SurfBorD
Systemunabhängig realisierte flexible Bedienoberfläche für relationale Datenbanken

Dirk Fischer, Heike Utermann

Lehrstuhl für Datenverarbeitung
Technische Universität München
80290 München
fischer@ei.tum.de
utermann@ei.tum.de

Zusammenfassung: Der Wechsel auf ein anderes DBMS (Datenbank-Management-System) oder das Erweitern einer Datenbank ist in der Regel mit viel Nachfolgearbeit verbunden, da auch die verschiedenen Anwendungen angepaßt werden müssen, die auf die Datenbank zugreifen. Mit SurfBorD wurde eine Anwendung in Java entwickelt, die es ermöglicht, beliebige Datenbanken abzufragen oder zu bearbeiten. Dabei ist es gleichgültig, von welchem Hersteller das DBMS stammt und auf welcher Plattform SurfBorD eingesetzt wird. Durch dynamisch ladbare Komponenten kann SurfBorD beliebig um neue Analyse- und Manipulationswerkzeuge erweitert werden.

1 Einleitung

Relationale Datenbanken sind heutzutage das gängige Mittel, um große Datenmengen strukturiert zu speichern und zu verwalten. Um das Potential der Datenbanken effektiv nutzen zu können, muß eine Möglichkeit zum Bearbeiten und zum Neueintrag von Daten vorhanden sein, welche einfach zu bedienen ist und keine detaillierten Kenntnisse über die Datenbankstruktur voraussetzt.

Es existieren viele Anwendungen, die einen transparenten und benutzerfreundlichen Zugriff auf Datenbanken bieten. Jedoch sind diese Lösungen häufig proprietär und nicht auf andere Anwendungsfälle übertragbar. Daneben gibt es den im Kernbereich standardisierten Zugriff mit Hilfe der Datenbankabfragesprache SQL (Structured Query Language). Eine Verwendung dieser Zugriffsart setzt jedoch detaillierte Kenntnisse der verwendeten Datenbankstruktur und eine größere Einarbeitungsphase voraus, so daß sie kein probates Mittel für den Endanwender darstellt.

Das Ziel war, die wesentlichen Eigenschaften der verschiedenen Zugriffsmöglichkeiten zu verbinden und eine Bedienoberfläche zu entwickeln, die unabhängig von der zugrundeliegenden Plattform benutzt werden kann. Die für den Zugriff notwendige Information soll aus der jeweiligen Datenbank erhältlich sein, so daß

keine Anpassung an der Anwendung vorgenommen werden muß, falls die Struktur der Datenbank geändert wird. Außerdem sollen möglichst viele Datenbank-Management-Systeme (DBMS) unterstützt werden. Gleichzeitig wird eine einfache und übersichtliche Bedienoberfläche angestrebt.

2 Zugriffsmöglichkeiten auf relationale Datenbanken

Die verbreitetsten Hilfsmittel, dem Benutzer eine Anwendung zum Befüllen und Auswerten einer Datenbank bereitzustellen, sind Maskengeneratoren und Programmbibliotheken. Maskengeneratoren ermöglichen es, auf einfache Weise Applikationen mit zweckmäßigen GUIs (Graphical User Interfaces) zu erstellen. Programmbibliotheken erlauben es einem Entwickler, alle Freiheiten seiner bevorzugten Programmiersprache zu nutzen und zusätzlich mit Hilfe von einfach zu gebrauchenden Funktionen auf Datenbanken zugreifen zu können. Diese Methoden sind für die meisten Betriebssysteme verfügbar, jedoch müssen die so erstellten Programme für jede zu verwendende Plattform neu angepaßt werden, was mit nicht unerheblichem Aufwand verbunden ist.

Zudem unterscheiden sich die Programmierstile bei Anwendungen (prozedural, objektorientiert, ...) und Datenbanken (SQL, 4GL) deutlich, so daß ein Entwickler grundlegendes Verständnis von beiden Bereichen haben muß. Durch die Verwendung von ESQL (Embedded-SQL) wird zumindest erreicht, daß die Programmquellen prägnanter formuliert werden können und für Menschen verständlicher zu lesen sind als bei direkten Bibliotheksaufrufen: Datenbankanweisungen werden direkt in SQL angegeben, wobei die Möglichkeit bereitgestellt wird, Programmvariablen in den SQL-Anweisungen zu verwenden. Auch wird das Anpassen an andere DBMS deutlich vereinfacht, da der Entwickler lediglich den passenden Pre-Compiler verwenden muß, um die ESQL-Anweisungen durch bestimmte Bibliotheksaufrufe zu ersetzen. Solange die ESQL-Anweisungen und die verwendeten Pre-Compiler dem Standard (ANSI/ISO) entsprechen, sollten keine wesentlichen Änderungen an den Programmquellen erforderlich sein. Allerdings muß das Programm neu übersetzt werden, bevor es ein anderes DBMS nutzen kann oder auf einer anderen Plattform eingesetzt werden kann.

Als alternative Zugriffsmethode bietet sich die Kombination von HTML (Hypertext Markup Language) mit CGI-Skripten (Common Gateway Interface) an, die eine Anbindung an das WWW und ein einfaches Erstellen von statischen Bedienoberflächen ermöglichen. Jedoch bleibt die Formularverwaltung von HTML weit hinter dem zurück, was man von den ursprünglichen Bedienoberflächen gewohnt ist.

Ansätze, die auf verteilten Objekten, wie z.B. CORBA oder DCOM, basieren, ermöglichen leistungsfähige verteilte Systeme. Jedoch sind die einzelnen Objekte im allgemeinen direkt abhängig vom verwendeten Betriebssystem und von der Rechnerarchitektur, auf der sie ausgeführt werden sollen. Damit ist zumindest eine erneute Übersetzung der Quelltexte erforderlich, falls ein Objekt auf einer anderen Plattform betrieben werden soll.

In Java erstellte Programme sollten diese Einschränkung nicht aufweisen. Die JVM (Java Virtual Machine) ist für die verbreitetsten Rechnerarchitekturen erhältlich und einmal erstellte Java-Programme können auf jeder dieser JVMs gestartet werden. Das objektorientierte Paradigma (OOP) von Java und die vielfältigen, frei verfügbaren Klassenbibliotheken unterstützen die Entwicklung flexibel einsetzbarer Software.

3 Realisierung der Zielvorgaben

Die Bedeutung einer einfachen Datenbankanbindung für das Erstellen von betrieblich genutzter Software wurde in den letzten Jahren immer deutlicher. Dieser Trend hat auch die Weiterentwicklung von Java beeinflußt und hat zu der Definition von JDBC (Java Database Connectivity) [3] geführt. Im folgenden soll analysiert werden, wie die Eigenschaften von Java und JDBC genutzt werden können, um die anfangs definierte Zielvorgabe zu realisieren.

3.1 Plattformunabhängige Bedienoberfläche

Um das mehrfache Entwickeln der Bedienoberflächen für verschiedene Plattformen in einem heterogenen Rechnernetz zu vermeiden, ist es erforderlich, die entsprechenden Anwendungen von der jeweiligen Plattform abzukoppeln.

Ein Weg, der große Flexibilität verspricht, ist die Implementierung der Anwendung in Java. Die virtuelle Maschine von Java stellt eine Schicht dar, die direkt auf der Architektur der verwendeten Plattform aufsetzt und die Java-Programmen eine einheitliche Laufzeitumgebung zur Verfügung stellt. Damit wird die geforderte Trennung der Anwendung von der Plattform erreicht und Java-Programme können auf allen Rechnern eingesetzt werden, auf denen die virtuelle Java-Maschine installiert wurde. Damit ist ein Einsatz auf den verbreiteten Plattformen (UNIX, WinNT, Win95, ...) in der vorgesehenen Weise möglich.

3.2 Universeller Zugang zu Datenbanken verschiedener Hersteller

Der Nutzen einer Datenbank nimmt deutlich zu, wenn es möglich ist, benutzerfreundliche Anwendungen zu entwickeln, die Daten aus der Datenbank auslesen können, um sie in aufbereiteter Form zu präsentieren. Diese Idee wird auch von den Herstellern von Datenbanken unterstützt, indem Programmbibliotheken und Maskengeneratoren angeboten werden, die das Entwickeln dieser Anwendungen vereinfachen. Diese Werkzeuge sind jedoch speziell auf die Datenbank des jeweiligen Herstellers zugeschnitten.

Mit JDBC wurde eine Architektur geschaffen, die sich an ODBC (Open Database Connectivity) [7] anlehnt und wie diese auf vier Ebenen basiert (Anwendung, JDBC Driver Manager, JDBC-Treiber, Datenbank). Damit ist es möglich, Anwendungen für beliebige Datenbanken zu entwickeln. Der Datenbank-Hersteller bietet zu seiner Datenbank einen JDBC-Treiber an, der die Funktionen der Datenbank auf eine standardisierte JDBC-API abbildet.

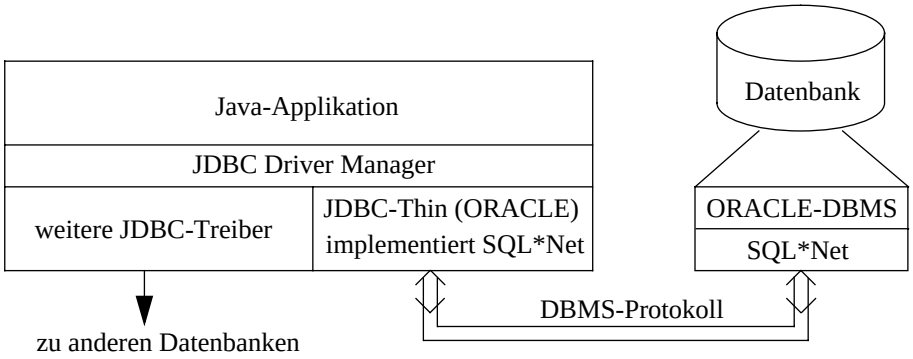

Abb. 1. Kommunikationsmodell von JDBC am Beispiel eines ORACLE-DBMS

Der JDBC-Driver-Manager identifiziert die einzelnen Datenbanken über einen eindeutigen URL (Uniform Resource Locator), der sowohl das verwendete Subprotokoll als auch die Bezeichnung der Datenquelle enthält. So kennzeichnet der URL „jdbc:oracle:thin:@ora.world" beispielsweise eine bestimmte Oracle-Datenbank. Bei einem Verbindungswunsch der Anwendung testet der Driver-Manager jeden verfügbaren Treiber, ob er eine Verbindung zur angegebenen Datenbank aufbauen kann. Der erste passende wird benutzt (Abb. 1).

Die erhältlichen JDBC-Treiber werden in vier Kategorien [4] unterteilt: Treiber der Kategorie 1 und 2 greifen auf Bibliotheken mit nativem C-Code zurück und nur die Kategorien 3 und 4 definieren rein in Java geschriebene Treiber. Um die gewünschte Plattformunabhängigkeit zu bewahren, ist es also erforderlich, sich auf JDBC-Treiber der Kategorie 3 oder 4 zu beschränken.

Mit JDBC-2.0 [5] werden Erweiterungen eingeführt, die einige Einschränkungen der ersten Version aufheben und den zunehmenden Funktionsumfang von DBMS berücksichtigen (BLOBS, SQL-3,...). Zudem werden Mittel bereitgestellt, die zu einer Leistungssteigerung bei JDBC-Anwendungen führen können, wie z.B. das Zusammenfassen von mehreren Transaktionen. Grundsätzlich hängt die Leistungsfähigkeit einer JDBC-Anwendung von vielen variablen Faktoren ab. Neben dem Einfluß der Hardware und des verwendeten DBMS hängt der Datendurchsatz, wie bei allen Java-Anwendungen, auch von der eingesetzten JVM ab. Durch die Verwendung von JITs (Just In Time Compiler) können sich Java-Anwendungen der Ausführungsgeschwindigkeit entsprechender Anwendungen annähern, die in Maschinencode vorliegen und direkt vom Prozessor ausgeführt werden. Zukünftige Entwicklungen (HotSpot [6]) sollen diesen Nachteil weiter verringern.

Durch den sinnvollen Einsatz der bereitgestellten Mittel kann auch der Entwickler die Leistungsfähigkeit der erstellten Anwendung beeinflussen. So bietet JDBC grundsätzlich drei Varianten an, um Datenbankanweisungen auszuführen: Neben den einfachen JDBC-Statements für einzelne Datenbankabfragen gibt es auch sogenannte Prepared Statements, die eine Anweisung übersetzen, um sie daraufhin beliebig oft mit unterschiedlichen Parametern aufrufen zu können.

Besonders bei gleichartigen Anfragen, die häufig vorkommen, bewirken Prepared Statements eine Beschleunigung des Programmablaufs. Die dritte Variante, Callable Statements genannt, macht von der Eigenschaft vieler DBMS Gebrauch, Anweisungen in Form von Stored Procedures direkt im DBMS ausführen zu können. In bestimmten Fällen kann damit eine weitere Beschleunigung erreicht werden.

Aufbauend auf JDBC gibt es Entwicklungen, die den Zugriff von Java-Anwendungen auf relationale Datenbanken weiter vereinfachen. So gibt es Erweiterungen, die Embedded-SQL auch unter Java ermöglichen (JSQL [8]) oder die eine Abbildung relationaler Datenbank-Tabellen auf Java-Klassen vornehmen, so daß der Programmierer überhaupt kein SQL kennen muß [9].

3.3 Automatische Anpassung an die Struktur der Datenbank

Die übliche Vorgehensweise beim Erstellen von Bedienoberflächen für Datenbanken führt häufig zu einer doppelten Definition der Datenbankstruktur:

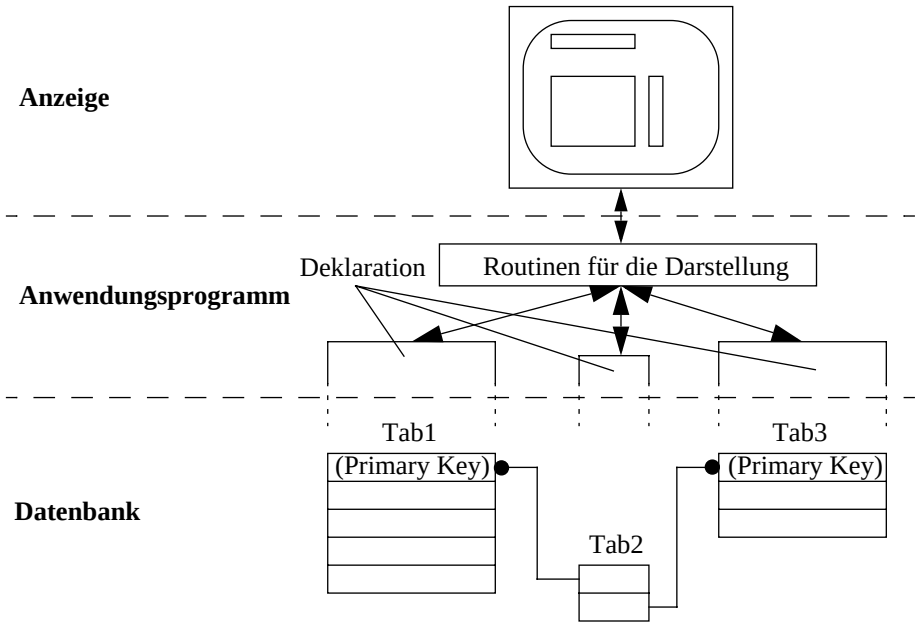

Abb. 2. Definition der Datenbankstruktur in Anwendung und Datenbank

Zunächst muß der DBA (Datenbankadministrator) die Struktur in Form einer DDL (Data Definition Language) festlegen und den einzelnen Anwendern Zugriffsrechte mit Hilfe einer DCL (Data Control Language) zuordnen. Die gleiche Struktur wird jedoch ein zweitesmal in den Programmen deklariert, über die die Benutzer auf den Datenbestand zugreifen können sollen (Abb. 2).

Der wesentliche Nachteil dieser Vorgehensweise wird deutlich, wenn die Struktur der zugrundeliegenden Datenbank geändert werden muß:

- Die Bedienoberflächen auf den verschiedenen Plattformen müssen ebenfalls geändert werden.
- Durch den größeren Zeitaufwand wird die neue Datenbank erst später verfügbar.
- Es muß sichergestellt sein, daß keine Bedienoberfläche im Rechnernetz versehentlich auf dem alten Stand bleibt.

Jede Datenbank enthält sogenannte Metadaten, die Information über die Struktur der Datenbank liefern. JDBC bietet mit Hilfe spezieller Methoden einen einheitlichen Weg, die Metadaten der verschiedenen Datenbanken abzufragen. Eine Anwendung kann die Metadaten auslesen und die Darstellung der Nutzdaten automatisch anpassen (Abb. 3).

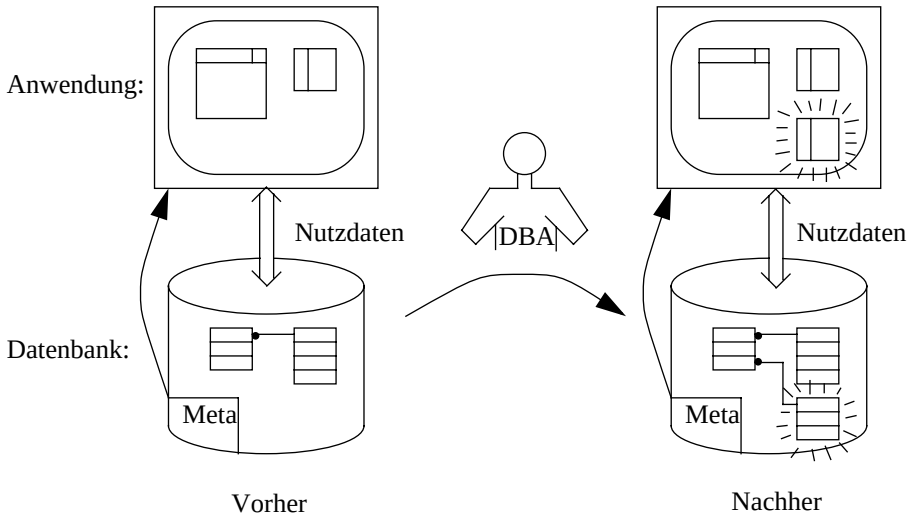

Abb. 3. Automatische Anpassung der Anwendung

Die Bedienoberfläche beschränkt sich automatisch auf die Daten, die der jeweilige Anwender auf Grund seiner Rechte und Privilegien benutzen darf. Wenn mehrere separate Datenbanken im Einsatz sind, kann der Anwender bequem zwischen den verschiedenen Datenquellen wechseln. Es genügt die Angabe des jeweiligen URLs, damit eine Verbindung zur angegebenen Datenquelle aufgebaut wird. Dabei ist es egal, auf welcher Plattform der Benutzer arbeitet, wie der Hersteller des DBMS lautet und welche Struktur in der Datenbank realisiert wurde. Die Bedienoberfläche paßt sich automatisch an.

Eine Beispielimplementierung, die von den aufgeführten Prinzipien Gebrauch macht, ist die Java-Applikation ,,SurfBorD". Die denkbaren Erweiterungen, die

in den folgenden Kapiteln beschrieben sind, haben sich in Teilimplementierungen als durchführbar erwiesen.

4 Weiterführende Konzepte

Die Grundidee war, größtmögliche Unabhängigkeit vom zugrundeliegenden Betriebssystem, dem jeweiligen Datenbanksystem und der datenbankinternen Tabellenstruktur zu erreichen. Dabei wird auf Daten und Mechanismen zurückgegriffen, die nahezu jede Datenbank standardmäßig anbietet. Im folgenden werden Konzepte vorgestellt, die auf zusätzlicher Information basieren, welche optional in einer Datenbank vorhanden sein können.

Die gewünschte Strukturunabhängigkeit bringt zunächst eine vorgegebene tabellarische Darstellung der verfügbaren Daten mit sich, da die tabelleninterne Semantik noch unbekannt ist. Einschränkungen dieser Art könnten mit einer in Tabellenform abgelegten Darstellungsempfehlung aufgehoben werden. Diese Information ist optional in der Datenbank verfügbar und wird von der Oberfläche gleichzeitig mit den Metadaten ausgelesen. Darauf basierend werden die Inhalte der Datenbank in einer, auf die speziellen Daten zugeschnittenen Form, dargestellt. Eine Visualisierung in Form verschiedener Diagrammtypen (z.B. Balken, Kurve, Tabelle) dient als Grundlage, aber auch eine komplexere Darstellung mit Multimediakomponenten (z.B. Video, Audio, 3-D-Graphik) soll möglich sein.

Die übliche Darstellung von Daten sind einfache Diagramme. Daher soll diese Art der Anzeige direkt in der Oberfläche implementiert sein. Für komplexere Darstellungsvarianten soll es dem jeweiligen Datenbankadministrator möglich sein, die dazu notwendigen Java-Klassen in der Datenbank abzulegen, so daß diese zur Laufzeit dynamisch als Komponenten in SurfBorD eingebunden werden können. Die für die Konfigurationsdaten verwendete Tabellenstruktur, die sogenannte SBMeta (SurfBorD-Metadaten), ist in Abbildung 4 dargestellt.

Zunächst muß eine Haupttabelle, die sogenannte PresentationTable angelegt werden, welche die allgemeine Darstellungsdefinition vorhält. Darin werden den Datenobjekten (Tabellen oder Views) Darstellungsobjekte (Kurven, 3D-Darstellungen, ...) und die zugehörigen Fenster zugeordnet. Über die Darstellungsschlüssel wird in einer weiteren Tabelle, der TypeTable, auf die jeweiligen Darstellungstypen verwiesen. Über den sogenannten Klassenindex wird auf eine dritte Tabelle referenziert, die ClassTable. In dieser wird, gegebenenfalls in einzelne Teile aufgespalten, die Definition der zu verwendenden Darstellungsklassen, die SurfBorD-Komponenten, als Byte-Code gespeichert.

Damit weiterhin die gewünschte Unabhängigkeit von der Oberfläche und der Datenbank besteht, muß die PresentationTable eindeutig identifizierbar sein. Das kann entweder durch die Vergabe eines standardisierten Namens erreicht werden oder der Oberfläche wird beim Start ein Parameter zur Identifikation dieser Haupttabelle übergeben. Ist keine PresentationTable in der betreffenden Datenbank verfügbar, so soll auf vorgegebene Darstellungen zurückgegriffen werden.

PresentationTable

Datenobjekt	Darstellungstyp	Parameter	Kommentar	Fenster
Tabelle1	Kurve	INDEX<...>VALUE<...>	---	1
Tabelle2	Balken	INDEX<...>VALUE<...>	‚Bemerkung‘	2
View3	3-D-Anim	INDEX<...> X<...> Y<...> Z<...>	‚Bemerkung‘	3

TypeTable

Darstellungstyp	Klassencodierung	Klassenindex
3-D-Anim	direct	7
Statistik	ascii	5
...	...	...

ClassTable

Klassenindex	Fragment-Nr.	Klassendefinition
7	1	<Byte Code>
7	2	<Byte Code>
5	1	<Byte Code>

Abb. 4. Tabellenstruktur der SBMeta

5 Realisierung der Erweiterung

Java hat die Eigenschaft, alle beteiligten Klassen erst laden zu müssen, wenn sie tatsächlich benötigt werden. Verzichtet ein Benutzer während der Ausführung eines Programms auf bestimmte Dienste des Programms, brauchen die betreffenden Klassen nicht geladen zu werden. Die Instanz, die dafür sorgt, daß benötigte Klassen automatisch nachgeladen werden, ist der sogenannte ClassLoader.

SurfBorD definiert einen eigenen ClassLoader, der es erlaubt, Java-Klassen bei Bedarf sowohl auf herkömmlichem Weg, z.B. bei Systemklassen, als auch aus der aktuell verbundenen Datenbank zu laden. Möchte ein Benutzer ein bestimmtes Datenobjekt betrachten oder verändern, stellt SurfBorD fest, in welcher Darstellungsform dieses Datenobjekt präsentiert werden soll und lädt gegebenfalls die entsprechende Klasse aus der Datenbank, um ein Darstellungsobjekt davon zu instantiieren.

Die Typtabelle kann jederzeit um neue Typen mit den zugehörigen Komponenten (Byte-Code der Java-Klasse) erweitert werden. Die Eigenschaften von Java erlauben es, daß diese Komponenten separat entwickelt werden können. Sobald der DBA eine neue Komponente in der Datenbank speichert, wird sie für alle SurfBorD-Benutzer sichtbar und kann sofort eingesetzt werden.

6 Konfiguration

Der Datenbankadministrator kann die Art der Darstellung einzelner Datenobjekte (Tabellen, Views,...) voreinstellen. Dazu gibt er eine entsprechende Darstellungsbeschreibung in SBMeta ein. Obwohl der DBA diese Formatierungsinformation auch manuell in die jeweilige SBMeta eintragen könnte, ist es zweckmäßig, hierfür ein Konfigurationswerkzeug einzusetzen. Damit hat der DBA die Möglichkeit, seine Vorstellungen von den verschiedenen Sichten auf die Daten, in übersichtlicher Form festzulegen und auf leichte Weise zu ändern. Wenn Gebrauch von den dynamischen Erweiterungsmöglichkeiten von SurfBorD, dem Nachladen von SurfBorD-Komponenten, gemacht werden soll, muß die Möglichkeit eingerichtet werden, diese Komponenten in der Datenbank abzulegen. Damit wird ein entsprechendes Konfigurationswerkzeug unverzichtbar. SBconfig ist eine abgewandelte Version von SurfBorD, die diese administrativen Mittel bereitstellt (Abb. 5).

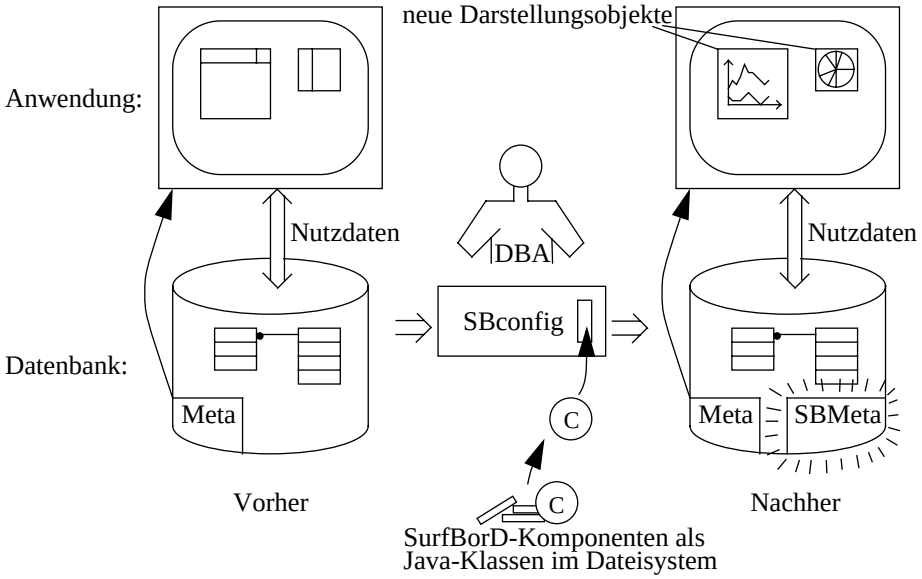

Abb. 5. Administration der Datendarstellung mit Hilfe von SBconfig

Das Programm legt die Tabellenstruktur für SBMeta an und befüllt sie nach den Vorgaben des Administrators. Über eine graphische Benutzerführung können den Datenobjekten der Datenbank beliebige Darstellungsobjekte zugeordnet werden, wobei die Darstellungseigenschaften der einzelnen Objekte interaktiv änderbar sind. Auch die Zuordnung mehrerer Darstellungsobjekte zu einzelnen Datenobjekten bzw. die Darstellung mehrerer Datenobjekte in einem Darstellungsobjekt ist möglich (Abb. 6). Außerdem können Java-Klassen aus dem

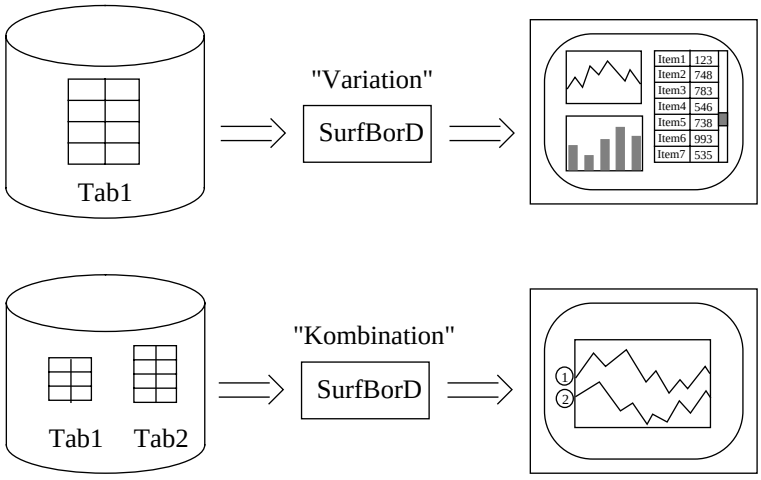

Abb. 6. Variation und Kombination der Darstellung

lokalen Dateisystem geladen und als dynamisch ladbare SurfBorD-Komponenten in der Datenbank gespeichert werden. Viele DBMS ermöglichen das direkte Speichern von Binärdaten, wie sie z.B. bei digitalisierten Bildern, Tönen usw. auftreten. In diesem Fall können die Java-Klassen direkt in der Datenbank gespeichert werden. Falls beliebige binäre Daten, z.B. BLOBs (Binary Large Objects), nicht unterstützt werden, verwendet SBconfig eine Codierung, die das Speichern von Java-Klassen in Textfeldern bzw. Zeichenketten ermöglicht. Gegebenenfalls können die Java-Klassen auch in mehrere Felder fragmentiert werden. Die Art der Codierung wird deshalb zusätzlich in SBMeta beschrieben.

7 Sicherheitsaspekte

Java-Programme, die in einen HTML-Browser geladen und dort gestartet werden können, sogenannte Applets, arbeiten unter der Kontrolle eines Security-Managers. Der Security-Manager ist eine Java-Klasse, die sich an alle Java-Funktionen anbindet, die sicherheitskritische Aufgaben erfüllen. Sobald eine dieser Funktionen aufgerufen wird, hat der Security-Manager die Möglichkeit den Zugriff auf die Ressource zu verweigern, indem er der Anwendung eine Ausnahmesituation meldet (Security Exception) und den Zugriffswunsch abbricht.

Die Java-Laufzeitumgebung erlaubt jedem Programm nur ein einziges Mal, sich einen Security-Manager einzurichten. Daher ist es nicht möglich, die Sicherheitspolitik nachträglich zu ändern. Die gebräuchlichen HTML-Browser verwenden einen Security-Manager, der es nicht zuläßt, daß sich ein Applet einen eigenen ClassLoader einrichtet. Ebenso ausgeschlossen ist eine Netzverbindung zu einem anderen Rechner als dem, von dem das Applet geladen wurde. Um alle Möglichkeiten nutzen zu können, die SurfBorD bietet, ist es deshalb momentan

noch erforderlich, SurfBorD als Applikation zu starten. Dadurch erhält SurfBorD alle Freiheiten, die das Betriebssystem auch dem Benutzer von SurfBorD eingeräumt hat.

Besteht der Wunsch, diese Freiheiten etwas einzuschränken, kann SurfBorD einen eigenen Security-Manager erhalten. Dieser Gedanke ist naheliegend, wenn berücksichtigt wird, das SurfBorD auch eine Komponente aus einer fremden Datenbank laden und ausführen könnte, die dort in der Absicht hinterlegt wurde, dem Benutzer zu schaden. Dazu kann bereits bei Programmstart über Kommandozeilenparameter oder auch zur Laufzeit über ein entsprechendes Konfigurationsmenü ein Security-Manager in Form einer Java-Klasse angegeben werden (Abb. 7).

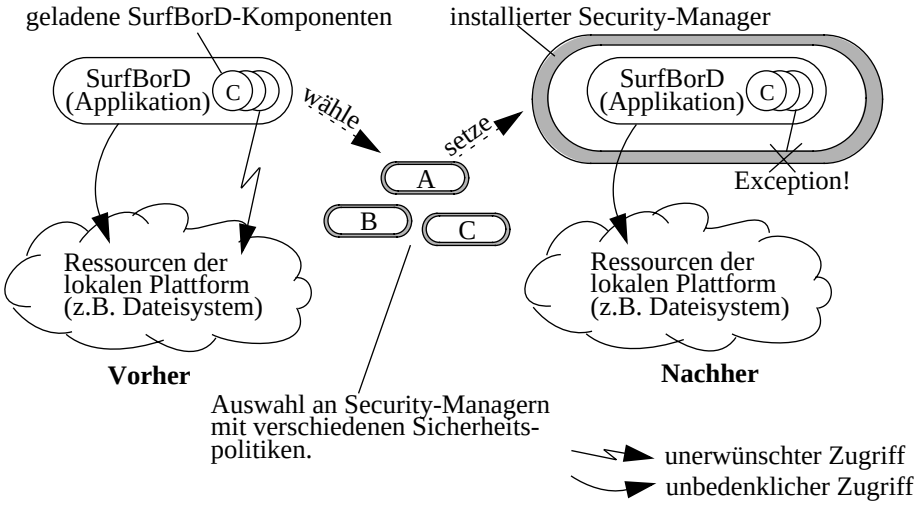

Abb. 7. Security-Manager

Eine andere, eventuell vertrauenswürdigere Alternative besteht darin, SurfBorD in einer sicheren Umgebung zu starten, die bereits einen Security-Manager vorgibt. Im Prinzip wird bei HTML-Browsern mit Java-Fähigkeit so verfahren, jedoch gibt es dort im allgemeinen keine Möglichkeit, eine eigene Sicherheitspolitik festzulegen. SurfBorD unterstützt dieses Vorgehen, indem es Einsprungpunkte implementiert hat, wie sie auch für Applets benötigt werden. So kann SurfBorD auch wie ein Applet in eine Umgebung mit bereits gesetztem Security-Manager geladen und dort gestartet werden.

Mit dem JDK-1.2 von SUN wird der Security-Manager durch einen sogenannten Access-Controller ersetzt, der es dem Endanwender ermöglicht, seine eigene Sicherheitspolitik umzusetzen und bestimmten Anwendungen gezielt Freiheiten einzuräumen. Damit erhält der Endanwender die Gewißheit, daß die Restriktionen, die er für SurfBorD definiert hat, tatsächlich eingehalten werden.

8 Ausblick

Die Parameter der Darstellungsarten in SBMeta geben nur Hinweise darüber, welche Attribute der einzelnen Datenobjekte bestimmte Bedeutungen für die Darstellung erhalten, wie z.B. Abszissen- und Ordinatenwerte einer Kurve. Eigenschaften, die nicht die Bedeutung der gespeicherten Daten betreffen, sondern die das konkrete Erscheinungsbild des jeweiligen Diagramms beeinflussen, wie z.B. Farbe und Signatur einer Kurve, werden bisher nicht in SBMeta gespeichert. Diese Eigenschaften sind als Attribute der Darstellungsobjekte ausgelegt, so daß jedes instantiierte Darstellungsobjekt separate Einstellungen erhalten kann. Dies entspricht der Vorgehensweise bei HTML-Dokumenten, wo ebenfalls an zentraler Stelle, beim Web-Server, die Strukturinformation zu einem Text gespeichert wird (Überschrift, Absatz, ...) und auf der Client-Seite definiert werden kann, wie die konkrete Darstellung aussehen soll (Schriftgröße, Farbe, ...).

Damit die persönliche Konfiguration der Darstellungsobjekte beim Beenden von SurfBorD nicht verloren geht, ist es erforderlich, die Attribute persistent zu speichern. Dies könnte im lokalen Dateisystem erfolgen, wenn es der eventuell eingerichtete Security-Manager zuläßt, oder auch in der Datenbank. Im zweiten Fall hätte die Datenbank mit den Konfigurationseinstellungen die Bedeutung einer ,,Home-Database", von der aus auf andere Datenbanken gewechselt werden könnte.

In diesem Zusammenhang wäre es sinnvoll, auch Querverweise zu anderen Datenbanken in der Home-Database zu speichern. Dabei könnte der JDBC-Treiber für die referenzierte Datenbank ebenfalls in der Home-Database gespeichert sein, so daß es beim Start von SurfBorD genügt, über den JDBC-Treiber für die Home-Database zu verfügen. Um diese Möglichkeiten anbieten zu können, ist es jedoch erforderlich, die einzelnen Depots für die benutzerdefinierten Einstellungen verwalten zu können, damit kein Konflikt mit den begrenzten Ressourcen der Datenbank auftritt.

Abhängig von den Entwicklungen bezüglich JDBC ist es auch denkbar, die bestehenden Konzepte auf andere Datenbanksysteme auszuweiten. Eine Anbindung an objekt-relationale, objektorientierte oder auch hierarchische Datenbanken wäre möglich, sofern die entsprechenden Treiber zur Verfügung stehen.

Literatur

1. G. Vossen, Datenmodelle, Datenbanksprachen und DBMS, Addisson-Wesley 1994
2. Sun, The Java Tutorial, http://java.sun.com/docs/books/tutorial
3. Sun, JDBC Guide: Getting Started, ftp://ftp.javasoft.com/docs/jdk1.1/jdbc.pdf
4. Sun, JDBC Drivers, http://java.sun.com/products/jdbc/jdbc.drivers.html
5. Sun, JDBC-2.0, http://java.sun.com/products/jdbc
6. Sun, HotSpot, http://java.sun.com/products/hotspot
7. Synergex, ODBC, http://www.synergex.com/odbc/odbc_man.htm
8. Oracle, JSQL, http://www.oracle.com/st/products/jdbc/html/stroadmap.html
9. Sun, Blend, http://java.sun.com/products/java-blend