JDBC w LoXiMie

Interfejs Java Database Connectivity
dla systemu LoXiM

Adam Michalik
2008

Sterownik JDBC – co to jest?

- Sterownik JDBC to zbiór klas implementujących interfejsy opisane w specyfikacji JDBC
- Sterownik JDBC pozwala na wykorzystanie zestandaryzowanego sposobu obsługi bazy danych w Javie
- Sterownik JDBC przyjmuje podejście relacyjne – implementacja dla LoXiMu będzie ciekawa

Sterownik JDBC do LoXiM – po co?

- Pokazanie, że LoXiMu da się używać!
- Używać = stosować jako bazę danych do aplikacji / większych systemów
- Zintegrować API klienckie LoXiMu z jakimś standardem
- Umożliwić testowanie LoXiMu w warunkach nie tylko laboratoryjnych, ale quasi-produkcyjnych
- Umożliwić programistyczne, a nie tylko konsolowe testowanie LoXiMu

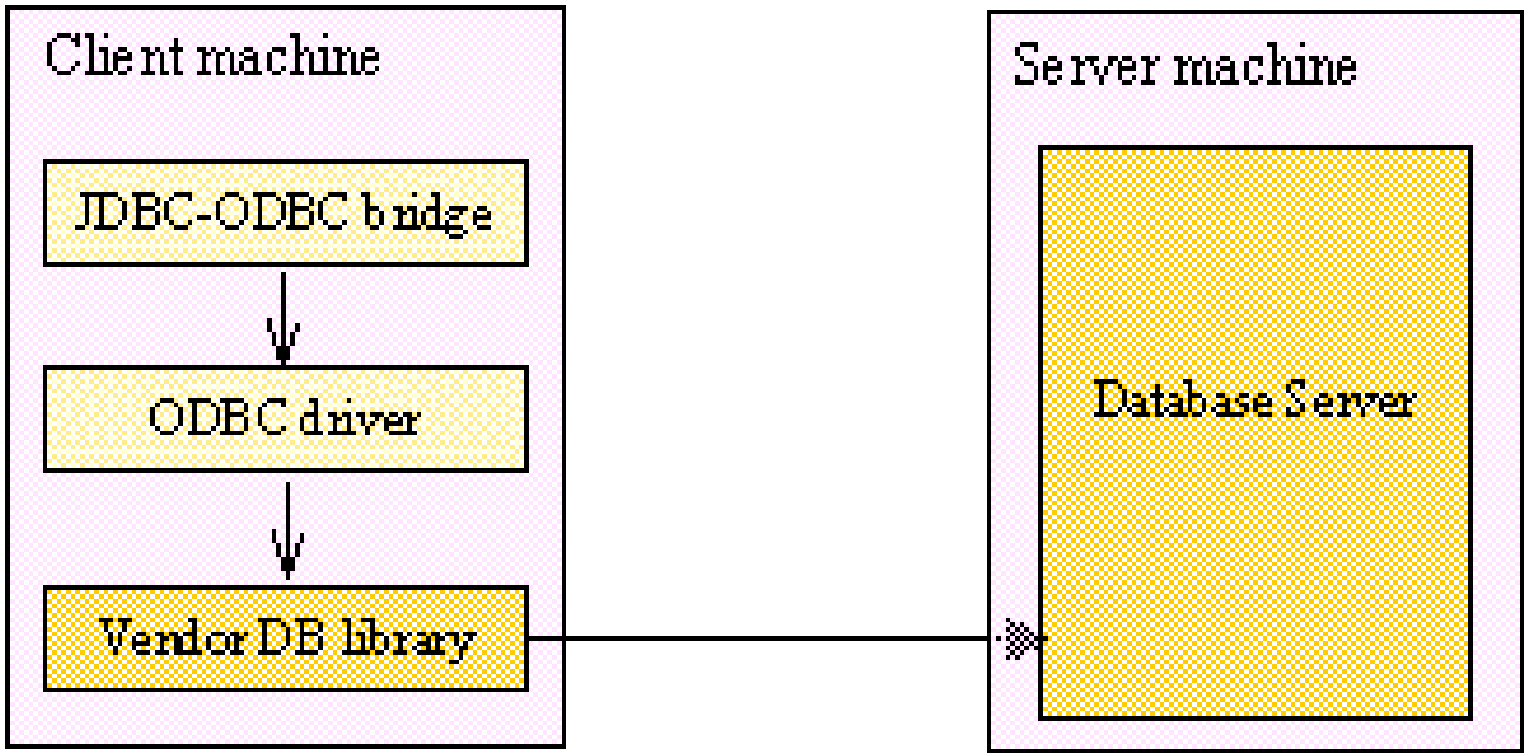
Sterownik JDBC do LoXiM – po co?

- Stworzyć podwaliny pod integrację LoXiMu z serwerami aplikacji przez systemy ORM (np. Hibernate), które korzystają ze sterowników JDBC. Takie rozwiązanie ostatecznie udowodniło by produkcyjną użyteczność LoXiMu

JDBC – typ 1

Mostek JDBC-ODBC

Mostek tłumaczy wywołania JDBC na ODBC (Open DataBase Connectivity) i wysyła je do serwera ODBC.



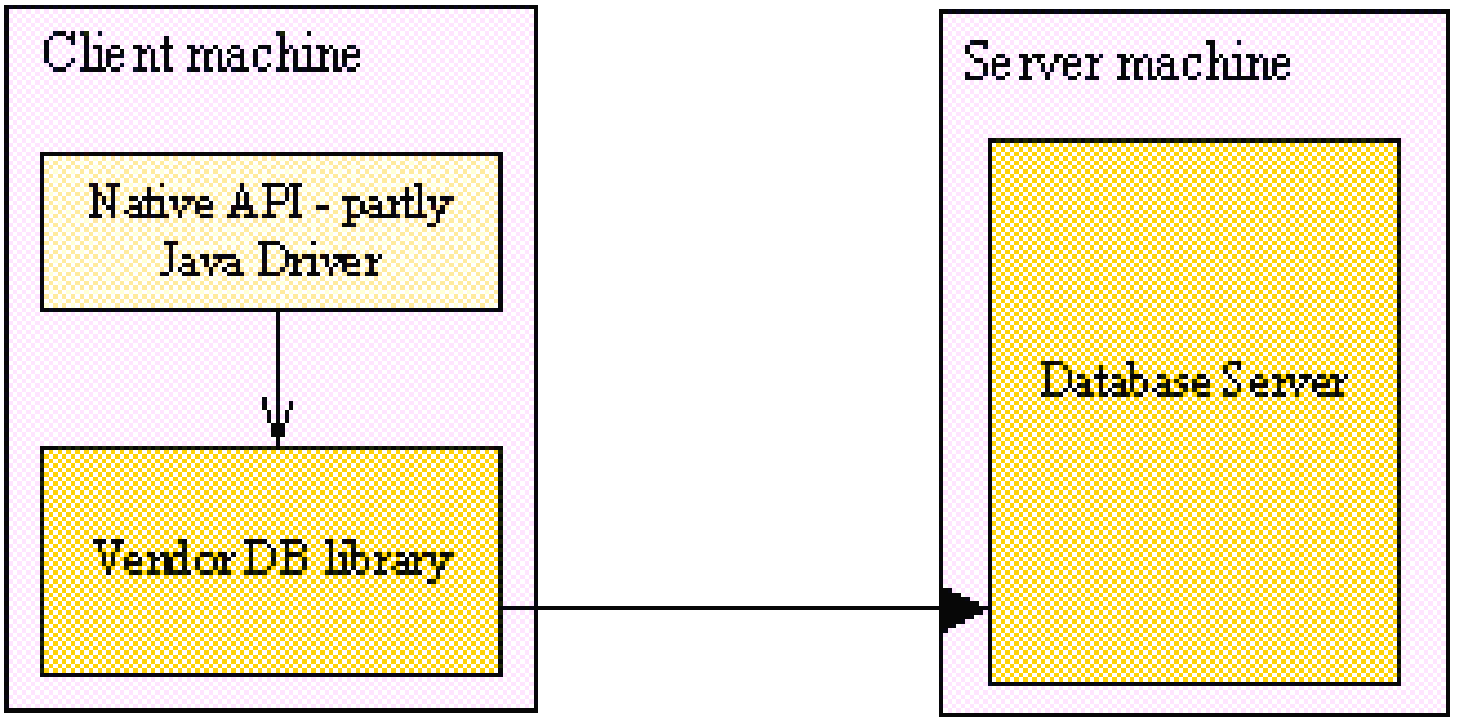
JDBC – typ 1

- Za:
  - jeśli istnieje sterownik ODBC (jak przypadku wielu systemów baz danych), mostek umożliwia łatwe użycie go w Javie
- Przeciw:
  - w przypadku LoXiM sterownika ODBC nie ma, więc ten typ w ogóle nie wchodzi w grę
  - obniżenie wydajności spowodowane tłumaczeniem wywołań JDBC na ODBC i w drugą stronę
  - sterownik ODBC musi być zainstalowany na maszynie klienta – uniemożliwienie używania np. apletów

JDBC – typ 2

Sterownik Javowy z kodem macierzystym

Sterownik pisany w Javie, ale z metodami operującymi na konkretnym systemie bazodanowym napisanymi w kodzie macierzystym



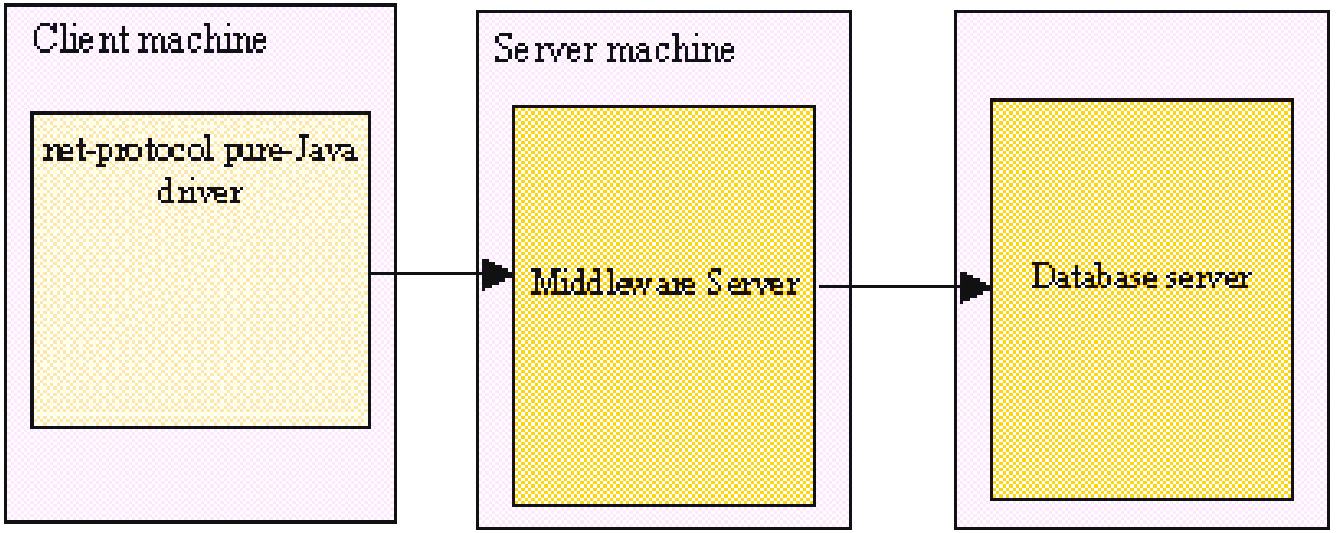
JDBC – typ 2

- Za:
  - lepsza wydajność niż typ 1
- Przeciw:
  - niższa wydajność niż typ 3 i 4
  - biblioteki macierzyste muszą być zainstalowane na maszynie klienta – uniemożliwienie używania np. apletów

JDBC – typ 3

Sterownik w pełni Javowy z serwerem pośredniczącym

Sterownik porozumiewa się z serwerem pośredniczącym za pomocą protokołu sieciowego, a serwer używa JDBC typu 1 lub 2, żeby łączyć się z bazą danych



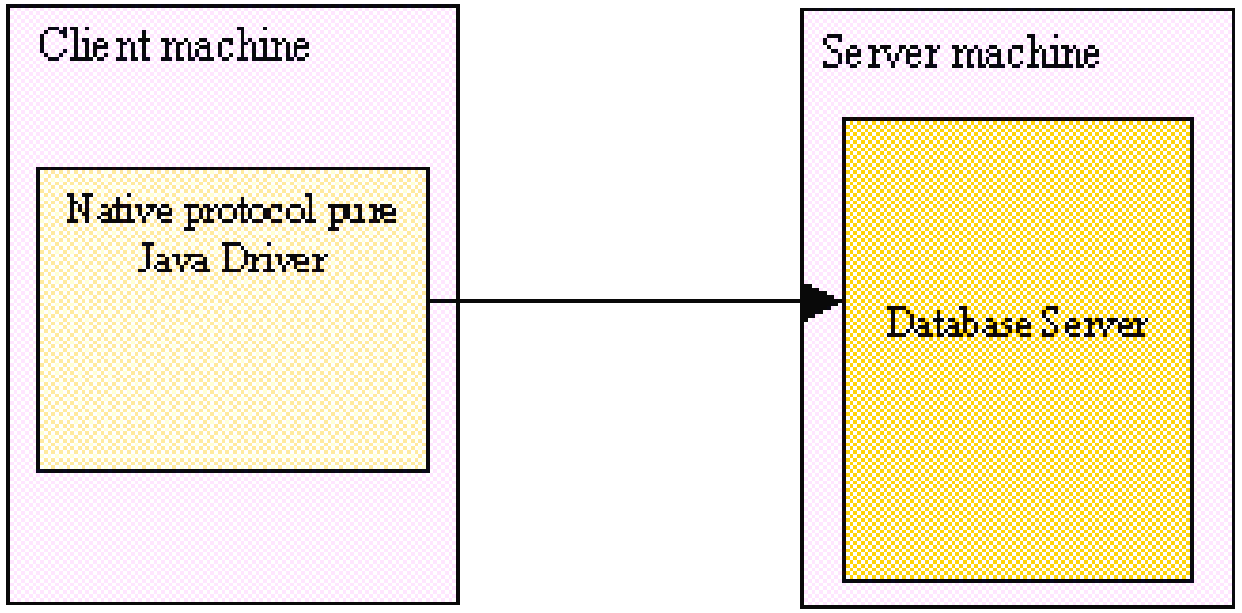
JDBC – typ 3

- Za:
  - sterownik po stronie klienta jest lekki, bez bibliotek odpowiedzialnych za konkretną obsługę bazy
  - sterownik najczęściej wspiera cache'owanie, równoważenie obciążenia, funkcje administracyjne (np. logowanie, nadzór)
- Przeciw:
  - potrzebny jest serwer pośredniczący ze sterownikiem typu 1 lub 2

JDBC – typ 4

Sterownik w pełni Javowy ze specyficznym protokołem

Sterownik w pełni napisany w Javie, porozumiewający się bezpośrednio z bazą danych po specyficznym dla niej protokole



JDBC – typ 4

- Za:
  - możliwość dynamicznego ładowania
  - nie trzeba instalować dodatkowych bibliotek na kliencie ani serwerze
  - wysoka wydajność (nie ma potrzeby tłumaczenia, jak w przypadku typów 1, 2, 3)
- Przeciw:
  - każdy system bazodanowy potrzebuje własnego sterownika

Składniki (interfejsy) JDBC 3.0

- JDBC składa się z ~20 interfejsów. 8 z nich jest kluczowych. W MySQL i HSQL większość z nich ma ~4000 linii kodu. Wniosek: implementuję podstawowe wymagania.
- Sterownik JDBC nie musi wspierać operacji, których nie wspiera sama baza danych
- Istnieje SUNowska JDBC API Test Suite 1.3.1 testująca czy sterownik spełnia nałożone wymagania

Składniki (interfejsy) JDBC 3.0

- Interfejsy podstawowe (trzeba zaimplementować je w pełni):
  - java.sql.Driver – nawiązuje połączenie z DB
  - java.sql.DatabaseMetaData – zwraca różne informacje o DB (mnóstwo metod typu „Czy DB obsługuje...”)
  - java.sql.ResultSetMetaData – zwraca różne informacje o kolumnach w zbiorze wyników (liczba, nazwy, typy kolumn, czy nulle są dozwolone itp.)

Składniki (interfejsy) JDBC 3.0

- Interfejsy podstawowe (niektóre metody są opcjonalne, jeśli np. DBMS ich nie wspiera):
  - CallableStatement – interfejs do wywoływania procedur SQL. Nie zamierzam implementować
  - Connection – połączenie (sesja) z DB. To tu są commit, rollback, close i tworzenie zapytań
  - PreparedStatement – prekompilowane zapytanie. Nie zamierzam implementować
  - ResultSet – ogromny interfejs do poruszania się po wierszach i kolumnach wyniku zapytania
  - Statement – zapytanie. Wykonujemy je i odbieramy wynik

Moje cele

- Napisać sterownik typu 4 zgodny z JDBC 3.0, ew. 4.0. JDBC 4.0 nie jest tak dobrze udokumentowany i rozszerza 3.0, więc na potrzeby LoXiMu wymagania JDBC 3.0 wystarczą, potem można oczywiście rozwijać.
- Napisać dobry, gruntownie przetestowany i udokumentowany kod umożliwiający późniejszy rozwój (lepiej podstawowe wymagania na 100 % niż więcej na 70 %)
- Sterownik JDBC ma sens tylko jeśli będzie później wykorzystywany

Moje cele

- Przejść jak najwięcej testów SUNa JDBC API Test Suite 1.3.1 z zagadnień objętych pracą magisterską
- Na SZBD-lab:
  - Implementacja protokołu sieciowego Piotrka Tabora. Integracja z serwerem Marka Dopiery
  - Kryterium sukcesu: sterownik łączy się (i rozłącza) z LoXiMem
- Na pracę magisterską:
  - Priorytet: wysyłanie zapytań SELECT i odbieranie wyników
  - Jeśli okaże się proste: UPDATE i DELETE

Z czego skorzystam?

- Z doświadczeń Marcina Pieli w egzekutorze SQL do LoXiM#, żeby skonwertować podejście relacyjne JDBC do potrzeb LoXiMu
- Z opisu interfejsu quasi-JDBC Odry
- Ze sterowników open-source: HSQLDB (Hypersonic) i MySQL. Oraz być może innych.
- Dokumentacja JDBC 3.0 i 4.0 SUNa
- JDBC API Test Suite 1.3.1
- Java 6