WebService-basierte Integration externer Datenquellen in relationale Datenbanksysteme

Lutz Schlesinger

Universität Erlangen-Nürnberg
Lehrstuhl für Datenbanksysteme
Martensstraße 3
D-91058 Erlangen
schlesinger@informatik.uni-erlangen.de

Wolfgang Lehner

Technische Universität Dresden
Arbeitsgruppe Datenbanken
Dürerstraße 26
D-01069 Dresden
lehner@inf.tu-dresden.de

Kurzfassung: Traditionell werden Datenbanksysteme als reine Datenverwaltungssysteme in unterschiedlichsten Anwendungsgebieten verwendet. Die Nutzung verteilt gehaltener Informationen und eine Verknüpfung mit lokalen Daten wird insbesondere bei der Rolle der Datenbanksysteme als zentrale Integrationsplattform immer wichtiger. Während kommerzielle Systeme proprietäre Schnittstellen als Erweiterungsmechanismus anbieten, wird in diesem Beitrag ein Rahmenwerk und eine Implementierung vorgestellt, welche durch Nutzung einer satzorientierten Schnittstelle nach dem ONC-Protokoll sowohl eine enge Kopplung auf Basis von Java-Objekten als auch eine lose Kopplung durch Nutzung von WebService-Technologien ermöglicht. Die Leistung des Systems besteht somit darin, dass beliebige über WebService bereitgestellte Dienste, die eine minimale vorgegebene Schnittstelle erfüllen, transparent als Datenbankobjekt in ein relationales Datenbanksystem integriert werden können.

1 Einleitung

Der Wunsch nach einem transparenten Zugriff auf unterschiedliche heterogene Datenquellen beschäftigt seit vielen Jahren sowohl die Forschung als auch die Industrie mit deren kommerziellen Systemen unter dem Schlagwort der Föderierten Datenbanken oder Datenbank-Middleware ([Conr97], [Rahm94]). Die Früchte der Diskussion und Implementierungen bestehen in proprietären Lösungen ([DiGe00]), einzelne Fremdsysteme in ein existierendes Datenbanksystem zu integrieren. Abbildung 1 illustriert diese Situation: Unterschiedliche Systeme weisen verschiedene Erweiterungsschnittstellen auf, sodass für jede einzubettende Datenquelle eigene Kapsel-(Wrapper-) Module ([RoSc97]) entwickelt werden müssen, um einen Zugriff auf den externen Datenbestand zu ermöglichen. Der Entwicklungs- und Konfigurationsaufwand artikuliert sich bei n Datenbanksystemen und m zu integrierenden Datenquellen in $n \bullet m$ unterschiedlichen Modulen in Form von Tabellenfunktionen (IBM DB2, [IBMI01a], [RPK+99]), Cartridges (Oracle, [Orac01a]), Data Blades (Informix, [IBMI01b]), etc.

Insbesondere die WebService-Technologie ([Newc02]) weckt jedoch den Wunsch, Fremdsysteme unabhängig von einem konkreten Datenbanksystem möglichst zur Laufzeit dynamisch einzubinden, um auf sie transparent zugreifen und mit lokalen Datenbeständen ver-

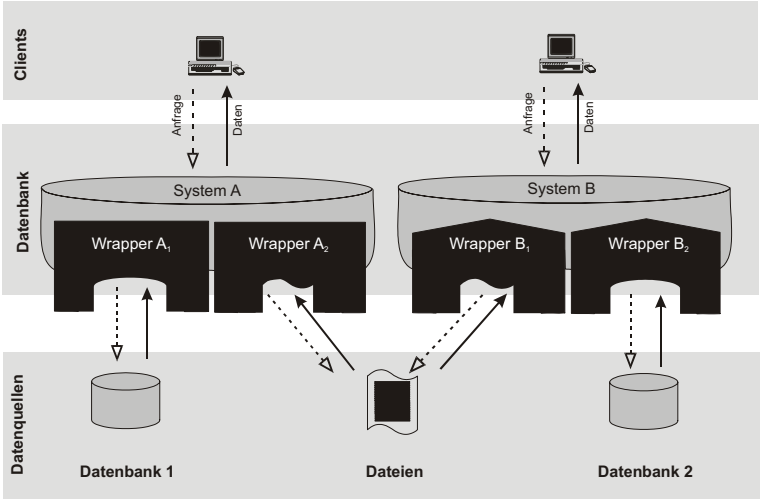
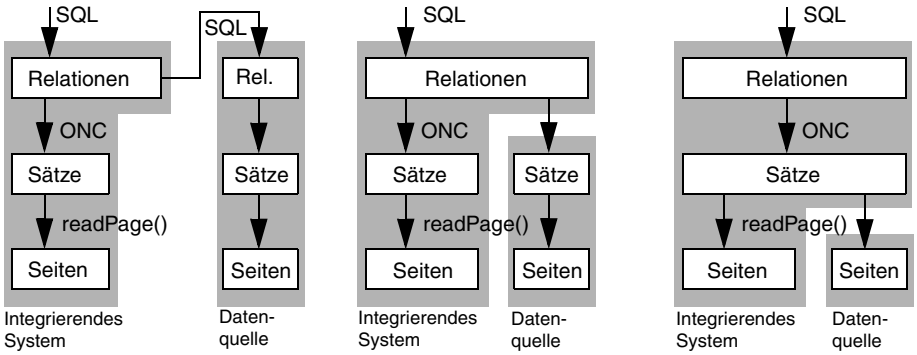


Abb. 1: Integration externer Datenquellen über proprietäre Wrapper-Module

knüpfen zu können. In diesem Beitrag wird ein Rahmenwerk vorgestellt, welches auf Ebene der Satzschnittstelle eine statische (via Java-Schnittstellen) und eine dynamische (via WebService-Diensten) Einbettung externer Datenquellen ermöglicht. Vor einer ausführlichen Diskussion des Rahmenwerks (Abschnitt 2) und einer Darstellung des aktuellen Stands der Realisierung am Beispiel von Oracle9i (Abschnitt 3) sind folgende Eigenschaften insbesondere als Abgrenzung zu bestehenden Ansätzen zu betrachten:

- *Funktionalität und Flexibilität über Effizienz*
  Mit der vorgestellten generischen Einbettungstechnologie wird Funktionalität über Effizienz gestellt. Wichtig für das in diesem Bericht angesprochenen Anwendungsszenario ist, dass überhaupt ein transparenter und möglichst zur Laufzeit dynamisch realisierter Zugriff auf eine Datenquelle möglich ist. Effizienz ist sekundär und kann durch proprietäre Einbettungstechniken erreicht werden.
- *ONC-Protokoll als Ebene der Integration*
  Bei der Wahl der Integrationsebene stehen grundsätzlich drei Alternativen zur Auswahl (Abbildung 2), die sich an der klassischen Grobarchitektur von Datenbanksystemen ([HäRa99]) orientieren. Im Wesentlichen besteht ein Datenbanksystem zunächst aus der Ebene der mengenorientierten Verarbeitung mit Relationen als Verarbeitungseinheiten und SQL als Zugriff. Die zweite Ebene repräsentiert eine satzorientierte Schnittstelle nach dem Open-Next-Close-Protokoll (ONC), während die unterste Ebene einzelne Seiten über elementare Operationen wie readPage(), writePage(), ... den höheren Schichten zur Verfügung stellt.
  Eine Integration auf höchster Ebene wird üblicherweise von DB-Middleware-Ansätzen realisiert (Abbildung 2a), wobei das zu integrierende System wiederum auf die mengenorientierte Schnittstelle der Datenquelle zurückgreift und die externen Zugriffe in SQL formuliert. Während einerseits Ausdrucksmächtigkeit gegeben ist, muss das Quellsystem sämtliche DB-Dienste bereitstellen. Ein Kopplung auf Ebene der Seitenzugriffe (Abbildung 2c) stellt das andere Extrem dar, wobei



a) DB-Middleware

b) Satzorientierter Zugriff

c) Seitenorientierter Zugriff

Abb. 2: Ebene der Integration externer Datenquellen

eine primitive Datenquelle eine Interpretation der Strukturen innerhalb einer Seite auf der integrierenden Seite voraussetzt. Als Kompromiss für eine flexible Kopplung bietet sich somit die ONC-Schnittstelle an (Abbildung 2b). Sowohl die statische als auch die dynamische Integration externer Datenbestände in dem vorgestellten System setzt auf dieser Schnittstelle auf.

2 Generische Integrationsschnittstelle

Die zentrale Idee, sowohl den Aufwand von Entwicklung und Installation zu reduzieren als auch eine Entkopplung von integrierendem Datenbanksystem und externer Datenquelle zu erzielen, besteht darin, das klassische Konzept der Wrapper-Module in einen datenbanksystem-spezifischen (PlugIn) und in einen datenquellen-spezifischen Anteil (Exposer) aufzuspalten. Die Schnittstelle zwischen PlugIn und Exposer sieht einen satzweisen Austausch einzelner Datensätze vor, weshalb die untere Schicht eine Transformation der Daten von einem beliebigen System in Satzform durchführen und die obere Schicht diese für eine Verwendung innerhalb des Datenbankkerns transformieren muss (Abbildung 3).

Die Schicht der Exposer übernimmt somit die klassische Wrapper-Funktionalität, indem beliebige Daten in eine Menge von Sätzen transformiert werden. Die obere Schicht der PlugIns liegt auf Seiten des Datenbanksystems und ist speziell an dessen technischen Möglichkeiten angepasst. Zentral ist, dass ein PlugIn pro Datenbanksystem nur einmal entwickelt und für alle Exposer mit einer satzorientierten Schnittstelle geeignet ist, sodass bei der Konfiguration einer externen Datenquelle das spezielle PlugIn nur mehrfach instanziiert wird. Die Exposer-Module selbst werden ebenfalls nur für jedes Quellsystem einmal implementiert und für jede Verbindung zu einem PlugIn instanziiert.

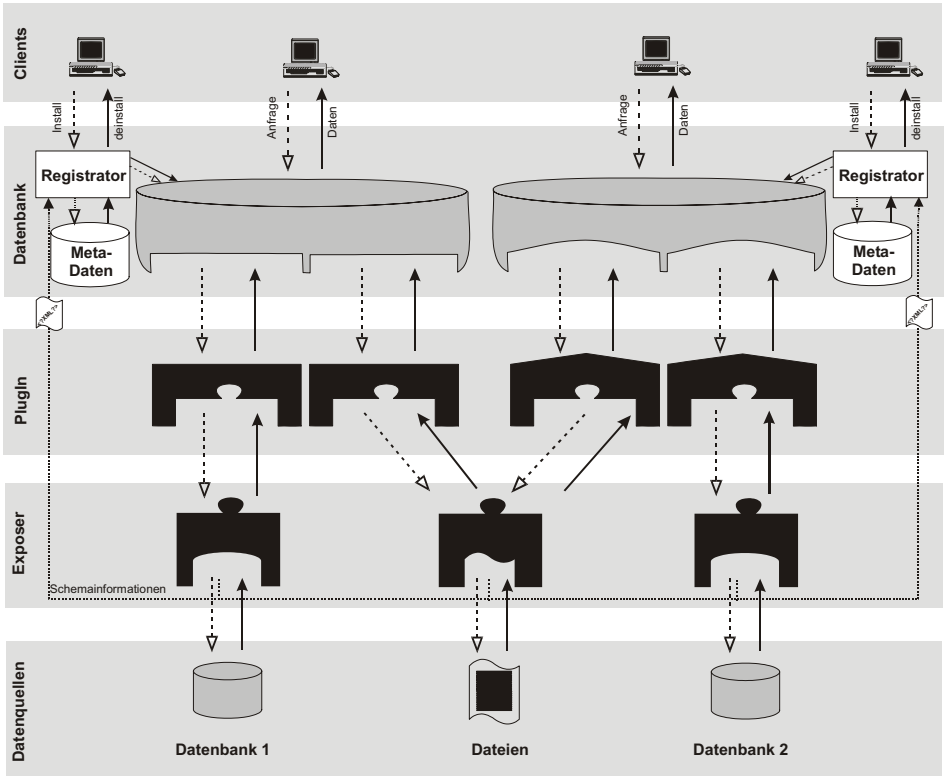


Abb. 3: Fünf-Schichten-Ansatz bei generischem Exposer/PlugIn-Ansatz

Konfiguration und Nutzung einer externen Datenquelle

Bei der Erzeugung* eines Exposer-Moduls für eine spezielle Datenquelle werden die Schemata der abgelegten und zugreifbaren Informationen in Form eines XML-Dokumentes ([W3C02b]) beschrieben und von einer Registrator-Anwendung eingelesen. Abbildung 4 zeigt das Schema der Quellenbeschreibung, wobei neben allgemeinen administrativen Informationen (Benutzername, Paßwort) die Satzbeschreibung in Form einer Menge einzelner Felder (column-Element) reflektiert wird†.

Der Registrator instanziiert das für das jeweilige Datenbanksystem existierende PlugIn, verbindet das PlugIn mit dem entsprechenden Exposer-Modul und erzeugt im integrierenden Datenbanksystem Hilfssichten, sodass die externe Quelle für den Benutzer transparent als 'normale' Relation sichtbar wird.

* Die Erzeugung kann in Abhängigkeit von der Datenquelle manuell, semiautomatisch oder automatisch erfolgen. Bspw. ist für den weiter unten diskutierten SOAP-fähigen Exposer in der aktuellen Implementierung eine weitgehende automatische Generierung möglich.

† In der aktuellen Implementierung existieren für relationale Datenbanksysteme Extraktionsprogramme, die Schemainformationen für einzelne Datenbankobjekte automatisch extrahieren und die Quellenbeschreibung erstellen.

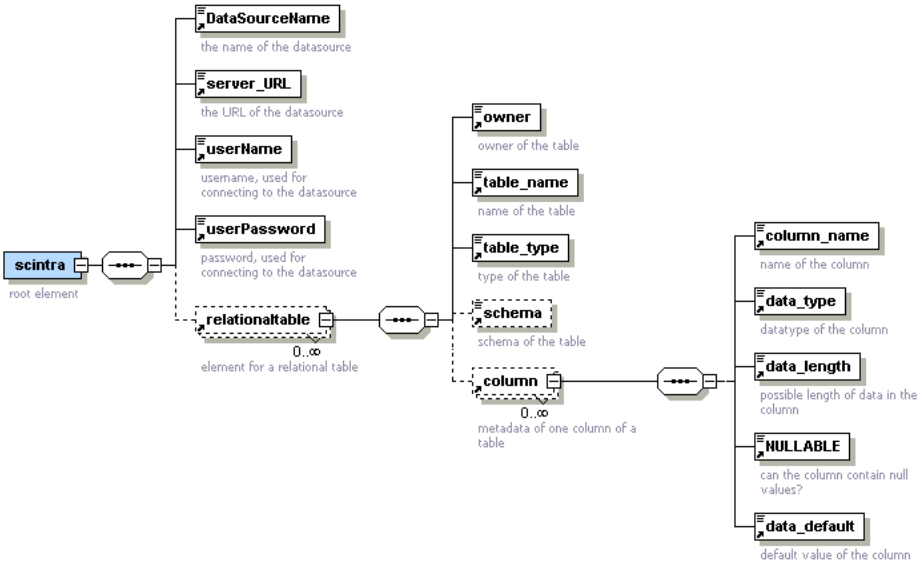


Abb. 4: XML-Schema der Quellenbeschreibung

Der Zugriff erfolgt nach dem ONC-Prinzip (open-next-close), wobei das PlugIn beim Exposer zunächst ein open() aufruft, wodurch eine Verbindung zum Quellsystem aufgebaut wird und die angeforderten Daten bereitgestellt werden. Anschließend wird bei jedem next()-Aufruf ein Datensatz vom Exposer an das PlugIn zur weiteren Verarbeitung übergeben. Sind alle Datensätze gelesen, so wird die Verbindung zum Quellsystem wieder geschlossen (close()). Wie in der klassischen Literatur ausführlich diskutiert ([ChSh93], [DuKS92], [LuOG93], [ONKE97], ...), sind beim Zusammenspiel von PlugIn und Exposer vielfältige Optimierungsstrategien denkbar, die vom einfachen Prefetching über Datensatz-Pooling bis hin zur lokalen Auswertung von Prädikaten reichen.

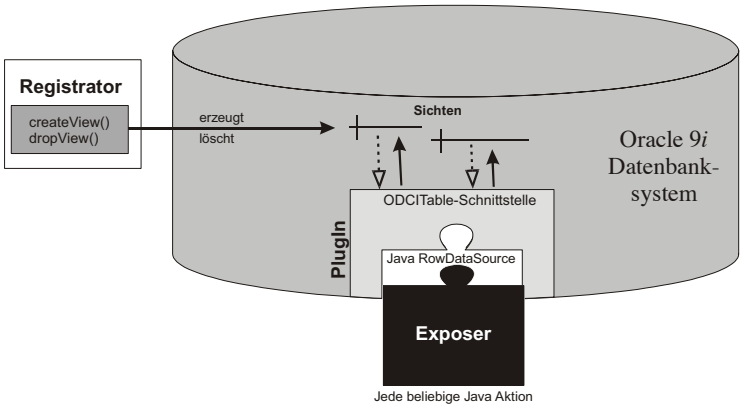


Abb. 5: Enge Bindung der Exposer-Modulee

Enge Bindung der Exposer-Module

Die erste (und klassische) Form der Integration externer Datenquellen in ein übergeordnetes relationales Datenbanksystem erfolgt auf dem Weg der engen Bindung der Exposer-Module (Abbildung 5). Sowohl das PlugIn als auch das Exposer-Modul laufen innerhalb des Datenbankkerns als Sammlung von Datenbankprozeduren, wobei in der aktuellen Implementierung (Abschnitt 3) auf Java als Programmiersprache der notwendigen PlugIn- und Exposer-Module zurückgegriffen worden ist. Im Fall des Datenbanksystems Oracle baut das PlugIn eine Brücke zwischen der proprietären ODCI-Table-Schnittstelle ([BaKM00], [Orac01a]) und der generischen JavaRowDataSource-Schnittstelle, die von Exposer-Modulen implementiert werden muss. Der quellenspezifische Exposer liegt somit in Form eines beliebigen Java-Objektes vor, welches zur Laufzeit innerhalb des Datenbankkerns abläuft und offensichtlich alle programmiersprachlichen Möglichkeiten ausschöpfen kann.

Insbesondere kann der Exposer über Prozedurfernaufruf mit anderen Komponenten, die außerhalb der Datenbank existieren, in Verbindung treten. Neben RMI, CORBA, etc. bietet sich somit auch die Nutzung von SOAP ([BEK+00]) als XML-basierte Kopplung an Fremdsysteme an. Zu beachten ist dabei, dass ein einziger SOAP-fähiger Exposer für unterschiedliche Datenbanksysteme Verwendung finden kann. Abbildung 6 skizziert die entsprechende Konfiguration, wobei dem Exposer die zusätzlich beliebige, jedoch vom System nicht explizit unterstützte Funktionalität weiterer WebService-Technologien (insbesondere von [UDDI02]) zur Verfügung steht.

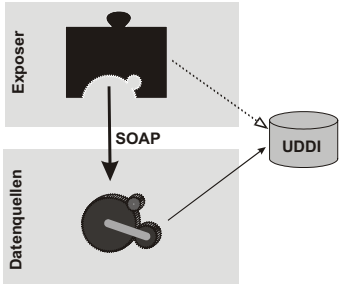


Abb. 6: SOAP-fähiger Exposer für enge Kopplung

Explizite Unterstützung von WebService-Technologien durch lose Kopplung

Eine explizite Unterstützung von WebService-Technologien zur flexiblen und zur Konfiguration dynamischer Integration von WebDiensten als reguläre Datenquelle in einem relationalen Datenbanksystem ist innerhalb des Rahmenwerks durch die Einführung einer losen Kopplung in Form eines WebService-basierten PlugIns möglich. Abbildung 7 illustriert diesen Ansatz im Vergleich zur engen Kopplung über die Bereitstellung einer Java-Schnittstelle.

Im Vergleich zur engen Kopplung, stellen Exposer-Module bei einer WebService-basierten Kopplung das Angebot, Daten über die generische ONC-Schnittstelle bereitzustellen, im UDDI in Form eines WSDL-Dokumentes ([W3C02a]) zur Verfügung, wobei als Kommentar der Diskursbereich der Datenquellen hinterlegt werden kann, sodass ein Benutzer durch Verwendung des Registrator-Moduls den entsprechenden Dienst im UDDI auffinden kann. Nach einer Entscheidung für die Nutzung einer Datenquelle wird – analog zur engen Kopplung – das Schema der Datenquelle über einen inquire()-Aufruf erfragt und ein WebService-PlugIn im Datenbanksystem instanziiert. Ein WebService-PlugIn realisiert nun analog zum PlugIn für die enge Kopplung über Java die satzorientierte Schnittstelle über SOAP, sodass jeder Exposer, der die Schnittstellen inquire(), open(), next() und close()

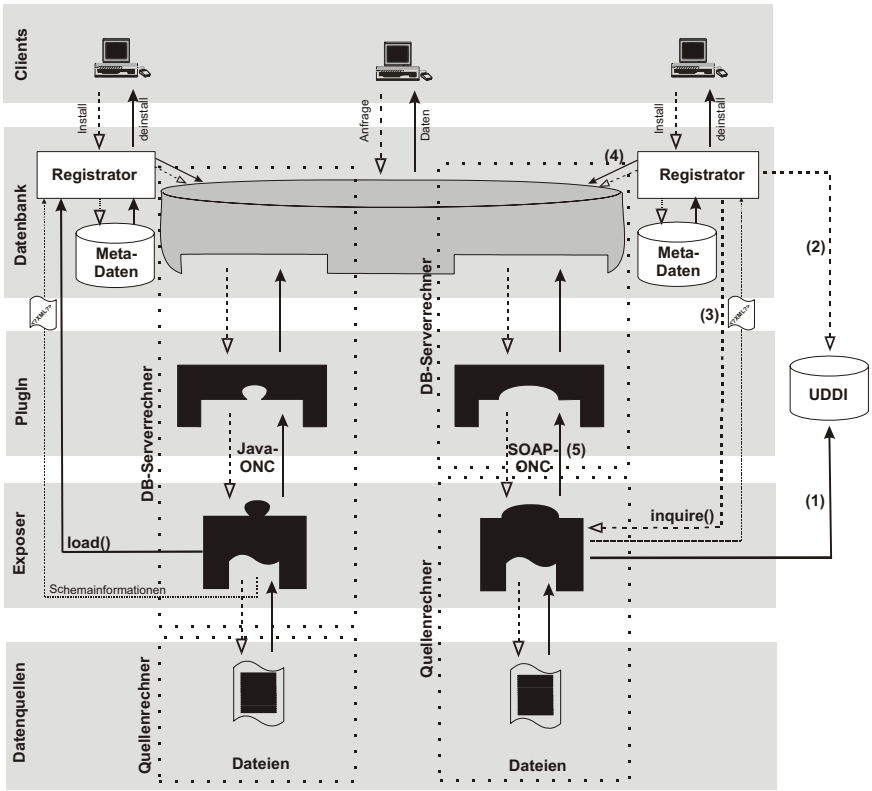


Abb. 7: Serverzentrierte Kapselung versus quellenzentrierte Kapselung

als SOAP-Methoden implementiert von diesem PlugIn als externe Datenquelle genutzt werden kann. Im UDDI erscheint eine Datenquelle in den Yellow-Pages als ein logisch einzelner Business Service.

3 Aktuelle Umsetzung

Die enge Form der Kopplung des vorgestellten Rahmenwerks ist im Kontext des SCINTRA-Projektes (›semi-consistent integrated timeoriented replication aspect‹) am Beispiel des Datenbanksystems Oracle 9i (Version 9.2) vollständig mit eine Satz von Exposer-Modulen für unterschiedliche Datenquellen realisiert. Das Oracle-System selbst bietet zwar eine Vielzahl von Anbindungsmöglichkeiten für externe Datenquellen wie die ›Heterogenous Services‹ für Datenbanken oder die ›External Tables‹ für Dateien ([Orac01b]), wobei die zu verwendenden Schnittstellen der umsetzenden Module bisher nur spärlich bis überhaupt nicht dokumentiert sind. Ferner decken die von Oracle bereitgestellten Wrapper-Module nicht alle Quellsysteme ab.

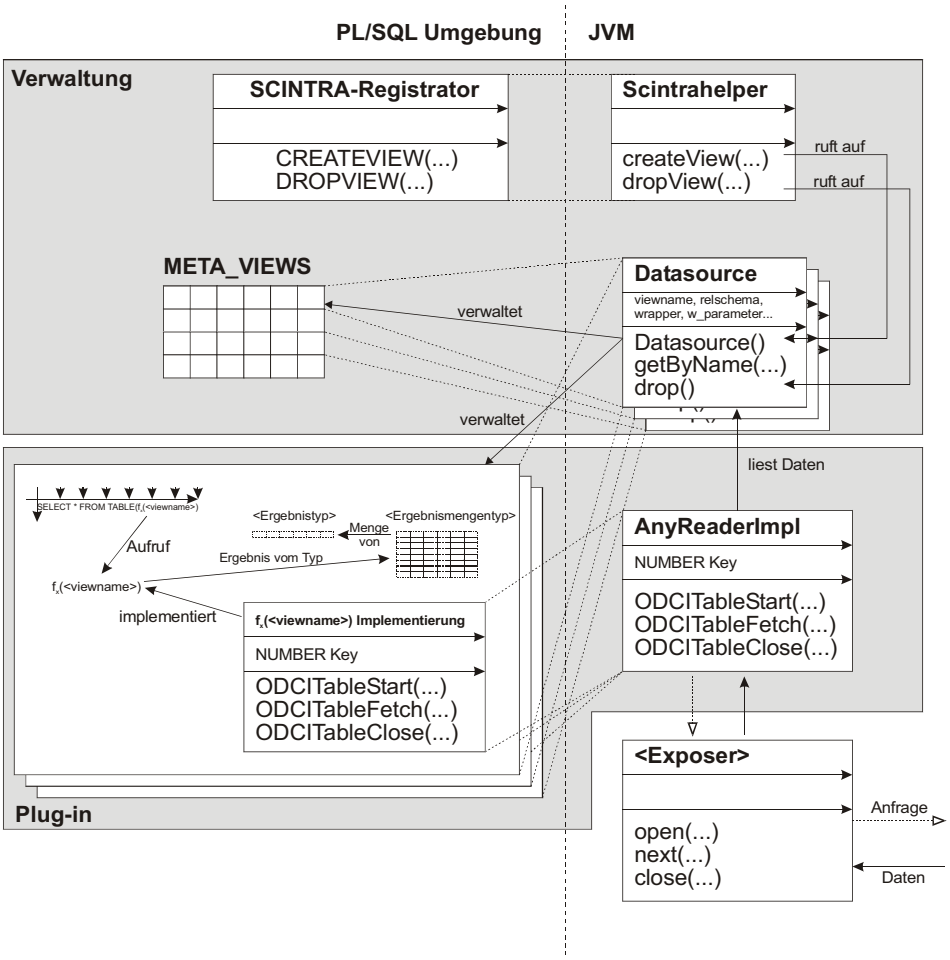


Abb. 8: Zusammenspiel von Objekten und Klassen der Oracle-basierten Implementierung

Zur Realisierung der Architektur wird auf die gut dokumentierten ›Pipelined Table Functions‹ zurückgegriffen. Das generische PlugIn für Oracle bildet ein Modul, welches in Form einer Oracle Data Cartridge realisiert ist. Die Java-basierte Implementierung wird im Fall von Oracle dadurch begünstigt, dass innerhalb des Datenbankkerns eine Java2-kompatible virtuelle Maschine nutzbar ist ([Orac01c]), so dass der Datendurchsatz im Vergleich zu externen Prozessen deutlich günstiger ausfällt. Abbildung 8 skizziert die Realisierung des Java-basierten PlugIn und die Schnittstelle zu den Exposer-Modulen.

Überblick über die Strukturen

Die Prozeduren createView() und dropView() werden vom externen Registrator aufgerufen und kapseln eine konkretes Objekt der Klasse Datasource zur Beschreibung einer externen Datenquelle in Form einer Sicht. Da eine Instanz der Klasse Datasource nur bei Operationen auf der Datenquelle existiert, werden die für die Instanziierung notwendigen Informa-

tioen wie das Schema der Quelle, die Namen der PL/SQL-Objekte und des zu verwendenden Exposers mit seinen Parametern, in der Tabelle META_VIEWS abgelegt. Bei der Instanziierung eines Datasource-Objekts werden die notwendigen PL/SQL-Objekte wie Sicht, Ergebnistypen und die Tabellenfunktionen inklusive Implementierungstyp erstellt. Die Factory-Methode getByName() dient der Reinstanziierung des Datasource-Objekts über den zuvor angelegten Sichtennamen, was wiederum von der Klasse AnyReaderImpl durchgeführt wird, um das Datasource-Objekt und die zum Aufruf der Exposer und zur Rückgabe der Ergebnisse nötigen Informationen zu erhalten. Diese zentrale Klasse AnyReaderImpl spiegelt eine flexible Implementierung der proprietären ODCITable-Schnittstelle wider, da sie je nach Datenquelle verschiedene Rückgabetypen haben kann. Da auf PL/SQL-Ebene diese Flexibilität nicht gegeben ist, muss für jede Datenquelle eine eigene Deklaration des Implementierungstyps erfolgen. Dieser muss dann jeweils als Tabellenfunktion veröffentlicht werden, so dass er über die erstellte relationale Datenbanksicht aufgerufen werden kann.

Überblick über den Ablauf

Sobald eine Datenquelle eingerichtet ist, d.h. mit createView() eine entsprechende Sicht erstellt worden ist, wird beim Zugriff auf die Sicht die entsprechende Tabellenfunktion aufgerufen. Zunächst ermittelt der Oracle-spezifische ODCITableStart-Aufruf die korrekte Instanz des Datasource-Objekts über die getByName()-Methode und instanziiert und initialisiert daraufhin den entsprechenden Exposer. Jeder folgende ODCITableFetch-Aufruf beim Zugriff auf die Daten wird in einen next()-Aufruf an den Exposer umgesetzt, der die Formate anpasst und den gelesenen Datensatz an das Laufzeitsystem des Datenbanksystem übergibt. Liegen keine Datensätze mehr vor, so ruft das Datenbanksystem die Methode ODCITableClose() auf, woraufhin der Exposer angewiesen wird, die Quelle zu schliessen. Im letzten Schritt wird das Datasource-Objekt wieder verworfen.

4 Zusammenfassung

Lokal in einem Datenbanksystem verwaltete Informationen mit externen Daten anzureichern bzw. ein Datenbanksystem nur noch als zentralen Zugang zu beliebig verteilt gehaltenen Datenbeständen einzusetzen, kann proprietär durch Nutzung spezifischer Erweiterungsmechanismen erreicht werden. Proprietäre Ansätze verhindern jedoch im Umfeld global zur Verfügung stehender Informationen die flexible und dynamische Nutzung von Datenbeständen. In diesem Beitrag wird ein Ansatz vorgestellt, der durch Einführung einer generischen ONC-Schnittstelle eine Entkopplung von datenbankspezifischen und datenquellenspezifischen Implementierungen erlaubt. Die Kopplung ist dabei entweder eng in Form von JAVA-Objekten auf Seite des Datenbanksystems oder lose durch Nutzung von WebService-Technologien zur Realisierung des ONC-Protokolls.

Während es in der aktuellen Implementierung bereits möglich ist, *jedes beliebige Java-Objekt*, welches die Methoden der vorgegebenen ONC-Schnittstelle implementiert, als Relation in ein klassisches Datenbanksystem einzubinden, beschäftigen sich die aktuellen Arbeiten damit, das Konzept auf die Nutzung der WebService-Technologie zu erweitern.

Es wird dann möglich sein, *jeden beliebigen Dienst*, welcher die ONC-Schnittstelle über SOAP implementiert, als Datenbankobjekt transparent für den Nutzer oder existierende Applikationen zur Verfügung zu stellen.

Die zukünftigen Arbeiten fokussieren zunächst die Evaluierung des vorgestellten Konzepts für unterschiedliche Datenbanksysteme. Des weiteren ist eine Flexibilisierung der einfachen ONC-Schnittstelle angedacht, sodass anstelle von einzelnen Sätzen bspw. ganze XML-Fragmente verarbeitet werden können. Ebenso soll eine weitere Dynamisierung des Ansatzes dahingehend durchgeführt werden, dass eine Quelle zusammen mit ihrem Exposer erst zur Anfrageausführungszeit ausgewählt werden kann. Schließlich sind die Einsatzmöglichkeiten von Sicherheitsmechanismen ([JeZe02]) zu untersuchen und umzusetzen.

Literatur

BaKM00 Banerjee, S.; Kishnamurthy, V.; Murthy, R.: *All Your Data: The Oracle Extensibility Architecture*. In: Geppert, A.; Dittrich K. R.: *Component Database Systems*. Morgan Kaufmann, San Mateo (CA), 2000

BEK+00 Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H.F.; Thatte, S.; Winer, D.: *Simple Object Access Protocol (SOAP) 1.1, Technical Note*. World Wide Web Consortium (W3C), 2000
(Elektronisch verfügbar unter: http://www.w3c.org/TR/SOAP/)

ChSh93 Chaudhuri, S.; Shim, K.: Query Optimization in the Presence of Foreign Functions. In: *Proceedings of the 19th International Conference on Very Large Data Bases* (VLDB'93, Dublin, Irland, 24.-27. Aug.), 1993, S. 529-542

Conr97 Conrad, S.: *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer Verlang, Berlin / Heidelberg / New York, 1997

DiGe00 Dittrich, K.; Geppert, A.: *Component Database Systems*. Morgan Kaufmann, San Mateo (CA), 2000

DuKS92 Du, W.; Krishnamurthy, R.; Shan, M.-C.: Query Optimization in Heterogenous DBMS. In: *Proceedings of the 18th International Conference on Very Large Data Bases* (VLDB'92, Vancouver, Kanada, 23.-27. August), 1992, S. 277-291

HäRa99 Härder, T.; Rahm, E.: *Datenbanksysteme - Konzepte und Techniken der Implementierung*. Springer-Verlag, Heidelberg u.a., 1999

IBMI01a o.V.: *Creating User-Defined Routines and User-Defined Data Types, Version 9.3*. IBM Corporation 2001

IBMI01b o.V.: *DataBlade Module Development Overview, Version 4.0*. IBM Corporation 2001

JeZe02 Jeckle, M.; Zengler, B.: SOAP - Aber sicher. In: *OBJEKTspektrum - Zeitschrift für Web und Objekttechnologie*, Januar/Februar 2002, S. 17-26

LuOG93 Lu, H.; Ooi, B.-C.; Goh, C.-H.: Multidatabase Query Optimization: Issues and Solutions. In: *Proceedings of the 3rd International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems* (RIDE-IMS'93, Wien, Österreich, 19.-20. April), 1993, S. 137-143

Newc02 Newcomer, E.: *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison Wesley, Boston u.a., 2002

ONKE97 Ozcan, F.; Nural, S.; Koksal, P.; Evrendilek, C.: Dynamic Query Optimization in Multidatabases. In: *IEEE Data Engineering Bulletin*, 20(1)1997, S. 38-45

Orac01a o.V.: *Oracle 9i Data Cartridge Developer's Guide, Release 1 (9.0.1)*. Oracle Corporation, 2001

Orac01b o.V.: *Oracle 9i Heterogeneous Connectivity Administrator's Guide, Release 1 (9.0.1)*. Oracle Corporation, 2001

Orac01c o.V.: *Oracle 9i Java Developer's Guide, Release 1 (9.0.1)*. Oracle Corporation, 2001

Rahm94 Rahm, E.: *Mehrrechner-Datenbanksysteme*. Addison-Wesley, Bonn / Paris, 1994

RoSc97 Roth M.T., Schwarz P.: Don't Scrap It, WrapIt! In: *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB'97, Athen, Griechenland, 25.-29. August), 1997, S. 266-275

RPK+99 Reinwald, B.; Pirahesh, H.; Krishnamoorthy, G.; Lapis, G.; Tran, B.; Vora, S.: Heterogeneous Query Processing through SQL Table Functions. In: *Proceedings of the 15th International Conference on Data Engineering* (ICDE'99, Sydney, Australien, 23.-26. März), 1999, S. 366-373

UDDI02 o.V.: *UDDI XML Structure Reference*, Version 3.0, Juli 2002 (Elektronisch verfügbar unter: http://www.uddi.org)

W3C02a o.V.: *Web Services Description Language (WSDL)*. World Wide Web Consortium (W3C), 2002 (Elektronisch verfügbar unter: http://www.w3c.org/TR/wsdl)

W3C02b o.V.: *Extensible Markup Language (XML)*. World Wide Web Consortium (W3C), 2002 (Elektronisch verfügbar unter: http://www.w3c.org/XML/)