[KuAck98] *Thomas Kubela, Ralf Ackermann;* **Agent-Q - das Diagnose- und Datenqualitätssystem;** IHS'98 GI-Jahrestagung Informatik '98, Magdeburg, 21.09.98

Agent-Q - das Diagnose- und Datenqualitätssystem

Ein Praxisbericht

Thomas Kubela[1], *Ralf Ackermann*[2]

1
debis Systemhaus GEI
Goebelstraße 1-3 • D-64293 Darmstadt

2
Technische Universität Darmstadt
Industrielle Prozeß- und Systemkommunikation
Fachbereich Elektrotechnik und Informationstechnik
Merckstr. 25 • D-64283 Darmstadt

tkubela@debis.com
Ralf.Ackermann@KOM.tu-darmstadt.de

Zusammenfassung

Agent-Q(uality) ist ein System, das Funktionalitäten einer aktiven Datenbank über heterogenen Datenbank-Management-Systemen, die diese Funktion nicht selbst anbieten müssen, realisiert. Es reagiert auf Ereignisse, überprüft Bedingungen und startet Aktionen nach dem ECA-Prinzip (Event-Condition-Action). Agent-Q kann mit beliebigen Datenbank-Management-Systemen, die über eine JDBC-Schnittstelle unmittelbar oder unter Nutzung von Middleware-Komponenten angesprochen werden können, kombiniert werden und die dort vorhandenen Funktionen sinnvoll ergänzen.

Der Beitrag beschreibt ausgehend von einer Darstellung der Erfordernisse und Probleme der Sicherstellung der Datenqualität in einem heterogenen und von einer Vielzahl von vorhandenen Anwendungen gekennzeichneten Umfeld die Konzeptbildung und Umsetzung einer portablen, flexiblen und skalierbaren Lösung, die durchgängig objektorientiert und unter Einsatz der Programmiersprache Java realisiert wurde. Das Potential dieser Sprache, Besonderheiten sowie Einsatzerfahrungen bei der Realisierung eines umfangreicheren Softwareprojektes werden exemplarisch dargestellt und Einsatzmöglichkeiten der entstandenen Implementierung aufgezeigt.

1 Einleitung

Verläßliche, qualitativ hochwertige Informationen gehören heute mehr denn je zum notwendigen Rüstzeug eines erfolgreichen Unternehmens. Laut einer Untersuchung der Gartner Group ist mangelnde Datenqualität jedoch eher die Regel als die Außnahme, was nicht zuletzt bei der Einführung von Data Warehouse Projekten und Executive Information Systems deutlich wird. Dabei treten oftmals Inkonsistenzen und fehlerhafte Daten sowohl innerhalb als auch zwischen den verschiedenen operationalen Systemen zu Tage.

Für Datenqualitätsprobleme gibt es verschiedene Ursachen. Ein Großteil entsteht durch Fehler in den einzelnen Prozessen, die die Dateneingabe, Zuordnung und Bearbeitung realisieren. Ebenso ist häufig das Applikationsdesign an sich verantwortlich für die schlechte Qualität der Daten.

Um die Datenqualität zu verbessern, sind zwei Aktivitäten notwendig. Zunächst muß eine Datenbereinigung des aktuellen Datenbestandes durchgeführt werden. Diese Datenbereinigung bringt die Daten kurzfristig in einen zuverlässigen Zustand, beseitigt jedoch nicht die Ursachen für deren mangelnde Qualität. Neu in die Systeme gelangende Daten können die Datenqualität wieder verschlechtern. Die langfristige Sicherstellung einer hohen Datenqualität wird nur mittels der Durchsetzung von Maßnahmen zum Datenqualitätsmangement erreicht. Dieses beinhaltet die kontinuierliche Überprüfung der Qualität der Daten und das Erkennen und Beseitigen der Ursachen für fehlerhafte oder unvollständige Daten.

Datenqualitätsmanagement wird von verschiedenen IT-Tools unterstützt.[Re96]

- Rule-discovery Produkte analysieren Daten in Archiven und Datenbanken, um Relationen zwischen den Daten festzustellen und Regeln zu selektieren, die die Art der Nutzung der Daten beschreiben. Diese Tools helfen z.B. schlecht dokumentierte Daten zu analysieren.

- Data cleansing/scrubbing Produkte analysieren Daten beim Portieren aus verschiedenen Archiven und Anwendungen in ein Data Warehouse. Sie finden Duplikate und transformieren Daten in korrekte oder wahrscheinlich korrekte Werte. Man kann dabei generalisierte und komplexe Transformations-Regeln definieren.
- Error-Prevention Produkte erzwingen Integritätsprüfungen an der Quelle der Daten. Diese Produkte können direkt von der Applikation aufgerufen werden oder man kann mit ihrer Hilfe Module entwickeln, die die Daten bei der Eingabe testen.
- Datenqualitäts-Analyse und Audit Produkte überprüfen Daten anhand von Business rules und decken gegebenfalls Inkonsistenzen auf.

Die im folgenden beschriebene Anwendung gehört in den Bereich der Datenqualitäts-Analyse Produkte. Es handelt sich dabei um ein automatisches Datenbankprüfsystem, das störende und auffällige Daten ermittelt und entsprechend vordefinierte Maßnahmen einleitet.

2 Aktive Datenbanksysteme

2.1 Funktionsprinzip

„Ein Datenbanksystem heißt aktiv, wenn es zusatzlich zu den üblichen DBS-Fähigkeiten in der Lage ist, definierbare Situationen in der Datenbank ... zu erkennen und als Folge davon bestimmte ... Reaktionen auszulösen [DiGa96]".

Wesentlicher Bestandteil aktiver Datenbanktechnik ist die Möglichkeit, Regeln zu definieren, die unter frei bestimmbaren Bedingungen die Inhalte der Datenbank überprüfen. Eine Regel setzt sich dabei aus drei Teilen zusammen:

- einem „Event" — dem Ereignis, das die Datenbankprüfung auslöst
- einer „Condition" — der Definition der Überprüfung
- einer „Action" — der Reaktion des Systems auf das Ergebnis der Überprüfung

Die Abarbeitung einer Regel erfolgt nach dem Event-Condition-Action (ECA) Prinzip. Das Auslösemoment ist durch den „Event"-Bestandteil der Regel definiert. Ein „Event" kann z.B. eine Operation in der Datenbank, das Erreichen eines festen oder sich regelmäßig wiederholenden Zeitpunktes oder ein über eine definierte, externe Schnittstelle eingehendes Signal sein, aber auch explizit durch einen Anwender ausgelöst werden.

Wird eine Regel durch ein Ereignis, welches mit dem in der Regel beschriebenen „Event" matched, aktiviert, überprüft das aktive Datenbanksystem den Datenbestand anhand der zur Regel gehörenden „Condition". Diese kann z.B. eine durch die Charakteristik des Events zusätzlich parametrisierte in SQL (Structured-Query-Language) formulierte Datenbank-Abfrage sein.

Das Ergebnis der Überprüfung einer „Condition" ist zunächst ein boolescher Wert. Ist dieser (je nach Spezifikation in der „Condition") true bzw. false, wird eine „Action" ausgelöst, der zusatzlich auch möglicherweise durch die „Condition" bestimmte Datensätze als Parameter übergeben werden können. Durch den „Action"-Teil der Regel wird spezifiziert, wie die weitere Verarbeitung erfolgt und welche Operationen resultieren. Die Regelbestandteile werden durch das aktive Datenbanksystem verwaltet, sie können über eine Nutzerschnittstelle eingegeben, modifiziert und zu Regeln kombiniert werden.

2.2 Abbildung von Mechanismen in einem heterogenen Einsatzumfeld

In einem Umfeld vorhandener heterogener Applikationen ist die Umsetzung der beschriebenen Funktionalität aktiver Datenbanken oft wünschenswert, wird aber von den eingesetzten DBMS nicht oder nur unangemessen unterstützt. So unterstützen in der Praxis gängige (R)DBMS bestenfalls Ereignisse, die auf Datenbankoperationen basieren (Trigger). Ein aktiver Abgleich von Daten zwischen verschiedenen DBMS ist i.d.R. nicht möglich.

Die individuelle Einbindung entsprechender Mechanismen in einzelne Applikationen, sei es mit Unterstützung des eingesetzten DBMS oder durch reine Realisierung im Applikationscode, ist aufwendig und kann insbesondere auch die Anforderungen von Prozessen, die mehrere zunächst autonome Datenbanken referenzieren (z.B.

Integration unterschiedlicher Geschäftsprozesse in ein einheitliches Modell, Migration von Daten) nur unbefriedigend erfüllen.

In der Literatur [ShLa90][CoTü95] wird ein Vorschlag zur Konsistenzsicherung in förderierten heterogenen Informationssystemen durch aktive Mechanismen in der Förderierungsebene sowie eine entsprechende Rahmenarchitektur vorgestellt.

3 Konzeption und Systemarchitektur

Die Funktionalität einer aktiven Datenbank wird durch eine Applikation nachgebildet, welche Regeln verwaltet und bei deren Aktivierung durch Events, den Zugriff auf die zu überprüfenden Datenbanken durchführt und die Ergebnisse verarbeitet. Das System besteht aus zwei wesentlichen Teilen, der Regelverwaltung und einem Event-Handler. Die Regelverwaltung dient zum Anlegen und Ändern von Regeln, während der Event-Handler den ausführenden Part übernimmt, d.h. die in der Regelverwaltung angelegten Regeln durchsucht und auf die zu behandelnden Datenbanken anwendet.

Die Kommunikation zwischen den beiden Teilen der Anwendung erfolgt allein über die Regeldatenbank, diese speichert alle im Gesamtsystem vorhandenen Regeln. Welche zu überprüfenden Datenbanken jeweils anzusprechen sind, wird über entsprechende Attribute vermerkt. Eine solche lose Kopplung ist sehr flexibel und ermöglicht z.B. den Betrieb der Event-Handler-Komponente bereits vor Fertigstellung des Nutzerinterfaces zur Manipulation von Regeln.

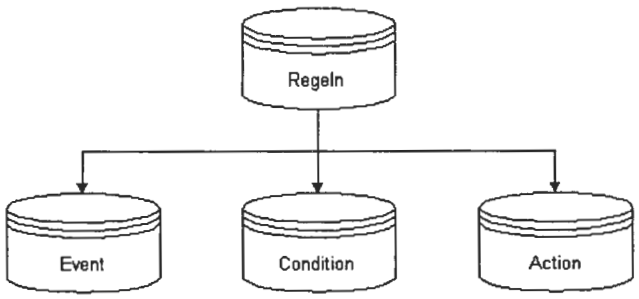

Abbildung 1: Bestandteile und Abspeicherung von Regeln in Agent-Q

Die Anwendung wird verteilt mit einem zentralen Server und einer Anzahl von durch die Anwender zu nutzenden Clients realisiert. Während der Server durchgehend aktiv ist, feststellt, ob Regeln aktiviert werden und diese dann abarbeitet, werden die Client-Systeme nur zur Nutzer-Interaktion mit dem System benötigt und können ansonsten deaktiviert werden. Von einem aktiven Client aus ist die Modifikation der Regelbasis sowie der Zugriff auf bei der Regelabarbeitung generierte und persistent abgelegte Daten möglich. Weiterhin wird das Gesamtsystem ebenfalls von den Clients ausgehend gesteuert.

Um zu gewährleisten, daß nur berechtigte Benutzer das Diagnose- und Datenqualitätssystem verwenden, muß sich der Anwender durch die Eingabe eines Benutzernamens und eines Paßwortes als berechtigt ausweisen. Aufgrund der unterschiedlichen Natur der durch den Anwender abrufbaren aufbereiteten Daten und der „Systemverwalter"-Komponente, wurde eine Lösung mit Nutzerklassen, denen unterschiedliche Rechte zugeordnet werden können, konzipiert und realisiert.

Damit hat ein Anwender die in Abbildung 2. aufgezeigte Sicht auf das Gesamtsystem.

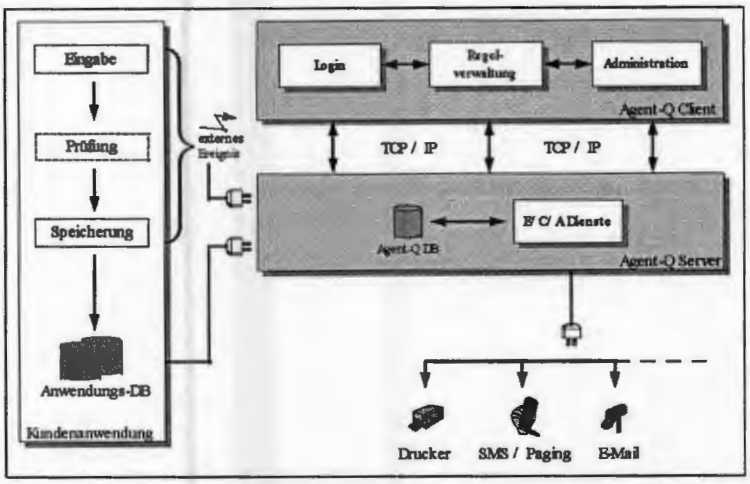

Abbildung 2: Nutzersicht auf das Gesamtsystem

Aus diesem Ansatz resultiert die in Abbildung 3. gezeigte und in der Beschriftung bereits auf eine Umsetzung in Java ausgelegte Gesamtarchitektur, der auch die Verteilung der Funktionalität entnommen werden kann.

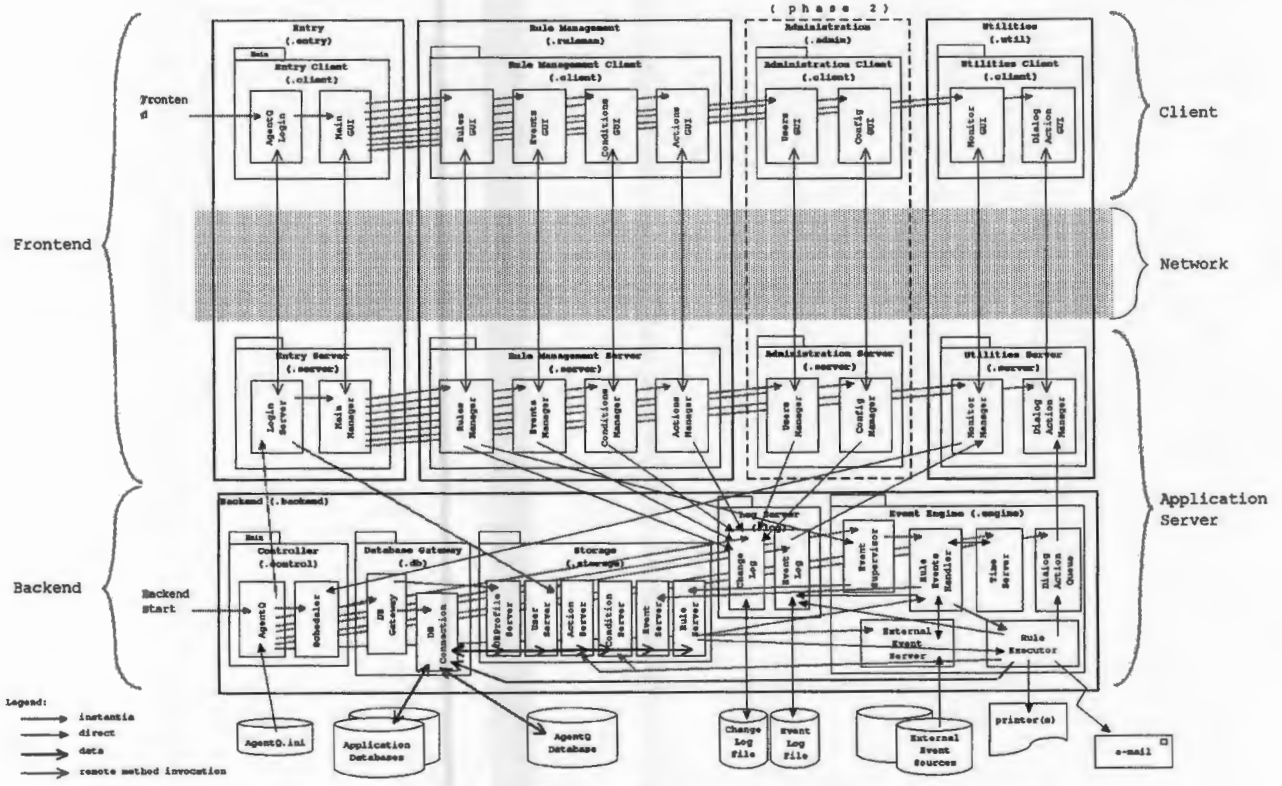

Abbildung 3: Systemarchitektur, Verteilung und Kommunikationsbeziehungen

Events werden in

- interne Events (durch Erreichen eines Zeitpunktes)
- externe Events (durch Interaktion oder Vorhandensein von Daten an einer definierten Schnittstelle)

eingeteilt und weitergehend danach unterschieden, ob sie nur einmalig oder nach einmaliger Aktivierung zyklisch wiederholt aktiv werden.

Aktionen beschreiben die Möglichkeiten des Systems, Ergebnisse aufzubereiten und weiterzugeben. Definiert sind zunächst:

- Bericht erzeugen und persistent speichern
- Batch-Datei starten
- Meldungen - zu unterscheiden sind hierbei optische (Dialog-) und akustische (Alarm-) Meldungen (deren Auswertung ist nur gegeben, wenn ein adressierter Client gerade aktiv ist, ansonsten gehen sie ohne weitere Behandlung verloren)
- Generieren einer ASCII-Ausgabedatei als Schnittstelle zu anderen IT-Systemen
- Bericht per E-Mail versenden
- Nachricht via Paging-Dienst versenden
- Tabelle mit Kopien der auffälligen Daten generieren

Die Möglichkeit Parameter zwischen den Regelbestandteilen zu übergeben, wurde vorgesehen, um eine erhöhte Flexibilität zu erreichen. Ein solcher Mechanismus ermöglicht z.B., in einer Eingabedatei, die einen externen Event auslöst, sowohl Parameter für die durchzuführende Untersuchung in der betroffenen Applikationsdatenbank zu spezifizieren, als auch Verarbeitungsschritte und Adressat für die resultierende Aktion anzugeben.

4 Realisierung

4.1 Objektorientierter Entwurf mit UML (Unified Modeling Language)

Bei der Unified Modeling Language (UML) [Fo97] handelt es sich um eine objektorientierte Modellierungssprache, die aus den bekannten Ansätzen von Grady Booch (OOD), Jim Rumbaugh (OMT) und Ivar Jacobson (OOSE) hervorgegangen ist. UML ist geeignet große und komplexe Systeme zu modellieren.

Für den Systementwurf wurden die Darstellungsformen:

- Klassendiagramme
- Klassenbezugsdiagramme
- Klassenbeschreibungen
- Anwendungsfälle und
- Ablaufdiagramme

von UML ausgewählt und unter Nutzung des Entwurfswerkzeuges Rational Rose durchgehend für Entwurf und Dokumentation verwendet. Zunächst wurde dieses Entwurfssystem nicht zur Codegenerierung eingesetzt, in seiner neuesten Version für Java verfügt es aber über die Fähigkeit zum Reverse-Engineering vorhandenen Codes. Die Nutzung dieser Möglichkeit ist im Rahmen der Weiterentwicklung der Anwendung vorgesehen.

4.2 Umsetzung mit Java

Die Programmiersprache Java

Java stellt eine mächtige und dennoch relativ einfach zu erlernende und effiziente objektorientierte Programmiersprache dar, die nach ihrer breiten Vorstellung durch Sun Microsystems schnelle Verbreitung gefunden hat und deren Anwendung keinesfalls auf die Programmierung von Applets zur Animation von Web-Seiten beschränkt ist.

Die Programmiersprache zeichnet sich einerseits durch Eigenschaften wie strenge Objektorientierung, Einfachheit, Robustheit, Sicherheit, Plattform-Neutralität, Portabilität und die inherente Unterstützung von Multithreading aus und profitiert andererseits vom Vorhandensein einer umfangreichen und mächtigen Basis-API (System-, Utility- und Container-Klassen, portable Unterstützung zur Programmierung von graphischen Nutzer-Interfaces mittels eines Abstract Window Toolkits (AWT), Netzwerkunterstützung für Verteilte Anwendungen, Remote Method Invocation (RMI) als Standard für die Kommunikation zwischen verteilten Java Anwendungen, standardisierter Datenbank-Zugriff über JDBC ...).

Bei der Übersetzung eines Java Programmes wird portabler Bytecode generiert, der von einer Java Virtual Machine (VM), die auf den verschiedensten Hard- und Software-Plattformen implementiert werden kann, abgearbeitet wird. Mit Java können sowohl eigenständig ablaufende Programme, die als Applikationen bezeichnet werden, als auch Applets, die in einer in die Laufzeitumgebung eines WWW-Browsers integrierten VM zur Ausführung kommen, geschrieben werden. Für Applets gelten aus Sicherheitsgründen Restriktionen hinsichtlich des Zugriffes auf die Resourcen des Rechners, auf denen sie ausgeführt werden aber auch hinsichtlich der ihnen gestatteten Netzwerkverbindungen. Diese Einschränkungen sind in der Implementierung des verwendeten SecurityManagers begründet und können z.B. für sogenannte "signed and trusted" Applets auch bewußt gelockert werden. Die Realisierung eines Programmes als Applet ist wegen der Möglichkeit dieses vollständig und ohne vorherige Verteilung von Software (mit Ausnahme des Browsers) über ein Netzwerk zu laden, gerade bei einer großen Anzahl potentieller Anwender oder kurzen Programmänderungszyklen attraktiv und sollte jeweils in Betracht gezogen werden.

JDBC als Möglichkeit zum Zugriff auf unterschiedliche Datenbanken

Mit JDBC ist eine Möglichkeit realisiert, SQL Datenbanken über ein standardisiertes Interface aus Java Programmen zu nutzen. Eine Java Anwendung kann dazu einen oder mehrere JDBC Treiber beim JDBC Driver Manager registrieren und anschließend mittels einer URL spezifizieren, zu welcher Datenbank und unter Nutzung welcher Mechanismen eine Verbindung aufgebaut werden soll. Dabei wird eine Zugriffskontrolle durch die Möglichkeit der Angabe eines Paßwortes beim Verbindungsaufbau unterstützt und kann genutzt werden, falls das angesprochene DBMS einen entsprechenden Zugangsschutzmechanismus besitzt.

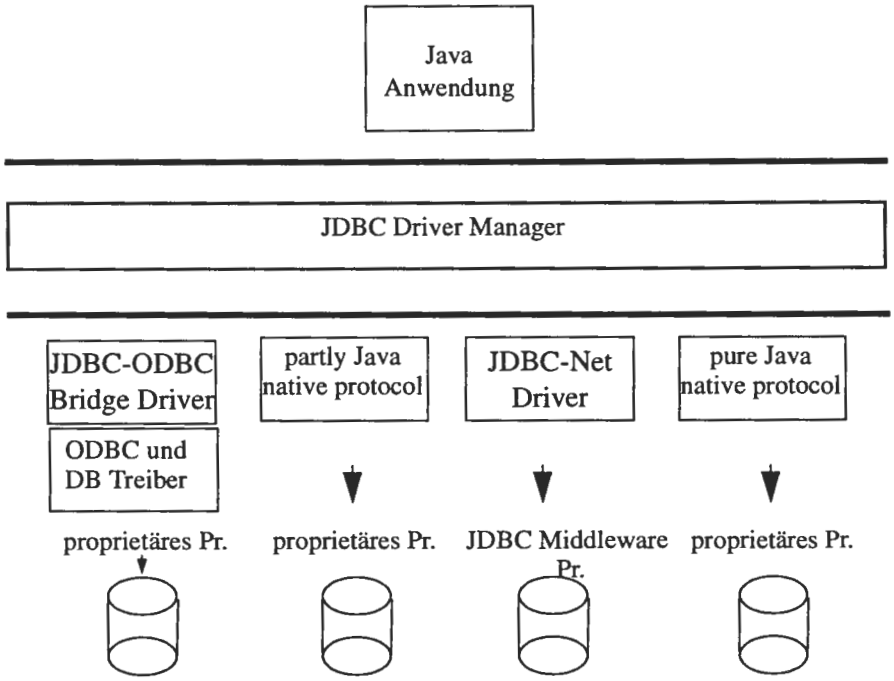

Abbildung 4: JDBC Architektur und Klassifizierung von JDBC Treibern [HaCaFi97]

Während Typ I Treiber mit einer JDBC-ODBC-Bridge und dem Einsatz von ODBC zum Datenbank-Zugriff nur eine ineffiziente und bis zur breiten Etablierung von JDBC-Schnittstellen teilweise noch notwendige Übergangslösung darstellen und Typ II "Native API partly Java" Treiber durch die Notwendigkeit des Ansprechens von native Code für Applets nicht anwendbar sind, kamen im Projekt Typ III bzw. Typ IV Treiber zum Einsatz. Diese vollständig in Java realisierten und damit auch dynamisch in Applets ladbaren Treiber bilden die JDBC Methodenaufrufe entweder auf ein generisches und DBMS-unabhängiges Netzwerk-Protokoll ab, welches beim DB-Server oder einem Gateway in das DB-Zugriffsprotokoll umgesetzt wird oder sprechen das DBMS unmittelbar über dessen Netzwerk-Protokoll an.

Eine Anzahl von Anbietern realisiert Gateway Lösungen wie z.B. DBAnywhere oder OmniGate, die eine Reihe von DB Schnittstellen transparent vor dem Anwender verbergen. Multi-tier Lösungen und der Einsatz von Middleware werden wegen ihrer guten Skalierbarkeit und der Möglichkeit zur Realisierung weiterer Funktionalität wie Zugriffskontrolle, dynamischer und transparenter Lastverteilung, erhöhter Ausfallsicherheit durch Redundanzeinbringung, effizient möglicher Behandlung in Firewall-Systemen oder auch Verschlüsselung auf der Transport-Ebene als überaus mächtig und zukunftsträchtig klassifiziert [OrHa97].

Neben der Bereitstellung eines einheitlichen Interfaces zum Zugriff auf verschiedenste Datenbanken, der Möglichkeit zur Formulierung von SQL-Abfragen und des Mappings der Abfrageresultate auf Java Datentypen bietet JDBC mit der standardisierten Möglichkeit zur Abfrage von Metainformationen über die DB-Inhalte einen weiteren sehr leistungsfähigen Dienst. JDBC gibt die Ergebnisse einer Abfrage in einem generischen ResultSet zurück. Dieses besitzt eine Methode zur Generierung eines ResultSetMetaData Objektes, das über eine Reihe von Methoden zur Bestimmung der Charakteristika der Rückgabedaten verfügt. Auf diese Art und Weise wird einerseits die für Java charakteristische strenge Typisierung realisiert, andererseits können Abfrageergebnisse zur Programmlaufzeit flexibel weiterverarbeitet werden. Dies unterscheidet die JDBC Schnittstelle von anderen Programmsystemen, die zum Zeitpunkt der Programmerstellung bereits explizites Wissen über das Layout der behandelten Tabellen der angesprochenen Datenbanken benötigen und daher weitaus weniger flexibel sind. Insbesondere im vorgesehenen Einsatzumfeld von Agent-Q mit der Notwendigkeit zu schneller Anpassbarkeit an die Spezifika der zu überprüfenden Datenbanken spielt dies eine wichtige Rolle.

Die Realisierung der Benutzerschnittstelle

Die Realisierung der graphischen Bedienoberfläche der Client-Systeme erfolgte entsprechend Abbildung 3. unter Anwendung des Model-View-Controler (MVC) Modells [Po96] mit Thin-Clients. Diese realisieren nur die eigentliche Interaktionskomponente (view) und sprechen für den Zugriff auf die Datenbank mittels RMI eine auf dem Server laufende korrespondierende Komponente (model) an. Das von Xerox PARC ursprünglich für die Generierung von graphischen Bedienoberflächen mit Smalltalk entwickelte Design Pattern eignet sich sehr gut für die Umsetzung der Anforderungen mehrerer gleichzeitig aktiver Klienten in einer verteilten Anwendung.

Die zwischenzeitlich praktizierte Generierung des GUI-Interfaces mittels der vom Programmiersystem Visual Cafe angebotenen DBAware-Methode, bei der Datenbankzugriffe vom Programmierwerkzeug bei der visuell unterstützten Oberflächenerstellung direkt im erzeugten Client-Code eingebaut werden, führt einerseits zu einer Verletzung der multi-tier Architektur und wurde andererseits sowohl von unserem Entwicklerteam als auch von Consultants der Firma Symantec als in erster Linie für Prototyp- und Adhoc-Anwendungen geringer Komplexität geeignet bewertet.

Die Programmierung der Oberfläche erfolgte daher individuell auf Basis des vom Abstract Window Toolkit und der Erweiterungen von Visual Cafe angebotenen Klassenumfanges. Während viele GUI-Bibliotheken für andere Programmiersprachen Oberflächen statisch generieren und die sichtbaren Komponenten fest positionieren, ermöglicht die Objektorientierung von Java in Kombination mit dem Einsatz von Layout-Managern flexiblere Lösungen, wie z.B. die Implementierung von generischen Browsern. [GeMcCl97]. Layout Manager, die in einem Basisumfang und mit unterschiedlicher Funktionalität und Mächtigkeit im AWT bereits vordefiniert sind, aber auch selbst implementiert werden können, verwalten darstellbare Komponenten und sind für deren Positionierung und Größe entsprechend vorgegebener Regeln und der zur Programmlaufzeit vorliegenden Bedingungen verantwortlich. So können z.B. in einem nur einmalig zu implementierenden Browser Attribute unterschiedlicher Anzahl und Charakteristik dargestellt werden. Fehlerträchtiges Kopieren von Code gleicher Funktionalität oder der Aufwand für die Änderung von GUI Layouts bei Hinzukommen oder Wegfall von Attributen kann bei Kenntnis und bewußtem Einsatz dieser Möglichkeit vermieden werden.

Im Rahmen der Implementierung von Agent-Q entstand eine Reihe von für andere Applikationen nutzbaren Klassen und Anwendungsszenarien. Auch ist zukünftig mit einer Erweiterung des Umfanges von Klassenbibliotheken für spezifische Oberflächenkomponenten zu rechnen. Für die Weiterführung des Projektes wird die

Benutzung der JavaBean Komponentenarchitektur und der swing Klassen erwogen.

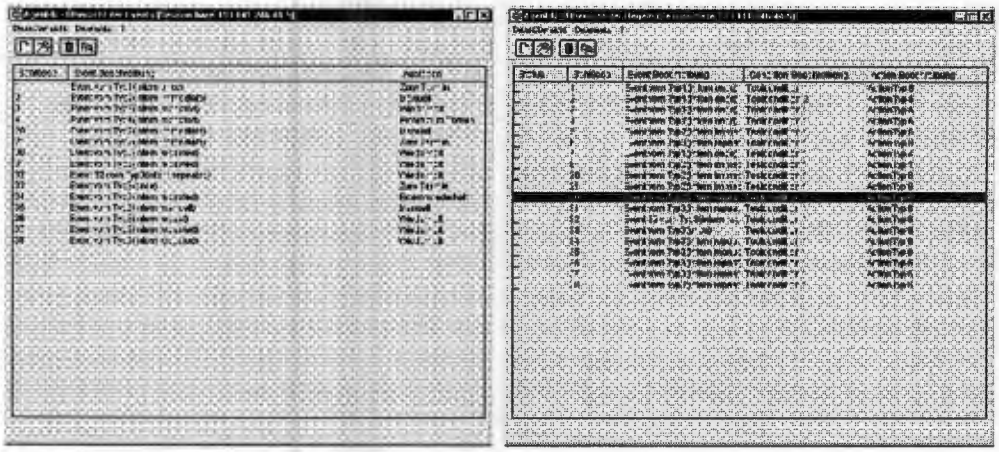

Abbildung 5: Layoutmanager ermöglichen generische dynamische Browser

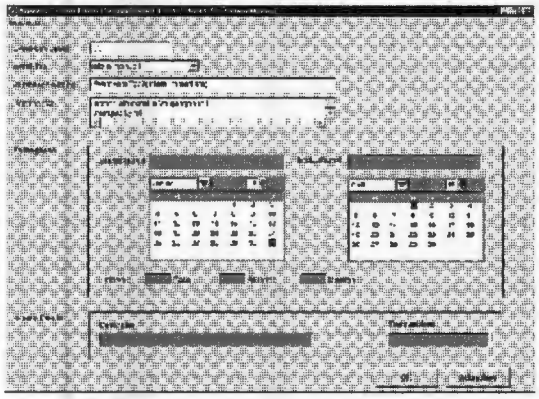

Abbildung 6: Beispieloberfläche mit Visual Cafe Erweiterung (Kalender)

Meldungen, Label und sonstige Texte im Programm wurden nicht hart codiert, sondern benutzen Resourcen. Wird eine entsprechende Resource Datei mit je nach Sprache angepaßtem Inhalt zur Verfügung gestellt, dann ist eine einfache Internationalisierung der Anwendung möglich.

Java-Threads

Multithreading auf Sprachebene und die Unterstützung von Primitiven zur Synchronisation stellen eine wichtige Eigenschaft von Java dar. Sie ermöglichen eine Reihe von Anwendungen in sehr effizienter und einfacher Umsetzung durch den Programmierer, ohne daß dieser wie z.B. in C Programmen spezielle Systemrufe des unterliegenden Betriebssystemes nutzen muß. Threads werden von Agent-Q hauptsächlich im Backend für die Event-Bearbeitung genutzt. Jedem eine Regel auslösenden Event wird zur Abarbeitung ein separater Thread zugeordnet. Threads sind z.B. auch anwendbar, um JDBC Anfragen vorzeitig zu beenden , falls diese nach einer vorzugebenden Zeitspanne noch keine Ergebnisse geliefert haben. Weiterhin erlauben sie die elegante Realisierung asynchroner Callbacks zur Benachrichtigung der Clients und heben die von herkömmlichen Systemen bekannte klare Trennung Client oder Server auf. So kann ein Client in einem von ihm erzeugten Thread einen Netzwerksocket öffnen und seinerseits Server-Funktionen übernehmen.

Java-Schnittstellen zur Einbindung in heterogene Umgebungen

Java bietet neben dem bereits beschriebenen Datenbank-Zugriff über JDBC und der Kommunikation zwischen Java Objekten über RMI eine breite Auswahl von Möglichkeiten zur Kommunikation mit anderen Komponenten.

So ist es in Applikationen möglich, externe Programme aufzurufen oder auch in anderen Programmiersprachen realisierten Native Code in die Java-Laufzeitumgebung einzubinden. Sowohl Applikationen als auch Applets können über in 100% pure java realisierten ONC Remote Procedure Call (RPC) oder CORBA IIOP Schnittstellen Daten mit anderen Anwendungen austauschen. Im Rahmen von Agent-Q wurde eine E-Mail Schnittstelle sowohl zum Versand von Nachrichten an konventionelle Mail-Empfänger als auch an Paging-Systeme (z.B. Scall, SMS, Quix ...) implementiert. Dies erfolgte durch explizite Argumentübergabe an einen auf dem Server installierten Mail User Agent, es ist aber auch das Ansprechen eines auf einem anderen System laufenden Mail Transfer Agents mit Nutzung des mittels der Socket API realisierbaren Simple Mail Transfer Protocol (SMTP) möglich.

Erfahrungen bei der Programmentwicklung

Für die Programmentwicklung wurde Visual Cafe und Java 1.1. eingesetzt. Die Stabilität und Benutzbarkeit der Tools wurde allgemein als gut eingeschätzt. Durchgängig wurde die In-line Dokumentation des Codes mit javadoc Kommentaren genutzt. Aus diesen kann automatisiert eine Hypertext HTML Dokumentation der programmierten Klassen generiert werden.

Insgesamt kann eingeschätzt werden, daß Java sehr gut für den beschriebenen Einsatzzweck geeignet ist.

5 Einsatz der Anwendung und Ausblick

Als eine vieler denkbarer Beispiel-Anwendungen wird die Kontrolle sich ändernder Kundendaten in einem Sze-

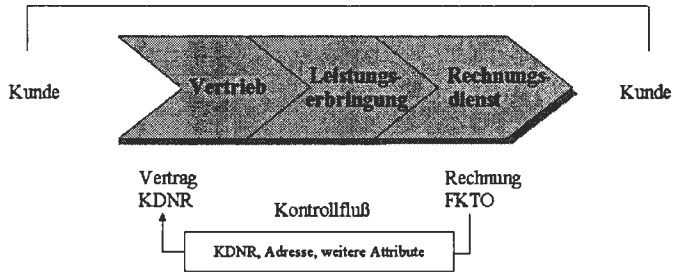

Abbildung 7: Beispielanwendung zur Verifikation geänderter Daten

nario, welches diese Kundendaten für 3 verschiedene Anwendungen jeweils in disjunkten Datenbanken verwaltet, aktuell konzipiert und getestet. Während die Aufgabe des Data Scleaning/Scrubbing des gesamten Datenbestandes in regelmäßigen Abständen durch ein leistungsfähiges Werkzeug mit Fähigkeiten zum unscharfen und phonetischen Vergleich realisiert ist, wird für die Kontrolle sich ändernder Kundendaten Agent-Q eingesetzt. In dem beschriebenen Szenario kann davon ausgegangen werden, daß sich ändernde Kundendaten zunächst in den Datenbanken des Vertriebes und der Leistungserbringung aktuallisiert werden. Wird bei einer Veränderung von Daten in der Datenbank des Rechnungsdienstes durch Agent-Q eine Inkonsistenz festgestellt, so löst dies eine Benachrichtigung aus. Eine unmittelbare Manipulation des Datenbestandes durch das System selbst ist zunächst nicht vorgesehen.

Neben der technischen Weiterentwicklung der Anwendung spielt die Untersuchung und Einführung von Mechanismen zum Qualitätsmanagement eine zentrale Rolle für den Einsatz der entstandenen Lösung. Diese beinhalten die Definition von Dimensionen und Metriken zur Datenqualität und den Einsatz von Rule-Discovery Produkten, die über die intuitiv aus der Kenntnis der bearbeiteten Prozesse formulierten Regeln hinaus versuchen, Regeln analytisch festzulegen, zu wichten und für den Einsatz zu selektieren.

Referenzen

[CoTü95] S. Conrad, C. Türker, "Active Integrity Maintenance in Federated Database Systems", Preprint Nr. 9, Fakultät für Informatik, Universität Magdeburg 1995

[DiGa96] K. R. Dittrich, S. Gatziu, "Aktive Datenbanksysteme - Konzepte und Mechanismen", Thomson's Aktuelle Tutorien, Thomson Publishing, 1996

[Fo97] M. Fowler, "UML Distilled - Applying the Standard Object Modelling Language", Addison-Wesley, 1997

[GeMcCl97] D. M. Geary, A. L. McClellan, "Graphic Java - Mastering the AWT", SunSoft Press, Prentice Hall, 1997

[HaCaFi97] G. Hamilton, R. Cattell, M. Fisher "JDBC Database Access with Java", The Java Series, Addison-Wesley, 1997

[OrHa97] R. Orfali, D. Harkey, "Client/Server Programming with Java and CORBA", Wiley Publishing, 1997

[Po96] M. Potel, "MVP: Model-View-Presenter - The Taligent Programming Model for C++ and Java", 1996 (http://www.ibm.com/java/education/mvp.html)

[Re96] T. C. Redman, "Data Quality for the Information Age", Artech House, 1996

[ShLa90] A. P. Sheth, J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogenous and Autonomous Databases", ACM Computing Surveys, 22(3):183-236, September1990