestaltungsideen und die Möglichkeit, mehr Zeit in die Inhalte als die äußere Form dieser Arbeit zu stecken.

Katrin, meiner wunderbaren Freundin, weil sie immer für mich da war, mich motiviert und in meiner Arbeit bestärkt hat.

Diese Diplomarbeit möchte ich meinen Eltern widmen, die mich während meines gesamten Studiums uneingeschränkt unterstützt haben und mich stets meinen Weg gehen ließen.

Kurzfassung

Bei der Benutzung von Software geraten Anwender gelegentlich in ungewöhnliche Problemsituationen, die sie an der erfolgreichen Bewältigung ihrer Arbeit hindern. Eine reibungslose Funktion ist aber gerade wegen der zunehmenden wirtschaftlichen Bedeutung von Softwaresystemen wichtig. Neben Fehlern in der Software liegen die Ursachen von Problemen mit komplexer Software oft auch in der fehlerhaften Benutzung durch die Anwender. Helfen Fehlermeldungen und die Produktdokumentation nicht, eine Lösung zu finden, muss auf externe Hilfesysteme zugegriffen werden. Vor allem der hohe Anteil manueller Tätigkeiten macht die Suche dort zeitaufwendig. Außerdem ist bisher beim Bereitstellen eigener Lösungen dauerhaftes Engagement nötig.

Damit Anwender einfacher als bisher Lösungen anbieten können und um existierende Lösungen wieder zu verwenden, anstatt sie erneut aufwendig suchen zu müssen, ist ein neues Hilfesystem nötig, dessen Grundidee von IBM als Ausgangspunkt für diese Arbeit zu Verfügung gestellt wurde.

Das neue Hilfesystem sieht vor, Lösungen in einer zentralen Datenbank inklusive einer Beschreibung des jeweiligen Problems abzulegen. So kann Wissen mit einmaligem Aufwand dauerhaft zu Verfügung gestellt werden. Durch eine Erweiterung um Kontextinformationen soll die Problembeschreibung aussagekräftiger gemacht werden als die bisherigen Prosabeschreibungen. Außerdem sollen durch eine einheitliche und maschinenlesbare Form Unklarheiten vermieden und eine automatische Lösungssuche ermöglicht werden. Ein Vergleich der Problembeschreibungen soll dabei ähnliche Probleme identifizieren, um nur potentiell relevante Lösungen auszugeben.

Ziel und Aufgabe der Diplomarbeit war die Realisierung eines solchen verteilten Hilfesystems. Dazu gehören insbesondere das Finden eines geeigneten Problembeschreibungsformats, die Ausarbeitung eines Matchingverfahrens zur automatischen Lösungssuche und das Finden und Implementieren einer skalierbaren und robusten Anwendungsarchitektur.

Implementiert wurde eine auf XML Web Services basierende Client-Server Anwendung im Umfang von ca. 8.000 Zeilen Programmcode, die das Ablegen und Suchen von Lösungen über das Internet erlaubt. Dabei werden als aussagekräftige und maschinenlesbare Problembeschreibungen die weit verbreiteten Java-Stacktraces verwendet. Als Benutzerschnittstelle dienen wahlweise ein browserbasiertes Webinterface und ein Plugin, das direkt in eine Anwendersoftware integriert ist. Das Hilfesystem wird intern erfolgreich bei IBM eingesetzt und hat dort mehrfach in Problemsituationen besser helfen können als es mit den bisherigen Hilfsmitteln möglich war.

Inhaltsverzeichnis

ANHANG A Abbildungsverzeichnis

KAPITEL 1 Einleitung

1.1 Aufbau der Arbeit

Diese Arbeit gliedert sich in folgende Bereiche:

Einführung

Dieser Abschnitt führt in die Fragestellungen der Arbeit ein und motiviert ihren Nutzen. Anhand eines realen Beispiels aus dem Umfeld der IBM soll gezeigt werden, wie zeitaufwendig die Beseitigung von Problemsituationen mit Software sein kann und warum die Konzeption eines neuen Hilfesystems sinnvoll ist. Als Grundlage der Betrachtungen werden existierende Hilfesysteme zur Lösung von Anwenderproblemen wie Fehlermeldungen, Herstellersupport, Mailinglisten und Newsgroups vorgestellt. Außerdem sollen die Stärken und Schwächen dieser Systeme identifiziert werden.

Grundidee des neuen Hilfesystems

In Hinblick auf die Schwächen existierender Systeme wurde bei IBM ein Grundkonzept für ein neues verteiltes Hilfesystems erarbeit und dient als Ausgangspunkt der Diplomarbeit. Seine Eigenschaften und die Verbesserungen gegenüber bestehenden Systemen werden in diesem Kapitel vorgestellt.

„SolutionBroker" – eine Beispielrealisierung

Dieses Kapitel beschreibt die Übertragung des vorliegenden Konzepts in eine konkrete Beispielanwendung und dokumentiert die nötigen Schritte und Entscheidungen. Es werden Anforderungen an die verwendeten Technologien formuliert und wichtige Systemkomponenten identifiziert. Für die Implementierung entscheiden wir uns für eine bestimmte Art von Problemsituationen und einem dazugehörigen Problembeschreibungsformat. Anhand typischer Fallbeispiele werden Voraussetzungen für den Matchingalgorithmus zur Lösungssuche definiert und dieser ausgearbeitet.

Implementierung

Die konzeptuellen Entscheidungen des vorhergehenden Kapitels werden nun in Software unter Verwendung konkreter Technologien umgesetzt. Das Kapitel beschreibt die Klassenarchitektur und nennt relevante Implementierungsdetails. Mittel zur Verbesserung der Performance werden ebenso genannt wie Aspekte zur Verbesserung der Benutzerfreundlichkeit. Mit Hilfe von Bildschirmfotos der beiden Clientanwendungen werden am Ende des Kapitels typische Benutzungsszenarien gezeigt.

Erweiterungs-möglichkeiten

Es werden Vorschläge für mögliche Erweiterungen von Konzept und Implementierung gemacht.

Ergebnisse der Diplomarbeit

Die Ergebnisse der Diplomarbeit und die Vorteile des neuen Hilfesystems im Vergleich mit existierenden Systemen werden hier noch einmal abschließend zusammengefaßt.

Anhang

Die beiden Anhänge am Anfang und am Ende des Dokuments enthalten ein Abbildungsverzeichnis und ein Literaturverzeichnis.

1.2 Motivation – Anwenderprobleme mit Software

Immer komplexere Software unterstützt immer mehr Menschen bei ihrer täglichen Arbeit. Mit Hilfe von Software werden Geschäftsvorgänge ganz oder teilweise automatisiert und immer effizienter erledigt. Die zunehmende Durchdringung großer Bereiche des täglichen Lebens führt im Gegenzug zu wachsender Abhängigkeit der Anwender von der reibungslosen Funktion solcher Softwaresysteme. Auch wenn die Bedienung und die eingebauten Hilfestellungen heutiger Software intuitiver sind als je zuvor, können Anwender in Problemsituationen geraten, die sie an der erfolgreichen Bewältigung ihrer Arbeit hindern.

Die Probleme sind oft weniger Ausdruck echter Fehlfunktionen der Software als der fehlerhaften Benutzung oder Konfiguration durch den Benutzer. So ist es durchaus sinnvoll, dass das Betriebssystem den Anwender am Löschen einer schreibgeschützten Diskette hindert und einen Fehler signalisiert. Zu einem Arbeitshindernis und einem Problem werden derartige Situationen trotzdem, wenn Fehlermeldungen dem Anwender nicht bei der Erkennung und Beseitigung des Fehlers helfen. Muss der Anwender mit ungenügenden Informationen die Lösung selbst finden, steht der Aufwand oft in keinem Verhältnis zur meist minimalen Ursache.

Die Schilderung des folgenden Problemfalles aus dem Umfeld der IBM soll dies noch anschaulicher machen. Auf einer AS/400 Maschine sollte Ende der neunziger Jahre ein Magnetband formatiert werden. Damals wurden Magnetbänder zur externen Datenspeicherung verwendet. Normalerweise sind mehrfach beschreibbare magnetische Speichermedien beim Kauf bereits vorformatiert, so dass durch eine Formatierung hauptsächlich bereits beschriebene Bänder gelöscht werden. Das dafür vorgesehene Kommando INZTAP des Betriebssystems hatte die betroffene Person zuvor etliche Male erfolgreich benutzt. Der Versuch, ein fabrikneues, unformatiertes Band zu formatieren, schlug allerdings jedes Mal mit folgender Fehlermeldung fehl:

„Error: Invalid format"

Die Fehlermeldung ist zu knapp und das Wort „`format`" in diesem Fall irreführend, da es sich sowohl auf das Format des eingelegten Bandes beziehen kann wie auch einen ungültigen Aufruf des Format-Befehls. Man nahm an, dass die Speicherkapazität des verwendeten Bandes nicht unterstützt wird. So wurden fabrikneue Bänder verschiedener Größen und Marken durchprobiert. Erst durch Formatierung eines bereits formatierten und beschriebenen Bandes kam man der Lösung näher, weil diesmal überraschenderweise der Inhalt des Tapes aufgelistet wurde, gefolgt von einer Rückfrage an den Benutzer, ob diese Daten tatsächlich überschrieben werden sollen.

Der eigentliche Grund für die Problemsituation war dann schnell gefunden: Das verwendete Kommando bietet die Möglichkeit, vor dem Formatieren den Inhalt des Bandes anzuzeigen, um ein versehentliches Löschen zu verhindern. Während dafür bisher ein zusätzlicher Parameter angegeben werden musste, war dieses Verhalten in der verwendeten neusten Version des Betriebssystems als Default eingestellt.

Der Versuch ein fabrikneues, unformatiertes Band zu lesen, schlug daher fehl und produzierte die Fehlermeldung.

Die Vielfalt möglicher Problemsituationen und der unterschiedliche Wissensstand der Anwender macht es für den Hersteller oft unmöglich, in allen Fällen adäquate Fehlermeldungen anzubieten. Oft muss ein Kompromiss zwischen detaillierten technischen und kurzen allgemeinen Meldungen gefunden werden.

Noch problematischer ist es, wenn Fehler auftreten, die dem Hersteller bei der Auslieferung der Software nicht bekannt waren. Dies ist z.B. dann der Fall, wenn die Software stark mit fremden Komponenten verzahnt ist, auf die der Hersteller keinen Einfluss hat. So kann eine bestimmte Software auf üblichen Systemen zwar reibungslos funktionieren, bei seltenen Kombinationen von Hard- und Softwarekomponenten aber Fehler provozieren.

Um in den Hilfesystemen der Hersteller und den Archiven der Internetforen relevante Hinweise zu finden, ist es entweder nötig, Experten das Problem aussagekräftig zu beschreiben oder treffende Suchbegriffe zu wählen. Eine verwirrende Fehlermeldung wie in unserem Beispiel macht aber genau dies unmöglich. Die Eingabe der Fehlermeldung in die Internetsuchmaschine Google (www.google.de) ergab über 100.000 Suchtreffer, von denen die meisten unbrauchbar waren, da sie andere Formen von Lesefehler beschreiben. Auch nach Erweiterung der Suchbegriffe um die Worte *„AS/400„* und *„tape"* konnte keine größere Relevanz der Treffer festgestellt werden.

In dem geschilderten Fall kostete es zwei Personen jeweils einen halben Tag, die Lösung selbst zu erarbeiten. Da dieses Problem durch eine Änderung des Betriebssystems entstand, ist es anzunehmen, dass zuvor viele andere Anwender dasselbe Problem hatten und wegen der irreführenden Fehlermeldung ähnlich lange mit dem Suchen einer Lösung zugebracht haben. Den IBM-Kollegen stand kein geeignetes Hilfesystem zur Verfügung, bei dem man zusätzlich zu der sehr allgemeinen Fehlermeldung die Suchanfrage durch aussagekräftigere Angaben zum Problemkontext erweitern hätte können. Auch war es nicht möglich, nachträglich mit geringem Aufwand eine vollständige Problembeschreibung und die gefundene Lösung einem großen Anwenderkreis dauerhaft zur Verfügung zu stellen.

1.3 Existierende Hilfesysteme

Das einleitende Beispiel sollte zeigen, wie aufwendig die Behebung von Problemsituationen sein kann, wenn Fehlermeldung und Produktdokumentation nicht weiterhelfen. In diesem Kapitel werden existierende Hilfsmittel beschrieben, die Anwender bisher in solchen Situationen benutzen. Dabei werden neben den Stärken der einzelnen Verfahren auch Mängel genannt, die die Lösungssuche in den beschriebenen Problemsituationen eventuell erschweren.

1.3.1 Herstellersupport

Niemand kennt die Funktionsweise eines Programms und mögliche Ursachen für Fehler besser als der Hersteller selbst. Die Suche in seinen Hilfeinformationen ist daher bei Problemen sinnvoll. Falls Fehler bei der Auslieferung des Produkts noch unbekannt waren oder wegen ihrer Seltenheit nicht in die Produktdokumentation aufgenommen wurden, wird oft der so genannte After-Sales-Support (Kundenbetreuung nach dem Kauf) des Herstellers in Anspruch genommen. Aktuelles Hilfematerial nach der Auslieferung bieten immer mehr Hersteller vor allem online im Internet an. Dort finden sich z.B. aktuelle Fassungen der Produktdokumentationen sowie Lösungsvorschläge und Updates (sog. Bugfixes oder Patches) zur Behebung von nachträglich bekannt gewordenen Fehlern.

Es gibt Problemsituationen, die so selten oder begrenzt auf die Umgebung des Benutzers sind, dass der Hersteller dafür keine Lösung veröffentlicht hat. Für solche Fälle bieten Hersteller Servicehotlines an.

Über Telefon können so auch ungewöhnliche Problemsituationen gelöst werden, da firmeneigene Experten großes Wissen über das Produkt haben und im Gegensatz zu automatisierten Hilfesystemen auch unscharfe Problembeschreibungen verstehen.

Der Vorteil dieser Einzelbehandlung durch Experten ist allerdings auch ein großer Nachteil von Hotlines – sie sind aufwendig und deshalb oft auch kostenpflichtig. Außerdem sind sie bei einer zu großen Zahl von Anfragen nicht mehr praktikabel.

Um Antworten wieder verwenden zu können und einer großen Zahl von Anwendern zur Verfügung zu stellen, bieten Hersteller außerdem so genannte FAQs (Abkürzung für „Frequently Asked Questions") an. Meistens findet sich auf einer Internetseite eine Sammlung von Antworten auf die häufigsten Fragen der Benutzer. FAQs lassen sich mit den Erfahrungen der Hotlinebetreiber leicht gewinnen und schnell veröffentlichen.

1.3.2 Mailinglisten & Newsgroups

Eine weitere Informationsquelle bei der Problemlösung nutzen Mailinglisten und Newsgroups – das Wissen erfahrener Anwender. Diese kennen zwar die internen Details einer Software nicht, haben aber große Erfahrung in der Benutzung und kennen häufig Workarounds bei Problemsituationen.

Beide Hilfesysteme erleichtern durch computerbasierte Mittel den Erfahrungsaustausch der Anwender untereinander. Im Gegensatz zum Herstellersupport können Anwender nicht nur als Informationssuchende, sondern auch als Anbieter von Lösungen aktiv werden. Es gibt unzählige Mailinglisten und Newsgroups in den verschiedensten Größen und zu den unterschiedlichsten Themen, die von Einzelpersonen, Interessengruppen, Organisationen angeboten werden. Selbst Hersteller haben eigene Server dafür eingerichtet, um den Austausch Ihrer Kunden zu fördern und um dadurch den eigenen Kundendienst zu entlasten.

Bei Mailinglisten wird über eine Art Email-Rundfunk kommuniziert. Die Idee dabei ist, dass registrierte Teilnehmer Nachrichten an eine zentrale Emailadresse schicken und eine spezielle Serversoftware, ein so genannter Listserver, sie automatisch an alle anderen Listenteilnehmer weiter sendet. Auf diese Art können Fragen gestellt, diskutiert und beantwortet werden.

Trotz der asynchronen Art von Email macht das zugrunde liegende Format es möglich, die chronologische und thematische Abfolge von Fragen und Antworten nachzuvollziehen. Um einmal ausgetauschtes Wissen wieder zu verwenden, werden die Emails in einem Archiv abgespeichert und meist per Volltextsuche verfügbar. Um unnötigen Emailverkehr zu vermeiden, ist es üblich, vor der Veröffentlichung eigener Fragen das Archivmaterial zu durchsuchen. Um an Mailinglisten teilnehmen zu können, muss man sich mit der eigenen Emailadresse kostenfrei registrieren, damit dem Server diese bei der Verteilung von Nachrichten zur Verfügung steht.

Nicht automatisch verschickt werden Nachrichten von Newsgroups. Ein so genannter Newsreader, eine speziellen Software, die bereits in viele Emailprogramme integriert ist, holt sie vom Server ab. Statt sich anzumelden, weist man das Programm an, Nachrichten bestimmter Themengruppen regelmäßig herunterzuladen (=abbonieren).

URL-ähnliches Namenssystem bei Newsgroups

Wie auch bei Mailinglisten ist weder die Verwaltung der Newsgroups noch die Vergabe der Gruppennamen zentral organisiert. Vergleichbar mit dem Namenschema von Internetdomains (z.B. www.ibm.com) wird durch ein hierarchisches Namenssystems eine Struktur und Ordnung erzwungen. Themengruppen werden anhand eines URL-ähnlichen Namens eindeutig identifiziert und können beliebig fein systematisiert werden.

Obwohl es verbreitete Namenskonventionen und Ordnungsschemata gibt, werden manche Newsgroups nicht gefunden, da sie unter unüblichen Namen abgelegt sind.

Wie die folgenden Beispiele aus der Praxis zeigen, reicht es bei der Suche nicht aus, sich auf die Einhaltung der Konventionen zu verlassen. So sind Themengruppen zu IBM Produkten sowohl in der allgemeinen Rubrik *Computer – Datenbanken*, als auch direkt unter dem Namen des Herstellers zu finden. Länderspezifische Ausprägungen eines Themas (z.B. deutscher Linux-Foren) finden sich so oft unter unterschiedlichen Namen.

- comp.databases.ibm-db2 (IBM Datenbank DB2)
- ibm.software.websphere (IBM Webserver WebSphere)
- de.comp.os.unix.linux (deutsche Linux-Foren)
- comp.os.linux.development.apps (enthält auch deutsche Foren)

📖 Weitere Informationen zu Mailinglisten und Newsgroups unter [1]+[2]

Die große Vielfalt und Menge von Gruppen und Listen ist nicht nur ein Vorteil bei der Suche nach Lösungen. Zwar gibt es spezielle Internetsuchmaschinen (z.B. google.com/groups) für solche Foren, ist es trotzdem oft schwierig im akuten Problemfall schnell Personen zu finden, die helfen könnten. Die Trefferqualität der Suchmaschinen hängt stark von der Formulierung aussagekräftiger Suchbegriffe ab. Ist die Problemsituation schwer zu beschreiben und geben unverständliche Fehlermeldungen keine Hinweise, fällt dies dem Anwender oft schwer.

Dies gilt nicht nur für das Finden von Gruppen und Listen, sondern auch für die Suche der eigentlichen Informationen in Archiven. Wissen kann oft nicht wieder verwendet werden, da es mit den rudimentären Volltextsuchen nicht gefunden wird. Um dann trotzdem Hilfe zu erhalten und selbst eine Frage zu veröffentlichen, registrieren sich Hilfesuchende dann bei der Mailingliste oder abonnieren eine bestimmte Newsgroup.

Nun sind solche Hilfesysteme längerfristige Kommunikationskanäle, in denen sich interessierte Anwender immer wieder treffen und sich oft auch bereits kennen. Neulinge, die nur die einmalige Hilfe im akuten Problemfall suchen, werden schnell erkannt und stoßen selten auf prompte Unterstützung. Um trotz der Emailflut von anderen Teilnehmer schnell Hilfe zu bekommen, ist aber gerade die Aufmerksamkeit und Hilfsbereitschaft anderer Teilnehmer nötig. Sie müssen die Emails aufmerksam lesen und anhand oft unvollständiger Prosabeschreibungen das Problem verstehen.

Besonders verärgert sind dauerhafte Teilnehmer dann, wenn Fragen gestellt werden, die trivial oder themenfremd sind oder die bereits mehrmals beantwortet wurden und somit im Archiv gesucht werden sollten. Da jede zusätzliche Email die Nutzbarkeit eines solchen Forums verringert und den Aufwand für alle Teilnehmer erhöht, ist diese Verärgerung verständlich.

Selbst bei großer Hilfsbereitschaft kann es mehrere Tage dauern, bis die richtige Lösung gefunden ist und die Problemsituation behoben werden kann. Zuschriften auf die eigene Frage müssen manuell gesichtet werden, um irrelevante Antworten auszusortieren und erfolgsversprechende Vorschläge zu testen. Außerdem müssen oft in weiteren Emails Rückfragen beantwortet und Angaben konkretisiert werden.

1.3.3 Internetsuchmaschinen

Um sich den Aufwand beim Suchen geeigneter Informationsquellen zu ersparen, beginnen die meisten Anwender ihre Lösungssuche direkt in einer regulären Internetsuchmaschine.

Suchmaschinen indizieren jede für sie erreichbare Internetseite unabhängig von ihrem Themengebiet. Sie finden deshalb auch die Inhalte der Archive von Mailinglisten bzw. Newsgroups und helfen damit, deren oft schlechte Suchsysteme zu umgehen. Nicht nur erzielen die ausgereiften Suchalgorithmen der Suchmaschinen (z.B. www.google.de) bessere Treffer, sondern sie decken mit einer einzigen Suche die gesamte ihnen verfügbare Information ab.

Allerdings sind Suchmaschinen nicht mit anderen Hilfesystemen vergleichbar. Sie sind reine Suchhilfen, bei denen die Bereitstellung von Information nicht möglich ist. Sie haben lediglich die Aufgabe, unter den existierenden Internetseiten, solche zu finden, die für die Suchbegriffe des Benutzers relevant sind. Die eigentlichen Informationen (z.B. Lösungen für Softwareprobleme) liefern die Betreiber der Internetseiten. Um als Anwender eigene Inhalte beizusteuern, müsste man also eine eigene Internetseite veröffentlichen oder die Möglichkeiten bestehender Foren nutzen. In beiden Fällen würde die Information aber nicht sofort für andere Anwender erreichbar sein, da Suchmaschinen neue Internetseiten oft erst nach Tagen oder Wochen erkennen und indizieren.

1.3.4 Präventive Fehlervermeidung

Sozusagen außer Konkurrenz sind Hilfesysteme, die Softwarefehler und andere Problemsituationen bereit vor ihrem Auftreten vermeiden. Ein derartiges System bietet z.B. *Microsoft* mit seinem *Windows Update* (windowsupdate.microsoft.com) an. So gibt es in vielen Microsoft Produkten die Möglichkeit, über eine Internetverbindung fehlende Updates automatisch zu identifizieren und nachträglich zu installieren oder Sicherheitslecks in einem Programm zu beheben. Überprüfung können dabei entweder manuell vom Anwender initiiert werden oder direkt von den einzelnen Programmen periodisch durchgeführt werden. Eine solche präventive Fehlervermeidung bringt für die Anwender offensichtliche Vorteile. Durch die automatische Aktualisierung der Software wird vermieden, dass Anwender erneut in Problemsituation geraten, für die es bereits Lösungen gibt.

Selbstverständlich können auch von diesen Hilfesystemen nur solche Probleme vermieden werden, die zumindest dem Hersteller bekannt sind und er Lösungen dafür bereitstellt. Die nötige enge Verflechtung mit der Software macht es aber unmöglich, solche Aktualisierungstools unabhängig von bestimmten Produkten und Herstellern anzubieten. Außerdem wirft die für den Anwender transparente Überprüfung lokaler Software über eine Internetverbindung zahlreiche Sicherheits- und Datenschutzfragen auf.

1.3.5 Schwächen existierender Hilfesysteme

Ziel des neuen Hilfesystems ist es, besser und schneller zu helfen, als dies bisher möglich war. Es konzentriert sich auf Problemsituationen, in denen weder die eingebauten Hilfsmittel der Software wie Fehlermeldungen und Produktdokumentation helfen das Problem sofort zu beheben, noch geeignet sind das Problem für die Suche in anderen Hilfesystemen richtig zu beschreiben.

Wir möchten hervorheben, dass die existierenden Systeme in anderen Situationen hervorragend arbeiten und meist gar nicht den Anspruch haben Probleme jeder Art perfekt zu lösen. Wir wollen die bestehenden Systeme weder verbessern noch ersetzen, sondern nur bisher unbefriedigend gelöste Fälle in Zukunft besser behandeln.

Im folgenden Abschnitt werden an konkreten Beispielen bestimmte Mängel einzelner Systeme erläutert.

aufwendige Suche geeigneter Informationsquellen

Internetsuchmaschinen leisten als zentrale Anlaufstellen bei der Suche für einen großen Teil der online verfügbaren Information wertvolle Arbeit. Je nach Leistungsfähigkeit werden bestimmte Informationsquellen aber entweder zu spät oder gar nicht gefunden. Außerdem werden oft irrelevante Informationen als Suchtreffer zurückgegeben. Da ihre primäre Aufgabe nicht das Finden von Problemlösungen ist, sind sie nicht geeignet um aus beliebigen Suchtreffern, solche mit konkreten Lösungsvorschlägen zu filtern.

Obwohl Mailinglisten und Newsgroups reichhaltige Wissensquellen sind, ist das Finden eines geeigneten Forums wegen ihrer Vielfalt und dem lockeren Namenschema oft schwierig.

schlechte Wiederverwendung vorhandener Lösungen

Je mehr Menschen komplexe Softwaresysteme benutzen, desto wahrscheinlicher ist, dass Anwender immer wieder in ähnliche Problemsituationen geraten. Es ist daher erstrebenswert, das Wissen von Experten die eine Problemsituation bereits erfolgreich gelöst haben, wieder zu verwenden und anderen Anwendern in einer ähnlichen Situation zur Verfügung zu stellen.

Zu diesem Zweck sammeln viele Hersteller häufige Fragen und Antworten und machen sie im Internet als FAQs zugreifbar. Lösungen für seltene oder dem Hersteller noch unbekannte Probleme sind dort aber nicht zu finden.

Zur Wiederverwendung von Antworten bieten Mailinglisten und Newsgroups Archive an, die bei Verwendung aussagekräftiger Suchbegriffe gute Ergebnisse liefern. Bei missverständlichen Fehlermeldungen fällt dem Anwender das Formulieren der Suchbegriffe oft schwer und der Nutzen solcher Archive bleibt gering. Oft werden deshalb häufige Probleme erneut mit großem Aufwand gelöst oder als Frage veröffentlicht. Erfahrene Anwender müssen dann oft ihr Wissen wiederholt manuell weitergeben.

aufwendiges Bereitstellen eigener Lösungen

Werden von Anwender neue Lösungen gefunden, ist es erstrebenswert, diese mit minimalem Aufwand anderen zur Verfügung zu stellen.
Dies geht bei bisherigen Systemen entweder überhaupt nicht (Suchmaschinen, Herstellersupport) oder nur mit hohem Aufwand. Um in Mailinglisten und Newsgroups eigenes Wissen beizusteuern, ist die längerfristige Teilnahme in verschiedenen Foren und die zeitaufwendige Beschäftigung mit den Problemen anderer Anwender nötig.

mangelhafte Problembeschreibung in Prosa

Keines der vorgestellten Hilfesysteme löst Probleme mit automatischen Mitteln selbst. Sie arbeiten lediglich als Makler zwischen Nachfragenden und Anbietern von Lösungen. Die eigentlichen Lösungen zu Behebung eines Problems produzieren nach wie vor Menschen. Bei allen vorgestellten Verfahren ist es nötig, die aufgetretene Problemsituation in Prosa zu beschreiben – entweder am Telefon oder in einer Nachricht an andere Forumsteilnehmer.
Eine treffende Beschreibung ist bei seltenen Fehlern oder unverständlichen Fehlermeldungen der Software nicht immer möglich. Obwohl Menschen auch unklare Fragen verstehen und durch Rückfragen Unklarheiten ausräumen können, sind wegen des hohen Aufwands und der schlechten Skalierbarkeit einer solchen direkten Kommunikation enge Grenzen gesetzt.

manuelles Aussortieren irrelevanter Information

Selbst wenn existierende Systeme trotz der beschriebenen Mängel, problemrelevante Informationen liefern, ist ein aufwendiges manuelles Sichten und Aussortieren nötig. So müssen die Zuschriften auf Fragen in News- oder Emailforen gelesen, verstanden und oft durch Rückfragen konkretisiert werden. Automatische Methoden zur Einordnung der Information nach Relevanz für das eigene Problem gibt es derzeit nicht.

1.4 Begriffsabgrenzung und Aufgabenstellung

Ziel dieser Diplomarbeit ist die Realisierung eines verteilten Hilfesystems auf Basis einer Grundidee von IBM. Diese wird im nächsten Abschnitt vorgestellt. Mit dem neuen Hilfesystem sollen in Ausnahmesituationen, in denen existierende Hilfesysteme nicht weiterhelfen, aus einem Bestand existierender Lösungen solche gefunden werden, die das Problem mit hoher Wahrscheinlichkeit lösen. Außerdem soll der Aufwand für das Bereitstellen eigener Lösungen minimiert werden.

In der Diplomarbeit beschränken wir uns dabei auf Problemsituationen, die bei der Benutzung von Software auftreten und bei denen das Problem sich in Form einer Fehlermeldung manifestiert. Unerwartete Programm- oder Systemabstürze oder Situationen in denen der Benutzer an der Arbeit gehindert wird, weil er beispielsweise sein Passwort vergessen hat, klammern wir damit aus. Solche Fälle können oder sollen nicht behandelt werden, da sie entweder nicht im Einflussbereich eines Hilfesystems liegen oder wie im Fall von Sicherheitsbarrieren genau der Intention einer Software entsprechen.

Als Umgebung für die Implementierung wurde der *IBM WebSphere Application Developer* gewählt.

Das Softwarepaket enthält einen Web Application Server und eine Entwicklungsumgebung zur Realisierung komplexer Geschäftsprozesse auf Basis von Web Services, Enterprise Java Beans und J2EE. Während der Anwender dort meist graphisch oder dialoggesteuert arbeitet, werden große Teile des Programmcodes automatisch generiert. Aufgrund verborgener Abhängigkeiten der erzeugten Komponenten führen nachträgliche Code- und Konfigurationsänderungen leicht zu Laufzeitfehlern in Form von Java Stacktraces.

KAPITEL 2 Grundidee eines neuen Hilfesystems

Das Grundkonzept für ein neues Hilfesystem zur Verbesserung der beschriebenen Problemsituation wurde bereits vor Beginn der Diplomarbeit von IBM erarbeitet. Zur Zeit läuft bei IBM eine Patentanmeldung um eine eventuelle kommerzielle Nutzung des Konzepts zu ermöglichen. Da das Konzept Ausgangspunkt dieser Arbeit und der in den nachfolgenden Kapiteln erläuterten Beispielrealisierung ist, werden seine Kernpunkte im Folgenden zunächst genauer betrachtet.

Im Einführungskapitel wurde festgestellt, dass es Problemsituationen bei der Benutzung von Software gibt, bei denen die bestehende Systeme Schwachpunkte haben und nicht zufrieden stellend helfen können. Genau an diesen Stellen wollen wir mit einen neuen System Verbesserungen erreichen. Wie bei anderen Hilfesystemen ist auch hier nicht der Anspruch, Problemsituationen mit Software generell zu verhindern. Dies ist wegen der wachsenden Komplexität der Anwendungen, unterschiedlichster Benutzerkreise und Durchdringung zahlreicher Lebensbereiche praktisch unmöglich. Stattdessen soll vermieden werden, dass dieselben Probleme parallel immer wieder von neuem gelöst werden müssen, weil vorhandenes Wissen nicht oder nur mit großem Aufwand weitergegeben werden kann. Unser System soll die Rolle eines Maklers übernehmen, der Anwender bei der Bereitstellung und der Vermittlung von Lösungen unterstützt. Als Namen für das neue System haben wir uns deshalb im Rahmen dieser Diplomarbeit für **SolutionBroker** entschieden.

Der Erfolg eines Maklers hängt von der Akzeptanz durch beide Seiten ab - den Personen, die ihr Wissen zur Verfügung stellen wollen und denen, die es zur Lösung Ihrer Probleme benötigen. Um seiner Rolle gerecht zu werden, muss der Makler beiden Seiten so viel Arbeit wie möglich abnehmen. Unser Ziel ist es, dies in einem höheren Maß zu tun als die vorgestellten Hilfesysteme. Dabei ist die Umsetzung der folgenden Eigenschaften in unserem Konzept von zentraler Bedeutung.

Hauptzielsetzungen des Konzeptes:

- geringerer Aufwand beim Finden relevanter Lösungen
- geringerer Aufwand beim Beisteuern eigener Lösungen

2.1 Zentrale Lösungsdatenbank

Es ist Stand der Technik, dass bisher nur Menschen in der Lage sind, Lösungen für komplexe Problemsituationen zu finden. Ein Mangel existierender Systeme ist aber die Notwendigkeit einer direkten Kommunikation zwischen Anwendern. Da sich dabei nur wenig automatisieren lässt, skalieren solche Verfahren schlecht und sind recht teuer.

Das Weitergeben von Wissen in Mailinglisten oder anderen freiwilligen Foren ist mit längerfristigem Engagement verbunden und erfordert von den Teilnehmern ein großes Maß an Hilfsbereitschaft.

Diesen Aufwand sind viele Anwender nach einer zeitintensiven Lösungssuche selten bereit aufzubringen. Das gilt besonders im professionellen Umfeld wo Problemlösungen zwar erfolgskritisch sind, Zeit aber knapp ist. Wertvolle Erfahrung wird deshalb oft nicht mit anderen Anwendern geteilt.

Außer beim Bereitstellen von Lösungen auf eigenen Internetseiten war bisher eine Kontaktaufnahme zwischen Anwendern nötig, weil es keine Möglichkeiten gab, gefundene Lösungen sofort zu veröffentlichen. Dies soll sich durch die Bereitstellung einer zentralen Lösungsdatenbank ändern. Dort sollen Anwender mit minimalem Zusatzaufwand gefundene Lösungen sofort veröffentlichen können.

Ablegen von Lösungen auf Vorrat möglich

Bisher wurden Problembeschreibungen und Lösungsvorschläge zeitlich getrennt voneinander abgelegt. Dies liegt hauptsächlich daran, dass Problembeschreibungen bzw. Lösungen von unterschiedlichen Personen stammen. Ein Anwender veröffentlicht eine Frage und bekommt eventuell von anderen Anwendern Lösungsvorschläge. Welcher diese Vorschläge aber schließlich sein Problem gelöst hat, ist entweder gar nicht aus den Archivdaten ersichtlich oder nur durch aufwendiges manuelles Sichten des kompletten Gesprächsfadens. In unserer Lösungsdatenbank hingegen sollen Probleme inklusive ihrer Lösung von ein und derselben Person abgelegt werden können. Damit wird garantiert das nur Einträge gefunden werden, die erfolgreiche Lösungen enthalten. Lösungssuchende können so auch nach langer Zeit auf die Einträge in der Datenbank zugreifen. Einmal gewonnenes Wissen geht so nicht verloren.

Ein besonderer Vorteil dieses Konzepts ist der asynchrone Wissensaustausch. Ein direkter Kontakt zwischen Anbietern und Nachfragern von Lösungen mit seinen Nachteilen ist nicht mehr nötig. Das Abspeichern in der Datenbank externalisiert das Wissen, dass längerfristiges persönliches Engagement in Mailinglisten und Newsgroups nicht erforderlich ist.

Eine zentrale Anlaufstelle für Software-Probleme jeder Art

Ein weiteres Problem war bisher die große Vielfalt unterschiedlicher Informationsquellen und die damit verbundene Schwierigkeit, relevante Datensammlungen oder Expertenrunden zu finden. Unsere Datenbank soll das Potential einer zentralen Anlaufstelle für Software-Probleme unabhängig von bestimmten Produkten oder Herstellern bieten und somit dieses Problem umgehen. Um von einem möglichst großen Anwenderkreis erreichbar zu sein, wird die Datenbank mit einfachen Mitteln über das Internet zugänglich sein.

Eine intuitive Benutzerführung soll die Hemmschwelle senken, die bisher viele Anwender abgehalten hat, ihr Wissen weiterzugeben. Wollten sie bisher eine gefundene Lösung anbieten, mussten sie diese entweder auf einer eigenen Seite im Internet veröffentlichen oder in Newsgroups und Mailinglisten nach Personen suchen, die genau dieses Wissen benötigen.

Viele Funktionen des Systems sollen automatisiert werden, um im Gegensatz zu teuren Hotlines einem großen Anwenderkreis preiswert zur Verfügung zu stehen.

2.2 Client-Server Architektur

Wegen der Rolle des Hilfesystems als Makler zwischen Anbietern und Nachfragenden von Lösungen bietet sich für die Umsetzung eine Client-Server-Architektur an. Dabei greifen die verschiedenen Benutzer über eine lokale Software (Clientanwendung) auf die Funktionen eines zentralen Computers zu (Serveranwendung).

Die Aufgaben der Clientanwendungen sind:

- Möglichkeit zu Eingabe Lösungsanfragen und -beiträgen
- Versand von Lösungsanfragen und -beiträgen
- Empfang und Darstellung der Ergebnisse vom Server

In unserem Fall sind die Aufgaben der Serveranwendung:

- Empfang von Suchanfragen und Lösungsbeitragen
- dauerhafte Speicherung von Lösungsbeiträgen in Datenbank
- Suche von Lösungen für Benutzerprobleme
- Versand der Suchergebnisse zurück an die Clients

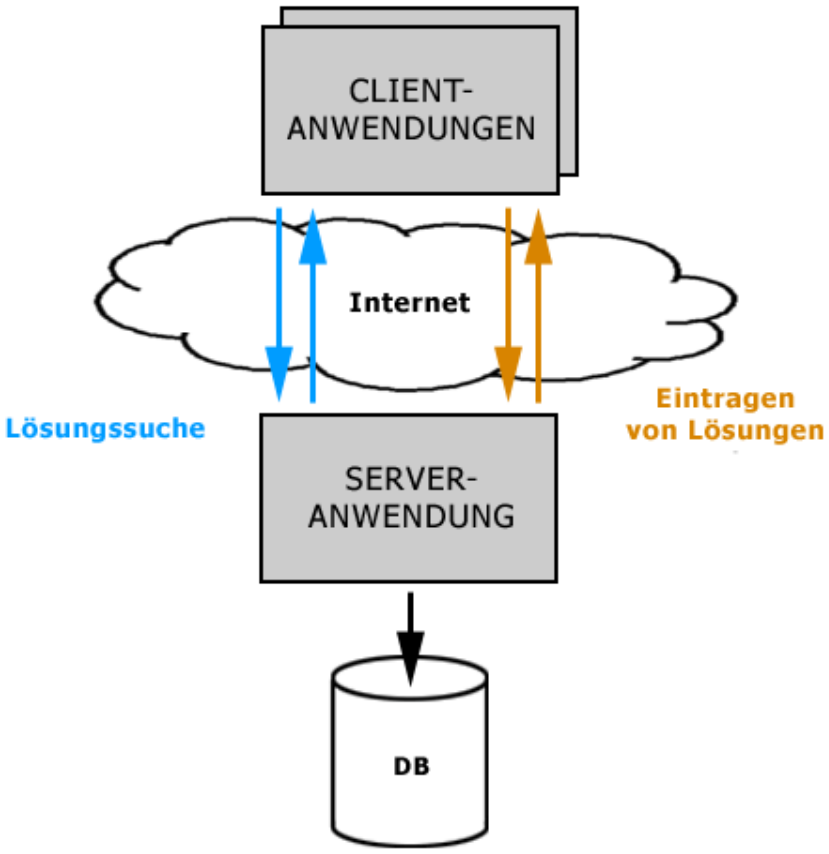

Abbildung 1 Architektur des neuen Hilfesystems

2.3 Einbeziehung von Kontextinformation

Die Qualität der Antworten menschlicher Experten hängt in hohem Maße von der aussagekräftigen Beschreibung der vorliegenden Problemsituation ab. Je komplexer die verwendete Software ist und je ungewöhnlicher die Problemsituation ist, desto schwerer fällt dem Anwender diese Beschreibung. Oft fehlen aussagekräftige Fehlermeldungen, um Art und Ursache des Problems näher zu spezifizieren. So weiß der Anwender zwar, dass er in einer Problemsituation steckt, nicht aber, wie er dort hineingeraten ist und wie der Fehler behoben werden kann. Je nach Kenntnisstand der Anwender wird die Situation dann unterschiedlich und eventuell falsch interpretiert. Ein Verständnis der Situation ist vor allem wichtig, wenn die Hilfe anderen Personen in den Hilfeforen in Anspruch genommen werden soll. Um ihre eigenen Erfahrung weitergeben zu können, müssen diese erkennen, dass es sich um ein Problem handelt, das sie bereits erfolgreich gelöst haben.

Um in solchen Fällen ein Problem klar zu beschreiben, ist oft die Kenntnis zusätzlicher Kontextinformationen wichtig. Solche Informationen sind z.B.:

- Was hat der Benutzer vor dem Auftreten des Problems gemacht?
- Wo im System tritt der Fehler auf?
- Welche fremden Software- und Hardwarekomponenten interagieren mit der fehlerhaften Software?

Aussagekräftige Problembeschreibung durch zusätzliche Kontextinformation

Die Kenntnis des Problemkontexts ist vor allem wichtig, wenn die Ursache in Systemteilen liegt, die nicht im Einflussbereich des Herstellers liegen. Unzählige Kombinationen möglicher Fremdeinflüsse machen weder die Kenntnis aller potentiellen Fehler noch deren detaillierte Behandlung durch Fehlermeldung möglich. Sind Fehlermeldungen allein nicht hilfreich, muss zusätzlich auf Daten zugegriffen werden, die unabhängig vom Hersteller zur Problemzeit vorliegen.

Bei der Auswahl von Kontextinformation sollen nur relevante Daten herangezogen werden, die das Problem eindeutig spezifizieren, ohne es überzuspezifizieren. Ein besseres Verständnis könnte so beispielsweise durch Kenntnis der Benutzeraktionen und der internen Prozesse vor dem Auftreten des Problems gewonnen werden, aber z.B. nur selten durch die Angaben der Systemzeit und des Benutzernamens.

Unser System soll die Qualität der Suchergebnisse vor allem dadurch verbessern, dass Probleme anhand solcher relevanten Kontextinformationen besser beschrieben werden. Nicht nur bei der Suche, sondern auch beim Ablegen von Lösungen sollen derart angereicherte Problembeschreibungen in der Datenbank mit abgelegt werden. So ist später erkennbar, ob ein vorliegendes Problem einem Datenbankeintrag ähnlich ist und die existierende Lösung wieder verwendet werden kann.

2.4 hohe Automatisierung

Ein Grund für den bisher hohen Aufwand bei die Suche und dem Beisteuern eigener Lösungen war, dass die meisten Arbeitsschritte manuell erledigt werden mussten. Um beispielsweise eine Frage in Mailinglisten oder Newsgroups zu stellen, muss erst eine Problembeschreibung in Prosa formuliert werden. Nach der Veröffentlichung dauert es oft Stunden oder Tage bis andere Teilnehmer mit Zuschriften darauf reagieren. Die Antworten enthalten nur selten sofort komplette Lösungen. Viele entpuppen sich als nicht direkt anwendbar oder bringen erst nach mehrmaligen Rückfragen die gewünschte Beseitigung des Problems. Bei der Verwendung ungeeigneter Suchbegriffe in den Archiven oder Suchmaschinen muss außerdem eine Vielzahl ungeeigneter Treffer aussortiert werden.

Maschinenlesbarkeit erlaubt automatisches Problem-Matching

Wir wollen diese Schwierigkeiten umgehen, indem die Problembeschreibungen von der Software automatisch und in einer maschinenlesbaren Form zur Verfügung gestellt werden. Dabei werden nicht nur die Fehlermeldung, sondern auch Kontextinformationen extrahiert. Neben der Tatsache, dass dabei Problembeschreibungen immer in einer einheitlichen und vergleichbaren Form vorliegen, ist durch ihre Maschinenlesbarkeit eine automatische Verarbeitung möglich. Nachteile von Prosabeschreibungen, wie widersprüchliche Aussagen, subjektive Einschätzungen und Auslassungen wichtiger Angaben werden so vermieden.

Die Automatisierung wichtiger Funktionen macht das Hilfesystem außerdem preiswerter, effizienter und besser skalierbar als andere Hilfesysteme.

Um die Relevanz existierender Lösungsvorschlägen zu beurteilen, musste der Anwender bisher das eigene Problem mit dem der fremden Lösung vergleichen. Nur so war festzustellen, ob es sich um ähnliche Probleme und damit eventuelle Lösungen handelt. Dieser Vergleich kann automatisiert werden, wenn auch die Lösungseinträge in der Datenbank eine maschinenlesbare Problembeschreibung enthalten. Mit Hilfe der Automatisierung können sofort für eine Anfrage passenden Lösungen gefunden werden, da die Serveranwendung die Problembeschreibung des Benutzers mit denen in der Datenbank vergleichen kann. Dabei werden nur solche Lösungsvorschläge ausgegeben, bei denen es sich um dasselbe oder zumindest ein ähnliches Problem handelt.

Andere Aspekte, die bei einer manuellen Suche nicht möglich waren, können ebenfalls zur Bestimmung der Relevanz von Lösungen beitragen. So könnte Anwendern die Möglichkeit gegeben werden, nachträglich den Nutzwert einer Lösung zu bewerten. Ähnlich wie bei den Produktbewertungen des Online-Buchhändler *www.amazon.de,* könnten dann solche Lösungen mit besserer Bewertung bevorzugt empfohlen werden.

KAPITEL 3 „SolutionBroker" – eine Beispielrealisierung

Aufgabe der Diplomarbeit war es, die Grundidee des neuen Hilfesystems aus dem vorhergehenden Kapitel in einer Beispielimplementierung zu realisieren und so dessen Praxistauglichkeit zu belegen.

Dazu werden im folgenden Kapitel jene Bereiche erörtert, zu denen das Grundkonzept keine Aussagen macht und die in der Diplomarbeit erarbeiteten Lösungen vorgestellt. Diese sind:

- Schaffung einer robusten und skalierbaren Anwendungsarchitektur,
- Finden eines geeigneten Formats zur Problembeschreibung,
- Erarbeiten eines Matchingalgorithmus zu automatischen Lösungssuche.

3.1 Anforderungen an die Anwendung

3.1.1 Verwendung von Standardtechnologien

Eine wichtige Voraussetzung für den Nutzen des **SolutionBrokers** ist es, möglichst viele Benutzer als Anbieter oder Nachfragende von Lösungen zu erreichen. Nur die Verwendung akzeptierter und plattformunabhängiger Technologien ermöglicht eine weite Verbreitung. Im Folgenden werden die verwendeten Technologien vorgestellt und begründet, warum sie für das System besonders geeignet sind.

Netzwerkkommunikation

Wegen seiner weiten Verbreitung bietet sich das Internet als Kommunikationsnetz besonders an. Zum Austausch von Daten und dem Aufruf entfernter Programmfunktionen über das Internet gibt es unter dem Namen „Verteilte Systeme" zahlreiche Übertragungsprotokolle. Diese sind nötig, um eine logische Verbindung zwischen entfernten Rechner aufzubauen, die fehlerfreie Übertragung der Daten zu ermöglichen und die Kommunikation zu synchronisieren.

Allerdings sind nicht alle dieser Mechanismen für unsere Zwecke geeignet. Wegen der gewünschten großen Verbreitung möchten wir uns nicht auf sprach- oder systemabhängige Konzepte wie Microsofts DCOM oder Suns Java RMI beschränken. Da anstatt einfacher Basistypen bei Anfragen und Ergebnissen strukturierte Daten ausgetauscht werden sollen, entfallen auch die klassischen RPC-Protokolle. Gegen Konzepte wie CORBA sprechen vor allem die Notwendigkeit teurer Softwarekomponenten, für unsere Zwecke unnötige Features und zahlreiche einschränkende Voraussetzungen.

Netzkommunikation über Standardtechnologien mit XML Web Services

Wegen seiner Einfachheit, der Verfügbarkeit freier Implementierungen und der raschen Verbreitung wollen wir stattdessen die relativ junge Technik XML-basierter Webservices nutzen. XML Web Services verwenden im Gegensatz zu älteren Protokollen bestehende Übertragungsstandards wie HTTP über IP und XML. Kommunikationspartner nehmen über eine IP-Adresse Kontakt auf und tauschen Daten aus. Als universelles Austauschformat dient XML.

Mit Hilfe der Strukturierung- und Typisierungsmöglichkeiten von XML und der Erweiterungssprache XML Schema ist es möglich, eigene komplexe Datenstrukturen zu definieren. Die technischen Details des Verbindungsaufbaus und bei der Übertragung der Daten übernehmen dabei so genannte Proxy-Objekte. Sie dienen jeweils auf Seite des Clients und des Servers als lokale Stellvertreterobjekte des entfernten Kommunikationspartners. Um den Programmierern die Implementierung dieser komplexen Programme zu ersparen, können sie automatisch aus einer Schnittstellenbeschreibung generiert werden. Bei XML Web Services liegen diese Informationen über Ablauf und Syntax der Kommunikation ebenfalls im XML-Format vor. Die konkrete Schnittstellenbeschreibung des **SolutionBrokers** findet sich weiter unten in den Implementierungsdetails der Serveranwendung.

🕮 Für eine ausführliche Beschreibung von XML Web Services siehe [16].

Für weitere Details zum Themenkreis XML Web Services sei auf die angegebene Onlinequelle verwiesen.

Anwendungsprogrammierung

Zur Programmierung der Client- und Serveranwendung bietet sich eine plattformunabhängige Sprache an, um eine große Verbreitung der Clients und eine leichte Portierung des Servers zu erlauben. Obwohl diese Voraussetzung auch von anderen Sprachen wie z.B. Microsofts C# erfüllt wird, macht die große Verbreitung der Programmiersprache Java diese zum besten Kandidaten. Außerdem unterstützt eine große Zahl von Webservern die Veröffentlichung java-basierter Web Applikationen. Dies gilt auch für die geplante Zielumgebung, den *IBM WebSphere* Webserver. Die Verfügbarkeit zahlreicher freier Funktionsbibliotheken für Java ist ein weiterer wichtiger Grund für die Entscheidung.

3.1.2 Robustheit

Robustheit ist eine typische Anforderung an Client-Serveranwendungen, die Tag und Nacht zuverlässig erreichbar sein. Ein System ist dann robust, wenn es gegenüber Fehlbenutzung, Ressourcenengpässen oder anderen seltenen kritischen Umständen unanfällig ist.

Zum Teil wird diese Anforderung bereits durch die verwendeten Basistechnologien erfüllt. So arbeiten DB2, WebSphere und die XML Web-Services auch bei hohen Zugriffslasten oder instabilen Netzwerkverbindungen stabil. Durch eigene Erweiterungen, wie z.B. einem JDBC-Connection Pooling zur ressourcensparenden Wiederverwendung von Datenbankverbindungen wird der Speicherbedarf der Serveranwendung verringert, um mehr gleichzeitige Benutzerzugriffe zu ermöglichen.

🕮 Tutorial zu Java Connection Pooling unter [6]

Durch die Kapselung der Hauptfunktionen des **SolutionBrokers** in separate Javaobjekte und Packages mit eigenen Exceptionklassen werden unerwünschte Abhängigkeiten vermieden und Fehler besser lokal behandelt.

3.1.3 Skalierbarkeit

Eine weitere Anforderung neben der Robustheit ist die Skalierbarkeit des Systems, also die Fähigkeit, mit steigenden Anforderungen und Belastungen ohne Qualitätseinbußen mitzuwachsen, ohne dass grundlegende Änderungen nötig sind.

Die kritischen Faktoren unseres Systems sind die Anzahl der Benutzer und die Zahl der Lösungseinträge in der Datenbank. Je mehr Benutzer gleichzeitig Lösungen suchen oder eintragen, desto mehr Anfragen müssen von der Serveranwendung parallel bearbeitet werden.

Je mehr Einträge in der Datenbank abgespeichert sind, umso mehr Vergleiche sind bei einer automatischen Lösungssuche nötig. Die Beispielimplementierung unseres Konzeptes wurde nur im kleinen Rahmen mit einem Dutzend Benutzern und wenigen tausend Datenbankeinträgen getestet. Bei Erreichen der gewünschten Zielgruppe sind allerdings schnell mehrere tausend Benutzer und hunderttausender Datenbankeinträge zu erwarten. Für den Fall steigender Benutzerzahlen bzw. steigender Datenmengen werden im Implementierungskapitel mögliche Optimierungen vorgestellt.

3.2 Systemkomponenten

Die folgende Grafik zeigt die Hauptbestandteile der Beispielrealisation. Die Hauptkomponenten und deren Aufgaben sind:

- *Relationale Datenbank* zur dauerhaften Speicherung von Lösungsdaten
- *XML Web Services* Schicht zur Kommunikation zwischen Client und Server
- *JSP-Browserinterface* Serverzugriff über Client in Standard-Webbrowsern
- *Eclipse-Plugin* als Clientintegration in Standardsoftware-Anwendung

In den folgenden Abschnitten werden die damit verbundenen technischen und konzeptuellen Entscheidungen begründet.

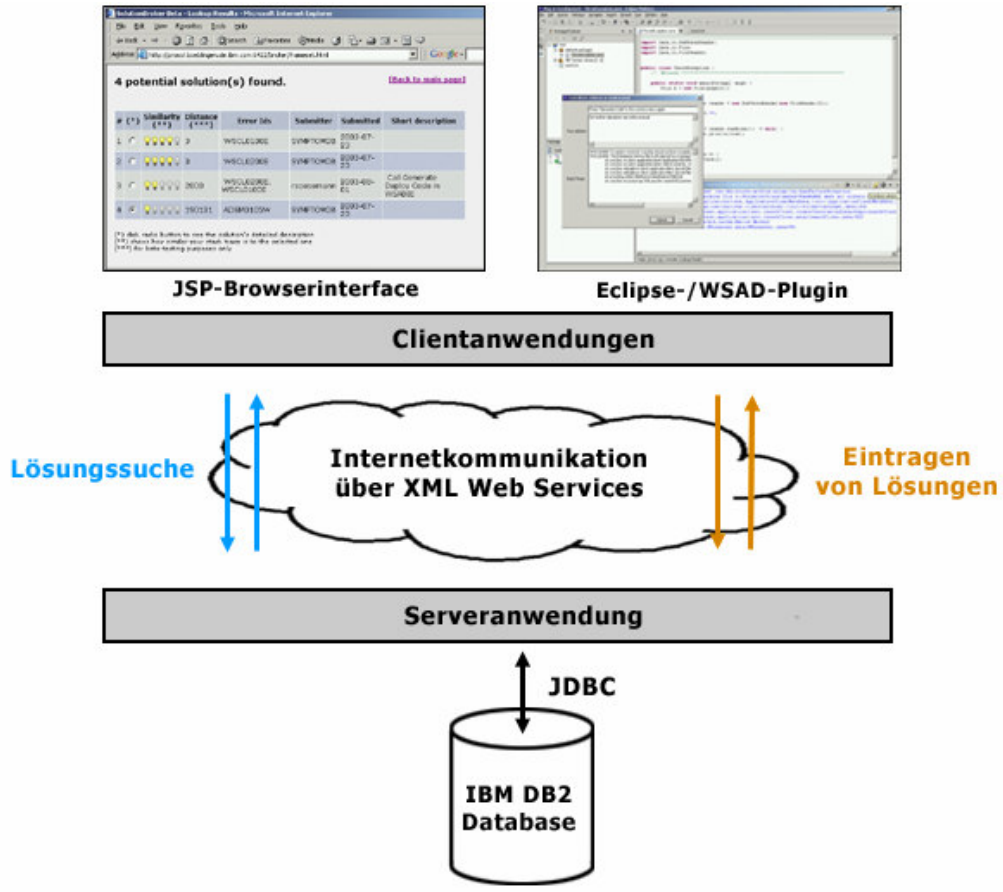

Abbildung 2 Komponenten des SolutionBrokers

3.2.1 Serveranwendung

Als Plattform für die Bereitstellung der Serveranwendung und des JSP-Interfaces im Internet wählen wir den *IBM WebSphere Application Server 5.0*, da er bereits die Verwendung von *XML Web Services* unterstützt. Dies ist nötig, da die Dienste des **SolutionBrokers** sowohl für das JSP-Browserinterface als auch entfernte Instanzen des Eclipse-Plugins über eine einheitliche XML Web Service Schnittstelle zur Verfügung gestellt werden sollen. Ein weiterer Vorteil ist, dass die zum WebSphere-Paket gehörende Entwicklungsumgebung die Erzeugung der Web Service Komponenten stark vereinfacht und eng mit dem Application Server zusammenarbeitet.

3.2.2 Datenbank

Wir benötigen auf dem Server eine performante und stabile Möglichkeit Daten über Problemsituationen und dazugehörige Lösungen abzuspeichern und abzufragen. Wir entscheiden uns für die Verwendung einer relationalen Datenbank. Diese Art der Datenspeicherung bietet zahlreiche Vorteile.

Durch die Zerlegung der Information in Tabellen und Attribute sind detaillierte und effiziente Abfragen möglich, was vor allem bei der automatischen Lösungssuche oder zur redundanzfreien Ablage von Daten wichtig ist. Die Reife der zugrunde liegenden technischen Konzepte (relationale Algebra) erlauben mit der Abfragesprache SQL Datenselektion in Geschwindigkeiten, die weit über die Möglichkeiten anderer Speicherungsmethoden liegen(z.B. Speicherung in Dateien). Außerdem bieten sie bereits viele Funktionen, die wichtige Voraussetzungen einer robusten Client-Server-Anwendung abdecken.

Stabilität und Performance durch RDBMS

Wie die meisten Client-Server Systeme ist auch der **SolutionBroker** eine typische Mehrbenutzeranwendung. Das Transaktionskonzept relationaler Datenbanken garantiert uns bei gleichzeitigem Zugriff mehrerer Benutzer die Konsistenz der Daten. Durch automatische Backups und Datenspiegelung bieten sie ein hohes Maß an Datensicherheit, die bei der hohen wirtschaftlichen Relevanz der gespeicherten Informationen unverzichtbar ist.

Ein solches Datenbanksystem ist zum Beispiel das hier verwendetet *DB2 Universal Database 7.2* von IBM. Aber auch die Verwendung anderer relationaler Datenbanksysteme (z.B. Oracle) ist möglich, da in der Implementierung keine herstellerspezifischen Erweiterungen verwendet werden.

3.2.3 Webbasierte Clientanwendung

Die Verwendung des Internets als Kommunikationsnetzwerk legen es nahe, für den Client die Software zu verwenden, die die meisten Anwender bereits installiert haben – einen Webbrowser. Als eine Ausprägung der Clientanwendung soll daher ein browserbasiertes Webinterface in Form einer Internetsite angeboten werden, über die mit der Serveranwendung kommuniziert werden kann.

Da abhängig von der Benutzeranfrage und der Datenbankinhalte Webinhalte in Form von HTML-Seiten angezeigt werden sollen, müssen diese auf Serverseite dynamisch erzeugt werden. Wegen der nahtlosen Zusammenarbeit mit der Programmiersprache Java entscheiden wir uns für die Technik der *Java Server Pages*. Andere mögliche Kandidaten sind das freie *PHP*, *Microsofts Active Server Pages* (ASP) oder *Allaires ColdFusion*.

📖 *Weitere Informationen über Java Server Pages in [3]*

Java Server Pages und Ihre Schwestertechnologie Java Servlets erweitern Java um die Möglichkeit, Webseiten dynamisch auf Serverseite zu erzeugen. Sie ähneln dabei syntaktisch statischen HTML-Seiten, die durch Einfügen zusätzlicher Javacode-Abschnitte um Logikelemente erweitert werden. Java Server Pages werden von einer Komponente des Webservers (dem sog. Servlet Container) verwaltet und in Javaprogramme (Servlets) übersetzt. Ruft ein entfernter Webbrowser eine Java Server Page auf, generiert der Servlet Container anhand der Logikbausteine eine statische HTML-Seite und schickt sie anschließend zurück. Typische Aktion während der dynamischen Generierung sind Datenbankabfragen und die Interaktion mit anderen Funktionskomponenten der Businesslogik.

3.2.4 Integriertes Client-Plugin

Um von größtmöglichem Nutzen zu sein, sollten die Dienste des **SolutionBrokers** dort zur Verfügung stehen, wo Problemsituationen üblicherweise auftreten oder gelöst werden – direkt in Anwendungsprogrammen.

Die Umsetzung des Clients als Website hat aber den Nachteil, dass z.B. bei der Abfrage von Lösungen zusätzlich der Webbrowser geöffnet und die Information über den Problemkontext manuell hineinkopiert werden muss. Eine zweite Clientanwendung soll direkt in ein Anwendungsprogramm eingebaut werden, um die Vorteile dieser Integration zu zeigen. Dadurch ist es möglich, dass Benutzer nur signalisieren, dass eine Problemsituation vorliegt und die Software dann automatisch die nötige Problembeschreibung zusammenstellt, um eine Anfrage beim Server zu starten.

📖 *Mehr Informationen zu Eclipse und dem Pluginkonzept siehe [4] bzw. [5]*

Als Umgebung für unsere integrierte Clientanwendung haben wir uns für die kommerzielle Erweiterung einer weit verbreiteten Anwendungssoftware entschieden – die Entwicklungsumgebung (IDE) *Eclipse*. Eclipse wurde von IBM und OTI entwickelt und im Jahr 2001 dem unabhängigen Eclipse-Projekt als Open Source kostenlos zur Verfügung gestellt, welches seit damals unabhängig dessen Verbreitung und Weiterentwicklung verfolgt.

Auch wenn alle Funktionen einer typischen Java-Entwicklungsumgebung bereits in Eclipse vorhanden sind, kann es durch Hinzufügen zusätzlicher Komponenten beliebig erweitert werden. Die integrierten Werkzeuge und das ausgefeilte Plugin-Konzept mit zahlreichen Schnittstellen haben dazu geführt, dass zahlreiche Hersteller und Privatnutzer eigene kommerzielle oder freie Erweiterungen von Eclipse anbieten.

Plugin lauffähig auf kostenlosem Eclipse und IBM WSAD

Zur Realisierung komplexer Enterpriseanwendungen auf Basis von J2EE und Enterprise Java Beans bietet IBM eine kommerzielle Erweiterung von Eclipse unter dem Namen *WebSphere Application Developer* (WSAD) an. Anwender können dort mit Hilfe meist graphischer Werkzeuge komplexe Webanwendungen und Geschäftsprozesse realisieren und auf dem Webserver veröffentlichen.

Die Einfachheit mit der sich auch in WSAD Plugins erstellen lassen und nahtlos integrieren lassen, machen es zur idealen Umgebung für den zweiten Client.

3.3 Stacktraces als Problembeschreibung

Im folgenden Abschnitt widmen wir uns der Forderung des neuen Hilfesystems nach einem aussagekräftigen und einheitlichen Beschreibungsformats zur Problemerfassung. Allerdings schlägt das Grundkonzept von IBM weder ein konkretes Format vor, noch legt es fest durch welche Kontextinformationen eine höhere Aussagekraft erreicht werden soll. Diesen Fragestellungen widmen sich die folgenden Abschnitte. Dort wird gezeigt, warum Java-Stacktraces in dieser Diplomarbeit als Problembeschreibungsformat für die Beispielimplementierung gewählt wurden und wie ein automatisches Matching für die Lösungssuche realisiert wurde.

Das vorgegebene Konzept sieht vor, daß Problembeschreibungen nicht nur die Frage nach Art und Ort des Fehlers beantworten, sondern durch kontextuelle Zusatzinformation Angaben über mögliche Fehler des Systems oder des Benutzers machen. Eine einheitliche und maschinenlesbare Form soll im Gegensatz zur bisherigen Prosaform eine automatische Verarbeitung der Problembeschreibungen ermöglichen.

Außerdem gibt es den Wunsch, das neue System so allgemeingültig wie möglich zu gestalten. Nur so können unterschiedlichste Problemsituationen bei beliebigen Softwareanwendungen von unserem System behandelt werden. Aus diesem Wunsch ergibt sich die Forderung nach einem standardisierten Kontextformat für Softwarefehler. Eine Einigung der Hersteller auf ein gemeinsames Format ist allerdings unwahrscheinlich, da die meisten eigene, zueinander inkompatible Fehlercodes verwenden. Aus diesem Grund ist es besser, nur solche Daten zur verwenden, die zur Laufzeit bereits heute bei vielen Programmen verfügbar sind.

📖 Für ausführliche Information zu Core Dumps siehe [7]

Auf einigen Hardwareplattformen(Unix, AS/400) erzeugen Programme, die vor ihrer Auslieferung in Maschinencode der Zielplattform übersetzt wurden, in Fehlersituationen so genannte Core Dumps. In Form von Textdateien zeigen Core Dumps eine Art Schnappschuss des Hauptspeichers an. Dieser enthält die geschachtelten Prozeduraufrufe des Programms während des Auftretens des Fehlers. Das folgende Beispiel zeigt einen Core Dump eines Programms, das in der Programmiersprache C geschrieben wurde. Obwohl das Programm über 10 Jahre alt ist, sind Core Dumps nach wie vor ein übliches Werkzeug zur Fehlersuche.

```
#0  0x400c4111 in __kill ()
#1  0x400c3d66 in raise (sig=6) at ../sysdeps/posix/raise.c:27
#2  0x400c5447 in abort () at ../sysdeps/generic/abort.c:88
#3  0x401000fc in free_check (mem=0x80526f0, caller=0x400517c1) at
    malloc.c:437
#4  0x400fe47f in __libc_free (mem=0x80526f0) at malloc.c:2900
#5  0x400517c1 in scm_remove_from_port_table (port=1075437712) at
    ports.c:411
#6  0x40051ec8 in scm_close_port (port=1075437712) at ports.c:576
#7  0x40047373 in scm_primitive_load (filename=1075437696) at load.c:121
```

Abbildung 3 Core Dump eines C-Programms

Auch bei Programmen, die in plattformunabhängigem Bytecode vorliegen und erst zur Laufzeit von einer Art virtueller Maschine interpretiert werden, steht wertvolle Laufzeitinformation in ähnlicher Form zur Verfügung. Die wichtigsten Vertreter solcher Programmiersprachen sind *Java* der Firma Sun und der *.NET* Dialekt C# (gesprochen C-Sharp)der Firma Microsoft. Diese Sprachen verwenden zur Behandlung von Laufzeitfehlern das Konzept von *Exceptions* (deutsch: Ausnahmen).

📖 *Mehr Informationen zum Exceptionkonzept finden sich unter [8].*

Dieses Sprachkonstrukt erlaubt es, Problemsituationen als programmiersprachliche Klasse zu kapseln und bei ihrem Auftreten gezielt programmatisch zu behandeln. Während Fehler sonst die gesamte Anwendung zum Absturz brachten, werden sie heute von der Laufzeitumgebung abgefangen und an benutzerdefinierte Behandlungsroutinen weitergeleitet. Weniger der Problembehandlung selbst als einer Randfunktionalität gilt unser Interesse. In derartigen Sprachen gibt die Laufzeitumgebung - üblicherweise ein Interpreter oder eine Virtuelle Maschine) - zahlreiche Informationen über die Problemsituationen in Form so genannter *Stacktraces* aus.

Java-Stacktraces als maschinenlesbare Problembeschreibung

Stacktraces und Core Dumps enthalten ungefähr dieselbe Informationsmenge. Die Kompilierung von Programmen beliebiger Sprachen in Maschinensprache hat zur Folge, dass Core Dumps zwar in einer sprachunabhängigen, aber eben auch in einer schwerer lesbaren maschinennahen Form vorliegen als Stacktraces. Um unsere Überlegungen in den folgenden Kapiteln anschaulicher darstellen zu können, haben wir uns für Stacktraces als Mittel der Problembeschreibung entschieden.

3.3.1 Stacktraces allgemein

An einem vereinfachten Beispiel aus der Programmiersprache Java sollen die Bestandteile eines Stacktraces genauer betrachtet werden.

```
java.cos.naming.NameNotFoundException: JNDI name 'broker' could not be
resolved
  at com.ibm.SolutionBrokerService.search(SolutionBrokerService.java:451)
  at com.ibm.SolutionBrokerService.lookup(SolutionBrokerService.java:233)
  at com.ibm.Test.main(Test.java:12)
```

Abb. 1: einfacher Java Stacktrace

Der oberste Teil des Stacktraces spezifiziert die Exception durch Angabe des Klassennamens (hier `NameNotFoundException`). Die meisten Exceptions haben einen sprechenden Klassennamen, an dem man die Art des Fehlers leicht erkennen kann. Um die Klasse eindeutig zu identifizieren, ist die zusätzliche Angabe des Packagenamen (`java.cos.naming`) nötig.

Exceptions decken meist klar abgegrenzte Fehlersituationen ab und Programmiersprachen liefern für die häufigsten Fehlerarten eine Vielzahl vordefinierter Exceptionklassen. Hersteller erweitern Ihre Software oft um weitere anwendungsspezifische Exceptions.

Exceptions aus dem Grundrepertoire der Programmiersprache Java sind z.B.:

- `java.io.IOException`: Fehler beim Zugriff auf Ein- & Ausgabegeräte
- `java.io.FileNotFoundException`: eine Datei wurde nicht gefunden
- `java.sql.SQLException`: Fehler bei SQL-Anfrage an eine Datenbank

Nach dem Doppelpunkt folgt oft noch eine Fehlerbeschreibung in Prosa (z.B. *JNDI name 'broker' could not be resolved*), die die Fehlerursache und eventuelle Lösungswege in Prosaform beschreibt.

📖 *Für Informationen zur Fehlersuche mit Java-Stacktraces siehe [9].*

Die folgenden Zeilen, die in Java mit dem Wort **`at`** beginnen, werden *Stackframes* genannt. Jede Zeile repräsentiert den Aufruf einer bestimmten Stelle im Programmcode. Ihre Abfolge entspricht dabei der Aufrufreihenfolge auf dem Laufzeitstapel der Virtuellen Maschine. Diese verwaltet die Rücksprungadressen bei geschachtelten oder rekursiven Methodenaufrufen in Form eines Stapels (engl. stack). Ausgelöst durch eine Aktion des Benutzers wird die erste Methode aufgerufen (unterster Stackframe).

Diese ruft ihrerseits weitere Methoden auf, bis es schließlich zu einem Fehler kommt (oberster Stackframe). Dabei zeigt die Position der Stackframes verschiedene Sichtweisen auf das Problem. So charakterisieren die Stackframes am unteren Ende Methodenaufrufe in logischer Nähe zur fehlerprovozierenden Benutzerinteraktion und damit die Sicht des Endanwenders. Man erfährt, welche Aktion des Anwenders letztendlich den Fehler verursacht hat. Die Stackframes im oberen Teil geben an, wo im Code die Exception ausgelöst wurde und stellen damit die Sicht eines Entwicklers dar.

Die Stackframes im mittleren Bereich beschreiben den Weg des Programms und zeigen, welche Zwischenschritte des Programms zu dem Fehler führten.

Stacktrace als Schnappschuss des Aufrufstacks zur Problemzeit

Der Stacktrace ist eine Art Schnappschuss der Virtuellen Maschine zum Zeitpunkt der Fehlersituation. Die einzelnen Stackframes stellen weniger ein zeitliches Nacheinander bei Abarbeiten von Methoden dar als die Schachtelung von Methodenaufrufen zu einem bestimmten Zeitpunkt. Damit unterscheiden sich Stacktraces von den so genannten *Traces* oder *Tracelogs*, bei denen während des normalen Betriebs eines Programms – z.B. zu Debuggingzwecken - die Aufrufabfolge der Methoden mitprotokolliert wird.

3.3.2 Informationsgehalt und Verbreitung

Stacktraces werden im Fehlerfall direkt auf dem Bildschirm oder in Protokolldateien (auch Logfiles) ausgegeben. Sie bieten wesentlich mehr Informationen als bloße Fehlermeldungen. Durch Angabe der voll qualifizierten Exceptionklasse ist die Fehlerart eindeutig identifiziert. Die Stackframes liefern wertvolle Zusatzinformation. Sie geben nicht nur Auskunft darüber, was und wo etwas passiert ist, sondern auch wie es dazu gekommen ist. Trotzdem stellen Stacktraces wegen ihrer systemnahen Semantik für die meisten Endanwender kein Hilfsmittel dar. Zumeist werden sie nur von Entwicklern und Softwareadministratoren zu Debuggingzwecken verwendet, um im Fehlerfall Problemstellen besser lokalisieren zu können.

Da Stacktraces Problemsituationen aber aussagekräftig beschreiben und wegen Ihrer relativ festen Syntax gut maschinell verarbeitbar sind, eignen sie sich hervorragend für die Anwendung als Problembeschreibung in unserer Implementierung des **SolutionBrokers**. Wir konzentrieren uns im Folgenden auf Stacktraces, die von Java-Programmen produziert werden. Dies stellt keine große Einschränkung dar, da Stacktraces auch von anderen Programmiersprachen verwendet werden. So zum Beispiel haben Stacktraces in *.NET* Dialekten ein Format, das Java sehr ähnelt, wie das untere Beispiel eines *C#* Stacktraces zeigt.

```
[Exception: CSE.MPS.DATAMANAGER: SQL Error EXCEPTIONMESSAGE: Invalid column name 'id'.]
  Microsoft.ErrObject.Raise(Int32 Number, Object, ..., Object HelpContext) +431
  Controls.DataManager.GenerateDS(String query,String connstring) in C:\DataManager.vb:74
  QuicktranetPortalControls.DataManager.GenerateDS(String query) in C:\ DataManager.vb:50
  quicktranet.Login.Page_Load(Object sender, EventArgs e) in C:\ login.aspx.vb:178
  System.Web.UI.Control.OnLoad(EventArgs e) +67
  System.Web.UI.Control.LoadRecursive() +35
  System.Web.UI.Page.ProcessRequestMain() +731
```

Abbildung 4 typischer .NET Stacktrace

Durch minimale Anpassungen der Implementierung des **SolutionBrokers** könnten auch andere Formate von Stacktraces erkannt und behandelt werden. Mit geringem Zusatzaufwand könnten auch die genannten Core Dumps verwendet werden. Dies wäre in Hinblick auf die gewünschte Allgemeingültigkeit des Konzepts von Vorteil.

Mit unserer Festlegung auf Java-Stacktraces als Problembeschreibungsformat beschränken wir uns jedoch auf die Behandlung von Problemsituationen mit Software, die solche Stacktraces erzeugen. Auch wenn es Problemsituationen gibt, in denen dies nicht der Fall ist (z.B. Benutzer hat Passwort vergessen), sind Stacktraces ein guter Indikator dafür, dass tatsächlich eine Problemsituation vorliegt.

3.3.3 Heuristik zur Stacktrace-Erkennung

Für die automatische Suche und das Beisteuern Lösungen benötigt der **SolutionBroker** einen Stacktrace zu Beschreibung eines Problems. Diese Information holt er sich entweder automatisch von der Software, indem er den Inhalt der Fehlerkonsole ausliest oder bekommt sie vom Benutzer, der den Stacktrace manuell per Copy & Paste eingibt. Beide Formen der Weitergabe sind problematisch, da sie nicht garantieren, dass nur brauchbare Informationen beim Parsen zur Verfügung stehen. So enthalten typische Logfiles manchmal auch mehrere Stacktraces und Protokollinformation, die für die Beschreibung des Fehlers nicht relevant sind. Auch bei der Eingabe durch den Benutzer können überflüssige Zeilen mitkopiert oder relevante Teile vergessen werden. So ist es beispielsweise üblich, dass Stacktraces per Email zwischen Benutzern ausgetauscht und erst dann ins System eingegeben werden. Dabei gehen unter Umständen auch charakteristische Zeilenumbrüche verloren. Eine tolerante Heuristik soll auch bei variierter Syntax den relevanten Stacktrace erkennen.

Leichte Variationen in Stacktrace-Syntax möglich

Anhand zweier Extrembeispiele soll im Folgenden eine sinnvolle Heuristik beschrieben werden. Das erste Beispiel zeigt einen einzelnen und leicht erkennbaren Java-Stacktrace, während der Auszug aus einer WebSphere Logdatei im zweiten Beispiel mit seinen Sonderfällen die Erkennung erschwert und uns zur Berücksichtigung von Sonderfällen motiviert.

Die Heuristik wird durch die nummerierten Pfeile am linken Rand der Stacktraces und durch eine Beschreibung der einzelnen Schritte und Sonderfälle im Folgenden näher beschrieben.

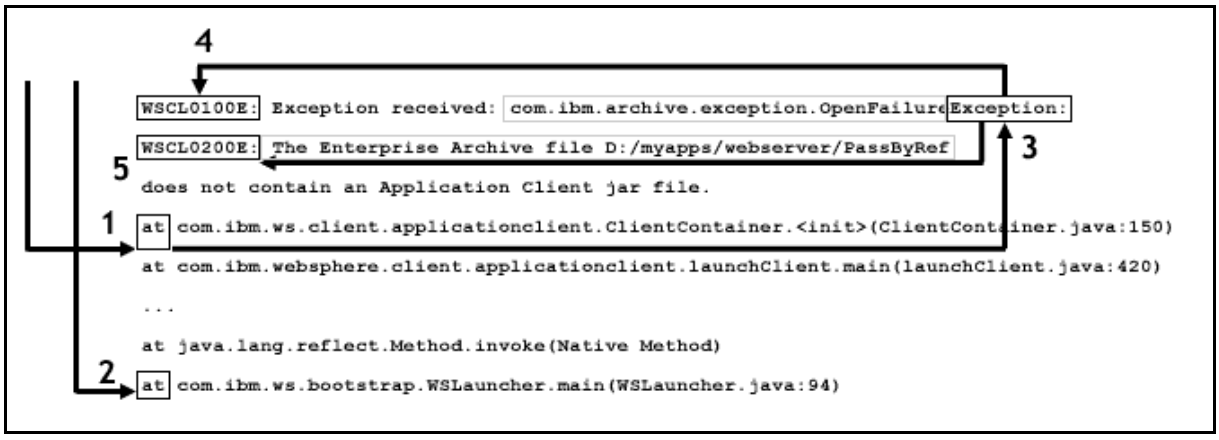

Abbildung 5 Heuristik zur Stacktrace-Erkennung (Version 1)

Erkennen der Stackframes:

1) Suche erstes Auftreten von „ `at` „ (= erster potentieller Stackframe)

2) Suchen letztes Auftreten von „ `at` „ (= letzter potentieller Stackframe)

Die Werte aus 1) und 2) können identisch sein, wenn nur ein Stackframe vorhanden ist.

Erkennen der Fehlerart:

3) Suche letztes Auftretens von „`Exception:`" vor dem ersten „ at „ (= extrahiere davor Klassen- / Packagenamens, danach Fehlermeldung)

4) Suche des letzten Fehlercodes vor „`Exception:`" (=Beschreibung der allgemeinen Fehlerart). Fehlercodes werden erkannt wenn sie der folgenden Syntax gehorchen: 4 alphanumerischen Zeichen + 4 Ziffern + abschließendes „W" oder „E".

5) Suche des ersten Fehlercodes nach „`Exception:`" (=genauere Fehlerbeschreibung)

Schlägt die Suche von 3) fehl, wird nur nach dem ersten Auftreten eines Fehlercodes gesucht. Dies ist vor allem dann nötig, wenn die Fehlerinformation nur aus einem Fehlercode besteht.

Der folgende Ausschnitt aus einer WebSphere Logdatei zeigt, dass die Heuristik um die Beachtung diverser Sonderfälle erweitert werden muss.

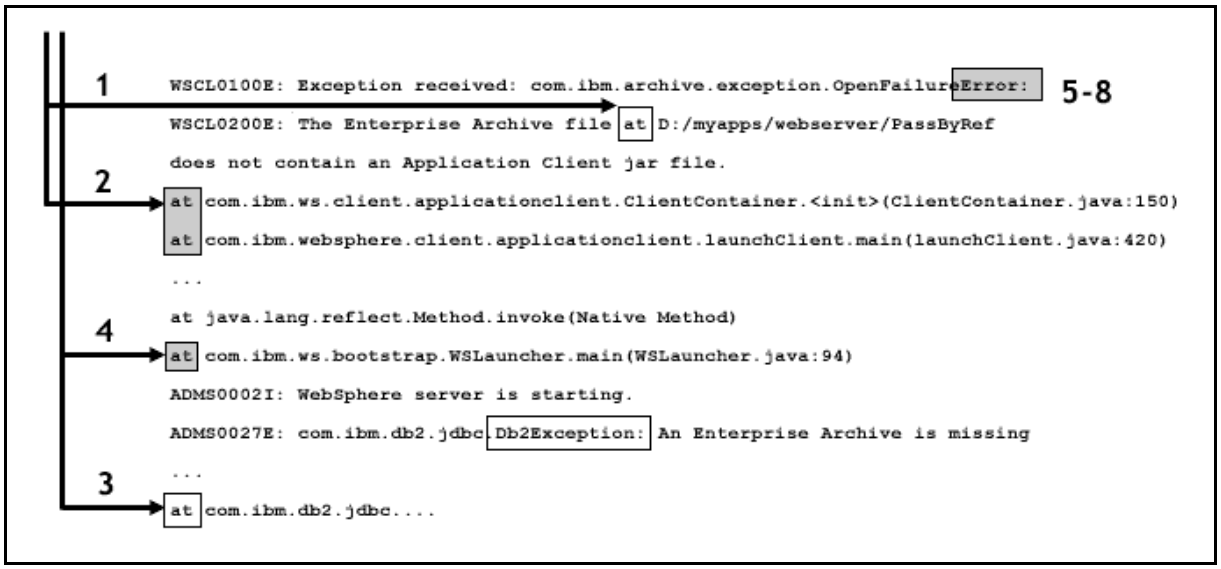

Abbildung 6 Heuristik zur Stacktrace-Erkennung (Version 2)

Erkennen der Stackframes:

1) Suche erstes Auftreten von „`at`„ (= erster potentieller Stackframe)

2) Suche nach einem zweiten Auftreten von „`at`„ Um Missdeutung des Wortes „`at`" außerhalb von Stackframes (z.B. in der Fehlermeldung) zu vermeiden, nehmen wir in Kauf, dass mit dieser Zusatzregel Stacktraces mit nur einem Stackframe ignoriert werden.

3) Suche letztes Auftreten von „`at`„ (= letzter potentieller Stackframe)

4) Suche letztes Auftreten von „`at`„ vor der nächsten Exception (wie in B)

Um nicht mehrere Stacktraces als einen einzigen zu identifizieren, lassen wir uns durch den kleineren Wert aus 3) und 4) das Ende des Stacktraces anzeigen.

Erkennen der Fehlerart:

5) Suche erstes Auftreten von „Exception:" oder „Error:"
6) Falls 5) erfolglos, suche erstes Auftreten von „:", allerdings nur wenn dieses einen führenden Package-Punkt „." hat
7) Falls 6) erfolglos, suche erstes Auftreten von „Exception" oder „Error"

Die zwei Sonderfälle sind nötig, da manche Exceptionklassen wie im Beispiel untypisch nicht mit dem Wort „Exception" bzw. „Error" enden, oder der Doppelpunkt danach fehlt (häufig bei JSP-Umgebungen).

8) Suche nach Fehlercodes wie in der ursprünglichen Heuristik

3.4 Automatische Lösungssuche

Wie das Kapitel über existierende Hilfesysteme gezeigt hat, war die Lösungssuche bisher mit einer zeitaufwendiger Suche und manuellem Aussortieren irrelevanter Information verbunden. Dank einer automatischen Suche soll dieser Aufwand auf ein Minimum reduziert werden. Anhand eines Matchingalgorithmus sollen in der Datenbank Lösungen ähnlicher Probleme gefunden werden und dadurch existierendes Wissen wieder verwendet werden.

Finden relevanter Lösungen durch Vergleich der Problembeschreibungen

Dies setzt voraus, dass er solche ähnlichen Einträge im System auch gefunden werden. Die Datenbank enthält Lösungen für Benutzerprobleme die durch einen Java-Stacktraces repräsentiert werden. Die Problemsituation die ein Anwender mit Hilfe des **SolutionBrokers** lösen will, manifestiert sich ebenfalls in Form eines Stacktraces. Dieser wird im Folgenden als *Benutzerstacktrace* bezeichnet. Zur Erkennung gleicher oder ähnlicher Probleme ist der Vergleich von Stacktraces nötig. Jeder Stacktrace der Lösungen in der Datenbank wird auf Ähnlichkeiten mit dem Benutzerstacktrace untersucht, um damit relevante Lösungen zu identifizieren.

Ziel dieses Kapitels ist die Definition eines Matchingalgorithmus. Dieser soll Problemsituationen in Form von Stacktraces vergleichen (matchen) und ein Maß für die Ähnlichkeit errechnen. Dieser Wert erlaubt dann eine Schätzung der Relevanz der dazugehörigen Lösung.

3.4.1 Anforderungen an einen Matchingalgorithmus

Das pragmatischste, aber gleichzeitig auch am schwierigsten umzusetzenden Ziel eines Matchingalgorithmus ist, nur solche Einträge auszugeben, die für die Lösung des Benutzerproblems hilfreich sind. Auch wenn diese Anforderung intuitiv verständlich ist, lässt sie sich nicht leicht auf einen Algorithmus übertragen. Es ist daher zunächst nötig, den Zusammenhang von Lösungen und Probleme besser zu verstehen.

Zwischen den konzeptuellen Entitäten Problem und Lösung besteht eine n-m Beziehung. Das heißt, zu einem gegebenen Problem kann es *keine*, *eine* oder *mehrere* Lösungen geben. Man spricht dann von einer Lösung, wenn das Problem mit ihrer Hilfe erfolgreich beseitigt werden kann. Im Gegenzug gibt es auch Lösungsvorschläge die *keine*, *eine* oder *mehrere* Problemsituationen erfolgreich beheben.

Für uns ist vor allem die Eigenschaft, dass Lösungen unter Umständen mehrere Probleme lösen, von Interesse. Genau in diesem Fall hätte man den gewünschten Effekt, einmal gefunden Lösungen wieder verwenden zu können. Dies gilt vor allem dann, wenn die Probleme gleich oder zumindest ähnlich sind.

Probleme annähern durch Stacktraces

Für unser System ist es nicht möglich, die Ähnlichkeit zweier Problemsituationen direkt festzustellen. Wie beschrieben, wird als Annäherung die maschinenverständliche Repräsentation einer Problemsituation in Form von Stacktraces verwendet. Damit lässt sich die Komplexität des Algorithmus reduzieren. Sind zwei Stacktraces identisch, gehen wir davon aus, dass es sich um das gleiche Probleme handelt und die zugehörige Lösung auch für das neue Benutzerproblem geeignet ist. [1]

Die Hauptaufgabe des Matchingalgorithmus ist, zu bestimmen, wie ähnlich zwei Stacktraces sind.

Die Gleichheit zweier Stacktraces ist einfach durch einen einfachen Test der beiden Zeichenketten auf Gleichheit überprüfbar. Wir definieren daher den Fall der Gleichheit als eine Intervallgrenze für unsere Ähnlichkeitsfunktion. Da sich die Null als intuitiver Wert anbietet, soll statt der Ähnlichkeit die Differenz berechnet werden.

Definition:

Die Funktion $\texttt{difference}(S_{P1}, S_{P2}) \rightarrow [0..\bullet]$ gibt an wie groß die semantische Differenz von Stacktrace S_{P1} und S_{P2} ist. Ist der Wert 0, sind S_{P1} und S_{P2} semantisch identisch.

Weniger intuitiv zu beantworten ist die Frage, wie groß die Differenz bei nicht identischen Stacktraces ist. In den folgenden Abschnitten wird daher zunächst die Bedeutung der einzelnen semantischen Einheiten eines Stacktraces untersucht. Je nach identifizierter Wichtigkeit sollen Unterschieden in den einzelnen Elementen verschieden hohe Kostenwerte zugewiesen werden, um diese später zu einer Gesamtdifferenz zu addieren.

3.4.2 Semantik unterschiedlicher Exceptions

Ein Vergleich der Exception muss vier Fälle abdecken.

Package- und Klassenname gleich:

Benutzerstacktrace:

```
com.ibm.db2.JDBCDriverException …
```

Datenbankstacktrace:

```
com.ibm.db2.JDBCDriverException …
```

Da es hier keinerlei Anhaltpunkte für eine unterschiedliche Problemsemantik gibt, erhält dieser Fall die Kosten 0.

Packagenamen verschieden:

Benutzerstacktrace:

```
com.ibm.db2.JDBCDriverException …
```

Datenbankstacktrace:

```
com.oracle.JDBCDriverException …
```

Dieser Vergleich sollte einen Kostenwert > 0 ergeben, da unterschiedliche Packagenamen z.B. oft auf verschiedene Entwicklerkreise hindeuten. Eine gleiche Intention bei der Auswahl des Exceptionnamens ist also nicht unbedingt gewährleistet.

[1] Wir vernachlässigen zu allgemeine Lösungen, wie z.B. die Empfehlung einen schnelleren Prozessor einzubauen, wenn schlechte Performance zu beliebigen Fehlern führt.

Kostenvariable EXCEPTION _HALFMISMATCH

Wegen der großen Transparenz von Javakonzepten (d.h. übliche Exceptionnamen werden häufiger gewählt) und der Verwendung sprechender – damit auch oft recht langer– Namen, erscheint uns eine niedrigere Differenz als bei kompletter Verschiedenheit angemessen.

Der Kostenwert erhält den Namen EXCEPTION_HALFMISMATCH.

Packagename in Datenbankstacktrace fehlt

Benutzerstacktrace:

```
com.ibm.db2.JDBCDriverException …
```

Datenbankstacktrace:

```
JDBCDriverException …
```

Kostenvariable EXCEPTIONPACKAGE _MISSING

Bei der Angabe der Exception in einem Stacktrace fehlt unter Umständen der Packagenamen. Man kann dann weder mit Bestimmtheit die Verschiedenheit noch die Gleichheit der zugrunde liegenden Problemsituation annehmen. Aus diesem Grund sollte die Differenz in einem solchen Fall zwischen den beiden vorangehenden Fällen angesiedelt sein. Der Bezeichner für den Kostenwert heißt EXCEPTIONPACKAGE_MISSING.

Klassennamen verschieden

Benutzerstacktrace:

```
com.ibm.db2.JDBCDriverException …
```

Datenbankstacktrace:

```
java.io.FileNotFoundException …
```

Kostenvariable EXCEPTION _FULLMISMATCH

Dieser Fall sollte Kosten erhalten, deren Wert größer ist als in den Fällen zuvor. Die Kostenvariable erhält den Namen EXCEPTION_FULLMISMATCH.

Zusammenfassend ergibt sich die folgende Reihenfolge bei der Gewichtung von Änderungen bei Exceptionklasse und –package:

Kostengewichtung bei Exception-Unterschieden:

0

^

EXCEPTIONPACKAGE_MISSING

^

EXCEPTION_HALFMISMATCH

^

EXCEPTION_FULLMISMATCH

Zusätzlich spielen bei der Relevanzbestimmung inhaltliche Unterschiede eine Bedeutung. Der folgende Abschnitt befasst sich beispielhaft mit der zusätzlichen Betrachtung von verschieden aussagekräftigen Exceptionarten. Er bietet durch eine Verfeinerung der Gewichtung eine Möglichkeit, den Algorithmus feiner auf die semantische Bedeutung von Stacktraces einzustellen.

Es folgt eine Optimierung des Matchingalgorithmus, die auf spezielle Besonderheiten verschiedener Exceptions eingeht und damit seine Genauigkeit verbessern kann.

Optimierung : Abwertung unspezifischer Exceptions

Fehlerarten, die bereits von Programmiersprachen behandelt werden, wie `NullPointerException` oder `ArrayIndexOutOfBoundsException`, treten sehr häufig auf. Da dies wegen der Allgemeingültigkeit der behandelten Fehlerart die unterschiedlichsten Ursachen haben kann, spezifizieren sie allein die Problemsituation nur ungenügend.

So kann beispielsweise die Eingabe einer negativen Zahl in ein Eingabeformular beim Online-Banking genauso eine `NullPointerException` zu Folge haben, wie eine abgebrochene Netzwerkverbindung zu entfernten Objekten eines Client-Server-Systems. In solchen Situationen wird erst durch die Stackframes die Situation näher spezifiziert.

Im Gegensatz dazu gibt es auch Exceptions, die bereits sehr genau auf die die Semantik eines Problems eingehen, wie:

- `com.ibm.mqseries.client.QueueNotInitializedException`
- `org.apache.soap.WSDLFileNotProvidedException`

Bei einer Exception, die häufig vorkommt, ist wenig Filterwirkung gegeben, wenn beim Vergleich diese Exception übereinstimmt. Als Folge erhalt man zu viele Matches. Es ist daher sinnvoll, je nach Informationsgehalt einer Exception, die Bedeutung der nachfolgenden Stackframes stärker oder schwächer zu gewichten. Durch Aufnahme des Gewichtungsfaktors STACKFRAMES_WEIGHT in den Matchingalgorithmus können bei weniger aussagekräftigen Exceptions Unterschiede in den nachfolgenden Stackframes stärker gewertet werden.

Gewichtungsfaktor STACKFRAMES_WEIGHT

Die Gewichtung von Exceptions könnte z.B. folgendermaßen umgesetzt werden:

- *manuell*: Definition von zu filternden Packages und Exceptions
- *automatisch*: schwächer Gewichtung bei langen (=viele „.“) Packagenamen
- *dynamisch*: aufsteigend je nach Häufigkeit der Exception in den bisherigen Datenbankeinträgen

3.4.3 Semantik unterschiedlicher Fehlermeldungen

Die Fehlermeldung ist wegen ihrer Prosaform nur begrenzt zur maschinellen Verarbeitung geeignet. Die Fehlermeldung enthält Elemente die oft sehr spezifisch auf eine Problemsituation eingehen und somit stark vom jeweiligen Umfeld abhängen. Ohne die Anwendung komplizierter Algorithmen zum Vergleich von Zeichenketten oder Verfahren der künstlichen Intelligenz bleibt uns nur der Test auf echte Gleichheit.

Eine mögliche Hypothese ist, dass zwei identische Fehlermeldungen dasselbe Problem beschreiben. Diese Annahme erweist sich aber als falsch, sobald Fehlermeldungen inhaltlich sehr allgemeingültig sind (z.B.: *„Eine nicht weiter spezifizierter Fehler ist aufgetreten“*).

Die folgenden zwei Meldungen zeigen, dass verschiedene Meldungen im Gegenzug nicht unbedingt auf verschiedene Problemarten hindeuten müssen. Die Meldungen…

- configuration file ‘C:\config.xml’ has wrong syntax
- configuration file ‘D:\config.xml’ has wrong syntax

unterscheiden sich nur im Laufwerksbuchstaben.

Dies ist für die Aussage, dass der Dateiinhalt nicht der erwarteten Syntax entspricht, kaum von Bedeutung. Die Grenzen eines einfachen Tests auf Gleichheit werden auch bei der Übersetzung der Fehlermeldung in eine andere Sprache deutlich.

- die Syntax der Konfigurationsdatei 'C:\config.xml' ist falsch

Kostenvariable EXCEPTIONMESSAGE _MISMATCH

Um der geringen Aussagekraft der Fehlermeldung Rechnung zu tragen, soll der Matchingalgorithmus ihr nur ein geringes Gewicht zuordnen. Den Kostenwert nennen wir EXCEPTIONMESSAGE_MISMATCH.

3.4.4 Semantik unterschiedlicher IBM Statuscodes

Als Umgebung für die Implementierung des **SolutionBrokers** wurden der IBM Application Server *WebSphere* und die dazugehörige Entwicklungsumgebung *WebSphere Application Developer* gewählt. Beide Produkte sind in der Lage, Fehler in Form von Stacktraces anzuzeigen. Dies Stacktraces enthalten zusätzlich noch spezielle IBM-Statuscodes, wie sie viele Softwarehersteller zur prägnanten Identifizierung interner Systemzustände verwenden.

Das folgende Beispiel zeigt einen typischen Stacktrace aus dem Umfeld von *WebSphere*.

```
WSCL0100E: Exception received: com.ibm.archive.exception.OpenFailureException:
WSCL0200E: The Enterprise Archive file D:/PassByRef does not contain an Application
Client jar file.
   at com.ibm.ws.client.appclient.ClientContainer.<init>(ClientContainer.java:150)
   at com.ibm.websphere.client.appclient.launchClient.main(launchClient.java:420)
   at java.lang.reflect.Method.invoke(Native Method)
   at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:94)
```

Abbildung 7 typischer Stacktrace produziert durch WebSphere Software

IBM Statuscodes (z.B. `WSCL0100E`) bestehen aus vier Zeichen, vier Ziffern, gefolgt von einem Zeichen. Die ersten 4 Buchstaben enthalten Information über die laufende Software. So stehen z.B. „WS" für WebSphere und „MQ" für das Middleware-Produkt MQ Series). Die vier Ziffern identifizieren die Situation genauer. Das letzte Zeichen gibt an, ob es sich bei der Meldung um unkritische Statusinformation (I für Information), eine Systemwarnung (W für Warning) oder eine Fehlermeldungen (E für Error) handelt.
Üblicherweise finden sich pro Stacktraces immer zwei solcher Statuscodes. Der erste signalisiert das Auftreten eines Fehlers, den der zweite durch die Angabe des Java Stacktraces genauer spezifiziert.

Kostenvariable ERRORCODE_MISMATCH

Um der unterschiedlichen Genauigkeit bestimmter Statuscodes Rechnung zu tragen, sollten zu allgemeine oder häufige Fehlercodes beim Vergleich weniger gewichtet oder nicht herangezogen werden. Dieses Ignorieren sehr häufiger Worte entspricht den Stopwords bei Internetsuchmaschinen wie Google (z.B. „and" und „the").

Filterliste ERRORCODE_IGNORE

Die Menge der ignorierenden Statuscodes wird mit dem Bezeichner ERRORCODE_IGNORE benannt. Bei Unterschieden in den verbleibenden Statuscodes sollte ein Gewicht vergeben werden, die mindestens so groß ist wie bei EXCEPTION_MISMATCH. Da Statuscodes vom Hersteller produktspezifisch ausgewählt wurden, ist es sinnvoll anzunehmen, dass sie genauer auf konkrete Problemsituationen zugeschnitten sind als Exceptions.

Es sollte also gelten:

EXCEPTION_MISMATCH <= ERRORCODE _MISMATCH

3.4.5 Unterschiede innerhalb des Stackframes

Bisher wurden Fälle betrachtet, in denen der komplette Stackframe verschieden war. Auch eine Betrachtung kleinerer Änderung innerhalb der Stackframes ist nötig.

Ein Stackframe der Form

```
at org.example.DBAccess.openWindow(DBAccess.java:5)
```

gibt eindeutig an, welche Zeile Code welcher Klasse aufgerufen wurde. Da dies bereits durch Angabe von Klassennamen und Zeilennummer eindeutig festgelegt würde, ist die zusätzliche Angabe des Methodennamen vermutlich nur als Hilfestellung beim Lesen des Stacktraces gedacht. Diese Vermutung wird durch das Weglassen einer Methodensignatur bestärkt. Da in Java beim sog. Überladen mehrere Methoden denselben Namen haben können, ist diese Angabe sowieso nicht eindeutig. Wir verwenden den Methodennamen nur als zusätzlichen Hinweis beim Vergleich. Allerdings ignorieren wir den Dateinamen (hier *DBAccess.java*), da er bis auf den seltenen Fall innerer Klassen identisch mit dem Klassennamen ist.

identische Stackframes

Da es hier keinerlei Anhaltpunkte für eine eventuell unterschiedliche Problemsemantik gibt, erhält dieser Fall den Differenzwert 0.

unterschiedliche Klassennamen

Die unterschiedliche Wichtigkeit der einzelnen Elemente lässt nicht jede Testkombination sinnvoll erscheinen und legt die Verwendung einer Reihenfolge mit absteigenden Differenzkosten nahe. So ist ein Vergleich von Methodennamen nur sinnvoll, wenn diese in derselben Klasse liegen, wie das Beispiel der `toString()`–Methode zeigt.

- `at java.sql.Connection.toString(Connection.java:145)`
- `at com.example.Customer.toString(Customer.java:72)`

Kostenvariable CLASS_MISMATCH

Wir brechen aus diesem Grund bei unterschiedlichen Klassennamen den Test ab und weisen ihm eine Differenz mit dem Namen CLASS_MISMATCH zu.

Wegen der geringen Bedeutung eines einzelnen Stackframes verzichten wir darauf den Spezialfall fehlender Packagenamen gesondert zu behandeln und gehen in solchen Fällen von einem vollständigen Unterschied aus.

unterschiedliche Methodennamen

Kostenvariablen METHOD_LINE_MISMATCH und METHOD_MISMATCH

Die einzelnen Methoden einer Klasse haben meist klar abgegrenzte Aufgaben. Zwei unterschiedliche Methodennamen weisen also oft darauf hin, dass es sich um zwei verschiedene Aufrufe handelt. Dieser Fall erhält den Differenzwert-Bezeichner METHOD_LINE_MISMATCH zu. Stimmen jedoch wenigstens die Zeilennummern überein, ist es möglich, dass bei einer Code-Umstrukturierung (auch Refactoring) lediglich der Methodenname geändert wurde, die restlichen Teile des Codes jedoch unverändert sind. Wir vergeben hier eine niedrigere Differenz mit dem Namen METHOD_MISMATCH.

unterschiedliche Zeilennummer

Ein Unterschied in der Zeilennummer kann verschiede Ursachen haben, z.B. Code-Änderungen der jeweiligen Klasse oder ein Versionssprung in verwendeten Fremdkomponenten. Aber auch nicht funktionale Änderungen, wie das Einfügen von Kommentarzeilen, das Neuformatieren des Quellcodes oder Refactoring schlagen sich in geänderten Zeilennummern nieder.

Um größere Änderungen der Zeilennummer stärker zu gewichten, könnte die Differenz abhängig von der absoluten Differenz berechnet werden. Eine Änderung von Zeile 1 auf Zeile 100 wiegt schwerer als die von 1 auf 10.

Kostenvariable LINE_MISMATCH

Das Einfügen oder Löschen einer Zeile am Anfang einer langen Datei ändert die Zeilennummern aller folgenden Methoden. Unterschiede in hohen Zeilennummern haben Ihre Ursache also meist nicht in der umgebenden Methode, sondern im Bereich davor. Beim Vergleich sollte die absolute Größe der Zeilennummern so betrachtet werden, dass eine Änderung von Zeile 10 auf Zeile 11 schwerer wiegt, als die von 1010 auf 1011.

Ein Spezialfall ist das Fehlen von Zeilennummern, wie in diesen Beispielen:

- `at org.example.DBAccess.openWindow(Compiled Code)`
- `at org.example.DBAccess.openWindow(Unknown Source)`
- `at org.example.DBAccess.openWindow(Native Code)`

📖 Weitere Informationen zu JIT-Compiler und dem Java Native Interface unter [10] bzw. [11].

Der Laufzeitumgebung steht keine Zeileninformation zur Verfügung, wenn Java Klassen z.B. zur Verbesserung der Performance von so genannten *Just-In-Time Compilern* in Maschinencode übersetzt werden. Statt einer Zeilennummer findet sich die Angabe `Compiled Code` oder `Unknown Source`. Bei Verwendung von Komponenten anderer Programmiersprachen wie C++ mit Hilfe des *Java Native Interface* erscheint statt einer Zeilennummer der Ersatztext `Native Code`.

Folgende Regeln für die Kostenvergabe erscheinen uns sinnvoll:

- Behandlung aller Ersatztexte wie Zeilennummer gleich 0
- der Vergleich eines Ersatztextes und einer numerischen Zeilennummer gibt dieselbe Differenz wie bei zwei unterschiedlichen Zeilennummern. Der Differenzwert erhält den Bezeichner LINENUMBER_MISMATCH

3.4.6 Parametrisierbare Gewichtung bei Stackframe-Unterschieden

Wir haben gesehen, dass es nicht ausreicht, als Hinweis für die semantische Ähnlichkeit zweier Stacktraces nur die Exception heranzuziehen. Auch die Stackframes liefern wertvolle Information. Je nach Kontext und internem Aufbau eines Programms sind Unterschiede auch an den verschiedenen Positionen unterschiedlich relevant für die Beurteilung einer Problemsituation. Anhand des folgenden Beispielprogramms und des dazugehörigen Stacktrace soll die Relevanz typischer Änderungen analysiert werden.

Das fiktive Programm `DBAccess` kann mit verschiedenen Benutzeroberflächen und über verschiedene Kommunikationswege SQL-Anweisung an eine Datenbank absetzen.

Die Klasse stellt dafür die folgenden Methoden zur Verfügung:

- `localAccess()` - über JDBC auf eine lokale Datenbank
- `remoteAccess()` - über RMI auf eine entfernte Datenbank
- `main()` - Eingabe der SQL-Query über die Kommandozeile
- `openWindow()` - Eingabe über Formular in GUI-Komponente

```
0:  package org.example;
1:  public class DBAccess {
2:      public void static main(String[] args) {...}
3:      public void static openWindow() {...}
4:      public void static localAccess(String sql) {...}
5:      public void static remoteAccess(String sql) {...}
6:       ...
7:  }
```

Abbildung 8 fiktives Beispielprogramm DBAccess

Im zunächst betrachteten Fall produziert das Programm eine `SQLException`, die Fehler beim Ausführen eines SQL Befehls auf einer Datenbank unabhängig vom speziellen Produkt signalisiert.

```
java.sql.SQLException: CREATE TABLE not executed because table with name "TEST"
already exists
   at com.ibm.db2.DB_V7Driver.executeQuery(DBDriver.java:78)
   ...
   at org.example.DBAccess.localAccess(DBAccess.java:15)
   at org.example.DBAccess.main(DBAccess.java:5)
```

Abbildung 9 Stacktrace produziert durch Beispiel-Programm

Um zu zeigen, wie sehr die Relevanz von Änderungen von äußeren Umständen abhängt, skizzieren wir für den unteren, mittleren und oberen Bereich eines Stacktraces zwei Fallbeispiele. Die Stackframes beider Fälle unterscheiden sich vom ursprünglichen Stacktrace aus Abbildung 9. Im ersten Fall handelt es sich um eine für die Problemsituation charakteristische Änderung, während im zweiten Fall der Unterschied in den Stackframes kaum fehlerrelevant ist.

Die Bedeutung der farbigen Hervorhebungen ist:

- gelb : Fehlermeldung für unterschiedliche Problemsituation
- rot : problemrelevanter Unterschied in den Stackframes
- grün : nicht problemrelevanter Unterschied in den Stackframes

Dabei ist anzumerken, dass die gelb hervorgehobenen Unterschiede in den Fehlermeldungen in unserer Argumentation nur als Indikator für die Verschiedenheit der Probleme dienen. Wie im Abschnitt 3.4.3 beschrieben, eignen sich Fehlermeldungen wegen ihrer Prosaform wenig für die maschinelle Analyse. Aus diesem Grund ist die Analyse der Bedeutung der Stackframes in den folgenden Abschnitt umso wichtiger.

Unterschiede im oberen Teil

Die Stackframes im oberen Teil des Stacktraces repräsentieren Programmaufrufe in unmittelbarer Nähe der Codestelle, an der eine Exception ausgelöst wurde. Das legt die Vermutung nahe, dass sie für auch für die Problemsituation charakteristisch sind.

Ohne das Beispielprogramm zu ändern, werde die Datenbank und der zugehörigen JDBC-Treiber gegen die Open-Source Datenbank MySQL ausgetauscht.

relevant Stackframes:

Wir nehmen an, unsere CREATE TABLE Anweisung enthielte eine so genannte Fremdschlüsseldeklaration. Diese werden von frühen Versionen von MySQL

nicht unterstützt. Das Ergebnis ist eine semantisch verschiedene Exception, die die Verwendung eines nicht unterstützten Features moniert. Hier wird die Problemursache erst durch die rot markierten Änderungen der Stackframes ersichtlich. In diesem Fall sind sie sehr relevant und sollten durch den Matchingalgorithmus stark gewichtet werden.

```
java.sql.SQLException: CREATE TABLE statement not executed: foreign key constraint not
supported by used driver
    at org.mysql.MySQLDriver.executeQuery(DBDriver.java:78)
    ...
    at org.example.DBAccess.localAccess(DBAccess.java:15)
    at org.example.DBAccess.main(DBAccess.java:5)
```

Abbildung 10 relevanter Unterschied in den oberen Stackframes

wenig relevante Stackframes:

Das zweite Beispiel verwendet statt MySQL die neuere Version 8 von DB2. Wegen der Abwärtskompatibilität des Treibers kommt es auch nach dieser Änderung des Backends zum gleichen Fehler. Lediglich im ersten Stackframe erscheint ein anderer Klassenname – der des neuen Datenbanktreibers. Die Änderungen in den oberen Stacktraces hat hier kein besonderes Gewicht.

```
java.sql.SQLException: CREATE TABLE statement not executed: table "TEST" already exists
    at org.mysql.DB2_V8Driver.executeQuery(DBDriver.java:78)
    ...
    at org.example.DBAccess.localAccess(DBAccess.java:15)
    at org.example.DBAccess.main(DBAccess.java:5)
```

Abbildung 11 irrelevanter Unterschied in den oberen Stackframes

Unterschiede im mittleren Teil

Um eine Änderung der Stackframes im Mittelteil zu erreichen, wird nicht wie bisher mit einer lokalen Datenbank kommuniziert, sondern über *Remote Method Invocation (RMI)*.

relevante Stackframes:

Wie das erste Beispiel zeigt, kann dieser Unterschied sehr charakteristisch für die Problemsituation sein. Die SQLException moniert hier einen Timeout der Datenbanktransaktion. Falls unsere RMI-Verbindung langsam ist und die CREATE TABLE Anweisung in SQL als Transaktion gekapselt wurde, ist dieser Fall durchaus wahrscheinlich. Auch hier sind also die Änderungen der Stackframes fehlerrelevant und sollten stark gewichtet werden.

```
java.sql.SQLException: CREATE TABLE statement not executed: transaction timeout
    at com.ibm.db2.DB_V7Driver.executeQuery(DBDriver.java:78)
    ...
    ... other RMI method calls
    ...
    at org.example.DBAccess.remoteAccess(DBAccess.java:15)
        at org.example.DBAccess.main(DBAccess.java:5)
```

Abbildung 12 relevanter Unterschied in den mittleren Stackframes

wenig relevante Stackframes:

Kaum relevant sind die Änderungen im Mittelteil allerdings im folgenden Beispiel. Hier wird derselbe Fehler produziert wie im Ausgangsbeispiel. Dabei ist es unbedeutend, dass der Datenbankzugriff eben nicht lokal, sondern per RMI stattfindet.

```
java.sql.SQLException: CREATE TABLE statement not executed: table "TEST" already exists
   at com.ibm.db2.DBDriver.executeQuery(DBDriver.java:78)
   ...
   ... other RMI method calls
   ...
   at org.example.DBAccess.remoteAccess(DBAccess.java:15)
   at org.example.DBAccess.main(DBAccess.java:5)
```

Abbildung 13 irrelevanter Unterschied in den mittleren Stackframes

Unterschiede im unteren Teil

Anhängig von äußeren Umständen ist auch die Relevanz von Änderungen im unteren Bereich des Stacktraces. Sie haben eine große semantische Nähe zur Aktion des Benutzers. Ob dies auch für eine bessere Charakterisierung seiner speziellen Problemsituation relevant ist, testen wir durch zwei Änderungen in der Aufrufsemantik unseres Programms. Statt direkt in der Kommandozeile mit Hilfe der `main()`–Methode eine SQL-Anweisung anzugeben, wird `openWindow()` verwendet. In unserem fiktiven Programm wird nun eine graphische Benutzeroberfläche in Form eines Eingabeformulars angezeigt.

relevante Stackframes:

Wir nehmen an, die Eingabemaske der Benutzeroberfläche erlaubt die Eingabe von Leerzeichen, während die Kommandozeile des Betriebssystems sie als Beginn eines separaten Parameters mißinterpretiert. Da auch die meisten Datenbanksysteme Leerzeichen in Tabellennamen nicht erlauben, moniert die `SQLException` hier die Verwendung ungültiger Zeichen in der CREATE TABLE Anweisung. Erst durch die Kenntnis des letzten Stackframes, der zeigt, dass `openWindow()` statt `main()` aufgerufen wurde, macht ein Verständnis des Fehlers möglich.

```
Java.sql.SQLException: CREATE TABLE not executed: invalid table name used (no blanks allowed)
   at com.ibm.db2.DBDriver.executeQuery(DBDriver.java:78)
   ...
   at org.example.DBAccess.localAccess(DBAccess.java:15)
   at org.example.DBAccess.openWindow(DBAccess.java:5)
```

Abbildung 14 relevanter Unterschied in den unteren Stackframes

wenig relevante Stackframes:

Die Forderung einer generell hohen Gewichtung der unteren Stackframes ist allerdings nicht sinnvoll, da in vielen Fällen deren Änderungen nur Ausdruck einer problemirrelevanten Kapselung von Komponenten ist. Werden wie im Beispiel Funktionen nur über erweiterte Darstellungskomponenten aufgerufen, sind die unteren Stackframes nur eine Art Zuckerguss.

```
java.sql.SQLException: CREATE TABLE not executed: because table  name "TEST" already exists
   at com.ibm.db2.DBDriver.executeQuery(DBDriver.java:78)
   ...
   at org.example.DBAccess.localAccess(DBAccess.java:15)
   at org.example.DBAccess.openWindow(DBAccess.java:5)
```

Abbildung 15 irrelevanter Unterschied in den unteren Stackframes

Empfehlungen

Wie die gegensätzlichen Beispiele gezeigt haben, sind je nach Anwendungskontext und Programminternas Unterschiede in den drei Bereichen des Stacktraces unterschiedlich relevant. Da eine Festlegung auf fixe Gewichtswerte nicht sinnvoll ist, soll eine situationsgerechte Anpassung des Matchingalgorithmus durch parametrisierbare Gewichte ermöglicht werden.

Als Richtlinie für die Einstellungen dienen die folgenden Empfehlungen.

Situation	Gewichtung oben TOP_WEIGHT	Gewichtung Mitte MIDDLE_WEIGHT	Gewichtung unten BOTTOM_WEIGHT
Grundeinstellung	**1**	**1**	**1**
Falls die Exceptions oder Fehlercodes eindeutig auf Problemsituationen hindeuten, ist es denkbar auf eine Betrachtung der Stackframes ganz zu verzichten.	**0**	**0**	**0**
Geringe Variabilität in der Art, wie Benutzer auf die Funktionskomponenten zugreifen (z.B. Zugriff allein über JSP Webinterface)	**-**[2]	**-**	**1**
Große Variabilität in der Art ,wie Benutzer auf die Funktionskomponenten zugreifen	**-**	**-**	**0**
Geringe Variabilität in der Art, wie die Software Funktionen aufruft (fixe Fremdbibliotheken, gleichen JDK, nur lokale Aufrufe)	**-**	**1**	**-**
Große Variabilität in der Art wie die Software Funktionen aufruft (Logging, Security, Middleware)	**-**	**0**	**-**
Geringe Variabilität der zugrunde liegenden Systemkomponenten (zentrales Backend)	**1**	**-**	**-**
Große Variabilität der zugrunde liegenden Systemkomponenten (verteiltes Backend)	**0**	**-**	**-**

Abbildung 16 empfohlenen Einstellungen der Gewichtungsparameter

Die Notwendigkeit manueller Anpassung von Parameter ist aufwendig und verringert die Akzeptanz und den Nutzen eines Hilfesystems. Die Erfahrung bei internen Testläufen hat allerdings gezeigt, dass auch die Verwendung der Grundeinstellungen schon sinnvolle Ergebnisse liefert.

Die Auswirkung der Parametrisierung wollen wir zum besseren Verständnis noch einmal an zwei konkreten Fällen verdeutlichen. Im ersten Szenario kommt der **SolutionBroker** bei Verwendung eines heterogenen Backends mit *verschiedenen* Datenbanksystemen zu Einsatz.

[2] (*Hinweis: Das Zeichen **„-„** wird verwendet, wenn ein Parameter keine Bedeutung für die jeweiligen Besonderheit der Situation hat und demnach beliebig gewählt werden kann.

Selbst bei demselben Problem ist dort mit Unterschieden im oberen Teil der Stacktraces zu rechnen (siehe dazu die Beispiele in Abschnitt 3.4.5). Durch die Wahl eines niedrigen Werts für TOP_WEIGHT verhindern wir, dass die Unterschiede zu große Differenzwerte ergeben und erlauben so, dass auch Benutzer mit verschiedenen Backends einander helfen können.

Im zweiten Szenario betrachten wir den umgekehrten Fall: die Verwendung eines einheitlichen Datenbanksystems durch alle Benutzer. Hier sind Unterschiede im oberen Teil des Stacktraces ein gutes Signal für die tatsächliche Unterschiedlichkeit der zugrunde liegenden Probleme. Die Wahl eines hohen Wertes für TOP_WEIGHT ist hier sinnvoll, um die Unterschiede stärker in die Gesamtdifferenz eingehen zu lassen. Diese Parametrisierung verringert das Risiko für irrelevante Lösungen.

3.4.7 Bestandteile des Matchingalgorithmus

Wie die unten stehende Formel zeigen soll, berechnet der Matchingalgorithmus die Gesamtdifferenz zweier Stacktraces x und y, indem er die Differenz der einzelnen Bestandteile bestimmt und diese – eventuell gewichtet - addiert.

$$D(x,y) = (D_{error}(x,y) * w_{ibm}) + D_{exc}(x,y) + D_{mess}(x,y) + (D_{fram}(x,y) * w_{fram})$$

Abbildung 17 Allgemeiner Matchingalgorithmus

Die in der Formel verwendeten Symbole korrelieren jeweils mit einer Java-Methode aus der Implementierung, die die Differenzberechnungen realisiert.

- **$D(x,y)$** → `difference()`: Gesamtdifferenz der Stacktraces x und x
- **D_{error}** → `errorDifference()`: Differenz bei IBM Fehlercodes
- **w_{ibm}** → `ERRORCODE_WEIGHT`: Gewichtungsfaktor für IBM Fehlercodes
- **D_{exc}** → `exceptionDifference()`: Differenz bei Exceptiondaten
- **D_{mess}** → `messageDifference()`: Differenz bei Fehlermeldung
- **D_{fram}** → `stackframesDifference()`: Differenz bei Stackframes
- **W_{fram}** → `STACKFRAMES_WEIGHT`: Gewichtungsfaktor für Stackframes bei unspezifischen Exceptionklassen (siehe dazu Abschnitt 3.4.2)

Im vorherigen Abschnitt wurde die Semantik der Bestandteile eines Stacktraces analysiert und ihre unterschiedliche Relevanz bei der Charakterisierung einer Problemsituation herausgearbeitet. Für die Umsetzung dieser Erkenntnisse, ersetzen wir nun die symbolischen Namen, die wir bisher für die Kosten verwendet haben, durch konkrete Werte, die die Relevanzreihenfolge sinnvoll wiedergeben. So wiegt z.B. ein einzelner Mismatch vom Typ EXCEPTION_FULLMISMATCH in der Praxis schwerer als mehrere Mismatches vom Typ LINE_MISMATCH.

Damit solche bedeutsamen Änderungen möglichst nicht durch die kumulierten Kosten kleiner Änderungen übertroffen werden, wird als Schrittweite der Faktor 10 verwendet. In der Praxis kann man diesen Faktor den tatsächlichen Gegebenheiten anpassen.

Beginnen wir mit Elementen geringer Relevanz in den Stackframes.

- MESSAGE_MISMATCH = 1
- LINE_MISMATCH = 1

Änderungen in der Fehlermeldung und bei Ersatztexten bei der Zeilennummer haben nur geringe Relevanz. Die Vergabe minimaler Kosten soll lediglich vermeiden, das Stacktraces eine Differenz von 0 erhalten, die nicht identisch sind. Die etwas höhere Aussagekraft einer zahlenmäßig unterschiedlichen Zeileninformation soll durch einen etwas höheren Wert ausgedrückt werden.

- METHOD_MISMATCH = 10
- METHOD_LINE_MISMATCH = 20
- CLASS_MISMATCH = 100

Eine Größenordnung höher angesiedelt sind die Kosten für Unterschiede beim Methoden- bzw. Klassennamen. Der Fall unterschiedlicher Methoden bei identischer Zeilennummer soll wegen seiner geringeren Relevanz mit den halben Kosten belegt werden.

```
1: int methodDifference(Context prob, Context sol) {
2:   int difference = 0;
3:   if(prob.getMethod() <> sol.getMethod()) {
4:      if(hasLineNumber && prob.getLine() == sol.getLine())
5:        difference = METHOD_MISMATCH;
6:      else
7:        difference = METHOD_LINE_MISMATCH;
8: }
```

Abbildung 18 Vergleich der Exception mit methodDifference()

Ein Unterschied in einem Stackframe „kostet" maximal 100 Differenzpunkte, da bei der Feststellung des schwerwiegendsten Unterschiedes, also einem verschiedenen Klassennamen, keine weiteren Tests im Stackframe durchgeführt werden. Die Kosten für Unterschiede bei den Exception- bzw. Fehlerinformation werden so hoch gewählt, um nicht von der Summe von Differenzwerte der Stackframes überstiegen zu werden. Unserer Erfahrung nach sind über 50 Stackframes in einem Stacktrace sehr selten. Die schwerwiegenden Unterschiede in der Exceptionklasse erhalten die Kosten 10.000. Außerdem definieren wir 10.000 als Maximalwert beim Vergleich von Stackframes (→ Begrenzung auf Maximalkosten bei EditDistance)

Der weniger relevante Fall bei EXCEPTION_HALFMISMATCH erhält nur die halben Kosten. Um die höhere Relevanz produktspezifischer Fehlercodes zu berücksichtigen, erhalten fehlercode-spezifische Unterschiede zusätzliche 1.000 Differenzpunkte. Die Wahl des Wertes ist willkürlich, aber verhindert, dass beim Vergleich Unterschiede bei den Fehlercodes schwerer gewichtet werden, als solche bei der Exception.

- ERRORCODE_MISMATCH = 6.000
- EXCEPTION_HALFMISMATCH = 5.000
- EXCEPTION_FULLMISMATCH = 10.000
- ERRORCODES_MISSING = 2.000
- EXCEPTIONPACKAGE_MISSING = 1.000

```
1:  int exceptionDifference(Stacktrace s1, Stacktrace s2) {
2:     int difference = 0;
3:     int STACKFRAMES_FACTOR = 1;
4:
5:     // Beispiel für Abwertung häufiger Exceptions
6:     if(s1.startsWith("java.")) STACKFRAMES_FACTOR = 2;
7:     if(s1.getPackage() != s2.getPackage()) {
8:        if(s1.getExceptionClass != sol.getExceptionClass())
9:            difference = EXCEPTION_MISMATCH;
10:        else difference = EXCEPTIONPACKAGE_MISMATCH;
11:    }
12:    // zusätzlicher Differenzwert bei fehlender Packageinformation
13:    else if(s2.getPackage() == "")
14:           difference = difference + EXCEPTIONPACKAGE_MISSING;
15:
16:    return difference;
17: }
```

Abbildung 19 Vergleich der Exception mit exceptionDifference()

```
1:  int errorcodeDifference(Stacktrace s1, Stacktrace s2) {
2:     int difference = 0;
3:
4:     if(s2.hasErrorCodes)  {
5:        if(errorCodesMatching == 0)
6:           difference = ERRORCODE_MISMATCH * 2;
7:        else if(errorCodesMatching == 1)
8:           difference = ERRORCODE_MISMATCH;
9:      }
10:     else difference = ERRORCODES_MISSING;
11:
12:     return difference;
13: }
```

Abbildung 20 Vergleich der Fehlercodes mit errorcodeDifference ()

```
1:  int messageDifference(Stacktrace s1, Stacktrace s2) {
2:     int difference = 0;
3:
4:     if(s1.getMessage() != s2.getMessage())
5:         difference = EXCEPTIONMESSAGE_MISMATCH;
6:
7:     return difference;
8:  }
```

Abbildung 21 Vergleich der Fehlermeldung mit messageDifference ()

Vergleich einzelner Stackframes

```
1:  int stackframesDifference(StackFrame s1, StackFrame s2) {
2:     int difference = 0;

       // Test des voll-qualifizierten Klassennamen
3:     if(s1.getClass() != s2.getClass()) difference = CLASS_MISMATCH;

       // Test der Methodennamen
4:     else if(s1.getMethod() != s2.getMethod()) {
5:        if(hasIdenticalLineNo)
                difference = METHOD_HALFMISMATCH;
6:        else
                difference = METHOD_FULLMISMATCH;
7:     }
       // Test der Zeilennummern und Ersatztexte
8:     else if (!hasIdenticalLineNo) difference = LINETOKEN_MISMATCH;

9:     return difference;
10: }
```

Abbildung 22 Vergleich einzelner Stackframes mit stackframesDifference ()

Vergleich aller Stackframes

Die größte Herausforderung bei der Realisierung des Matchingalgorithmus liegt im Vergleich der Stackframes. Ein reiner Zeichenketten-Vergleich benachbarter Stackframes ist ungeeignet, da bereits das Löschen oder Einfügen eines einzigen Stackframes alle folgenden Positionen so verschiebt, dass diese als verschieden eingestuft würden. Es ist daher ein Algorithmus nötig, der die Auswirkungen solcher Änderungsaktionen in Betracht zieht. Geeignet erscheint uns der so genannte Diff-Algorithmus und die so genannten Levenshtein Distanz mit dem Edit-Distance Algorithmus.

📖 Eine Implementierung des DIFF-Algorithmus in Java findet sich unter[12].

Der Diff-Algorithmus (Diff ~ difference, *deutsch*: Unterschied) wird oft zum zeilenweisen Vergleich von Textdateien verwendet. Durch mehrfaches Traversieren des Textes findet er ähnliche Blöcke auch dann, wenn diese durch Einfügungen bzw. Löschungen verschoben wurden. Der Edit-Distance Algorithmus bzw. die Levenshtein-Distanz weißt den Unterschieden in Folgen einen konkreten Kostenwert zu. Wegen der Ähnlichkeit von Stackframes zu Folgen und der großen Verfügbarkeit von Codebeispielen für diesen Algorithmus haben wir uns für den Edit-Distance Algorithmus entscheiden.

📖 Mehr zur Levenshtein-Distance und dem Edit-Distance Algorithmus unter [14].

Dieser führt zwei Folgen durch Anwenden elementarer Änderungsoperation wie Einfügen, Löschen und Ersetzen ineinander über. Die Operationen können je nach Semantik der Folgen mit verschiedenen Kosten belegt werden. Diese werden aufsummiert und stellen eine Art von Differenzmaß dar. Dabei wird automatisch der abnehmenden Wahrscheinlichkeit größerer Verschiebungen Rechnung getragen.

Im folgenden Pseudocode sehen wir, wie die Differenz zwischen den Folgen x und y berechnet wird. In der Tabelle EDIT, deren Dimension den Längen der beiden Folgen entspricht, werden sukzessive die minimalen Kosten zur Überführung aller Teilfolgen gebildet. Der letzte Tabelleneintrag entspricht den minimalen Kosten zur Überführung von x nach y.

```
1:  m = |x|
2:  n = |y|
3:  EDIT [-1,-1] = 0
4:  for j = 0 to n-1 do
5:     EDIT [-1,j] = EDIT[-1,j-1] + INSERT
6:     for I = 0 to m-1 do
7:       EDIT[i,-1] = EDIT[i-1,-1] + DELETE
8:       for j = 0 to n-1 do
9:           EDIT[i,j] = min{EDIT[i-1,j-1] + SUBSTITUTE,
10:                        EDIT[i-1,j] + DELETE,
11:                        EDIT[i,j-1] + INSERT}
12: return EDIT[m-1,n-1]
```

Abbildung 23 Edit-Distance mit konstanten Kosten

In [14] schlägt John Lambert eine Erweiterung des Edit-Distance Algorithmus vor, bei der die Kostenberechnung an die Semantik von Java Stackframes angepasst wird. Um Änderungen an verschiedenen Positionen unterschiedlich zu gewichten, ersetzt Lambert die konstanten Kosten INSERT, DELETE und SUBSTITUTE durch die Funktionen INSERT(i,x) bzw. DELETE(i,x) und SUBSTITUTE(i,x,j,y).

Kostenfunktion für INSERT/DELETE und SUBSTITUTE

Wir wollen nun die Erkenntnisse aus dem letzten Abschnitt in konkrete Kostenfunktionen umsetzen.

Um die gewünschte parametrisierbare Gewichtung der einzelnen Bereiche vornehmen zu können, führen wir die drei Strategieparameter TOP_WEIGHT, MIDDLE_WEIGHT, BOTTOM_WEIGHT ein. Es handelt sich dabei um Gleitkommazahlen zwischen 0.0 und 1.0. Wie durch die Regler eines Equalizers an Audioanlagen lassen sich durch sie die einzelnen Bereiche stärker oder schwächer gewichten.

```
1:  double getStackframeWeight (int i, StackFrame[] x) {
2:    int length = x.length;
4:    int center = Math.abs(length / 2);
5:    float step = 1 / length;
6:
7:    float top = (1 - i * step) * TOP_WEIGHT;
8:    float middle = (1 - (Math.abs(i - center) * step)) * MIDDLE_WEIGHT
9:    float bottom = (step + i * step) * BOTTOM_WEIGHT
10:
11:   return Math.max(top, Math.max(middle, bottom)) * STACKFRAMES_FACTOR;
12: }
```

Abbildung 24 Berechnung des Gewichtungsfaktors mit getStackframeWeight()

```
1:  public float substitute(int i, StackFrame[] x, int j, StackFrame[] y)
2:  {
3:    int costs = getWeightFactor(i,x) * x[i++].compareTo(y[j++]);
4:    return costs;
5:  }
```

Abbildung 25 Parametrisierbare SUBSTITUTE-Funktion

```
1:  public float insdel(int i, StackFrame[] x)
2:  {
3:    int costs = getWeightFactor(i,x) * INSDEL_COST;
4:    return costs;
5:  }
```

Abbildung 26 Parametrisierbare INSDEL-Funktion

Der Kostenwert der `INSDEL()`–Funktion sollten im Edit-Distance Algorithmus nur ausgewählt werden, wenn es der Unterschied mit hoher Wahrscheinlichkeit durch ein Einfügung oder Löschung und nicht durch eine Änderung des semantisch gleichen Stackframes zustande gekommen ist. Damit er Um bei der Minimumsbestimmung ausgewählt zu werden, wählen wir daher einen Kostenwert der höher ist als der Maximalwert bei Änderungen derselben Zeile.

- INSDEL_COST = 200

Zusammenfassung – die stacktraceDifference()-Funktion

Die Gesamtdifferenz zweier Stacktraces ergibt sich aus der Summe aller Teildifferenzen. Wenn nicht beide Stacktraces Stackframes haben, wird eine Strafgebühr addiert. Sonst wird das Resultat der Edit-Distance Funktion in die Summe mit aufgenommen. Je nach Einstellung der parametrisierbaren

Gewichte TOP_WEIGHT, MIDDLE_WEIGHT und BOTTOM_WEIGHT werden dort durch die Kostenfunktionen die verschiedenen Differenzwerte berechnet.

```
1:  public int difference(Stacktrace s1, Stacktrace s2) {
2:    int difference = 0;
3:    difference += errorcodeDifference(s1, s2);
4:    difference += exceptionDifference(s1, s2);
5:    difference += messageDifference(s1, s2);

6:    if(s1.getStackframes() == null XOR s2.getStackframes() == null)
        difference += STACKFRAMES_MISSING;
7:    else {
8:      difference += editDistance(s1.getStackframes(),s2.getStackframes())
8:    return difference;
9: }
```

Abbildung 27 Automatisierte Lösungssuche mit difference()

KAPITEL 4 Implementierung

Die folgenden Abschnitte widmen sich der praktischen Umsetzung der Komponenten und zeigen wichtige Implementierungsdetails.

4.1 Datenmodell

Die Datenbank besteht aus vier Tabellen:

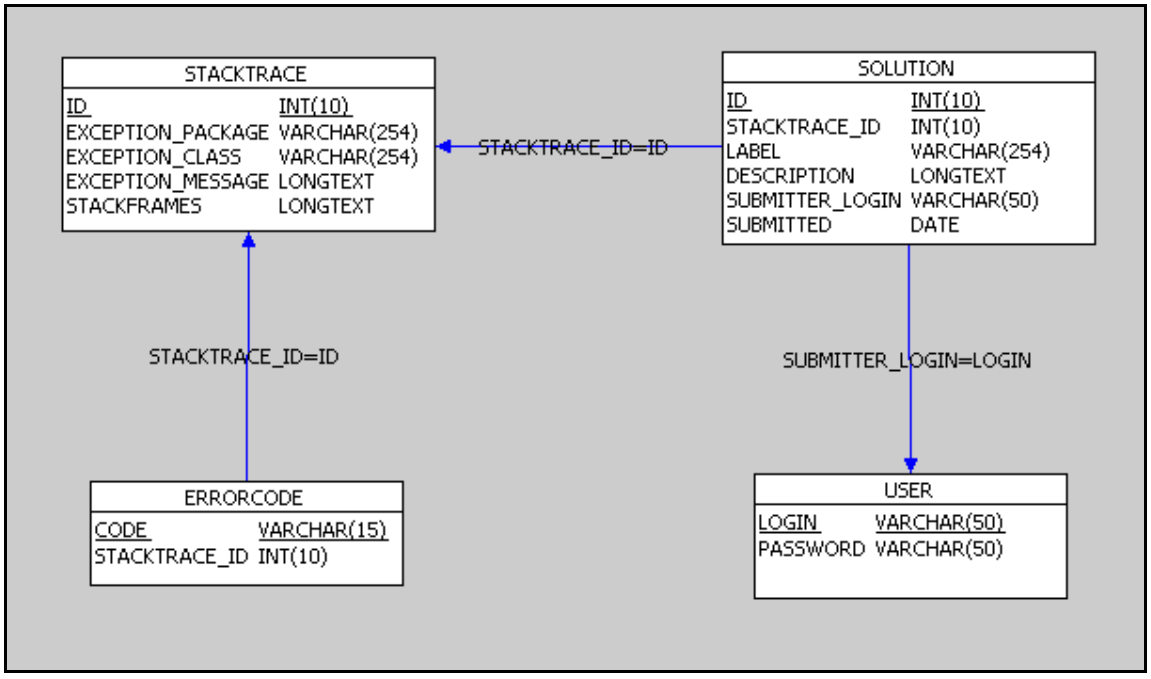

Abbildung 28 Datenbankschema

4.1.1 Tabelle STACKTRACE

Jeder Eintrag enthält die Informationen über eine bestimmte Problemsituation in Form eines Stacktrace. Um einen gezielten Zugriff auf einzelne Elemente eines Stacktraces zu ermöglichen (z.B. bei der Vorauswahl), werden sie in getrennte Attribute abgelegt. Eine weitere Zerlegung der Stackframes ist nicht nötig, da sie für ihren komplexen Vergleich sowieso komplett ausgelesen und in ein Java-Objekt umgewandelt werden müssen. Eine Längenbeschränkung bei Package und Klasse auf 255 Zeichen (Maximallänge des Typs VARCHAR) erscheint unserer Erfahrung nach ausreichend, während für Fehlermeldung und Stackframes als Datentyp CLOBs (Character Large Objects) ohne einer Maximalgröße nötig sind.

4.1.2 Tabelle SOLUTION

enthält die Lösungsbeschreibung eines Anwender zu einem bestimmten Problem. Die 1-n-Beziehung zur Tabelle USER wird durch das Fremdschlüsselattribut SUBMITTER_LOGIN ausgedrückt. Durch Hinzufügen weiterer Attribute (z.B. USEFULLNESS nach menschlicher Bewertung) können in dieser Tabelle weitere lösungsspezifische Daten gespeichert werden.

4.1.3 Tabelle ERRORCODE

IBM-Stacktraces können bis zu zwei Fehlercodes enthalten und gleiche Fehlercodes können in den verschiedensten Stacktraces auftreten. Daher modellieren wir diese n-m-Beziehung durch eine separate Tabelle, die über die Fremdschlüsselbeziehung CONTEXT_ID mit dem Primärschlüssel der Tabelle CONTEXT verbunden ist.

4.1.4 Tabelle USER

Benutzer können unter einem Login-Namen beliebige Lösungen in die Datenbank eintragen. Um diese Daten redundanzfrei zu halten, werden die Benutzerdaten in einer eigenen Tabelle abgelegt. Durch das Hinzufügen weiterer Attribute, wie PASSWORD, können hier bei Erweiterungen des Systems weitere benutzerspezifische Daten gespeichert werden (z.B. ein Punktekonto eines Anreizsystems, siehe dazu auch 6.1.3)

4.1.5 Integration der IBM Symptom DB

Bei verteilten Hilfesystemen wie Mailinglisten, Newsgroups und auch unserem System folgt das Benutzerverhalten üblicherweise dem Motto „erst nehmen, dann geben". Das heißt, dass Anwender wahrscheinlich zuerst Lösungen mit unserem System suchen werden, bevor sie auch eigene Einträge beisteuern. Es war daher sinnvoll, die Datenbank mit einem Grundbestand an Lösungen zu füllen.

📖 Mehr zum LogAnalyzer unter[15].

Für viele IBM Produkte gibt es bereits einen großen Bestand an Fehlerdiagnosen und Lösungsvorschlägen in Form so genannter Symptom-Datenbanken (=Symptom DB). Eine in die verschiedenen Produkte integrierte Software durchsucht Fehlerausgaben nach bekannten Symptomen und gibt Diagnosemöglichkeiten aus der Symptom DB des Produkts aus. Gefüllt werden die Symptomdatenbanken ausschließlich von IBM.

Die Daten liegen im XML-Format vor, wie der folgende Ausschnitt aus der Symptomdatenbank des *WebSphere Process Choreographers* zeigt. Als Fehlerbeschreibung liegen im Element `<matchsymptomv>` ein Java Stacktraces oder IBM Fehlercodes vor. Der dazugehörige Diagnosevorschlag findet sich in Prosaform im Element `<symptominfov>`.

```
...
<symrec recordid="Symptom_0" symptomtype="" symptomstatus="diagnosed">
    <matchsymptomv>
        BPEA0010E COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver]
        SQL30082N  Attempt to establish connection failed with security
        reason 24 (USERNAME AND/OR PASSWORD INVALID).  SQLSTATE=08001
        at COM.ibm.db2.jdbc.app.SQLExceptionGenerator.throw_SQLException
        at COM.ibm.db2.jdbc.app.SQLExceptionGenerator.check_return_code
        at COM.ibm.db2.jdbc.DB2XAConnection.<init>;
        at COM.ibm.db2.jdbc.DB2XADataSource.getXAConnection
    </matchsymptomv>
    <symptominfov>
        During the execution an unexpected exception occurred.
        Check the Username/Password provided for the Data Source.
    </symptominfov>
</symrec>
...
```

Abbildung 29 typischer Eintrag in der Symptom DB (XML-Fragment)

Die Daten der Symptom DB entsprechen weitgehend dem Datenmodell des **SolutionBrokers** und können mit relativ geringem Migrationsaufwand in die bestehende Datenstruktur übernommen werden.

Wir verwenden als Grundbestand eine Version der Symptom DB, die die Einträge vieler verschiedener IBM Produkte vereint. Die aktuelle Version ist kostenlos online abrufbar unter der URL ftp://ftp.software.ibm.com/software/websphere/info/tools/loganalyzer/symptoms/adv/. Sie enthält ungefähr 3000 Einträge, davon ca. 400 in Form von Stacktraces und weitere 2500 Einträge in Form von IBM Fehlercodes. Ein kleines Javaprogramm mit dem Namen `MigrateSymptomDB` übernimmt die Migration der Daten. Zu den Aufgaben des Programms gehört:

- sequentielles Einlesen der XML-Dateien mit event-basierten XML-Parser SAX (für große Datenmenge geeigneter als DOM-basierte Parser)
- Konstruktion von `Stacktrace`-Objekten aus den Inhalten der `<matchsymptomv>`–Elemente
- Konstruktion von `Solution`-Objekte aus den Inhalten der `<symptominfov>`–Elementen (als Urheber der Lösung wird der Name „SYMPTOM_DB" gespeichert
- Übertragen syntaktisch korrekter Daten in die relationale Datenbank

Je nach Version und Produktumfeld variiert die Struktur der XML-Daten einer bestimmten Symptom DB. Unser Migrationstool erkennt solche Unterschiede und bringt die gesammelten Daten vor der Weitergabe an die Kontruktoren für `Stacktrace`- bzw. `Solution`-Objekte in eine einheitliche Form.

Da unsere Implementierung zunächst intern bei IBM getestet werden soll, ist es keine große Einschränkung, dass wir mit den Daten der Symptom DB in der Anfangsphase nur Problemsituationen mit IBM-Produkten erkennen und dafür Lösungen anbieten.

4.1.6 Datenbank-Vorauswahl bei der Lösungssuche

Bei der automatischen Lösungssuche muss die Problemsituation des Benutzers mit den Einträgen in der Datenbank verglichen werden. Der Aufwand für diesen Vergleich wächst erwartungsgemäß mit der Anzahl der Einträge. Problematisch ist dabei weniger das Auslesen der Lösungen aus der Datenbank als der Vergleich selbst. Die Komplexität des Matchingalgorithmus macht es unmöglich, den Vergleich als Datenbankquery in SQL umzusetzen. Stattdessen muss er in Java stattfinden. Dabei ist es nötig, alle Einträge aus der Datenbank auszulesen und in Javaobjekten umzuwandeln. Selbst bei nur einem Benutzer müssten so kurzfristig eventuell Hunderttausende Objekte im Speicher des Servers gehalten werden. Neben hohen Speicheranforderungen ist auch eine hohe Belastung der CPU bei vielen Vergleichoperationen zu erwarten.

Eine Vorfilterung potentieller Vergleichspartner erscheint aus Performancegründen sinnvoll. Mit der Festlegung auf Stacktraces als interne Darstellung einer Problemsituation können nun konkrete Filtereigenschaften festgelegt werden. Die Vielfältigkeit möglicher Probleme und Lösungen macht es in der Praxis unwahrscheinlich, dass mehr als eine Handvoll von Einträgen relevant für den Benutzer sind. Indem man einen Teil des Vergleichs bereits von der Datenbank ausführen lässt, lassen sie nur offensichtlich geeignete Vergleichspartner extrahieren.

Um mit den begrenzten Mitteln von SQL einen solchen Filter zu implementieren, müssen besonders charakteristische Eigenschaften von Problemen identifiziert werden.

Diese müssen derart relevant für die Art der Problemsituation sein, das Einträge, die sich in dieser Eigenschaft unterscheiden, als Lösungen praktisch nie in Frage kommen. In der Datenbanktabelle sollte diese Eigenschaft als Attribut abgelegt werden, um leicht per SQL SELECT-Anweisung abgefragt werden zu können. Wegen Ihrer hohen Relevanz und einfachen Struktur verwenden wir in der Implementierung die Informationen zu Exceptionpackage und –klasse sowie die IBM Fehlercodes.

4.1.7 Verwendung von Tabellenindexen

📖 Detailinformationen zu Indexen und B-Bäumen unter [18].

Um die Vorauswahl und andere Abfragen der Datenbank zu beschleunigen, ist das Anlegen von Datenbankindexen bei häufig abgefragten Attributen sinnvoll. Mit Hilfe einer speziellen Datenstruktur (balancierte B-Bäume) erlauben es Indexe, bestimmte Einträge mit einem konkreten Attributwert besonders schnell zu finden. Ein Index muss einmal explizit angelegt werden und steht dann bei jeder Abfrage zu Verfügung.

In einer Testreihe haben wir den möglichen Geschwindigkeitsvorteil für den **SolutionBroker** konkret unter *DB2* überprüft. Dazu wurden zwei Datenbanken mit jeweils drei Tabelle mit 3.750, 37.500 und 370.000 Tupeln angelegt und mit zufälligen, nicht gleichverteilten Werten gefüllt. Die Tabellenstruktur(Attribute, Schlüsselbeziehungen,…) entsprachen dabei genau der Tabelle `Stacktrace`, da hier eine schnellere Vorfilterung von Einträgen bei der Lösungssuche durch Indexe besonders wünschenswert ist. Die drei Tabellen der ersten Datenbank wurden mit Indexen auf den abzufragenden Attributen versehen, während ihre Gegenstücke in der anderen Datenbank unverändert gelassen wurden.

Mit einem Profiling-Tool zur Geschwindigkeitsmessung und Optimierung von Datenbankenabfragen, das direkt in DB2 integriert ist, wurde anschließend an alle 6 Tabellen dieselbe Leseanfrage abgesetzt. Um wegen der unterschiedlichen Tupelzahl der Tabellen vergleichbare Ergebnismengen zu garantieren, wurde überall ein zusätzliches Tupel eingefügt. Dieses Tupel wurde auch als einziges Ergebnis von allen Anfragen zurückgegeben.

Die gemessenen Zeitunterschiede motivieren die Verwendung von Datenbankindexen. Während eine Verzehnfachung der Tupelzahl in der Datenbank ohne Indexe zu einer 10- bis 20-fach höheren Zugriffszeit führt, stellt man bei der Verwendung von Indexen nur eine 3- bis 6-fache Zugriffszeit fest.

	3.750 Tupel	**37.500 Tupel**	**375.000 Tupel**
ohne Index	596 Ti.	5908 Ti.	119.000 Ti.
mit Index	240 Ti.	1495 Ti.	3698 Ti.

Abbildung 30 Geschwindigkeitsvorteil durch Datenbankindexe

Als Maßeinheit für die Geschwindigkeit verwendet das Programm eine eigene Einheit mit dem Namen *Timerons*, um trotz plattform- und kontextabhängiger Einflüsse vergleichbare Messungen zu erlauben.

4.2 Serveranwendung

Die Hauptaufgaben der Serveranwendung sind:

- das Verwalten der Datenbank
- die syntaktische Erkennung von Stacktraces (engl. Parsing)
- die automatische Lösungssuche (engl. Matching)
- die Bereitstellung von Schnittstellen für die Clients

Die folgende Grafik soll einen Überblick über die Architektur der Serveranwendung geben und zeigen wie und wo diese Aufgaben erfüllt werden. Es beinhaltet die verwendeten Packages, deren Klassen und gibt anhand farbiger Pfeile einen ersten Eindruck des Daten- bzw. Interaktionsflusses. Die Aktionen der einzelnen Klassen und deren Zusammenspiel werden anschließend detailliert beschrieben.

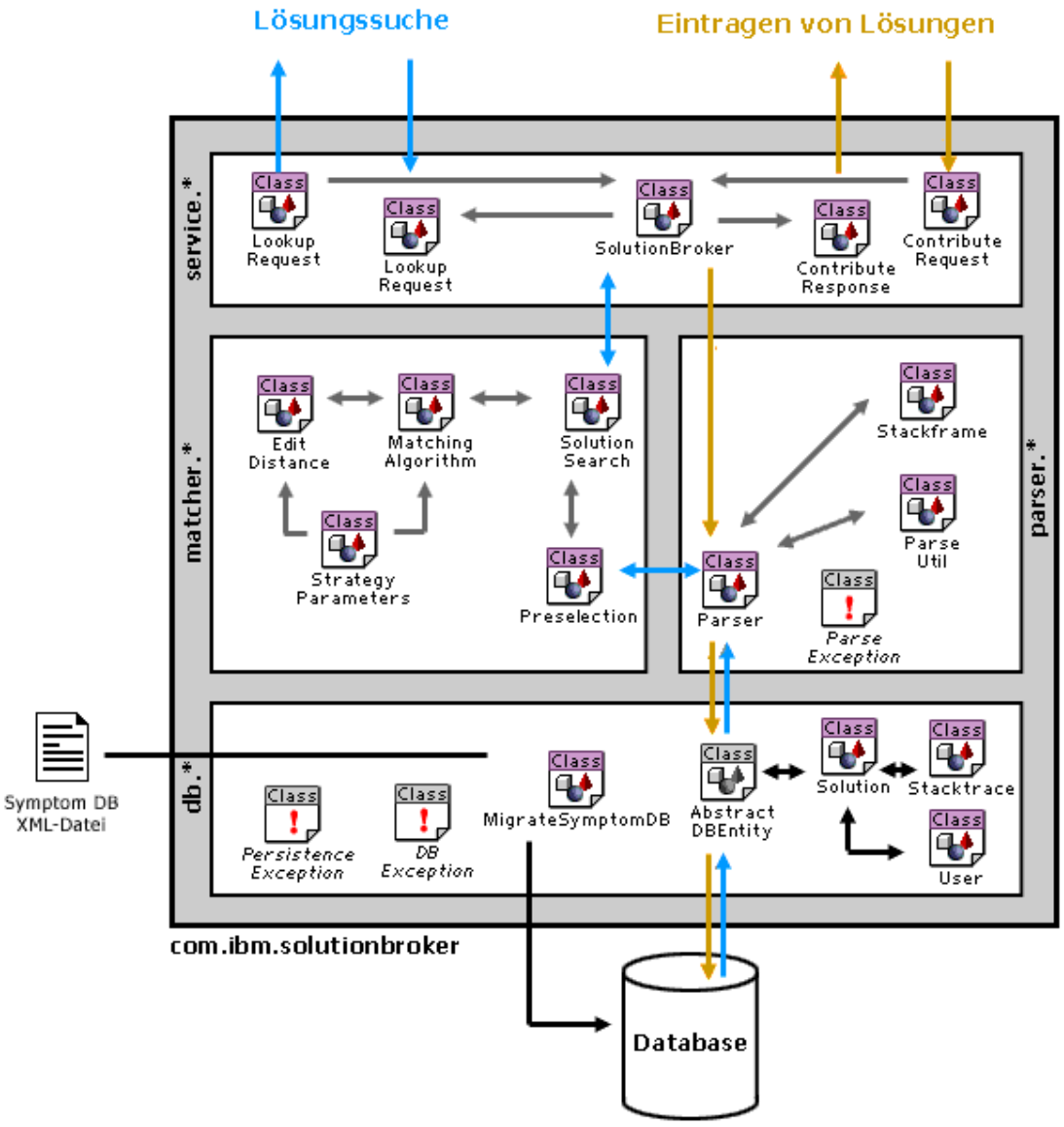

Abbildung 31 Klassenarchitektur der Serveranwendung

4.2.1 Client-Schnittstelle com.ibm.solutionbroker.service

Die Klassen dieses Package bieten eine Schnittstelle nach außen, über die Clientanwendungen mit Hilfe von XML Web Services auf die Dienste der Serveranwendung zugreifen können.

Service-Package als Schnittstelle zwischen Clients und Server

Als Kommunikationsmethode zwischen Clients und Server verwenden wir XML Web Services. Die zur Netzwerkkommunikation nötige technische Infrastruktur generieren die meisten Web Service Frameworks wie *Apache Axis* (siehe dazu http://ws.apache.org/axis) und die von uns verwendete Entwicklungsumgebung *WebSphere Application Developer* automatisch.

Als zentraler Zugriffspunkt dient die Klasse **`SolutionBroker`**, über die später jede Kommunikation laufen soll. Die Klasse hat die zwei Methoden `doLookup()` und `doContribute()`, die den Ablauf von Lösungssuche bzw. Lösungseintrag steuern und an die verantwortlichen Komponenten delegieren. Um lange Parameterlisten zu vermeiden und komplexe Rückgabedaten zu erlauben, werden diese durch eigene Klassen gekapselt:

- **`LookupRequest`**, Eingabeparameter bei der Suche; kapselt Problembeschreibung in Form eines ungeparsten Stacktraces und die gewünschte Anzahl maximal anzuzeigender Lösungen (`maxResults`)
- **`LookupResult`**: Rückgabeparameter bei der Suche: kapselt gefundene Lösungen und Relevanzinformation als Array
- **`ContributeRequest`**: Eingabeparameter beim Beisteuern von Lösungen; enthält Login-Daten des Autors und ein Beschreibung der Problemsituation in Form eines ungeparsten Stacktraces und die Lösung
- **`ContributeResponse`**: Ausgabeparameter bei Beisteuern von Lösungen; enthält Statusmeldungen zur Anzeige in den Clientanwendungen. Anwender erhalten so z.B. eine Bestätigung einer erfolgreichen Datenbankaktion oder Fehlermeldungen.

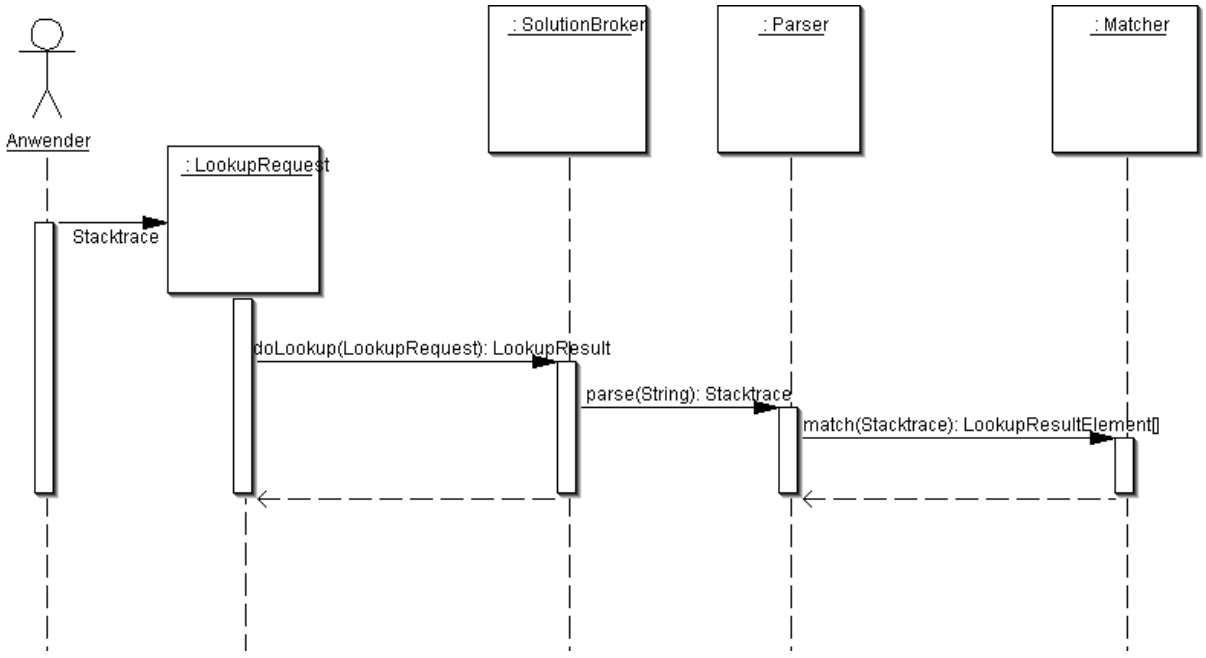

Abbildung 32 Sequenzdiagramm Lösungssuche

Am Beispiel einer Lösungssuche zeigt das obige Sequenzdiagramm den Ablauf und die Interaktion der Servicekomponenten mit anderen Systemteilen (hier dem Parser und dem Matcher).

Im Folgenden finden sich Ausschnitte der generierten Schnittstellenbeschreibung und eine kurze Beschreibung wichtiger Teile. Für detaillierte Informationen zur Syntax und Bedeutung aller WSDL-Elemente empfehlen wir die Onlineresource[16].

📖 Für eine ausführliche Beschreibung von XML Web Services siehe [16].

XML Web Services sind wie viele andere Technologien für verteilte Systeme programmiersprachenunabhängig. Aus diesem Grund wird die Java-Typinformation vor dem Austausch in die XML-Typisierungsprache XMLSchema überführt.

```
    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="unqualified"
            targetNamespace="http://solutionbroker.ibm.com/"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
            xmlns:xsd1="http://solutionbroker.ibm.com/">
            <import namespace="http://schemas.xmlsoap.org/wsdl/"
                    schemaLocation="http://schemas.xmlsoap.org/wsdl/"/>
            <import
                namespace="http://schemas.xmlsoap.org/soap/encoding/"
                schemaLocation="http://schemas.xmlsoap.org/soap/encoding/"/>
            <complexType name="LookupResultElement">
                <all>
                    <element name="submitted" nillable="true" type="dateTime"/>
                    <element name="errorIds" nillable="true" type="string"/>
                    <element name="label" nillable="true" type="string"/>
                    <element name="submitterLogin" nillable="true" type="string"/>
                    <element name="description" nillable="true" type="string"/>
                    <element name="solutionContext" nillable="true" type="string"/>
                    <element name="difference" type="int"/>
                </all>
            </complexType>
            <complexType name="ArrayOfLookupResultElement">
                <complexContent>
                    <restriction base="soapenc:Array">
                        <sequence/>
                        <attribute ref="soapenc:arrayType"
                                   wsdl:arrayType="xsd1:LookupResultElement[]"/>
                    </restriction>
                </complexContent>
            </complexType>
... // gekürzt
            <complexType name="LookupRequest">
                <all>
                    <element name="maxResults" type="int"/>
                    <element name="rawString" nillable="true" type="string"/>
                </all>
            </complexType>
... // gekürzt
            <complexType name="ContributeRequest">
                <all>
                    <element name="label" nillable="true" type="string"/>
                    <element name="submitterLogin" nillable="true" type="string"/>
                    <element name="rawString" nillable="true" type="string"/>
                    <element name="description" nillable="true" type="string"/>
                </all>
            </complexType>
        </schema>
    </types>
```

Abbildung 33 Datentypen in <wsdl:types>

Die Struktur der auszutauschenden Nachrichten (EXCEPTIONMESSAGEs) wird im Element `<EXCEPTIONMESSAGE>` definiert und anhand der Typinformation aus die Syntax der Ein- und Ausgabeparameter festgelegt.

```
<EXCEPTIONMESSAGE name="doLookupRequest">
        <part name="request" type="xsd1:LookupRequest"/>
</EXCEPTIONMESSAGE>
<EXCEPTIONMESSAGE name="doLookupResponse">
        <part name="result" type="xsd1:LookupResult"/>
</EXCEPTIONMESSAGE>
... // gekürzt
```

Abbildung 34 Nachrichtenformate in <wsdl:message>

Wichtig für die Synchronisierung der Kommunikationspartner ist die Festlegung von Anfragen und Antworten. Die Art (one-way, request-response) und Beschaffenheit (Ein- und Ausgabeparameter) dieser so genannten Operationen werden im Element `<operation>` spezifiziert. Hier finden sich die Gegenstücke zu den zwei Methoden `doLookup` und `doContribute` aus der `SolutionBroker`-Klasse.

```
<portType name="SolutionBroker">
        <operation name="doLookup" parameterOrder="request">
            <input EXCEPTIONMESSAGE="tns:doLookupRequest" name="doLookupRequest"/>
            <output EXCEPTIONMESSAGE="tns:doLookupResponse" name="doLookupResponse"/>
        </operation>
        <operation name="doContribute" parameterOrder="cr">
            <input EXCEPTIONMESSAGE="tns:doContributeRequest" name="doContributeRequest"/>
            <output EXCEPTIONMESSAGE="tns:doContributeResponse" name="doContributeResponse"/>
        </operation>
</portType>
```

Abbildung 35 Operationen in <wsdl:portType>

Basierend auf der Information dieser Schnittstellenbeschreibung, lassen sich automatisch lokale Stellvertreterobjekte für die Clients und den Server generieren. Mit ihrer Hilfe können die Clientanwendungen so programmiert werden, als ob sie auf ein lokales `SolutionBroker`-Objekt zugreifen. Dies ist möglich, da die lokalen Anfragen intern in XML-Nachrichten übersetzt (Marshalling) und über das Internet zum Server übertragen werden. Dort wird die Nachricht von einem weiteren Stellvertreter- bzw. Proxyobjekt wiederum in eine lokale Javaanfrage zurückübersetzt (Demarschalling) und an die Serverklassen weitergeleitet.

4.2.2 Datenbank-Modul - com.ibm.solutionbroker.database

Die Klassen dieses Packages übernehmen die Speicherung und das Auslesen von Lösungen, Problembeschreibungen, Fehlercodes und Benutzerinformation aus der Datenbank

📖 Für eine ausführliche Beschreibung von JDBC-Schnittstelle siehe [17].

Die Kommunikation zwischen Java und der DB2 Datenbank regelt die JDBC-Schnittstelle (=Java DataBase Connectivity). Um die Inhalte der einzelnen Tabellen auch in Java programmatisch zugreifbar zu machen, kapseln die drei Klassen **`Solution`**, **`Stacktrace`** und **`User`** die Daten der gleichnamigen Tabellen. Sie realisieren die Konvertierung eines Tabelleneintrags in ein Java-Objekt oder persistieren umgekehrt Java-Objekte aus dem Hauptspeicher in die Datenbank. Da Teile dieses Vorgangs (z.B. die nötigen SQL-Anweisungen) für alle drei Datenbankentitäten ähnlich sind, sind die Gemeinsamkeiten in die abstrakte Basisklasse **`AbstractDBEntity`** ausgelagert, um redundanten Code zu vermeiden.

Verbergen der Datenbank-kommunikation in eigenem Package

Um SQL-Anweisungen an die Datenbank abzusetzen zu können, muss eine Verbindung zu ihr aufgebaut werden. Während der Kommunikation werden die Verbindungsdaten als Java-Objekt im Speicher gehalten. Anschließend muss die Verbindung wieder geschlossen werden, um den benötigten Speicher wieder frei zu geben. Da bei einer Mehrbenutzeranwendung viele Nutzer gleichzeitig auf die Daten zugreifen wollen, ist ein ständiges Beenden und Neuinitialisieren ineffizient. Statt bei Anfragen immer neue Verbindungen zu öffnen, können unbenutzte bestehende Verbindungen wieder verwendet werden. Benötigt ein Client eine Verbindung nicht mehr, wird diese in einem Verbindungspool offen gehalten und für Anfragen anderer Clients wieder verwendet. Die Klasse **`ConnectionPool`** verwaltet einen Pool von Datenbankverbindungen - Instanzen der Klasse **`PooledConnection`** - und sorgt so für eine schnelle und ressourcensparende Datenbankkommunikation. Zwei Exceptionklassen behandeln typische Fehler beim Schreiben und Lesen der Datenbank. Beide erweitern die `SQLException` um zusätzliche anwendungsspezifische Informationen.

- **`DBException`**: kapselt Fehler beim Verbindungsauf- und -abbau, über das Connection-Pooling und andere eher technische Probleme. Die Exception speichert dabei zusätzlich das verwendete SQL-Kommando. Dies ist vor allem für Debuggingzwecken praktisch.
- **`PersistenceException`**: kapselt semantische Fehler beim Schreiben und Lesen der Datenbank. Solche Fehler treten auf wenn z.B. versucht wird Objekte zu lesen, die nicht in der Datenbank vorhanden sind, oder Objekte unter Primärschlüsseln neu anzulegen, die bereits existieren.

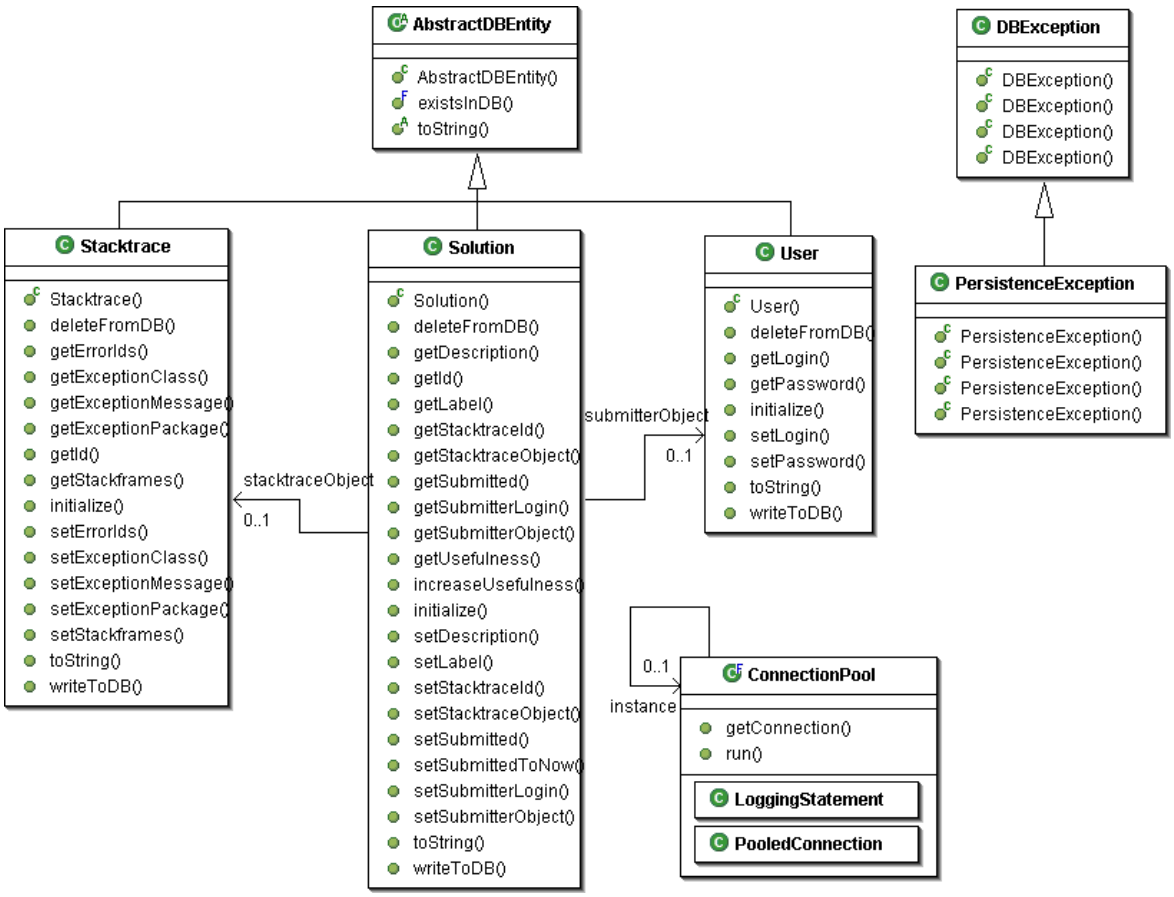

Abbildung 36 Klassendiagram Datenbankschicht

4.2.3 Parser-Modul - com.ibm.solutionbroker.parser

Die Klassen des Parser-Moduls extrahieren aus Zeichenketten Problembeschreibungen in Form von Java-Stacktraces.

Wie schon im Abschnitt über die Heuristik zur Erkennung von Stacktraces beschrieben, sollte der Parser mit Abweichungen von der üblichen Syntax umgehen können und gegenüber Eingabefehlern tolerant sein. Um dem Benutzer das manuelle Durchsuchen langer Fehlerausgaben zu ersparen, soll das System als Eingabe z.B. auch komplette Logfiles akzeptieren. Dabei soll nur relevante Information extrahiert werden.

Die Klasse **`Parser`** versucht nach der angegebenen Heuristik einen Stacktrace zu extrahieren. Dabei werden die einzelnen Elemente von separaten Methoden verarbeitet (z.B. parseErrorCodes() zur Extraktion der Fehlercodes). Diese Zerlegung des Parsingvorgangs in einzelne Schritte macht das Programm verständlicher und für spätere Änderungen leichter anpassbar.

Eine zusätzliche Hilfsklasse **`ParseUtil`** stellt Methoden zur Verfügung, die Sonderzeichen und Zeilenumbrüche in ein einheitliches Format konvertieren, um verschiedene Sonderfälle nicht erst während des Parsens beachten zu müssen.

Wie oben beschrieben, werden Stackframes zwar in der Datenbank nicht als einzelne Tupel abgelegt, müssen aber später von der Serveranwendung einzeln verarbeitet werden. Um ihre einzelnen Informationselemente programmatisch zugreifbar zu machen, übernimmt die Klasse **`Stackframe`** das Parsen einzelner Stackframes und deren Eigenschaften (z.B. Methodenname, Zeilennummer,…).

Gelingt die Erkennung eines gültigen Stacktraces nicht, wird eine Exception vom Typ **`ParseException`** ausgelöst, die Informationen enthält, wieso die Eingabe nicht verarbeitet werden konnte. Diese Information kann dann z.B. als Benutzerfeedback an den Client zurückgegeben werden.

4.2.4 Matching-Modul - com.ibm.solutionbroker.matcher

Die Klassen dieses Packages implementieren den vorgestellten Matchingalgorithmus und sind für die automatische Lösungssuche verantwortlich.

Die Hauptklasse **`SolutionSearch`** nimmt die vom Parsermodul in ein `Stacktrace`-Objekt überführte Problembeschreibung entgegen und delegiert und koordiniert die Schritte für die automatische Lösungssuche.

Mit Hilfe der Klasse **`Preselection`** wird eine Vorauswahl von Lösungseinträgen zur Performanceverbesserung vorgenommen. Für den Auswahlprozess in Java werden nur solche Lösungen aus der Datenbank geholt, für deren jeweilige Problembeschreibung eine der folgenden Bedingungen gilt:

- Übereinstimmung in Exceptionklasse/-package
- Übereinstimmung in mindestens einem Fehlercode

Eine Beschreibung der Datenbankvorauswahl findet sich in Kapitel 4.1.6.

Da für jede Bedingung separate SQL-Anweisungen nötig sind, sind als Ergebnis Duplikate möglich. Duplikate sind Lösungen, deren Problembeschreibung sowohl in Exception als auch den Fehlercodes übereinstimmen. Das Herausfiltern dieser Fälle übernimmt die Klasse ebenfalls.

Die extrahierten Problembeschreibungen werden nun der Reihe nach mit dem Benutzerproblem verglichen. Den Vergleich von Exception- und Fehlercodeinformation implementieren Methoden der Klasse **`MatchingAlgorithm`**. Ein wichtiges Kriterium bei der Umsetzung des Matchingalgorithmus war die Anpassbarkeit der Einzelkosten und sonstigen Strategieparameter. Wir haben sie daher als Konstanten in das separate Interface **`StrategyParameters`** ausgelagert, die von den einzelnen Matchingmethoden zur Laufzeit ausgelesen werden.

Der vorgestellte Edit-Distance Algorithmus wird durch die gleichnamige Klasse **`EditDistance`** realisiert, sie enthält auch die Kostenfunktionen INSDEL() und SUBSTITUTE().

4.2.5 Paralleler Problemvergleich

Mit wachsender Akzeptanz des **SolutionBroker** wächst nicht nur die Datenmenge, sondern auch die Anzahl der Benutzer, die mit Ihren Clients gleichzeitig auf die Funktionen des Servers zugreifen. Die verwendeten Basistechnologien bieten für die Behandlung der sich dabei ergebender Probleme bereits eine gute Grundlage.

- Konsistenzerhaltung der Daten durch Transaktionen (DB2)
- Wiederverwendung von Datenbankverbindungen durch Connection Pooling durch Java und DB2
- Parallele Bearbeitung von Serveranfragen durch Multithreading durch den WebSphere Application Server
- Wiederverwendung (Caching) von bereits erzeugten Webseiten durch den WebSphere Application Server

Aber auch mit diesen Features reicht die Verarbeitungsleistung einer CPU oder einer Datenbank nicht für beliebig viele Anfragen. Das Verteilen der Last auf mehrere parallele Kopien des Systems ist daher sinnvoll.

Beliebige Skalierbarkeit durch Parallelisierung

Ein Mechanismus mit dem Namen „load-balancing" kann je nach Auslastung eines Servers weitere CPUs zu Berechnung oder Kopien der Daten hinzuschalten, um eine parallele Bearbeitung zu ermöglichen. Während die parallele Abarbeitung von Vergleichen auf mehreren CPUs konzeptuell einfach zu realisieren ist, ergibt sich bei der Spiegelung der Datenbank folgendes Problem. Um alle Datenbankkopien identisch zu halten, müssen Einträge in einer Datenbank auf die anderen Systeme kopiert werden. Diese würde das Problem der Last nur verschieben. Ohne eine solche Synchronisierung der Datenbestände, würden Benutzer je nachdem, mit welcher Kopie der Datenbank sie interagieren, jeweils einen anderen Teil der Lösungen sehen.

Um das Problem zu entschärfen, bietet sich die Trennung der Datenbank in zwei Teile an. Neue Lösungen könnten zuerst in eine separate Datenbank abgelegt und zu Zeiten geringer Last mit den Ausgabedatenbanken abgeglichen werden. Dies stellt einen Kompromiss dar, der auf Kosten der Aktualität der Daten eine beliebige Skalierung des Systems ermöglicht.

4.3 Clientanwendung 1: das JSP-Webinterface

Die Stärke der browserbasierten Clientanwendung, liegt in der Möglichkeit praktisch von überall auf die Funktionen des **SolutionBrokers** zuzugreifen. Dies ist dann nötig, wenn der Benutzer zur Problemlösung keine Software mit integriertem Plugin zur Verfügung hat oder gefundene Lösungen nachträglich von einem anderen als dem Fehlersystem eintragen will. Ein regulärer Web-Browser steht selbst dann meistens zur Verfügung.

Die Nachteile dieser Client-Variante, nämlich die zusätzlichen manuellen Schritte zur Eingabe der Problembeschreibung sollen durch hohe Benutzerfreundlichkeit und einer Vermeidung sich wiederholender Arbeitsschritte wettgemacht werden. Es folgen daher zunächst einige realisierte Ideen zur Verbesserung der Usability,

4.3.1 Benutzerfreundlichkeit

Anders als in Eclipse, gibt es im Browser keine Webseiten-spezifischen Voreinstellungen. Die Speicherung eines Loginnamens, zur Verwendung bei Eintragen von Lösungen muss hier anders gelöst werden. Um ein wiederholtes Einloggen bei jeder Benutzung zu vermeiden, setzen wir so genannte *Cookies* ein. Benutzerdaten werden dabei in Form einer Textdatei – dem Cookie - auf Clientseite abgelegt und können beim nächsten Besuch automatisch verwendet werden. Ein wiederholtes Anmelden ist nicht nötig. Da zur Identifizierung des Benutzers keinerlei sicherheitskritische oder datenschutzrelevante Information benötigt werden, sind die bei Cookies sonst üblichen Sicherheitsbedenken nicht angebracht. Man muss aber davon ausgehen, dass verschiedene Benutzer denselben Webbrowser benutzen. Um zu verhindern, dass Einträge unter dem Namen eines anderen Benutzers abgelegt werden, wird ähnlich zu anderen Internetseiten (z.B. www.amazon.de, www.ebay.de) der Benutzernamen an prominenter Stelle angezeigt und die Möglichkeit angeboten sich unter anderen Namen anzumelden.

Automat. Anmeldung des Benutzers mit Cookies

Beim Eintragen neuer Lösungen, werden einige Benutzer zunächst prüfen wollen, ob ähnliche Einträge bereits vorhanden sind. Eintragen werden sie Ihren Lösungsvorschlag nur dann, wenn er nicht bereits im System gespeichert ist. Um nach der Suche und dem Drücken des „Zurück-Buttons" nicht erneut den Stacktrace eingeben zu müssen, bietet es sich an die Eingabefelder der Suchfunktion und auch der Eintragsfunktion auf einer Seite zu vereinen.

Unnötige Wartezeiten und Belastungen des Servers lassen sich durch sinnvolle Vorbelegungen der Eingabefelder und syntaktische Überprüfungen der Benutzereingaben auf Client-Seite vermeiden. Die Tests können mit Mitteln des Browsers (Javascript oder HTML-Formulare) realisiert werden.

Bei unseren Überprüfungen testen wir, ob:

- wichtige Fehler freigelassen wurden (z.B. Eintragen „leerer Lösungen")
- Zahlenfelder nur gültige numerischen Zeichen enthalten
- maximale Feldlängen eingehalten werden (z.B. Länge des Benutzername)

Komplizierte Test, wie die auf syntaktische Korrektheit der Problembeschreibungen, können allerdings erst von der Serveranwendung durchgeführt werden.

Die Ergebnisliste bei der Lösungssuche enthält unter Umständen viele Treffer. Die Art der Darstellung der Lösungen (Anzeige in HTML-Frames) macht es nötig, dass bereits beim Laden der Seite alle Daten an den Client

übermittelt werden. Da eine Anfrage unter Umständen mehrere Dutzend potentieller Lösungen liefert, kommt es zu merklichen Verzögerungen beim Seitenaufbau und unübersichtlichen Trefferlisten. Wir bieten daher die Möglichkeit, festzulegen, wie viele Treffer auf einmal dargestellt werden sollen.

Hervorzuheben ist, dass dies die Lösungssuche, also das Matching auf der Serverseite, nicht beschleunigt. Es führt im Gegensatz sogar zu zusätzlichem Rechenaufwand. Die Art der Differenzberechnung bei der Anzeige von 100 Lösungen macht z.B. bei jedem Zehnerschritt einen erneuten Vergleich *mit allen* Datenbankeinträgen nötig, da eben nur die Kenntnis *aller* Differenzwerte eine Aussage über die zehn besten Einträge erlaubt. Allerdings ließe sich dieser Mangel durch eine temporäre Zwischenspeicherung der Differenzwerte in der Serveranwendung beheben. Bei 100 Treffern müssten einmal alle Differenzen berechnet werden und gespeichert werden, die aber erst bei einer Anfrage der nächsten Zehnerschritte verwendet und übermittelt würden.

4.3.2 Umgang mit Eingabefehlern

Auf ein häufiges Problem bei der Anzeige von Benutzereingaben auf Webseiten soll gesondert eingegangen werden. Ähnlich wie bei Programmiersprachen, gibt es auch in der Gestaltungssprache für Webseiten HTML reservierte Zeichen oder Zeichenketten, die nicht im eigentlichen Inhalt verwendet werden dürfen. Da es sich bei HTML um eine XML-ähnliche Auszeichnungssprache handelt, dürfen z.B. spitze Klammer nicht direkt im Text vorkommen, da diese vom Browser sonst fälschlicherweise als Teile von Markup-Elementen missverstanden werden können. Um nicht als Starttag für die Fettschrift (<B>) interpretiert zu werden, muss man z.B. statt

a < b

die spitze Klammer durch die Escape-Zeichenkette

a < b

ersetzen. Andernfalls erhält man eine fehlerhafte Darstellung der Webseite. Dies ist auch bei der Verwendung sprachspezifischer Sonderzeichen, wie den deutschen Umlauten, nötig (so muss z.B. ö durch ö ersetzt werden).

Wir wollen die Verwendung solcher Zeichen nicht verbieten, da die Eingaben im Stil von „Fehler tritt nur auf wenn HeapSize < 1024“ als Lösungstext keine Seltenheit sind. Sonderzeichen können entweder einmalig beim Ablegen in die Datenbank oder jedes Mal bei der Anzeige durch eine Clientanwendung ersetzt werden. Auch wenn die erste Variante wegen des einmaligen Umwandelns effizienter erscheint, ist es in unserem Fall von Nachteil Escapezeichen in der Datenbank zu speichern. Bei der Ausgabe in anderen Umgebungen (z.B. dem Eclipse-Client) wäre eine Rückkonvertierung nötig, da Eclipse die HTML-spezifischen Escapesequenzen nicht interpretieren kann.

Ein anderes Problem würde sich durch die Verlängerung des Texts bei der Ersetzung ergeben. Hierzu ein kurzes Beispiel. Ein fiktives Datenbankattribut hat die Maximallänge von 20 Zeichen. Die Eingabe

Speichergröße > 1024

erfüllt diese Bedingung. Wird vorher allerdings das ö und > ersetzt, verletzt der neue Wert

Speichergrösse > 1024

die Maximallänge.

Um Probleme dieser Art zu umgehen und die Inhalte in der Datenbank systemunabhängig zu halten, werden Sonderzeichen erst auf Clientseite bei der Ausgabe auf Webseiten ersetzt.

Außerdem sollen typische Eingabefehler des Benutzers, die eine Fehlfunktion des Systems provozieren könnten, erkannt und durch Fehlermeldungen an den Benutzer kommuniziert werden. Die meisten Test lassen sich bereits auf Seite des Client mit Methoden des Browsers (HTML Formulare und Javascript) durchführen.

Diese sind:

- Eingabe von nicht numerischen Zeichen in Zahlenfelder
- Überschreiten der Maximallänge eines Feldes
- Freilassen zwingend auszufüllender Felder

4.3.3 Behandlung typischer Internetprobleme

Wir verwenden für unseren webbasierten Client Funktionen des Browsers wie Java Script, Frames, und Cookies. Da die Einstellungen des Benutzers dies eventuell unterbinden oder ältere Browser sie gar nicht unterstützten, muss der Benutzer auf solche Voraussetzungen hingewiesen werden oder vom System eine Alternativmöglichkeit angeboten bekommen. Browserfunktionalität, auf die wir nicht verzichten wollen, sind Frames und Javascript. Werden sie vom Browser nicht unterstützt, wird eine Warnung angezeigt, dass Teile des **SolutionBroker** eventuell nicht korrekt funktionieren.

Um Lösungsbeiträge von Benutzern in der Datenbank später eindeutig identifizieren zu können, ist eine Möglichkeit, dass dieser sich bei der Benutzung des Systems mit einem eindeutigen Namen anmeldet. Um die wiederholte Eingabe auf demselben Computer zu vermeiden, werden diese in einem so genannten Cookie abgelegt. Obwohl wir die Lebensdauer der Cookies auf den maximalen Wert von 1 Jahr setzen, müssen wir damit rechnen, dass nach Ablauf dieser Zeit oder durch manuelles Löschen das Cookie nicht vorhanden ist. In diesem Fall sollte das System ebenfalls ohne Fehler arbeiten. Die dafür nötigen Tests wurden implementiert.

Um auf Serverseite Informationen zwischen den einzelnen Seiten der Benutzeroberfläche auszutauschen, wird das so genannte Session-Objekt verwendet. Dieses speichert auch benutzerspezifische Daten wie die Lösungsanfrage oder den Login-Namen. Ähnlich wie Cookies können auch Session-Objekte nach einer Zeit vom System gelöscht werden. Damit die Serveranwendung bei Anfragen ungültiger Objekte keine Fehler produziert, wird dieser Fall ebenfalls durch Test abgefangen. Benutzer werden auf die Startseite verwiesen, falls Cookie oder Session nicht mehr existieren.

4.3.4 Benutzungsszenario

Die nachfolgenden Screenshots zeigen die Schritte bei der Benutzung der beiden Clientanwendungen – einmal bei der Suche bzw. beim Eintragen von Lösungen.

Abbildung 37: Um die Akzeptanz bei den IBM-internen Tests zu erhöhen, wurden Gestaltung und Text des Anmeldungs-Bildschirms an die WebSphere-Administrationskonsole angepasst. Der Benutzer wird auf den Sinn der Anmeldung hingewiesen und vor der Benutzung sicherheitskritischer Passwörter als Loginnamen gewarnt.

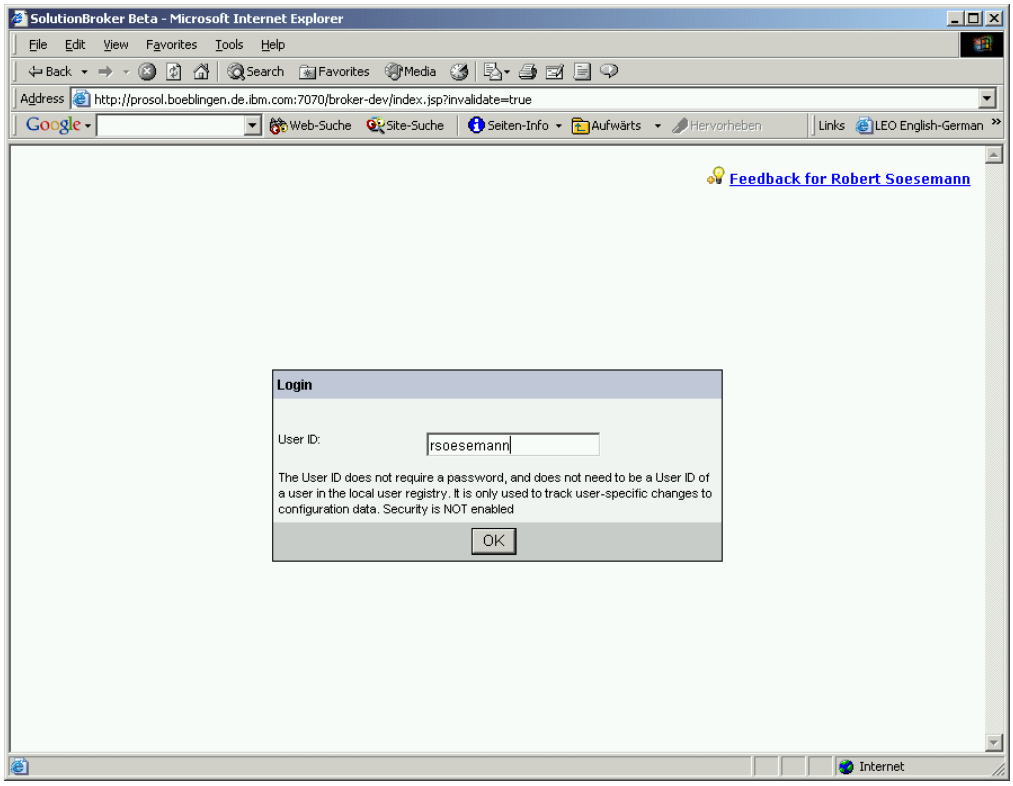

Abbildung 37 JSP-Clientanwendung : Login-Seite

Abbildung 38: Die Eingabefelder für die Suche und das Beisteuern eigener Lösungen finden sich beide auf der Hauptseite, die der Benutzer nach der erfolgreichen Anmeldung angezeigt bekommt.

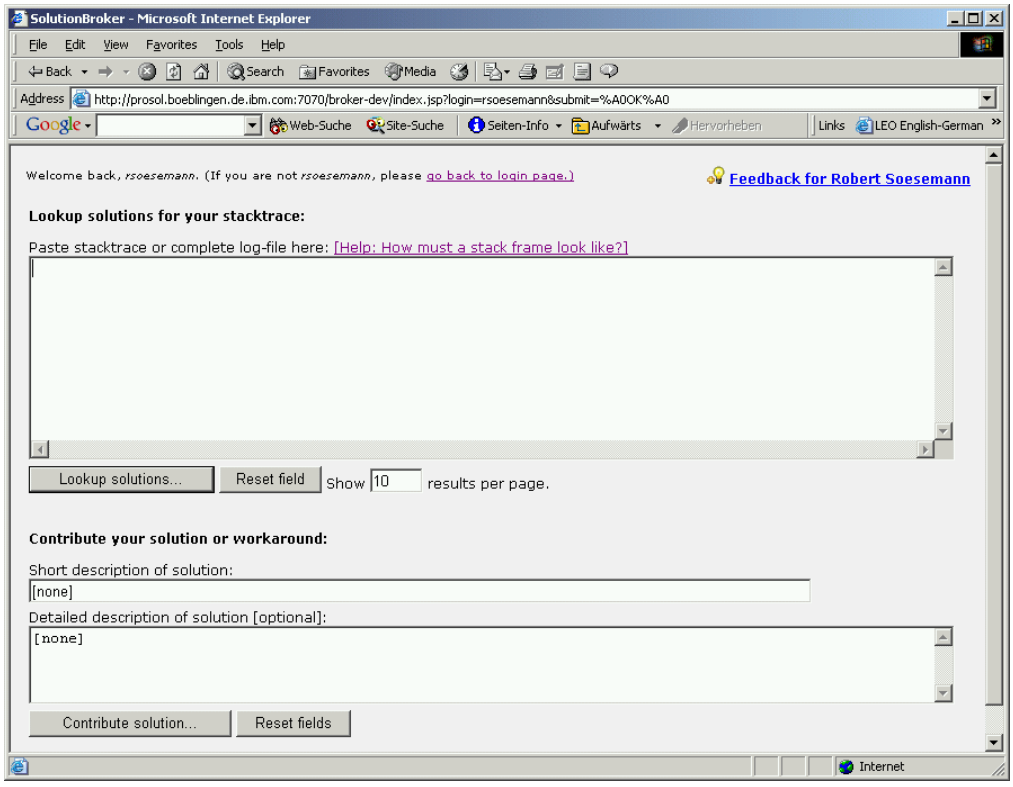

Abbildung 38 JSP-Clientanwendung : Hauptseite

Die Felder im unteren Teil sind nur für die Eingabe eigener Lösungen relevant, während im Eingabefeld oben immer eine Problembeschreibung in Form von Stacktraces eingefügt werden muss.

Außerdem findet sich dort ein Feld zur Festlegung der maximal anzuzeigenden Treffer, und Buttons zum Absenden der Abfrage sowie zum Löschen bzw. Rücksetzten der Felder auf Defaultwerte. Die Abbildung 39 zeigt den häufigen Fall, dass aus einem Emailprogramm nicht nur ein Stacktrace, sondern eventuell eine komplette Email eingefügt wird.

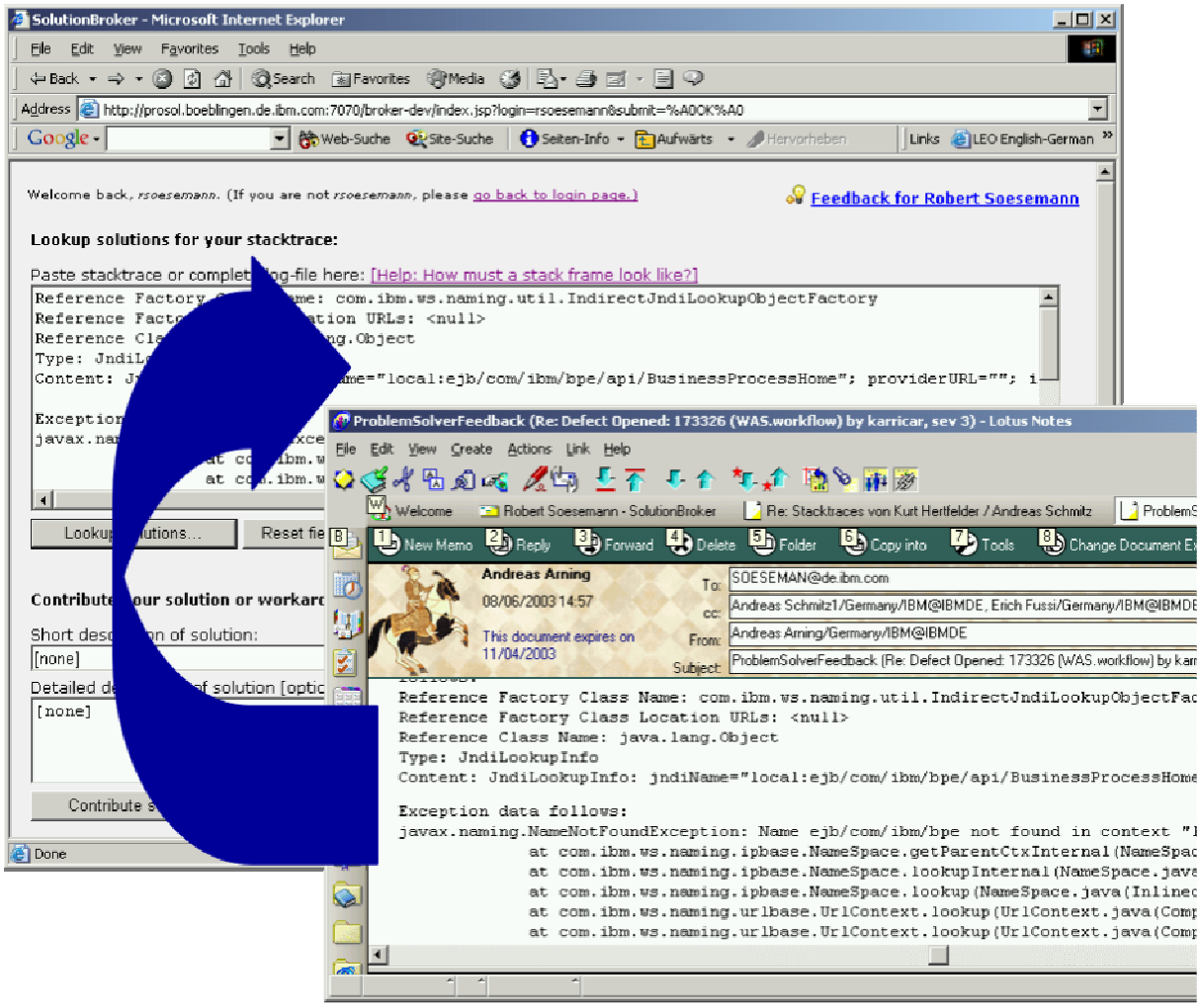

Abbildung 39 JSP-Clientanwendung : Lösungssuche mit Copy&Paste aus Emailprogramm

Um bei den IBM-internen Tests den Benutzern eine bequeme Möglichkeit für Anmerkungen zu geben, findet sich auf der Hauptseite des Webinterfaces ein Link, der schnelles Feedback über ein teilweise ausgefülltes Emailformular ermöglicht.

Abbildung 40: Für den erkannten Stacktrace wurden 4 Lösungen vom System gefunden. Da vorher die Grundeinstellung nicht geändert wurde, werden aber nur 10 Treffer pro Seite angezeigt. Diese werden absteigend nach Relevanz sortiert in der Tabelle oben links dargestellt.

Zusätzlich zu den errechneten Differenzpunkten werden als intuitiveres Gütemaß ein an die AMAZON Bewertungssterne angelehntes System von 5 Glühbirnen (1 Birne: geringe Relevanz - 5 Birnen: hohe Relevanz) angezeigt. Die Tabelle enthält außerdem Informationen wie Erstellungsdatum und Autor einer Lösung sowie einer Kurzbeschreibung und die Fehlercodes des gelösten Problems.

Durch Anklicken eines Treffers wird im rechten Bildschirmbereich die ausführliche Beschreibung der ausgewählten Lösung angezeigt, sofern eine solche in der Datenbank vorhanden ist. Damit die Anwender den eigenen Stacktrace und den aus der Datenbank selbst noch einmal vergleichen können, werden beide am unteren Rand angezeigt.

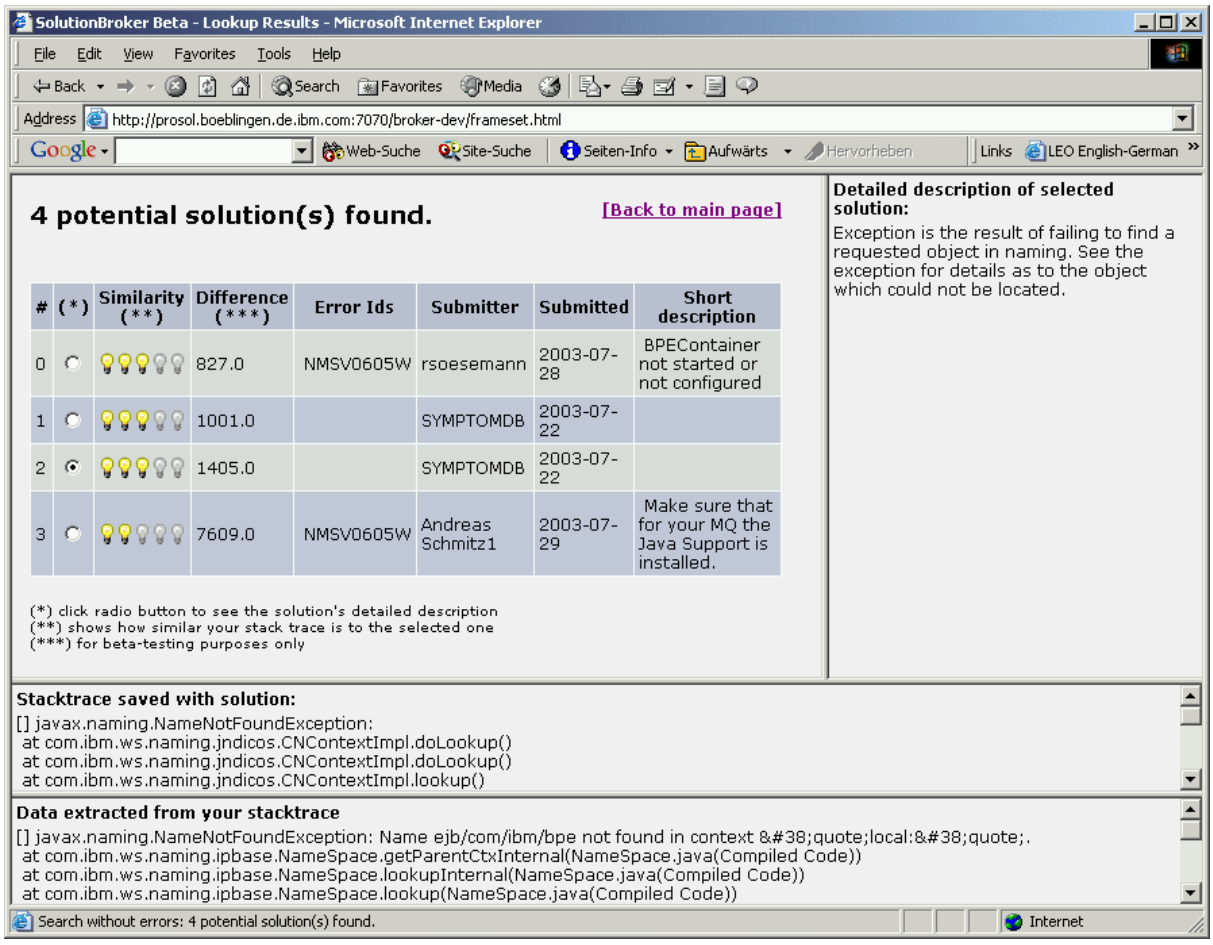

Abbildung 40 JSP-Clientanwendung : Trefferliste nach Lösungssuche

Ein Anwender, der eine Lösung veröffentlichen möchte, verfährt ganz ähnlich. Bei identischen Schritten wie der Anmeldung verzichten wir deshalb auf Bildschirmfotos.

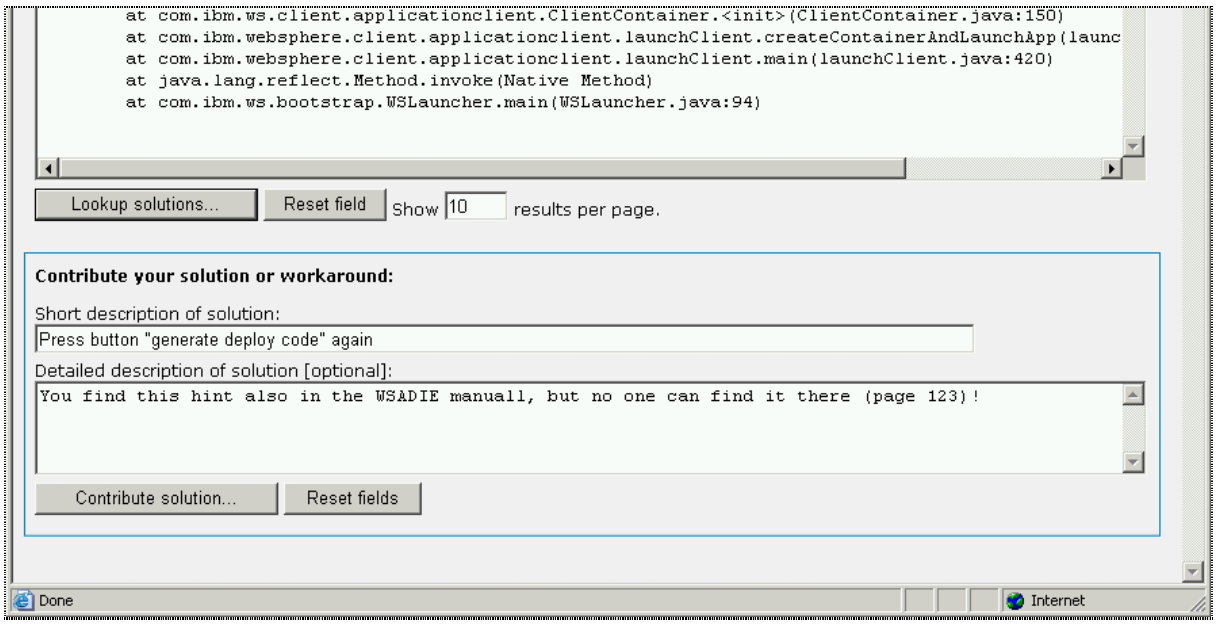

Abbildung 41 JSP-Clientanwendung : Eingabe einer Lösung

Abbildung 41: Er trägt dazu zusätzlich zum Stacktrace im oberen Teil der Hauptseite in die Felder darunter noch die Beschreibung seiner Lösung ein. Dabei ist nur die Angabe einer Kurzbeschreibung erforderlich. Die Eingabe eines längeren Textes im zweiten Feld ist optional. Die erfolgreiche Übermittlung des Lösungseintrages oder eventuelle Fehler werden anschließend im oberen Teil des Bildschirms angezeigt (siehe Abbildung 42).

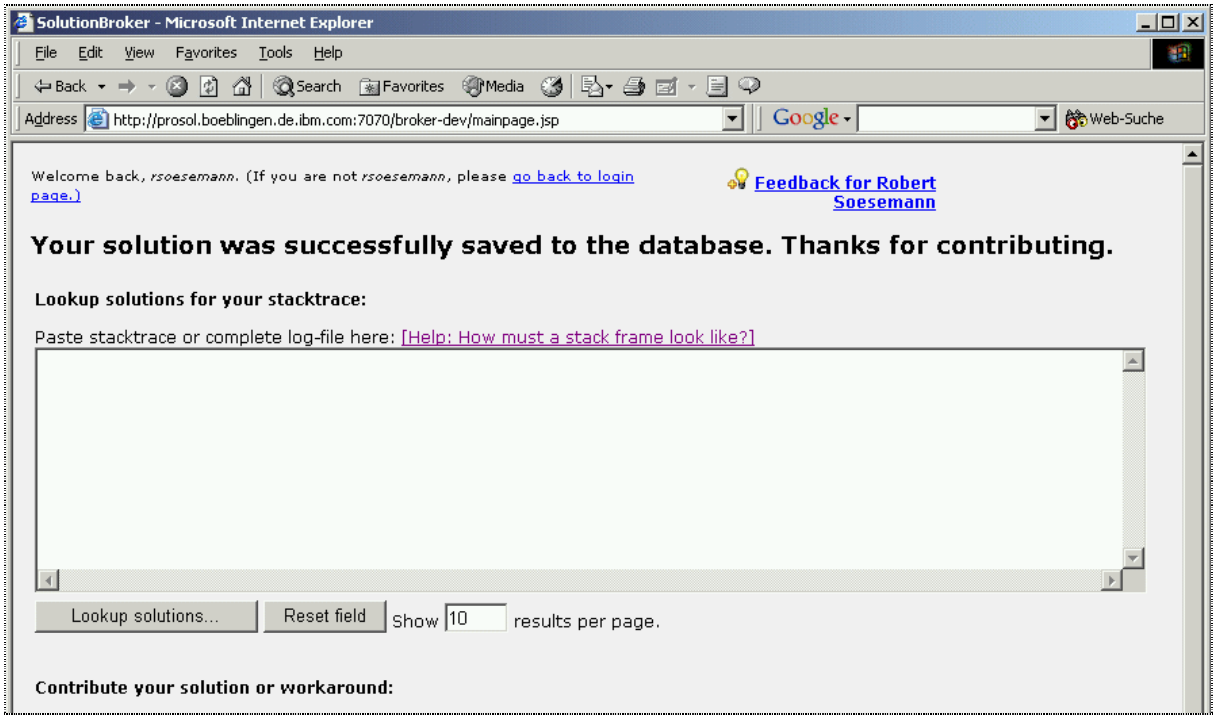

Abbildung 42 JSP-Clientanwendung : Bestätigung nach Eintragen einer Lösung

4.4 Clientanwendung 2: das Eclipse-Plugin

4.4.1 Vorteile der IDE-Integration

Die Integration der Clientanwendung in Anwendersoftware bietet den Vorteil, dass ohne Umwege Problemsituationen dort behoben werden, wo sie auftreten. Die schnelle Behandlung von Problemsituationen bei der Arbeit mit WSAD bzw. Eclipse ist also die große Stärke des Client-Plugins.

Typische Aktionen, die zu Fehlern in der IDE führen können sind:

- manuelle Änderung von generiertem Programmcode
- Änderungen an Komponenten, die auf dem Server veröffentlicht sind
- Deployment von Komponenten auf dem integrierten Test-Server

In solchen Situationen werden Stacktraces direkt im so genannten Console-View, einem kleinen Textausgabefenster in der IDE (siehe Abbildung 44) angezeigt. Genau an dieser Stelle soll der **SolutionBroker** seine Dienste anbieten. Signalisiert der Benutzer durch Klicken eines Buttons einen Fehlerfall, kann die dort angezeigte Information automatisch ausgelesen und in ein für die Anfrage von Lösungen nötiges `Stacktrace`-Objekt konvertiert werden. Die Behandlung von Fehlern, die beim manuellen Kopieren von Fehlerstacktraces passieren können, ist somit überflüssig.

In WSAD bzw. Eclipse stehen außerdem wesentlich ausgereiftere GUI-Komponenten zu Verfügung als bei den HTML-Formularen unseres JSP-Webinterfaces. So verwendet beispielsweise die Trefferliste eine Tabelle, die eine Sortierung der Spalten erlaubt. Die interne Funktionsweise solcher Tabelle macht auch eine Beschränkung auf eine bestimmte maximale Anzahl von Lösungen überflüssig. Um bei großen Trefferlisten dennoch nicht die Übersicht zu verlieren, gibt es die Möglichkeit, im Client innerhalb der Treffer nach bestimmten Kriterien zu suchen.

Auch beim Beisteuern eigener Lösungen werden die Vorteile von WSAD bzw. Eclipse genutzt. Damit Benutzer sich nicht jedes Mal wieder beim System anmelden müssen, wurde die Entwicklungsumgebung um eine **SolutionBroker**-eigene Voreinstellungsseite (preference page) erweitert. Dort können Benutzer Ihren Loginnamen dauerhaft abspeichern. Der Client liest diese Information dann ähnlich wie bei der Verwendung von Cookies in der JSP-Clientanwendung automatisch aus.

Zur Eingabe der Lösungsbeschreibung öffnet sich durch einen Buttonklick ein Dialogfenster. Die Problembeschreibung wird wie oben automatisch aus der Konsole extrahiert.

4.4.2 Benutzungsszenario

Um die beiden Clientanwendungen besser vergleichen zu können, zeigen die folgenden Screenshots des Eclipse-Plugins ähnliche Schritte wie die des JSP-Browserinterfaces.

Bevor der Benutzer Lösungen ins System eintragen kann, ist es nötig, dass er seinen Loginnamen in den Voreinstellungen des Plugins eingibt. Dafür stellt WSAD bzw. Eclipse ein standardisiertes Dialogfeld und Mittel zur Syntax- und Längenüberprüfung zur Verfügung (siehe Abbildung 43). Die Speicherung dieser Daten und die Weitergabe an das Plugin übernimmt dabei WSAD bzw. Eclipse.

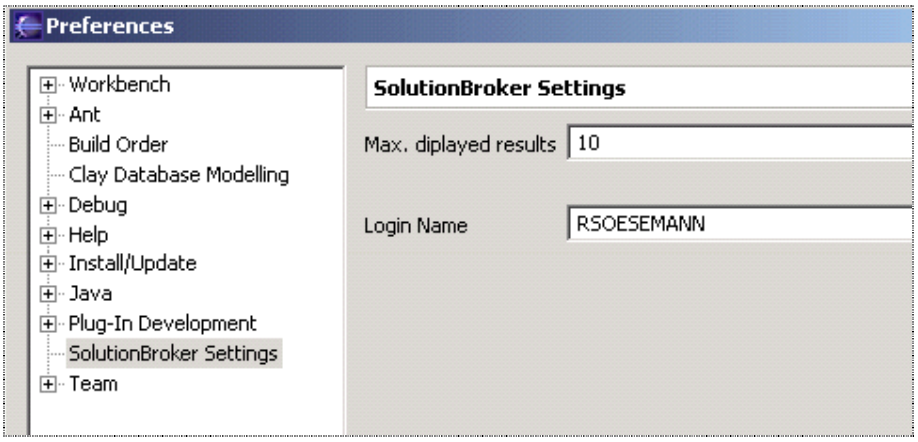

Abbildung 43 Eclipse-Plugin : Voreinstellungsdialog

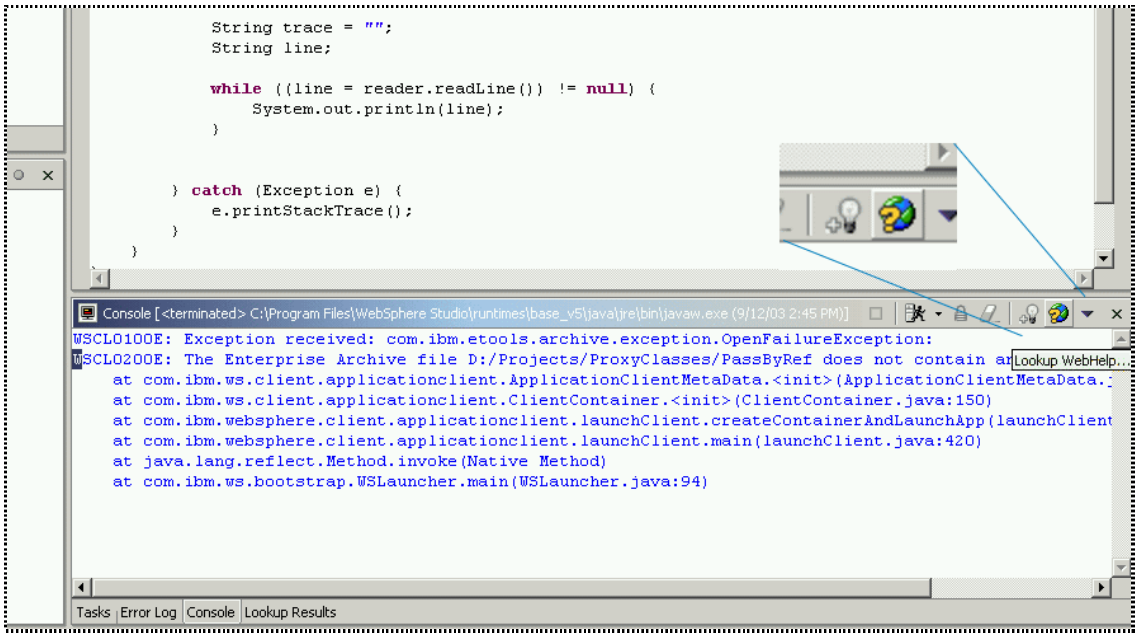

Abbildung 44 Eclipse-Plugin : Erweiterung der Eclipse-Fehlerkonsole

Wird nun während der Benutzung der IDE eine Exception ausgelöst, wird am unteren Bildschirmrand eine Textkonsole mit der Fehlerausgabe angezeigt (siehe Abbildung 44). Die Konsole wurde um einen Button erweitert, mit dem der Benutzer signalisieren kann, dass er mögliche Lösungen angezeigt bekommen möchte. Dabei wird die Fehlerausgabe automatisch extrahiert und mit Mitteln der XML Web Services an die Serveranwendung übermittelt.

Die Antwort des Servers wird ebenfalls durch Web Service-Komponenten in Javaobjekte umgewandelt und in der Trefferansicht (siehe Abbildung 45) als Tabelle dargestellt. Dort werden dieselben Informationen angezeigt wie bei der JSP-Clientanwendung. Auch hier können durch Anklicken eines Treffers die detaillierte Lösungsbeschreibung und der Problem-Stacktrace angezeigt werden. Mit der Filterfunktion kann die Trefferliste dialoggesteuert eingeschränkt werden.

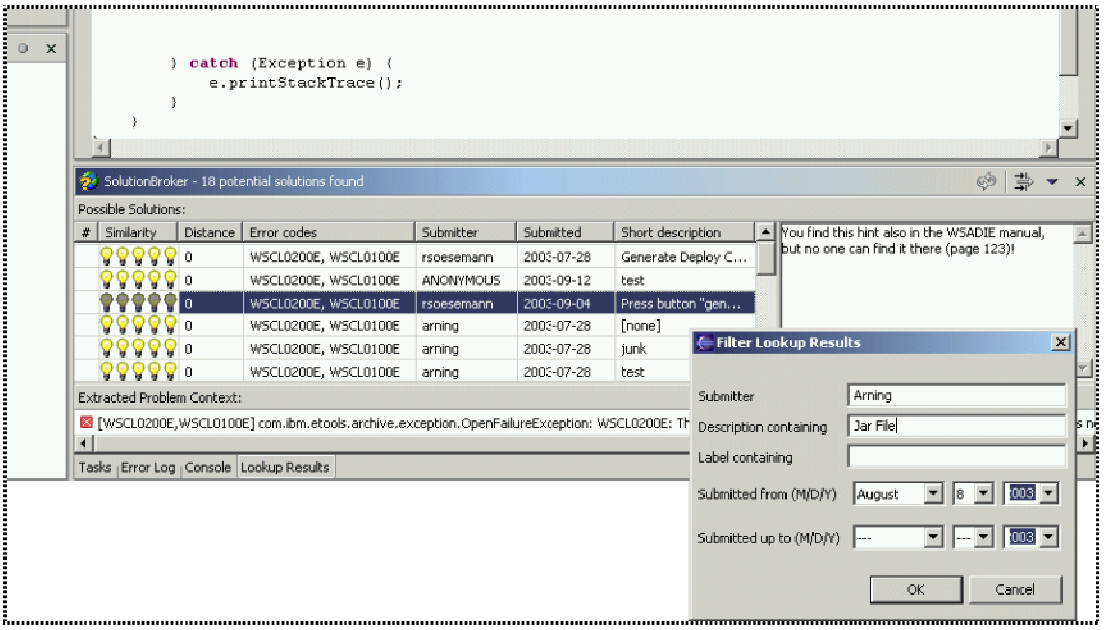

Abbildung 45 Eclipse-Plugin : Trefferliste und Filterdialog nach Lösungssuche

Ähnlich wie bei Lösungssuche können über einen Button in der Fehlerkonsole eigene Lösungsbeiträge erstellt und über XML Web Services an den Server versandt werden. Abbildung 46 zeigt das dazugehörige Eingabefenster.

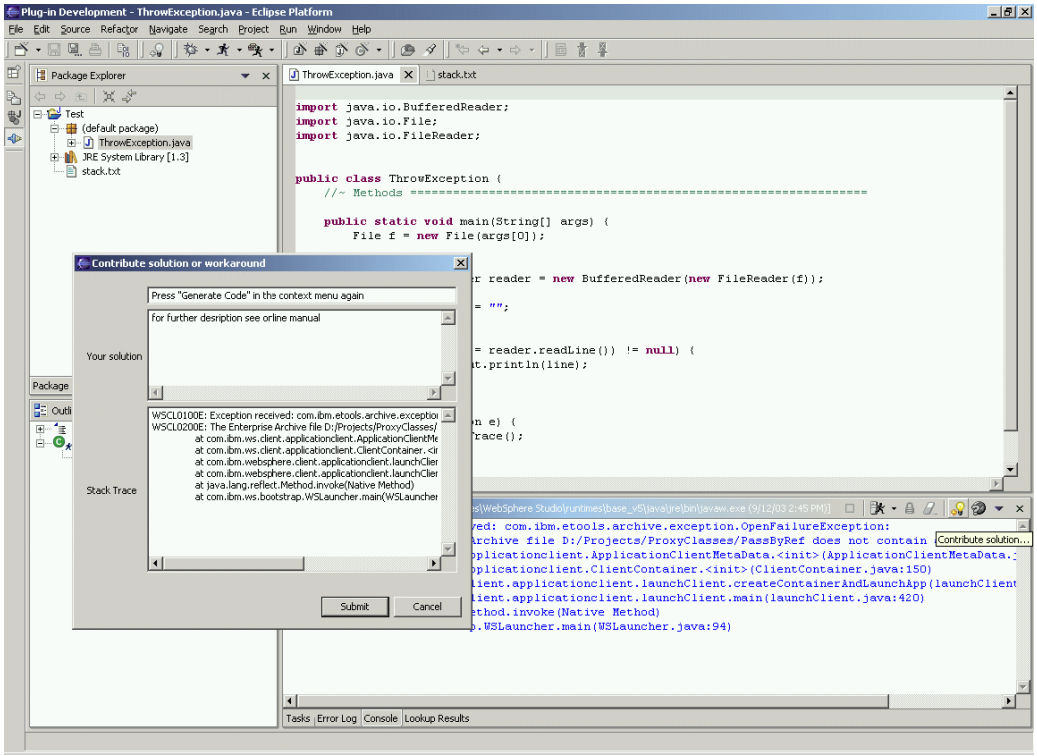

Abbildung 46 Eclipse-Plugin : Dialogfeld zum Beisteuern von Lösungen

KAPITEL 5 Ergebnisse der Diplomarbeit

Im Rahmen dieser Arbeit wurde das Konzept und die Implementierung eines neuen Hilfesystems mit dem Namen **SolutionBroker** vorgestellt. Durch die Bereitstellung einer zentralen Datenbank, in der Lösungen für Softwareprobleme mit minimalem Aufwand abgelegt und von anderen Anwendern gesucht werden können und durch die Automatisierung zahlreicher Arbeitsschritte, können Lösungen schneller gefunden oder zur Wiederverwendung Anderen bereitgestellt werden, als dies mit existierenden Hilfesysteme bisher möglich war.

Die wichtigsten Vorteile des **SolutionBrokers** sind:

- **Bidirektional**: Möglichkeit mit einem Hilfesystem sowohl existierende Lösungen im System zu finden, als auch eigene Lösungen anderen Anwendern zur Verfügung zu stellen
- **Zeitnah**: Neue Lösungen sind sofort nach ihrer Bereitstellung verfügbar. Dies war bei vielen bisherigen Systemen nicht möglich, da z.B. Suchmaschinen neue Internetseiten erst finden und indizieren müssen oder Hersteller nur Lösungen für häufige Problemsituationen in ihre FAQ aufnehmen.
- **Kontextinformation:** Um die Aussagekraft der Problembeschreibung bei der Suche und dem Beisteuern von Lösungen zu erhöhen, werden zusätzlich zur Fehlermeldung noch Informationen über den Problemkontext mit abgelegt.
- **Automatische Lösungssuche:** Ein Matchingalgorithmus extrahiert nur solche Lösungseinträge, die zu einem ähnlichen Problem gehören. Diese Vorfilterung ist durch einen Vergleich der Problembeschreibungen des Anwenders und denen in der Datenbank möglich. Die Problembeschreibungen werden in einer einheitlichen maschinenlesbaren Form abgelegt. Der Aufwand eines manuellen Aussortierens irrelevanter Suchergebnisse reduziert sich dabei stark. Unsere Beispielimplementierung verwendet dafür Java-Stacktraces.
- **Geringer Aufwand beim Beisteuern von Lösungen:** Eigene Lösungen können sofort und mit minimalem Aufwand dauerhaft anderen Benutzern zur Verfügung gestellt werden. Im Gegensatz zu Mailinglisten oder Newsgroups ist dazu weder die Interaktion mit anderen Personen noch längerfristiges Engagement in einem Internetforum nötig.
- **Hersteller- und Produktunabhängig**: Bei der Problembeschreibung werden nur Informationen verwendet, die von den meisten Programme und Herstellern zur Verfügung gestellt werden (z.B. Stacktraces). Damit soll ein möglichst großer Anwenderkreis erreicht werden.
- **Skalierbar:** Der **SolutionBroker** stellt keinen direkte Kontakt zwischen Experten und hilfesuchenden Anwendern her, sondern vermittelt (makelt) nur das Expertenwissen. Da dabei kein aufwendiger Dialog zwischen Menschen mehr nötig ist und wegen der Automatisierung bei der Lösungssuche, kann das System einem fast beliebig großem Anwenderkreis zu niedrigen Kosten zur Verfügung gestellt werden.

Nicht in allen Fällen in denen Softwarebenutzer Hilfestellung benötigen, kann unser System helfen. Dazu gehören vor allem die folgenden Fälle:

- **Behebung ungelöster Probleme:** Lösungen für die hier betrachteten Problemsituationen können bisher nicht automatisch, sondern nur von Menschen erarbeitet werden. Existiert für ein bestimmtes Problem eine Lösung noch nicht, kann der **SolutionBroker** im Gegensatz zu Systemen, die einen Kontakt zu Experten ermöglichen (z.B. Mailinglisten oder Hersteller-Hotlines) nicht helfen, da er sich auf den Austausch bereits erarbeiteter Lösungen konzentriert.
- **Problemprävention:** Im Gegensatz zu automatischen Aktualisierungssytemen wie *Microsoft Windows Update* kann und will der **SolutionBroker** nur Lösungen für Probleme liefern, die bereits beim Anwender aufgetreten sind. Eine präventive Problemvermeidung ist nicht das Ziel des vorgestellten Konzepts. Statt das Hilfesystem stark an ein Produkt, eine Fehlerart oder einen Hersteller zu knüpfen, wurde ein offener und verteilter Mechanismus geschaffen, mit dem beliebige Personen, hersteller- und produktunabhängig Hilfe mit minimalem Aufwand anbieten und finden können.

Die Stärken und Schwächen des **SolutionBrokers** im Vergleich mit existierenden Hilfesystemen zeigt die folgende Übersichtstabelle noch einmal auf einen Blick.

Eigenschaft	FAQ	Hersteller -Hotline	Präventiv-systeme	Such-maschinen	Mailing-listen & News-group	Solution Broker
Bidirektionalität					x	x
Zeitnah						x
Kontextinformation	x	x			x	x
Automatisierte Lösungssuche						x
Problemprävention			x			
Geringer Aufwand bei Beisteuern						x
First Time		x			x	
Hersteller- und produktunabhängig				x		x
Skalierbar	x		x	x		x

Abbildung 47 Featurevergleich mit anderen Hilfesystemen

Während der Arbeit an der Diplomarbeit wurde der **SolutionBroker** von Mitarbeitern des Böblinger IBM Labors getestet. Vor allem für die Testabteilung, die täglich dutzende Problemsituationen in Form von Stacktraces provoziert, bot die Implementierung eine einfache Möglichkeit, Fehler und Lösungsbeschreibungen für den internen Gebrauch abzulegen. Es ist geplant, das System auch für andere IBM Labors weltweit nutzbar zu machen und es später eventuell in IBM Produkte zu integrieren.

KAPITEL 6 Erweiterungsmöglichkeiten

Obwohl unsere Implementierung in IBM-internen Test erfolgreich eingesetzt wurde und sehr stabil lief, stellt sie weniger eine fertige Anwendung dar als ein „Proof of Concept", also einen Beleg für die Umsetzbarkeit des vorgestellten Konzeptes. Durch geeignete Erweiterungen ließen sich der Praxisnutzen und die Akzeptanz noch steigern. Wir möchten im Folgenden einige Ideen für mögliche Erweiterungen skizzieren.

6.1.1 Zusätzliche Kontextinformation

Um die Treffsicherheit des **SolutionBrokers** bei der Lösungssuche noch zu erhöhen, könnte man weitere Kontextinformationen in die Betrachtung miteinbeziehen, da außer Stacktraces auch andere Informationen über die Aktionen des Nutzers und das System während der Problemsituation für die Problembehebung von Interesse sein können.

Da Problemsituationen auch durch Fehler in fremden Komponenten verursacht werden können, bieten zusätzliche Informationen über die verwendete Hardware und systemnahe Software (z.B. Betriebssystem) weiteren Aufschluss über mögliche Fehlergründe.

Eine andere Erweiterung sind benutzerspezifische Daten. So gibt es sicherheitskritische Anwendungen (z.B. Bank- und Redaktionssysteme), die nur abhängig von Zugriffsrechten die Benutzung bestimmter Funktionen und Daten ermöglichen. Hat der Anwender die nötigen Rechte nicht, erhält er unter Umständen unverständliche Fehlermeldungen, die bei zusätzlicher Kenntnis der Benutzerdaten von unserem System besser behandelt werden könnten. Allerdings ergeben sich bei der Speicherung benutzerspezifischer Daten auch Datenschutzprobleme.

Durch die Integration des Clients in ein Anwendungsprogramm stehen weitere Informationen zur Verfügung. So haben die meisten modernen Programme eine so genannte Undo-History. Damit Benutzeraktionen bei Bedarf rückgängig gemacht werden können, protokolliert die Software diese mit (z.B. Mausklicks, Eingaben, Funktionsaufrufe). Diese Protokoll stellt eine Art erweiterten Stacktrace dar, da dort nicht nur Fehler im Systemcode erkannt werden könnten, sondern auch Bedienfehler in anderen Komponenten. Man muss bei der Verwendung dieser Information bedenken, dass sie meist stark zwischen den einzelnen Anwendungen und Herstellern variiert. Ein allgemeingültiger Fehlerkontext liegt somit oft nicht mehr vor und ein Vergleich beliebiger Fehler ist entsprechend schwieriger.

6.1.2 Automatische Suchen und Bereitstellen von Lösungen

An mehreren Stellen ließe sich das System weiter automatisieren, dass der Zusatzaufwand für seine Benutzung minimiert wird. Denkbare Erweiterungen wären die automatische Erkennung bzw. Beseitigung von Problemsituationen.

Um die Dienste des **SolutionBrokers** bei der Problembehebung nutzen zu können, muss der Anwender bisher dem Clientprogramm signalisieren, dass eine Problemsituation vorliegt.

Stattdessen könnte die Clientanwendung Problemsituation selbst erkennen und zur Lösungssuche die Serveranwendung kontaktieren.

Mögliche Ausprägungen sind:

- Permanente Überprüfung der Log-Dateien und Fehlerausgaben
- automatisches Anbieten von Lösungen für wiederkehrende Benutzerfehler
- Unterscheidung nach Schwere des Problems (manuelle Definition im Client oder in separater Serverdatenbank)

Zusätzlich zur Erkennung könnte die Software das Problem anhand der besten Lösung selbst beheben. Dazu ist es allerdings nötig, dass neben den Informationen über die Problemsituation auch die Lösung in einer maschinenlesbaren Form vorliegt, eine manuelle Eingabe einer Lösung durch Benutzer, die ein Problem behoben haben, ist dann nicht mehr praktikabel.

Aber auch ein automatisches Erkennen und Ablegen von Lösungswegen, die der Benutzer selbst gefunden hat, ist prinzipiell möglich. So könnten das beim Auftreten einer Fehlersituation die Aktionen des Benutzers mitprotokolliert werden und solche als Lösungsschritte extrahiert werden, die zur Beseitigung des Problems geführt haben. Die Software müsste automatisch feststellen, dass ein bestehendes Problem durch den Anwender behoben wurde und welche Aktionen dafür nötig waren. Diese Daten könnten dann in maschinenlesbarer Form abgelegt werden.

Bei einer derart hohen Automatisierung sind aber noch viele Fragen zu klären. Auch wenn wir diese hier nicht lösen können, möchten wir sie wenigstens stichpunktartig nennen:

- Übertragbarkeit von Lösungsaktionen zwischen Softwareprodukten: D.h. konkrete Aktionen wie Mausklicks oder Befehlskommandos lassen sich leicht durch Software erkennen, sind aber auch sehr programmspezifisch und nur begrenzt übertragbar
- Gefahr der Anwendung ungeeigneter Lösungsvorschläge: D.h. während beispielsweise bei Benutzer A das Formatieren der Festplatte ein Problem behebt, entstehen am System von Benutzer B durch diese Aktion irreversible Schäden
- Datenschutzverletzungen: Die ständige Beobachtung durch die Software und die unmerkliche Veröffentlichung von Informationen über das eigene Benutzerverhalten wird von vielen Anwender wahrscheinlich als unangenehm empfunden und kann gültige Datenschutzbestimmungen verletzen. Diese Funktionen müssten daher vorzugsweise deaktivierbar sein.

6.1.3 Kombination mit Anreizsystemen

Der Erfolg des **SolutionBroker** hängt entscheidend von der Menge und Qualität der Lösungseintrage ab, die Benutzer dem System zur Verfügung stellen. Da unser System in seiner Rolle als Makler zwischen Anbieter und Nachfrager tritt, erzwingt es im Gegensatz zu bisherigen Hilfesystemen wie Mailinglisten und Newsgroups keine Kontaktaufnahme der beiden Seiten. Anreize wie die Dankbarkeit anderer Menschen stehen uns also als Anreiz für die Hilfsbereitschaft nicht zur Verfügung.

Unser System minimiert den Aufwand, der nötig ist anderen zu helfen. Trotzdem ist beim Ablegen ein kleiner Zusatzaufwand nötig, der für den Benutzer keinen direkten Vorteil bringt. Die Einführung eines Anreizsystems könnte dieses Manko beseitigen.

Eine Möglichkeit wäre ein Punktesystem im Stile des Online-Buchhändlers Amazon (www.amazon.de). Je nach Anzahl geschriebener Buchkritiken wird der Name des Verfassers durch Darstellung von mehreren Sternen positiv hervorgehoben.

Zwei Fälle sollten von System getrennt gefördert und belohnt werden:

- Anzahl Lösungen (=viele Einträge in die Datenbank)
- Nutzerwert der Lösungen (= wie oft wurde anderen geholfen)

Um zu vermeiden, dass Benutzer mit der Speicherung unsinniger Einträge ihr Punktekonto vergrößern, sollten nur solche belohnt werden, die andere Benutzer als hilfreich empfunden haben. Dies könnte einfach durch Einbau einer Feedback-Rückkanals in die Clients umgesetzt werden. Jedes Mal, wenn ein Benutzer einen Eintrag als hilfreich bewertet, steigt das Punktekonto des Eintrages und des Autor in der Datenbank. Diese Rankinginformation durch Menschen könnte außerdem zur Verbesserung des Matchingalgorithmus verwendet werden.

Um nicht auf die rein ideelle Wirkung eines Punktesystems zu setzen, könnte man gesammelte Punkte mit konkreten Vorteilen verbinden. So bietet z.B. das Internetauktionhaus *eBay* aktiven Verkäufern attraktive Zusatzfunktion bei der Erstellung von Angeboten (Texthervorhebungen, Einbindung zusätzlicher Produktfotos).

📖 ***Weitere Informationen über das Anreizsystem von Experts-Exchange befinden sich unter [19].***

Eine interessante Erweiterung eines solchen Punktesystems um marktwirtschaftliche Prinzipien verwendet das Internetforum *Experts-Exchange*. Um ein Frage in das Forum stellen zu dürfen, benötigt man so genannte *Question Points*. Je nach persönlicher Wichtigkeit der Frage vergibt man mehr oder weniger von diesen Punkten. Diese Punkte werden nach erfolgreicher Beantwortung vom Fragesteller an einen oder mehrere Helfer verteilt. Die Fragen mit vielen Question Points sind daher für andere attraktiver und werden schneller beantwortet.

Um der Wichtigkeit von schnellen Problemlösungen im professionellen Sektor gerecht zu werden, ist es auch denkbar, die virtuelle Währung durch echtes Geld mit Hilfe eines Bezahlsystems (Micropayment-System) zu ersetzen. Anbieter und Nachfrager könnten festlegen, wie viel Lösungen eines Problems wert sind. Die Besonderheiten des immateriellen Guts Informationen lassen allerdings bei Fragen nach einem Rückgaberecht von unbrauchbaren Informationen viele Fragen ungeklärt.

Neben dem Anreiz für die Weitergabe von Wissen hat die Anwendung eines Bewertungssystems einen weiteren positiven Nebeneffekt. Der **SolutionBroker** verhindert nicht das erneute Eintragen bereits vorhandener Lösungen. Das System könnte zwar mehrfach vorhandene Problembeschreibungen herausfiltern, da diese maschinenlesbar sind, jedoch keine redundanten Lösungsbeschreibungen, da ihr Inhalt wegen seiner Prosaform nur für Menschen erfassbar ist. Dieses Herausfiltern ist aber gar nicht erwünscht, da es ja zu bestimmten Stacktraces verschiedene Lösungen geben kann.

Bisher gibt es kein automatisches Mittel, ungewollte Lösungen nachträglich aus dem System zu löschen, die eine offensichtlich unsinnige oder böswillig falsche Lösung enthalten. Bei Anwendung eines Bewertungssystems würden diese aber bald von gut bewerteten Einträgen auf den Trefferlisten auf derart niedrige Plätze zurückgestuft, dass die meisten Anwender gar nicht mit ihnen in Kontakt kämen.

ANHANG B Literaturverzeichnis

[1] Harley Hahn: *Understanding Mailing Lists*. 1999
www.harley.com/mailing-lists/

[2] Peter Doshi: *Understanding and Using Newsgroups*. 1999
http://www.sju.edu/infotech/webadmin/usenet/

[3] Sun Microsystems Inc: *Powering the Web Experience with Dynamic Content*.
http://java.sun.com/products/jsp/jsp_servlet.html

[4] Object Technology International Inc.: *Eclipse Platform Technical Overview*. 2003
http://www.eclipse.org/whitepapers/eclipse-overview.pdf

[5] Azad Bolour: *Notes on the Eclipse Plug-in Architecture*. 2003
http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html

[6] Hans Bergsten: *Improved Performance with a Connection Pool*. 1999
http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/conpool.html

[7] Techtarget Website: *Core Dump - search390.com Definitions*.
http://search390.techtarget.com/sDefinition/0,,sid10_gci211844,00.html

[8] Sun Microsystems Inc.: *The Java Tutorial - Lesson: Handling Errors with Exceptions*.
http://java.sun.com/docs/books/tutorial/essential/exceptions/

[9] Calvin Austin: *An Introduction to Java Stack Traces*. 1998
http://developer.java.sun.com/developer/technicalArticles/Programming/Stacktrace/

[10] T. Suganuma et al.: *Overview of the IBM Java Just-in-Time Compiler*. 1999
http://www.research.ibm.com/journal/sj/391/suganuma.html

[11] Beth Stearns: *The Java Tutorial – Trail: Java Native Interface*.
http://java.sun.com/docs/books/tutorial/native1.1/

[12] Darwin Open Systems: *Diff/Compare in Java*.
http://www.darwinsys.com/freeware/index.shtml

[13] Michael Gilleland: *Levenshtein Distance in Three Flavors*.
http://www.merriampark.com/ld.htm

[14] John Lambert: *Using stack traces to identify failed executions in a Java distributed system*. 2002 http://www.jlambert.com/john-lambert-thesis.pdf

[15] IBM Alphaworks: *Log and Trace Analyzer for Autonomic Computing*. 2003
http://alphaworks.ibm.com/tech/logandtrace

[16] Sun Microsystems Inc.: *The Java Web Services*. 2002
http://java.sun.com/webservices/docs/1.0/tutorial/index.html

[17] Maydene Fisher: *The Java Tutorial – Trail: JDBC Database Access*.
http://java.sun.com/docs/books/tutorial/jdbc/

[18] Lehman et al.: *A Study of Index Structures for Main Memory Database Management*. 1986
http://citeseer.nj.nec.com/context/256598/0

[19] Experts-Exchange Website: *Experts Exchange Help Pages*.
http://www.experts-exchange.com/help/index.jsp#14