Table of Contents

Mit JVx werden üblicherweise Multi-Tier Anwendungen entwickelt, mit dem Schwerpunkt auf Datenbanken. Um den Datenaustausch zwischen Client und Enterprise Tier zu ermöglichen, wird eine Transportschicht benötigt. Diese ist in JVx einerseits sehr abstrakt definiert und andererseits wurde bereits eine Implementierung basierend auf http(s) umgesetzt.

Die Transport- bzw. Kommunikationsschicht wurde Protokoll unabhängig designed. Dadurch ist es problemlos möglich, die Kommunikationsklassen, ohne Anpassungen, für unterschiedlichste Protokoll Implementierungen wiederzuverwenden.

Nachfolgend ein Beispiel für die Protokoll unabhängige Verwendung:

```
HttpConnection con = new HttpConnection(URL);

MasterConnection macon = new MasterConnection(pConnection);
macon.setApplicationName("demo");
macon.setUserName("demo");
macon.setPassword("demo");
macon.open();
```

Durch das http Protokoll wird definiert, das ein ein Web- bzw. Applikationsserver eingesetzt werden muss. Dieser Umstand erschwert natürlich die Entwicklung, da immer ein Applikationsserver gestartet werden muss bevor die eigentliche Business Logik getestet werden kann. Natürlich gibt es mit Jetty einen leichtgewichtigen Applikationsserver oder auch die Eclipse WTP mit integrierter Tomcat Unterstützung. Dennoch müssen Abhängigkeiten berücksichtigt und Konfigurationen durchgeführt werden. Darauf kann auch gerne verzichtet werden. Vor allem wenn es um die Lokalisierung von Kommunikationsfehlern geht. Je weniger Komponenten berücksichtigt werden müssen, umso besser gestaltet sich die Suche.

Um den Entwickler bestmöglich zu unterstützen enthält JVx neben der HttpConnection auch die VMConnection und DirectServerConnection.

Die VMConnection kann als pendant zur HttpConnection gesehen werden, mit dem Unterschied das die Kommunikation ohne Applikationsserver und one das http Protokoll auskommt. Der Server wird automatisch in der aktuellen VM gestartet und wird wie auch bei der HttpConnection über Streams angesprochen. Die zu übertragenden Objekte werden immer serialisiert.

Für den Entwickler ist diese Art der Kommunikation schon eine enorme Vereinfachung, doch es geht noch etwas besser.

Sowohl die VMConnection als auch die HttpConnection serialisieren bzw. deserialisieren die Objekte. Wenn der Server bereits in der selben VM wie der Client läuft, könnte auf die Serialisierung doch ganz verzichtet werden.

Genau für diesen Fall wird die DirectServerConnection verwendet. Damit werden Server Funktionen direkt über die Server Klasse aufgerufen und Objekte direkt an Methoden übergeben. Die Serialisierung entfällt und das wirkt sich natürlich auch positiv auf die Performance aus.

Hinweis

Der Entwickler muss zu keiner Zeit auf das Kommunikationsprotokoll Rücksicht nehmen, da dieses von JVx gekapselt wird. Er muss sich jedoch bewusst sein, daß zwischen DirectServerConnection und

VMConnection bzw. HttpConnection ein Unterschied im produktiven Betrieb besteht. Denn die Objekt Serialisierung muss bei der Fehlersuche berücksichtigt werden. Außerdem wird bei VMConnection und der DirectServerConnection, mit implizit gestartetem Server, der Server beim Applikationsende ebenfalls beendet.

Im Entwicklungsprozess empfehlen wir den Einsatz der DirectServerConnection. Wenn spezielle Serializer implementiert oder Serialisierungsfehler debugged werden sollen, führt kein Weg an der VMConnection vorbei.

Im produktiven Betrieb ist die HttpConnection oder eine spezielle IConnection Implementierung notwendig.