# Eine Fallstudie:
# Einbindung von Legacy-Datenbanken über JDBC

*Dr. Rainer Kerth*

IBM Unternehmensberatung GmbH
Mitglied im Bundesverband Deutscher Unternehmensberater BDU e.V.
Mitglied der IBM Consulting Group

## *Übersicht*

- **Projektkontext und Ziele**
- **Die Auswahl der Programmiersprache**
- **Übersicht über die Gesamtarchitektur des entwickelten Systems**
- **Das Persistenz-Framework der Datenzugriffsschicht**
- **Die Schnittstelle zur relationalen und zur hierarchischen Datenbank**
- **Übersicht über die Host-Anbindung**
- **Die Funktionalität des JDBC Treibers**
- **Performance**
- **Transaktionskoordinierung**

## *Das Projekt diente der Evaluierung einer alternativen Anwendungsentwicklung in einem Unternehmen der Versicherungsindustrie*

- **Der traditionelle Ansatz für die Anwendungsentwicklung war stark durch proprietäre Standards geprägt.**
  - Die Abhängigkeiten von Herstellern beschränkten die Auswahlmöglichkeiten des Unternehmens bei strategischen Entscheidungen.
  - Dadurch war es schwierig, auf Entwicklungen am Markt schnell zu reagieren.
- **Die technischen Grundlagen entsprachen nicht mehr dem heutigen Standard.**
  - Die im Unternehmen gewachsenen Programmbibliotheken waren nur schwer zu pflegen.
  - Eine Wiederverwendung von bestehenden Bibliotheken wurde durch eine Vermischung von Geschäfts-, Anzeige- und Datenzugriffslogik erschwert.

## *Als Basis für die neue Anwendungsentwicklung wurde die Java-Plattform gewählt.*

- **Java ist leicht zu erlernen.**
  - Das Unternehmen verfügte zu Beginn des Projektes über lediglich über geringe Erfahrungen mit Objekttechnologie.
  - Die Anfangsphase des Projektes wurde durch intensive Schulungen begleitet. Nach 3-6 Monaten war die Umstellung abgeschlossen.
  - Aus heutiger Sicht hat die Wahl von Java viel zum Erfolg des Projektes beigetragen.
- **Java ist plattformunabhängig.**
  - Der tägliche Entwicklungsbetrieb konnte weitgehend von systemadministrativen Maßnahmen freigehalten werden.
  - Java ermöglicht eine transparente Umstellung der zugrundeliegenden Hardware.
- **Auch die Firma Bull hat weitgehende Unterstützung für Java angekündigt.**
  - Somit bietet die Entwicklung in Java auch interessante Perspektiven für das Hostumfeld.

## *Die Anwendungsarchitektur besteht aus einem verteilten Model-View-Controller-Pattern, basierend auf CORBA.*

- **CORBA ist ein offener Standard, der eng mit Java integriert ist.**
  - CORBA ermöglicht eine weitgehende Netzwerktransparenz.
  - Durch eine Vielzahl von vordefinierten Diensten wird der Anteil der System-programmierung am Entwicklungsprozeß stark reduziert.
- **Durch das Model-View-Controller-Pattern wird eine klare Trennung von Geschäfts- und Präsentationslogik erreicht.**
  - Modellobjekte leben auf dem Server und haben keine direkte Kenntnis ihrer Präsentationsschicht. Sie kapseln alle Datenzugriffe.
  - Controller leben auf dem Client und koordinieren die Netzwerkkommunikation.
  - Views werden aus graphischen Komponenten zusammengesetzt und durch den Controller mit Daten versorgt.

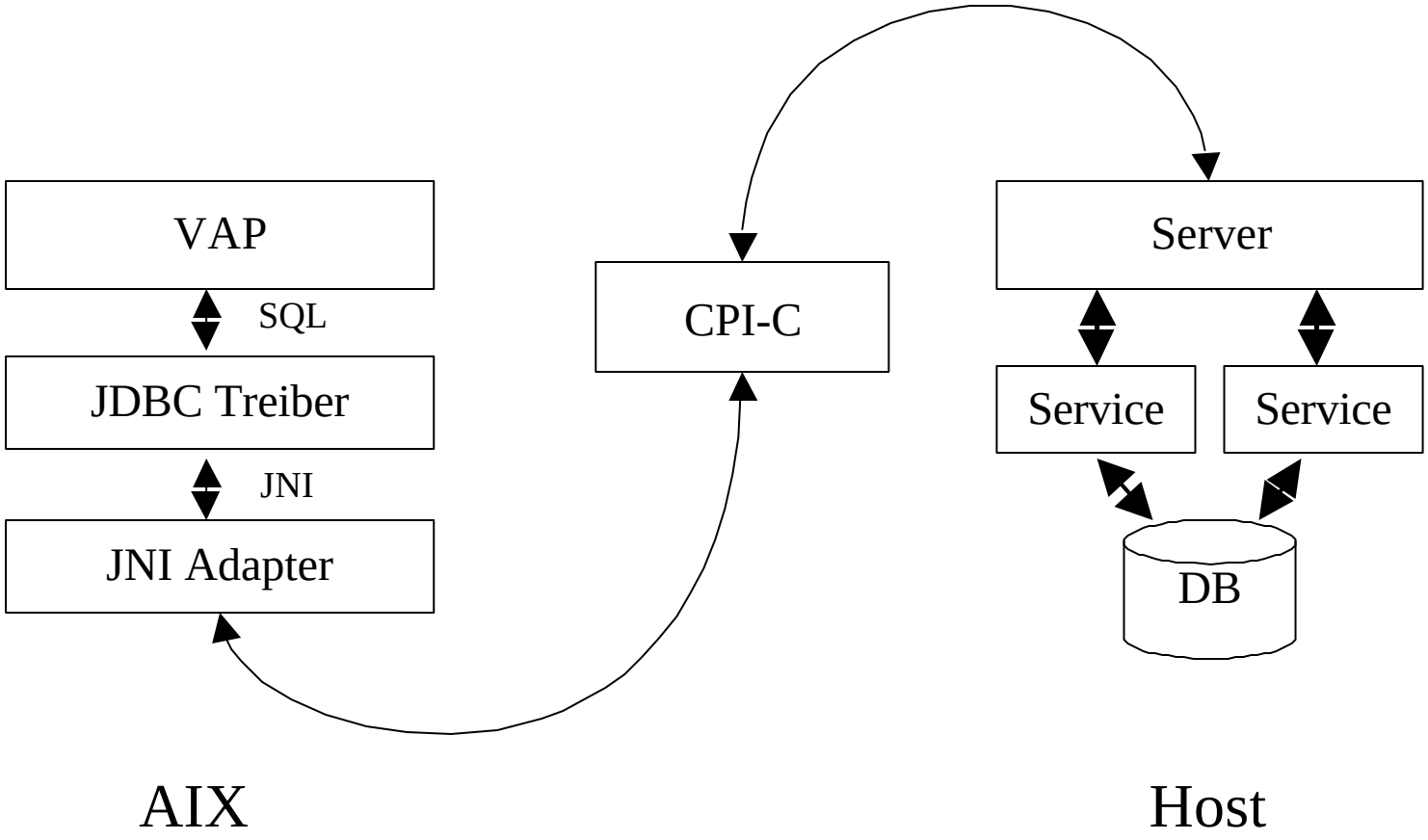
## *Für die Abbildung der Modellobjekte auf verschiedene Datenquellen kam ein OR-Framework zum Einsatz.*

- **Erste Versuche mit einem neuen, als Beta-Version zur Verfügung stehenden Framework waren wenig erfolgreich.**
  - Zeitverlust von 2 Monaten
- **Es erfolgte dann eine Umstellung auf VisualAge Persistence (VAP).**
  - VAP wird kostenlos als Teil von VisualAge für Java ausgeliefert.
  - VAP war eines der ersten OR Mappings, das sich am EJB Standard orientierte.
  - VAP steht als Quellcode zur Verfügung
- **VAP ermöglicht eine Integration mit Function Call Backends und mit JDBC Treibern**
  - Die Schnittstellen zu Function Call Backends sind bisher jedoch noch nicht standardisiert für Java (demnächst: Connectors).

## *Um größtmögliche Homogenität beim Datenzugriff zu erreichen, erfolgte die Anbindung des Host-Systems durch einen im Projekt entwickelten JDBC-Treiber.*

- **Dieser Ansatz erlaubt eine Gleichbehandlung von Daten in relationalen und in hierarchischen Datenquellen.**
  - Eine für die Zukunft zu erwägende Migration von hierarchischen Daten in eine relationale Datenbankstruktur wird dadurch erleichtert.
- **Aus Sicht des Entwicklungsprozesses werden die APIs vereinheitlicht.**
  - Es ist lediglich erforderlich, eine relationale Sicht auf die Daten zu unterstützen.
  - Eine explizite Navigationslogik für hierarchische Datenbestände entfällt.
- **JDBC ist ein offener Standard, der sowohl von Datenbankherstellern als auch von Middleware-Herstellern unterstützt wird.**
  - Hierdurch wird insbesondere ein Einsatz eines EJB Servers möglich.
  - Es gibt Überlegungen von Seiten des Herstellers der hierarchischen Datenbank, einen JDBC Treiber für den Zugriff auf die Daten zur Verfügung zu stellen.

## *Graphische Übersicht über die Gesamtstruktur der Hostanbindung*



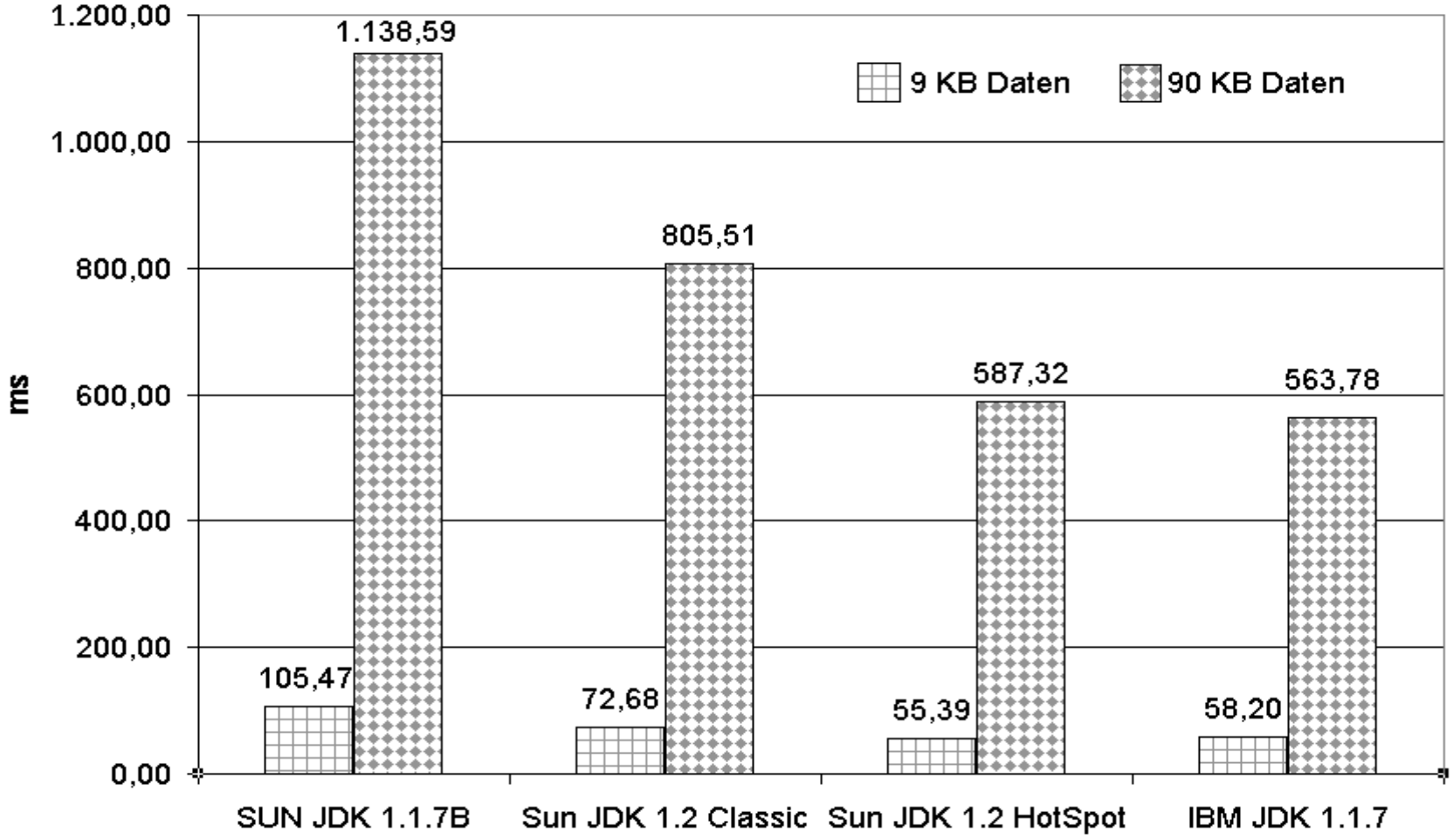
## *Der JDBC-Treibers erlaubt einen Zugriff auf hierarchische Daten über SQL.*

- **Die Hostdaten werden in ein relationales Format umgewandelt.**
  - Die Daten werden innerhalb des JDBC-Treibers als Tabellen gespeichert.
  - Hierarchie-Relationen werden über Fremdschlüssel abgebildet.
  - Dieser generische, konfigurierbare Mechanismus, zusammen mit passenden SQL-Abfragen, ersetzt die traditionelle Navigationslogik über die hierarchische Datenbank.
- **Die Daten werden vor der Auswertung durch VAP gefiltert.**
  - Die für die Instantiierung von Objekten benötigten Daten werden gezielt selektiert.
  - Diese Daten werden durch eine (eingeschränkte) SQL-Klausel beschrieben.
- **Nicht verwendete Daten werden in einem Cache für weitere Abfragen bereit gehalten.**
  - Dadurch wird die Netzbelastung reduziert.
- **Der JDBC-Treiber bietet auch eine Unterstützung für Stored Procedures.**
  - Services können nicht nur auf Datenbanken, sondern auch auf bestehende Host-Programme zugreifen und deren Rückgabewerte in den JDBC-Treiber weiterleiten.

## *Die genannten Funktionen des JDBC-Treibers beanspruchen zur Laufzeit 15% bis 20% der Gesamtperformance.*

- **Ein Großteil der Performance wird damit für andere Funktionen im Rahmen der Hostanbindung aufgewendet.**
  - Die Netzwerkkommunikation benötigt ca. 30%-35% der Performance.
  - Die Ausführung der Programme auf dem Host beansprucht die restlichen 50% der Zeit.
- **Die trotz allem bestehende Performance-Belastung muß gegen die Vorteile abgewogen werden, die durch standardisierte Schnittstellen gegeben sind.**
  - Klare Trennung von Datenzugriffslogik und Geschäftslogik.
  - Kompatibilität mit kommerziell verfügbarer Middleware (d.h. mit einem OR Framework oder einem EJB Server).
- **Die Performance des JDBC Treibers sollte bei Verwendung von adaptiven JVMs noch besser werden.**
  - Auf NT wurde eine Verbesserung um einen Faktor 2 gemessen.

# *Graphische Darstellung des Offline-Verhaltens des JDBC-Treibers in verschiedenen JVMs (durchschnittliche Zugriffszeit über 10000 Iterationen)*



## *Problematisch ist derzeit die Transaktionskoordination zwischen Datenquellen.*

- **Aus Sicht von VAP ist der Host eine von mehreren relationalen Datenquellen.**
  - Abfragen über verschiedene Datenquellen innerhalb einer Transaktion werden von VAP im Prinzip transparent verwaltet.
  - Zum Zeitpunkt des Commit werden die Änderungen über einen entsprechenden JDBC-Treiber an die beteiligten Datenquelle weitergegeben.
- **Die im Projekt verwendete Version von VAP basiert jedoch auf JDBC 1.22.**
  - Deswegen ist z. Zt. kein volles 2pc mit den Datenquellen möglich.
  - VAP verwendet intern bereits ein Zwei-Phasen Protokoll.
- **Auch auf der Host-Seite steht eine Unterstützung eines 2pc noch aus.**
  - Eine solche Unterstützung ist aber im Laufe dieses Jahres zu erwarten.
- **Deswegen werden die Daten derzeit innerhalb von kurzen Transaktionen zunächst in einen Bewegungspool geschrieben.**
  - Dieser wird dann durch ein Batch-Job in den eigentlichen Datenbestand überführt.
  - Inkonsistenzen im Bewegungspool müssen während der Überführung erkannt werden.