Sequoia 2.4 Tutorial

von
Rainer Brinkmöller

Inhaltsverzeichnis

Einleitung.... 2
Voraussetzungen zur Einrichtung der Beispielumgebung.... 3
Sequoia Installation (grafik modus).... 3
Sequoia Installation (silent setup / unattended installation).... 8
Entpacken der binären Distribution im Zip-Fromat.... 10
Script Dateien zum Starten des Controller und der Console.... 10
Konfigurationsdateien.... 10
Erstellen der Beispieldatenbanken.... 16
Die Beispielapplikation.... 17
Die Testumgebung.... 19
Sequoia Starten.... 20
Die ersten Administrationstätigkeiten.... 21
Der erste Applikationszugriff.... 22
Testszenario – Ausfall einer backend Datenbank.... 23
  Tabelle löschen.... 23
  Applikationszugriff testen.... 23
  Tabelle wiederherstellen.... 23
  Konsistenten Zustand der Sequoia-Umgebung herstellen.... 24
  Fazit.... 25
Testszenario – Datenbanksicherung erstellen.... 26
  Ein paar Worte zu Octopus.... 26
  Vorbereitung der backend Datenbanksicherung.... 27
  Datenbanksicherung durchführen.... 27
  Backend reaktivieren.... 28
  Fazit.... 28
Eclipse als grafisches Administrationsfrontend.... 29
Sequoia Controller als Dienst unter Verwendung des Java Service Wrapper.... 29
Anhang A – Inhalt der Datei wrapper.conf.... 30
Epilog, Danksagung und Copyright-Hinweis.... 33

Einleitung

Sequoia (http://sequoia.continuent.org/HomePage) ist eine Datenbank Cluster Middleware, die jegliche Java(TM)-Applikationen (oder Servlet, bzw. EJB(TM)-Container, etc.) transparenten JDBC(TM)-Zugriff auf Datenbank-Cluster ermöglicht.
Dazu ist es nicht notwendig, Anpassungen an den Client-Applikationen[1], an Applikationsserver- oder Datenbankserversoftware vorzunehmen. Sie müssen lediglich den Datenbanktreiber durch den Sequoia Treiber ersetzen.
Sequoia ist ein freies, Open Source Projekt. Eine Fortsetzung des C-JDBC Projektes (http://c-jdbc.objectweb.org) durchgeführt vom ObjectWeb Consortium (http://www.objectweb.org/). Sequoia ist lizenziert unter der Apache v2 Lizenz (http://www.apache.org/licenses/LICENSE-2.0.html) während C-JDBC unter der (LGPL) (http://www.gnu.org/copyleft/lesser.html) GNU Lesser General Public License erhältlich ist .
Sequoia stellt ebenfalls Treiber für Applikationen zur Verfügung die nicht mit Java entwickelt wurden. Diese Entwicklungen finden Sie im Carob Projekt (http://carob.continuent.org). Ein Eclipse-Plugin für Sequoia ist ebenfalls vorhanden. Dieses Plugin finden Sie im Oak Projekt (http://oak.continuent.org).

Auf den folgenden Seiten möchte ich Ihnen einige Basisfunktionalitäten von Sequoia anhand einfacher Beispiele näher bringen. Auf die Demos, welche dem Produkt beigefügt sind, wird nicht näher eingegangen. Das Ziel ist es, Schritt für Schritt einige mögliche Anpassungen darzustellen um Ihnen die Basis der Konfiguration und administrativen Tätigkeiten bei der Einbindung von Sequoia zu vermitteln. Da hier eine Beispielumgebung dargestellt wird, erwarten Sie bitte keine vollständige Anleitung der gesamten Konfiguration bzw. aller Administrationsbefehle. Weiterführende Dokumentation können Sie über die Web-Präsenz des Projektes beziehen.
Diese Anleitung richtet sich an Entwickler, Administratoren und Interessenten, welche Erfahrung in der Einrichtung von Applikationen mit Datenbankanbindung haben.
Bei technischen Problemen oder Fragen zum Produkt wenden Sie sich bitte an die Sequoia-Mailinglist: https://forge.continuent.org/mailman/listinfo/sequoia

[1] Eine Anpassung der Applikation kann dann notwendig werden, wenn diese für eine spezielle Datenbanksoftware geschrieben wurde und alle Verbindungsdaten bereits im Quelltext der Applikation enthalten sind. In den meisten Fällen werden diese Daten jedoch in externen Konfigurationsdateien gespeichert um die Verwendung alternativer Datenbanksoftware zu ermöglichen.

Verwendete Komponenten

- Betriebssystem:
  Microsoft - Im Rahmen der Erstellung dieses Tutorial wurde Windows 2000 Professional verwendet.
  Weitere mögliche Betriebssysteme: Solaris, HP-UX, AIX, Unix, Linux, *BSD, etc. (da Sequoia 100% Java(TM) ist, können alle Betriebssysteme verwendet werden für die es Java gibt)

- Datenbank Software:
  PostgreSQL 8.1 (http://www.postgresql.org/)
  Weitere mögliche Datenbanken: MySQL, Oracle, DB2, Microsoft SQL Server, etc.

- Datenbanktreiber: sequoia-driver.jar (Sequoia 2.4) und postgresql-8.1dev-403.jdbc2ee.jar
  (je nach verwendeter Datenbank ist der entsprechende JDBC-Treiber des Typ1, 2, 3 oder 4 zu verwenden. Sie können auch einen ODBC-Treiber mit einer JDBC-ODBC-Bridge verwenden.
  Den Sequoia Datenbanktreiber finden Sie nach der Installation im Unterverzeichnis 'drivers'. Den PostgreSQL-Datenbanktreiber erhalten Sie unter http://jdbc.postgresql.org/download.html)

- Java Software Development Kit: Zum Beispiel j2sdk-1_4_2_10-windows-i586-p.exe
  (http://java.sun.com/j2se/1.4.2/download.html)

- grafische Adminstrationsoberfläche:
  Eclipse (http://www.eclipse.org/downloads/index.php)
  inkl. Plug-in vom Oak Projekt (http://oak.continuent.org/HomePage)

- Java Service Wrapper (http://wrapper.tanukisoftware.org/doc/english/introduction.html)

- Java Tool für die Ausführung von insert, select und delete Befehlen.
  (Der Quelltext für einfache Test-Applikationen sind in diesem Dokument enthalten)

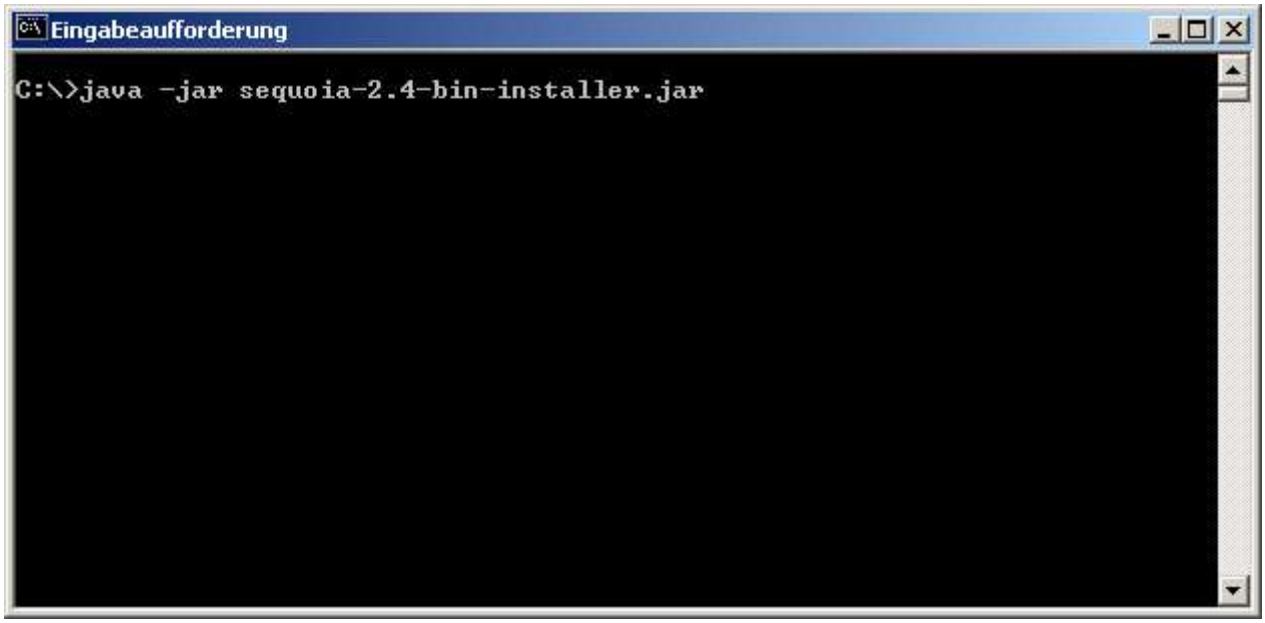
- Sequoia 2.4 (https://forge.continuent.org/frs/?group_id=6)

Voraussetzungen zur Einrichtung der Beispielumgebung

Sie benötigen mindestens einen PC mit eingerichteter Netzwerkkarte. Die Verwendung einer IP-Adresse wie 192.168.0.1 oder 10.0.0.1 ist ausreichend. Ein Betriebssystem Ihrer Wahl sollte vorinstalliert und konfiguriert sein. Sofern Sie die hier vorgestellten Beispielkonfigurationen übernehmen wollen, sollten Sie ein Microsoft Windows(TM) Betriebssystem verwenden (z.B. Microsoft Windows 2000 Professional oder Server bzw. Microsoft Windows 2003).
Ausreichend freier Festplattenplatz (ca. 150 MB für PostgreSQL und ca. 20 MB für Sequoia) und Arbeitsspeicher (512 MB empfohlen). Installieren Sie PostgreSQL 8.1 (C:\Lab\PostgreSQL) und das Java Software Development Kit (C:\Lab\j2sdk1.4.2_10). Setzen Sie alle notwendigen globalen Umgebungsvariablen (Insbesondere JAVA_HOME= C:\Lab\j2sdk1.4.2_10) und erweitern Sie die PATH Variable um den Eintrag des Pfades zum Java-Bin-Verzeichnis (Ausschnitt: PATH=.;C:\Lab\j2sdk1.4.2_10\bin;...)

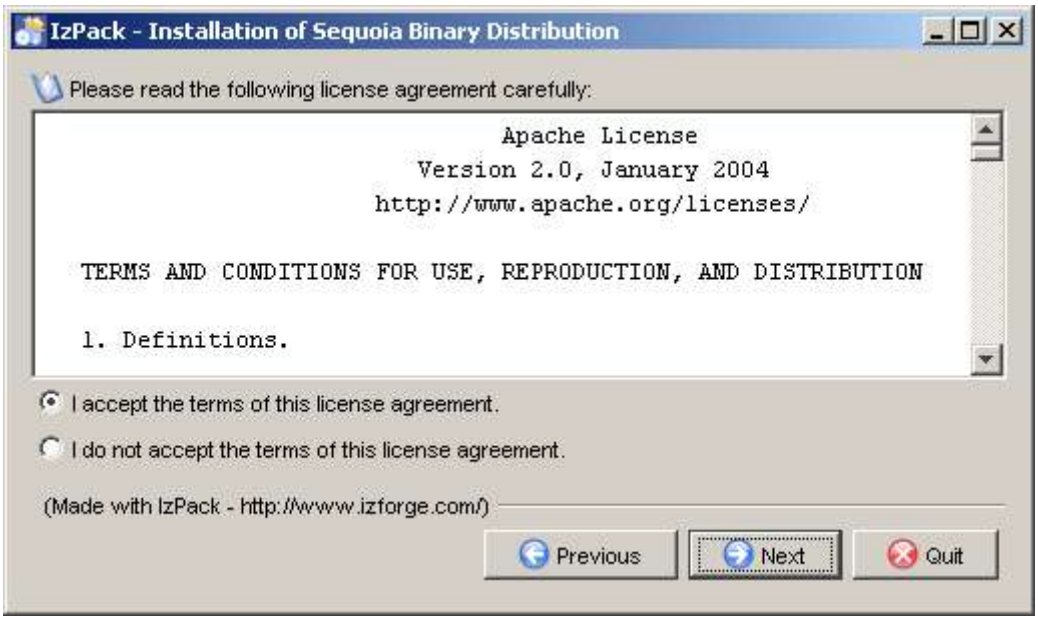
Sequoia Installation (grafik modus)

Nichts leichter als das. Eine sogenannte wwf (weiter, weiter, fertig) Installation. Hier die Screenshots.

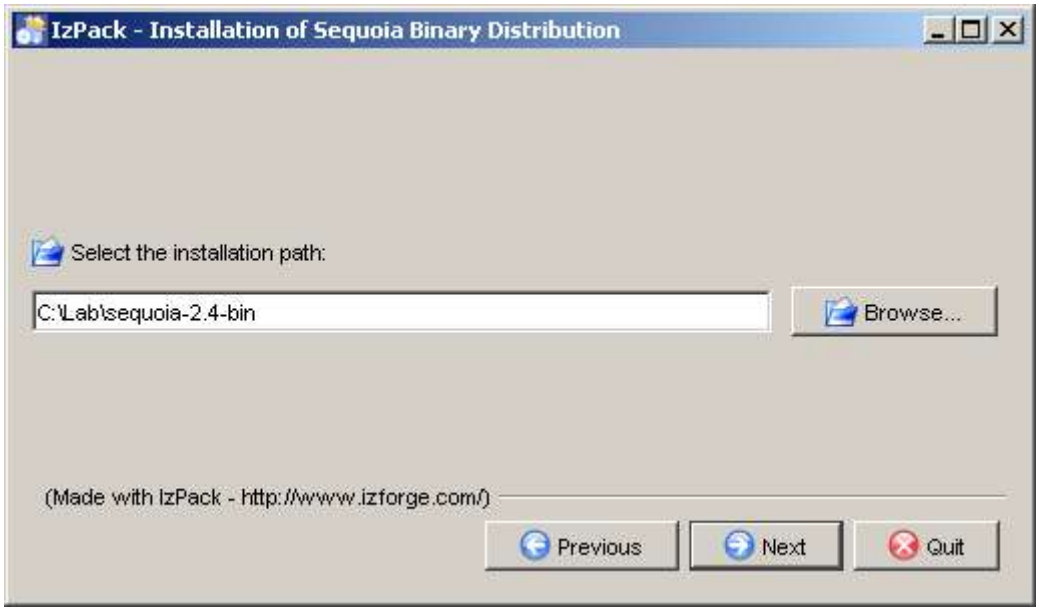


Nachdem Sie den Befehl “java -jar sequoia-2.4-bin-installer.jar“ ausgeführt haben wird die grafische Oberfläche des Installationsprogrammes gestartet.

Aktivieren Sie den Radiobutton “I accept the terms of this license agreement“ um die Lizenzbedingungen zu akzeptieren und die Installation fortsetzen zu können (anderenfalls ist an dieser Stelle der Spaß für Sie vorbei).

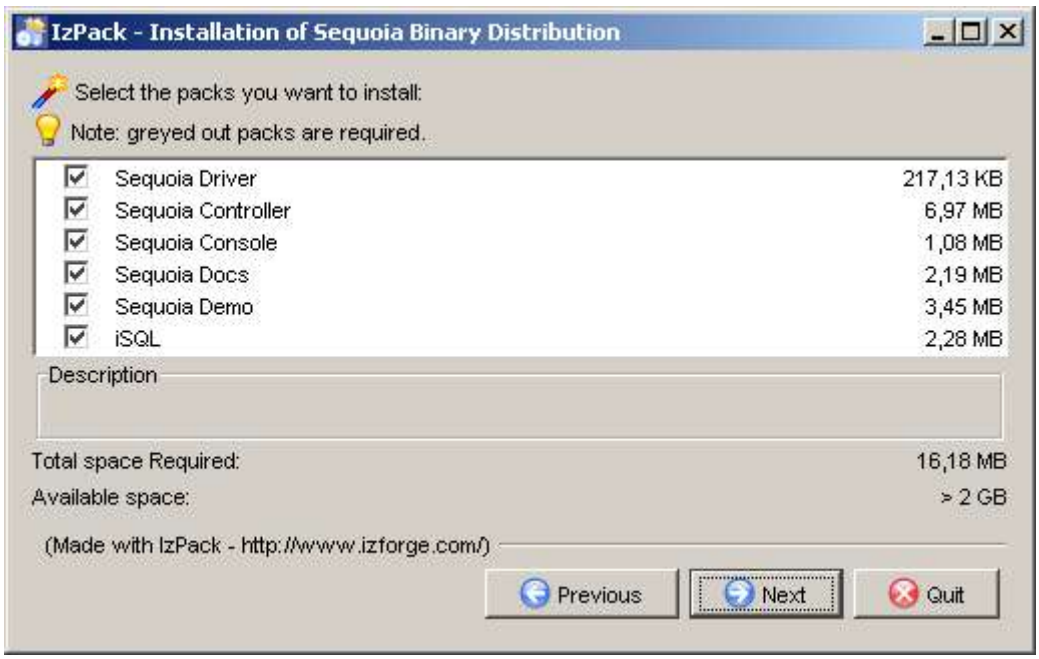


Aktivieren Sie den Button “Next“ um zum nächsten Dialogfenster zu wechseln.

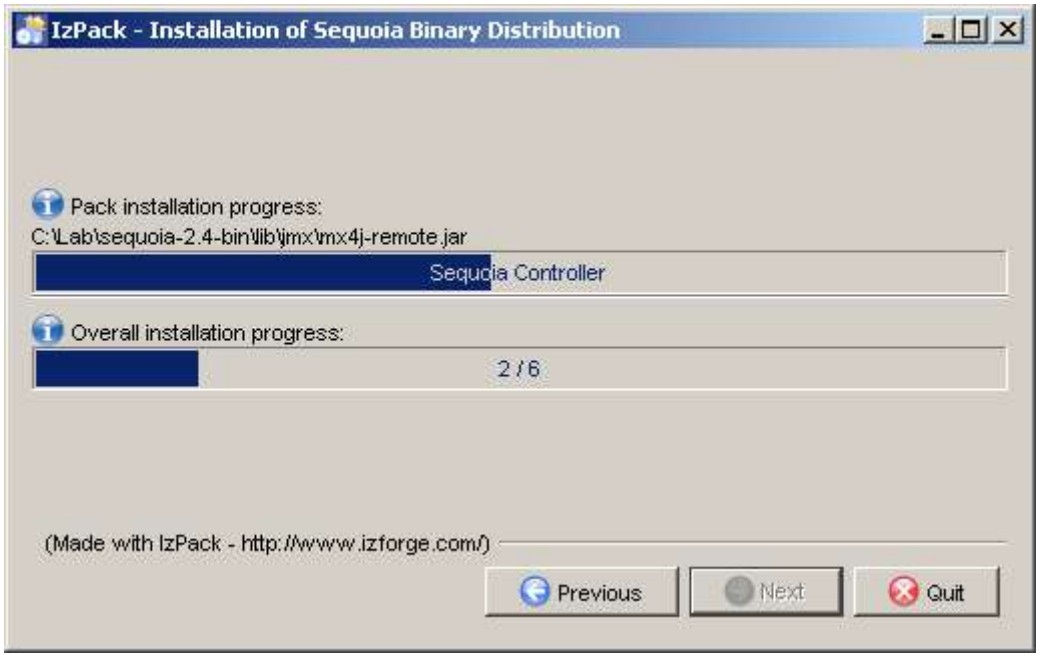


Passen Sie den Installationspfad dahingehend an, daß dieser auf das in der Abbildung dargestellte Verzeichnis verweist.

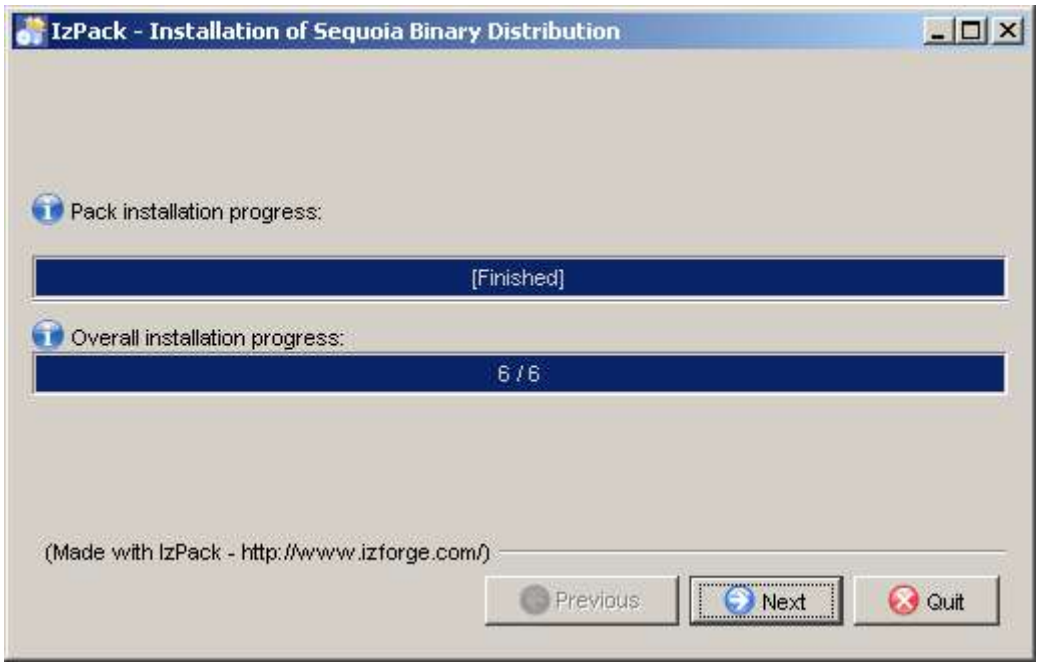
Da dieses Verzeichnis noch nicht existiert muss es erstellt werden. Dies ist der Hinweis, daß die Installation dies für Sie übernimmt.



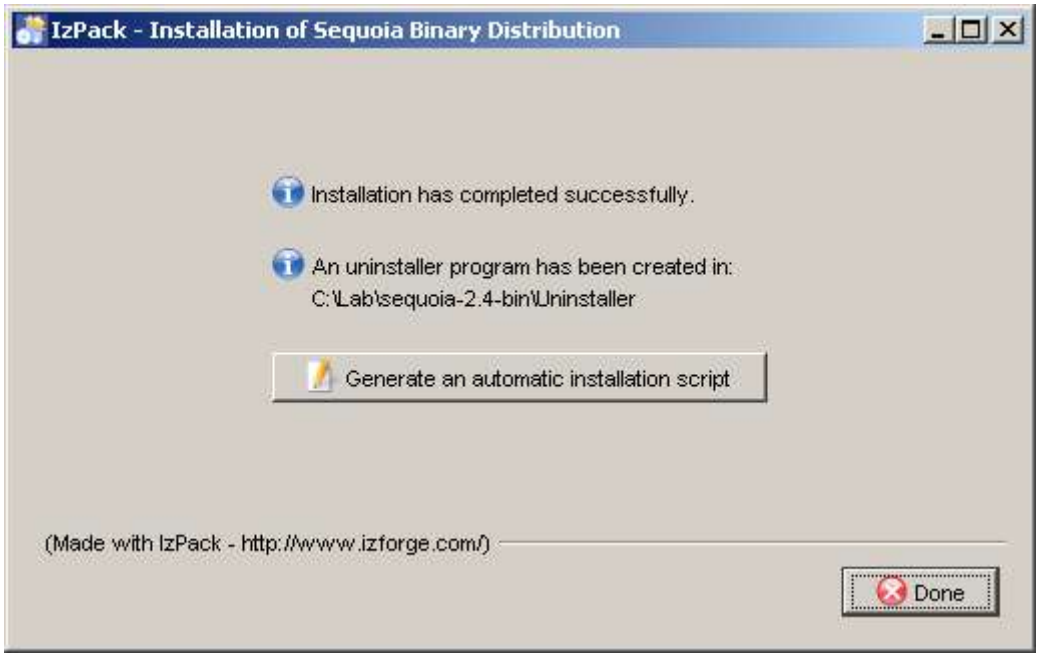
In einer produktiven Umgebung werden nicht alle der hier aufgeführten Komponenten benötigt. Für unsere Tests behalten Sie die Vorauswahl bei und fahren mit der Installation fort.



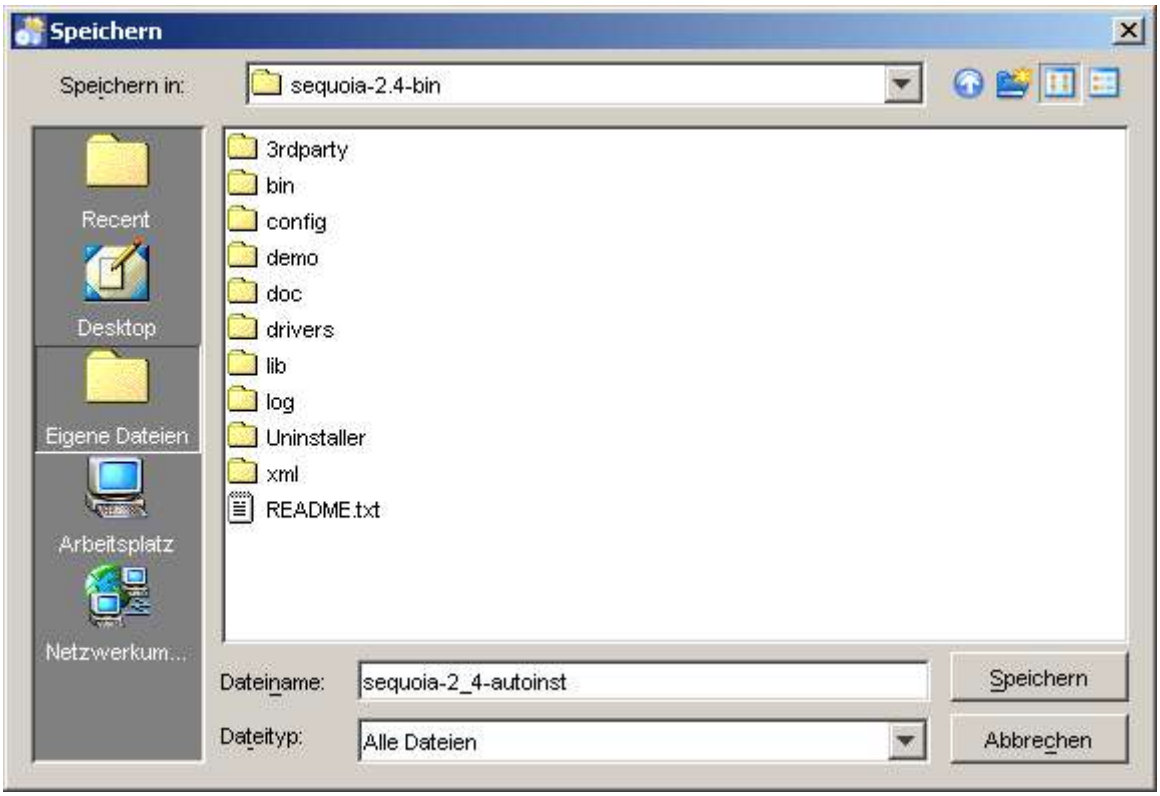
Anhand der Fortschrittsbalken erkennen Sie den Installationsstatus.



Sobald alle Komponenten installiert sind können Sie zum letzten Dialogfenster wechseln, indem Sie den Button “Next“ aktivieren.



Sie erhalten hier die Möglichkeit ein Installationsscript zu erstellen um eine automatische Installation mit der zuvor getroffenen Auswahl zu wiederholen, oder auf weiteren Computern durchzuführen. Aktivieren Sie hierzu den Button “Generate an automatic installation script“.



Das Dialogfenster zum Speichern wird geöffnet. Wählen Sie ein Zielverzeichnis und einen Dateinamen und aktivieren Sie den Button “Speichern“.

Möchten Sie dieses Script nicht erstellen, betätigen Sie den Button “Done“ im vorherigen Dialogfenster um die Installation abzuschließen.

Sequoia Installation (silent setup / unattended installation)

Im vorherigen Kapitel wurde die Installation mit grafischer Oberfläche beschrieben. Hierbei wurde abschließend aufgezeigt, daß diese Installationsmethode ermöglicht ein “automatic installation script“ zu erstellen. Dieses Script können Sie dazu verwenden ein silent setup (auch unattended installation genannt) durchzuführen.
Sie müssen jedoch nicht erst eine Installation mit grafischer Oberfläche durchführen um die Antwortdatei zu erstellen. Sie können diese auch selbst erzeugen. Im folgenden wird der Inhalt dargestellt und erläutert.

```
<AutomatedInstallation langpack="eng">
   <com.izforge.izpack.panels.HelloPanel/>
   <com.izforge.izpack.panels.LicencePanel/>
   <com.izforge.izpack.panels.TargetPanel>
      <installpath>C:\Lab\sequoia-2.4-bin</installpath>
   </com.izforge.izpack.panels.TargetPanel>
   <com.izforge.izpack.panels.PacksPanel>
      <selected>
         <pack index="0"/>
         <pack index="1"/>
         <pack index="2"/>
         <pack index="3"/>
         <pack index="4"/>
         <pack index="5"/>
      </selected>
   </com.izforge.izpack.panels.PacksPanel>
   <com.izforge.izpack.panels.InstallPanel/>
   <com.izforge.izpack.panels.FinishPanel/>
</AutomatedInstallation>
```

Lassen Sie mich Ihnen den Inhalt kurz beschreiben. Dies wird Ihnen ermöglichen, ein an Ihre Anforderungen angepasstes Installationsscript zu erstellen.

```
<AutomatedInstallation langpack="eng">
...
</AutomatedInstallation>
```

Alle weiteren Informationen müssen zwischen diesen Tags eingefügt werden. Der Wert von langpack bestimmt die Sprache, in welcher die Installation durchgeführt wird. Hierbei ist englisch nicht nur die voreingestellte Sprache, sondern derzeit auch die einzig mögliche.

```
<com.izforge.izpack.panels.HelloPanel/>
```

Dieser Tag wird genau an dieser Stelle erwartet. Er repräsentiert den Abschnitt der Installation, in welcher in der grafischen Oberfläche die Begrüßungsseite dargestellt wird.

```
<com.izforge.izpack.panels.LicencePanel/>
```

Dieser Tag wird genau an dieser Stelle erwartet. Er repräsentiert den Abschnitt der Installation, welcher in der grafischen Oberfläche die Bestätigung der Lizenzbedingungen abfragt. Im unbeaufsichtigten Modus wird vorausgesetzt, daß Ihnen diese bekannt sind und Sie ihnen auch zustimmen.

```
<com.izforge.izpack.panels.TargetPanel>
   <installpath>C:\Lab\sequoia-2.4-bin</installpath>
</com.izforge.izpack.panels.TargetPanel>
```

Dieser Bereich dient zur Angabe des Installationspfades. Selbstverständlich handelt es sich auch hierbei um eine Pflichtangabe.

```
<com.izforge.izpack.panels.PacksPanel>
   <selected>
      <pack index="0"/>
      <pack index="1"/>
      <pack index="2"/>
      <pack index="3"/>
      <pack index="4"/>
      <pack index="5"/>
   </selected>
</com.izforge.izpack.panels.PacksPanel>
```

Anhand dieses Abschnitt werden die zu installierenden Komponenten festgelegt. Die Zählung beginnt bei 0 und entspricht der Reihenfolge, wie sie in der grafischen Installation dargestellt wird. Somit steht der <pack index="0"/> für den Sequoia Driver, der <pack index="1"/> für den Sequoia Controller, der <pack index="2"/> für die Sequoia Console, der <pack index="3"/> für die Sequoia Dokumentationen, der <pack index="4"/> für die Sequoia Demo und der <pack index="5"/> für iSQL. Beabsichtigen Sie die Installation z.B. ohne die Demos durchzuführen erreichen Sie dies, indem Sie die gesamte Zeile mit dem Eintrag <pack index="4"/> löschen.

```
<com.izforge.izpack.panels.InstallPanel/>
```

Dieser Tag wird genau an dieser Stelle erwartet. Er repräsentiert den Abschnitt der Installation, welcher in der grafischen Oberfläche die Fortschrittsbalken darstellt.

```
<com.izforge.izpack.panels.FinishPanel/>
```

Dieser Tag wird genau an dieser Stelle erwartet. Er repräsentiert den Abschnitt der Installation, in welcher in der grafischen Oberfläche das abschließende Dialogfenster dargestellt wird.

Anhand des Befehls "java -jar sequoia-2.4-bin-installer.jar sequoi-2_4-autoInst" können Sie die Installation nun unbeaufsichtigt durchführen. Während der Installation werden die einzelnen Schritte angezeigt.

```
C:\>java -jar sequoia-2.4-bin-installer.jar sequoi-2_4-autoInst
[ Starting automated installation ]
[ Starting to unpack ]
[ Processing package: Sequoia Driver (1/6) ]
[ Processing package: Sequoia Controller (2/6) ]
[ Processing package: Sequoia Console (3/6) ]
[ Processing package: Sequoia Docs (4/6) ]
[ Processing package: Sequoia Demo (5/6) ]
[ Processing package: iSQL (6/6) ]
[ Unpacking finished. ]
[ Writing the uninstaller data ... ]
[ Automated installation done ]
```

Entpacken der binären Distribution im Zip-Fromat

Neben dem Installer-Jar wird die Distribution unter anderem auch als Archiv im Zip-Format zum Download bereit gestellt. Somit haben Sie auch die Möglichkeit, Sequoia in Ihrer Umgebung durch das Entpacken der Archivdatei unter Verwendung eines beliebigen Pfades bereitzustellen. Nach dem Entpacken der Zip-Datei sollten Sie die JDBC-Treiber der von Ihnen verwendeten Datenbanksoftware im Unterverzeichnis "drivers" hinterlegen und diese in der CLASSPATH-Variable in den Scripten zum starten des Controller und der Console aufnehmen. Setzen Sie zudem einleitend in diesen Batch-Scripten die Umgebungsvariablen JAVA_HOME und SEQUOIA_HOME.

Script Dateien zum Starten des Controller und der Console

Im Unterverzeichnis “bin“ finden Sie nach der Installation unter anderem die beiden Batch-Dateien controller.bat (zum Starten des Sequoia Controller) und console.bat (zum Starten der Administrations-Console im Befehlszeilenmodus). Eine Anpassung dieser Dateien ist nur in Ausnahmefällen notwendig (siehe oben). Es wird lediglich vorausgesetzt, daß die globale Umgebungsvariable JAVA_HOME korrekt eingerichtet ist und sich alle notwendigen Datenbank-Treiber im Unterverzeichnis “drivers“ befinden.

Konfigurationsdateien

Es gibt drei Arten von Konfigurationsdateien, welche ggf. für Ihre Umgebung angepasst werden müssen. Dabei handelt es sich um die Konfigurationsdateien mit der Endung .xml, .properties und .dtd, welche Sie in den Unterverzeichnissen “\config“, “\config\controller“, “\config\virtualdatabase“, “xml“ bzw. “\lib\octopus\xml\conf“ befinden oder erstellt werden müssen. Sofern das Unterverzeichnis “\config\virtualdatabase“ nach Ihrer Installation nicht vorhanden ist, liegt es wahrscheinlich daran, daß Sie die Komponenten für die Demo nicht mitinstalliert haben. In diesem Fall müssen Sie die entsprechende Verzeichnisstruktur nachträglich manuell erstellen.

Zunächst erstellen wir die Konfigurationsdatei “\config\controller\controller.xml“.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE SEQUOIA-CONTROLLER PUBLIC
 "-//Continuent//DTD SEQUOIA-CONTROLLER 2.4//EN"
 "http://sequoia.continuent.org/dtds/sequoia-controller-2.4.dtd">
<SEQUOIA-CONTROLLER>
  <Controller port="25322">
    <Report/>
    <JmxSettings>
      <RmiJmxAdaptor/>
    </JmxSettings>
    <VirtualDatabase configFile="vdb.xml" virtualDatabaseName="vDBtest"
      autoEnableBackends="false" checkpointName="Initial_empty_recovery_log"/>
  </Controller>
</SEQUOIA-CONTROLLER>
```

Einige wichtige Bereiche sollten kurz erläutert werden:

<Controller port="25322">

Hier geben Sie die Port-Nummer an, unter der Clients (Sequoia Treiber) sich verbinden. Die vorgegebene Port-Nummer ist 25322. Dieser Port kann angepasst werden, sofern er in Ihrer Umgebung bereits belegt ist. Beachten Sie aber, daß eine Port-Nummer unter 1024 voraussetzt, daß der Controller mit privilegierten Rechten ausgeführt wird (root User unter Unix bzw. Administrations- oder System-konto unter Windows).

<JmxSettings>
<RmiJmxAdaptor/>
</JmxSettings>

Anhand dieses Eintrags wird der RMI-Adapter aktiviert, welcher benötigt wird um den Controller lokal oder von einem fernen Client zu administrieren (Console, Eclipse-Plugin oder eigenes JMX-Tool).

<VirtualDatabase configFile="vdb.xml" virtualDatabaseName="vDBtest"
  autoEnableBackends="false" checkpointName="Initial_empty_recovery_log"/>

Das VirtualDatabase-Element enthält alle notwendigen Informationen zur Konfiguration der virtuellen Datenbank.
Der Eintrag configFile="vdb.xml" weist darauf hin, daß die Konfiguration der virtuellen Datenbank, in der Datei vdb.xml vorgenommen wurde. Das Programm setzt beim Starten des Controller voraus, daß diese Datei im Unterverzeichnis “\config\virtualdatabase“ vorhanden ist. Wurde diese Konfiguration in einem anderen Verzeichnis gespeichert, ist der gesamte Pfad anzugeben.
Der Eintrag virtualDatabaseName="vDBtest" verweist auf den Namen der virtuelle Datenbank deren Konfiguration beim Starten des Controller vom Programm ausgelesen werden soll.
Der Eintrag autoEnableBackends="false" verhindert die automatische Reaktivierung der backends vom letzten, als gesichert bekannten Status seit des letzten Herunterfahrens. Um die backends beim Starten automatisch zu reaktivieren, ersetzen Sie den Wert durch “true“. Der letzte mögliche Wert ist “force“, mit dem Sie einen neuen checkpoint einrichten.
Der Eintrag checkpointName="Initial_empty_recovery_log" definiert den Bezeichner des checkpoints, der zusammen mit dem recovery log verwendet wird um backends vom zugeordneten Status zu aktivieren. Wenn checkpoint nicht angegeben wird, wird der letzte bekannte checkpoint verwendet.

ACHTUNG: Verwenden Sie den Wert “force“ des Attribut “autoEnableBackends“ sehr sorgfältig. Ihre Daten könnten in einen inkonsistenten Zustand versetzt werden, wenn Sie einen ungültigen checkpoint zuordnen.

HINWEIS: Eine virtuelle Datenbank wird von einer Applikation wie eine reale Datenbank angesprochen. Es sind lediglich die Angabe zur verwendeten Klasse, der URL, der User-ID und des Passwortes entsprechend des Sequoia-Treiber und der Werte aus der Konfiguration der virtuellen Datenbank anzupassen. Sequoia leitet alle SQL-Befehle an die bekannten backend Datenbanken weiter und speichert die schreibenden SQL-Befehle (INSERT, UPDATE, DELETE, etc.) zusätzlich in der Recovery-Datenbank.

Lassen Sie uns nun die Konfigurationsdatei “\config\virtualdatabase\vdb.xml“ erstellen.

```
<?xml version="1.0" encoding="UTF8"?>
<!DOCTYPE SEQUOIA PUBLIC "-//Continuent//DTD SEQUOIA 2.4//EN"
"http://sequoia.continuent.org/dtds/sequoia-2.4.dtd">

<SEQUOIA>

 <VirtualDatabase name="vDBtest">

  <Monitoring>
   <SQLMonitoring defaultMonitoring="off">
    <SQLMonitoringRule queryPattern="^select" caseSensitive="false"
     applyToSkeleton ="false" monitoring="on"/>
   </SQLMonitoring>
  </Monitoring>

  <Backup>
   <Backuper backuperName="Octopus"
    className="org.continuent.sequoia.controller.backup.backupers.OctopusBackuper"
    options="zip=true"/>
  </Backup>

  <AuthenticationManager>
   <Admin>
     <User username="aduser" password="adsecret"/>
   </Admin>
   <VirtualUsers>
    <VirtualLogin vLogin="apuser" vPassword="secretcon"/>
   </VirtualUsers>
  </AuthenticationManager>

  <DatabaseBackend name="testNode1" driver="org.postgresql.Driver"
   url="jdbc:postgresql://localhost:5432/testdb1" connectionTestStatement="select now()">
   <ConnectionManager vLogin="apuser" rLogin="dbuser" rPassword="secret">
    <VariablePoolConnectionManager initPoolSize="10" minPoolSize="5"
     maxPoolSize="50" idleTimeout="30" waitTimeout="10"/>
   </ConnectionManager>
  </DatabaseBackend>

  <DatabaseBackend name="testNode2" driver="org.postgresql.Driver"
   url="jdbc:postgresql://localhost:5432/testdb2" connectionTestStatement="select now()">
   <ConnectionManager vLogin="apuser" rLogin="dbuser" rPassword="secret">
    <VariablePoolConnectionManager initPoolSize="10" minPoolSize="5"
     maxPoolSize="50" idleTimeout="30" waitTimeout="10"/>
   </ConnectionManager>
  </DatabaseBackend>

  <RequestManager>
   <RequestScheduler>
    <RAIDb-1Scheduler level="passThrough"/>
   </RequestScheduler>
```

```
    <RequestCache>
      <MetadataCache/>
      <ParsingCache/>
    </RequestCache>

    <LoadBalancer>
      <RAIDb-1>
        <WaitForCompletion policy="first"/>
        <RAIDb-1-LeastPendingRequestsFirst/>
      </RAIDb-1>
    </LoadBalancer>

    <RecoveryLog driver="org.postgresql.Driver"
      url="jdbc:postgresql://localhost:5432/testrec" login="recdbuser" password="secretrec">
      <RecoveryLogTable tableName="RECOVERY" idColumnType="BIGINT NOT NULL"
        vloginColumnType="VARCHAR NOT NULL" sqlColumnType="VARCHAR NOT NULL"
        extraStatementDefinition=",PRIMARY KEY (id)"/>
      <CheckpointTable tableName="CHECKPOINT"
        checkpointNameColumnType="VARCHAR NOT NULL"/>
      <BackendTable tableName="BACKEND"
        databaseNameColumnType="VARCHAR NOT NULL"
        backendNameColumnType="VARCHAR NOT NULL"
        checkpointNameColumnType="VARCHAR NOT NULL"/>
      <DumpTable tableName="DUMP" dumpNameColumnType="VARCHAR NOT NULL"
        dumpDateColumnType="VARCHAR NOT NULL"
        dumpPathColumnType="VARCHAR NOT NULL"
        dumpFormatColumnType="VARCHAR NOT NULL"
        checkpointNameColumnType="VARCHAR NOT NULL"
        backendNameColumnType="VARCHAR NOT NULL"
        tablesColumnType="VARCHAR NOT NULL"/>
    </RecoveryLog>

  </RequestManager>
 </VirtualDatabase>
</SEQUOIA>
```

Um den Rahmen nicht zu sprengen, werden im folgenden lediglich die fett formatierten Abschnitte der Konfigurationsdatei erläutert.

<VirtualDatabase name="**vDBtest**">

Der diesem Tag zugewiesene Wert muss identisch mit dem Wert von virtualDatabaseName in der Konfigurationsdatei controller.xml sein (in unserem Beispiel also “vDBtest“).

```
<AuthenticationManager>
 <Admin>
  <User username="aduser" password="adsecret"/>
 </Admin>
 <VirtualUsers>
  <VirtualLogin vLogin="apuser" vPassword="secretcon"/>
 </VirtualUsers>
</AuthenticationManager>
```

In diesem Bereich werden zwei Arten von User für die Authentifizierung eingerichtet. Im Bereich “Admin“ legen Sie die User-ID und das Paßwort für den administrativen Zugriff auf Sequoia fest (z.B. für die Anmeldung an der Console).
Im Bereich “VirtualUser“ legen Sie die User-ID und das Paßwort für den Zugriff zwischen Applikation und Sequoia fest (mit welchem User auf die virtuelle Datenbank zugegriffen wird).
Es besteht die Möglichkeit, in den Bereichen “Admin“ und “VirtualUser“ mehrere User-ID's und Paßwörter zu hinterlegen. Dazu fügen Sie in den jeweiligen Bereichen lediglich eine weitere Zeile mit den entsprechenden Werten hinzu. In unserem Beispiel sind diese Angaben jedoch ausreichend.

<DatabaseBackend name="testNode1" driver="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/testdb1" connectionTestStatement="select now()">

Diesen Tag finden Sie in der Konfiguration zweimal in fast identischer Form (das zweite mal im darauf folgenden Bereich). Hier wird die Verbindung zum realen Datenbank backend konfiguriert. In unserem Beispiel richten wir zwei identische Datenbank backends ein, um im Falle des Ausfalls der ersten Datenbank weiterhin Verfügbarkeit durch die zweite Datenbank zu erreichen.

Die Bezeichnung des ersten backend lautet in unserem Fall “testNode1“.

Der Treiber hängt von der backend Datenbank ab und ist in unserem Fall “org.postgresql.Driver“ (Sie erinnern sich, wir werden eine PostgreSQL 8.1 Datenbank verwenden). Die backend Datenbank wird über den Standard-Port 5432 auf der lokalen Maschine mit dem Namen “testdb1“ eingerichtet. Dementsprechend lautet die url “jdbc:postgresql://localhost:5432/testdb1“.
Insbesondere bei der Initialisierung und beim backup wird ein Testzugriff durchgeführt um die Erreichbarkeit der backend Datenbank zu verifizieren. Hierzu bedarf es eines entsprechenden Befehls, welcher als Wert von “connectionTestStatement“ hinterlegt wird.

Hier eine Liste der derzeit bekannten Werte für “connectionTestStatement“ je nach verwendeter Datenbanksofware:

- MySQL: select 1
- PostgreSQL: select now()
- Apache Derby: values 1
- HSQL: call now()
- SAP DB (MySQL MaxDB): select count(*) from versions
- Oracle: select * from dual
- Firebird: select * from rdb$types
- InstantDB: set date format "yyyy/mm/dd"
- Interbase: select * from rdb$types
- Microsoft SQL server 2000: select 1
- IBM DB2 (nur in der Version 8.1 FP 9a mit db2jcc Treiber verifiziert): values 1

Es kann durchaus vorkommen, daß Sie eine Version der hier aufgeführten Datenbanken mit einem JDBC(TM)-Treiber verwenden, welcher die aufgeführten Befehle nicht unterstützt. Oder Sie verwenden eine Datenbanksoftware, welche hier nicht aufgeführt ist. In diesem Fall sollten Sie die Dokumentation Ihrer Datenbanksofware zu Rate ziehen. Es besteht aber auch die Möglichkeit, eine Tabelle mit einer Spalte zu erstellen und hier z.B. den Wert “true“ zu hinterlegen. Als Wert von “connectionTestStatement“ geben Sie in diesem Fall einen entsprechenden Select Befehl an (z.B. “select * from CONTESTTBL where CONTESTVAL = 'true' “ - sofern Sie die Tabelle mit dem Namen “CONTESTTBL“ und die Spalte mit dem Namen “CONTESTVAL“ angelegt haben).

<ConnectionManager vLogin="apuser" rLogin="dbuser" rPassword="secret">

Die User-ID und das Paßwort für den Zugriff auf die backend Datenbank muss hier natürlich auch noch angegeben werden. Die Zuordnung erfolgt über die zuvor eingerichtete User-ID des virtuellen Users. Dies ermöglicht eine 1:1 Zuordnung im Rahmen der Anwenderberechtigungen.
Da wir bisher nur einen virtuellen User eingerichtet haben, werden wir diesen verwenden um eine Zuordnung zum, in unserem Beispiel einzigen, Datenbankanwender zu hinterlegen. Der Wert von "vLogin" (der virtuelle User) lautet also "apuser". Er repräsentiert den Datenbankanwender mit der User-ID "dbuser" und dem Paßwort "secret".

<RecoveryLog driver="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/testrec" login="recdbuser" password="secretrec">

Die Wiederherstellungsprotokolle (Sequoia Recovery Logs) enthalten die Informationen (die jeweiligen SQL-Statements) zu den Schreibzugriffen und Transaktionen zwischen den logischen checkpoints. Diese Protokolle können lediglich in einer Datenbank (bzw. in einem Datenbank-Cluster) gesichert werden.
Selbstverständlich werden auch die Parameter für den Zugriff auf diese Datenbank benötigt. Um den administrativen Aufwand so gering wie möglich zu halten empfiehlt es sich, auch hier dieselbe Datenbanksoftware wie für die Applikationsdatenbanken zu verwenden (auch wenn es möglich wäre, für jedes backend eine andere Datenbanksoftware zu verwenden, fallen mir im Moment nur wenige Beispiele ein, in denen dieses Vorgehen auch sinnvoll bzw. notwendig wäre).

In unserem Beispiel verwenden wir also wieder eine PostgreSQL Datenbank auf dem lokalen Computer. Wir richten diese Datenbank mit dem Namen "testrec" ein (driver="org.postgresql.Driver" url="jdbc:postgresql://localhost:5432/testrec"). Jedoch richten wir einen dedizierten User für den Zugriff auf diese Datenbank ein (login="recdbuser" password="secretrec").
Um die Tabellen müssen wir uns nicht weiter kümmern. Dies übernimmt Sequoia für uns (hierfür werden die weiteren Informationen in diesem Bereich benötigt).
Im Rahmen unseres Beispiels ist es nicht notwendig, an weiteren Kofigurationsdateien Anpassungen vorzunehmen. Dennoch möchte ich Ihnen einige aufzählen, welche Sie sich näher betrachten sollten, sofern Sie zum Beispiel andere Datenbanksoftware verwenden.

\config\controller_default.properties
(nehmen Sie hier nur Anpassungen vor, wenn Sie genau wissen was sie tun und es unumgänglich ist).

\xml\sequoia.dtd
(Dies ist die XML DTD - Formatierungsbeschreibung - für die Sequoia Konfigurationsdateien der virtuellen Datenbanken)

Weitere Informationen zu den ggf. notwendigen Anpassungen finden Sie in der Dokumentation, die mit der Software mitgeliefert wird (Sie finden diese unter "\doc\userGuide\html\userGuide.html").

Im Unterverzeichnis "\lib\octopus\xml\conf" finden Sie weitere Konfigurationsdateien im XML-Format für die Konfgurationen im Rahmen der Verwendung des mitgelieferten Backup-Tools Octopus.

Erstellen der Beispieldatenbanken

Nun ist es an der Zeit das Augenmerk auf die Datenbanken zu legen. Wie bereits mehrfach erwähnt, erstellen wir drei PostgreSQL Datenbanken. Zwei identische Datenbanken, die somit das ausfallsichere backend für den Datenzugriff einer Beispielapplikation repräsentieren. Und die Recovery Datenbank für die Sequoia Protokollierung.

In der Befehlszeilenumgebung setzen wir die folgenden drei Kommandos zur Erstellung der Datenbanken ab:

```
createdb --owner=dbuser --encoding=UTF8 --tablespace=pg_default testdb1
createdb --owner=dbuser --encoding=UTF8 --tablespace=pg_default testdb2
createdb --owner=recdbuser --encoding=UTF8 --tablespace=pg_default testrec
```

Damit ist die Vorbereitung für die Recovery Datenbank abgeschlossen. Die beiden anderen Datenbanken benötigen jeweils mindestens eine Tabelle. Die Tabelle für die beiden backend Datenbanken haben in unserem Beispiel folgenden Aufbau:

```
CREATE TABLE test_person ( person_name varchar(50) NOT NULL, person_id int4 NOT NULL );

ALTER TABLE test_person ADD CONSTRAINT pk_test_person_nr PRIMARY KEY(person_id);

CREATE SEQUENCE test_person_person_id_seq
   INCREMENT 1
   MINVALUE 1
   MAXVALUE 9223372036854775807
   START 1
   CACHE 1;

ALTER TABLE test_person ALTER COLUMN person_id
   SET DEFAULT nextval('test_person_person_id_seq'::regclass);
```

Sie erkennen schon, daß es sich hierbei um eine relativ einfache Datenbank handelt. Dennoch ist diese völlig ausreichend. Immerhin wollen wir uns ja auf die Konfiguration und Administration von Sequoia konzentrieren.

Die Beispielapplikation

Die Beispielapplikation gestalten wir ebenso einfach wie die Datenbanken. Alle Informationen, die für die Verbindung notwendig sind, sind bereits im Quelltext enthalten (hier fett formatiert - in der Regel sollten diese Informationen in entsprechenden Konfigurationsdateien gespeichert werden). In unserem Beispiel sind die Funktionen INSERT und SELECT völlig ausreichend. Allerdings ist die Test-applikation für den INSERT etwas fehleranfällig. Wenn Sie beim Ausführen vergessen einen Wert als Aufrufparameter mitzugeben, welcher in die Tabelle geschrieben wird bekommen Sie eine Exception als Antwort. Beachten Sie, daß dieses Programm nur dann funktioniert, wenn Sie sich bei der bisherigen Konfiguration an den Beispielvorgaben gehalten haben. Sie erhalten hier also keine professionelle Applikationen; den Anforderungen an unsere Beispielumgebung genügen sie aber.
Da wir den Standardport “25322“ verwenden (kann bei Bedarf in der controller.xml angepasst werden) müsste dieser nicht explizit angegeben werden.

```
// TestInsert.java
// -----------------------
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.io.*;

public class TestInsert {
  public static void main(String[] args) {

    String url = new String("jdbc:sequoia://localhost:25322/vDBtest");
    String user = new String("apuser");
    String password = new String("secretcon");

    try {
      Class.forName("org.continuent.sequoia.driver.DataSource");
    } catch ( ClassNotFoundException e ) {
      e.printStackTrace();
    }
    try {
      Connection dbcon = DriverManager.getConnection(url, user, password);
      Statement st = dbcon.createStatement();
      st.executeUpdate("insert into test_person (person_name) values ('" + args[0] + "')");
      st.close();
      dbcon.close();
    } catch ( SQLException e ) {
      e.printStackTrace();
    }
  }
}
```

Kopieren Sie den Quelltext in eine Datei mit dem Namen “C:\Lab\TestInsert.java“ (Achtung: Beachten Sie die Groß-/Kleinschreibung beim Dateinamen). Anschliessend kompilieren Sie den Quelltext mit dem Befehl “javac TestInsert.java“. Damit erhalten Sie die Datei TestInsert.class, die das eigentliche Programm darstellt, welches wir später ausführen werden.

Als nächstes benötigen wir noch ein Programm für die Ausführung eines SELECT-Befehls.

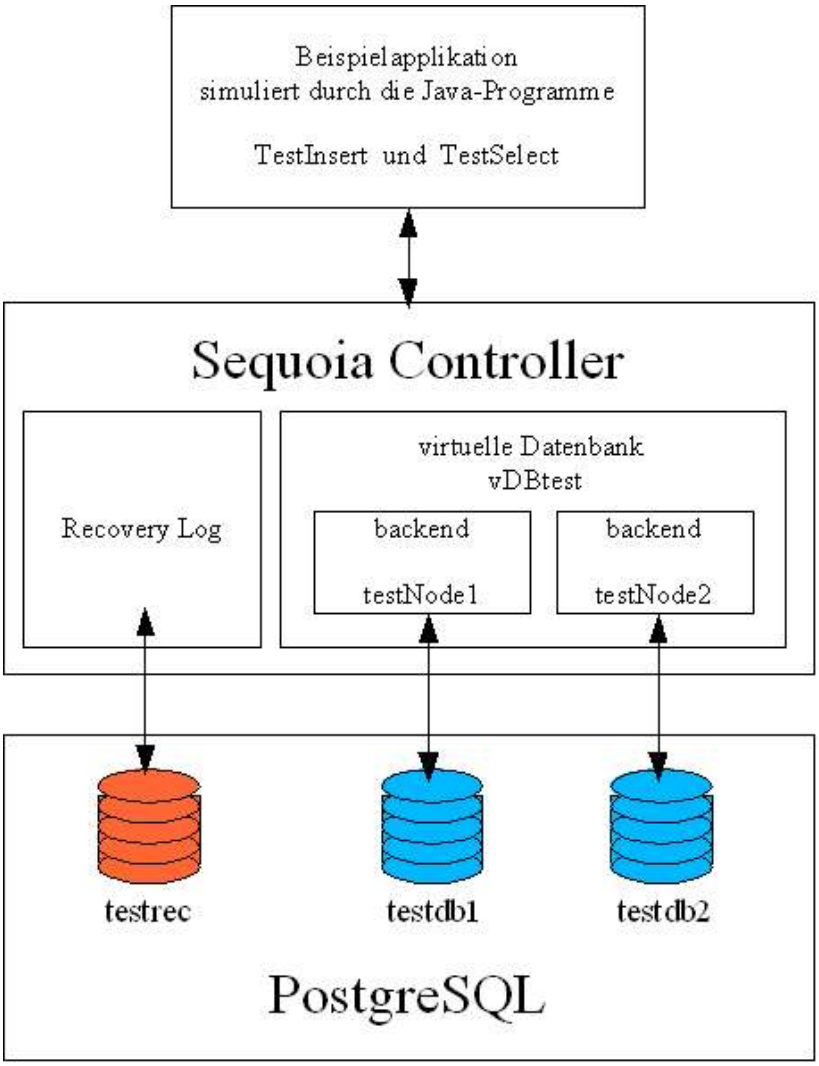
```
// TestSelect.java
// -----------------------
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.io.*;

public class  TestSelect {
  public static void main(String[] args) {

    int colCount;
    String url = new String("jdbc:sequoia://localhost:25322/vDBtest");
    String user = new String("apuser");
    String password = new String("secretcon");
    String dbselect = new String("SELECT * FROM TEST_PERSON");

    try {
      Class.forName("org.continuent.sequoia.driver.DataSource");
     } catch ( ClassNotFoundException e ) {
      e.printStackTrace();
    }
    try {
      Connection dbcon = DriverManager.getConnection(url, user, password);
      Statement st = dbcon.createStatement();
      ResultSet rs = st.executeQuery("select * from test_person");
      ResultSetMetaData meta = rs.getMetaData();
      colCount = meta.getColumnCount();

      System.out.println( meta.getColumnLabel(1) + "\t" + meta.getColumnLabel(2) );

      while ( rs.next() ) {
        System.out.println( rs.getString(1) + "\t\t" +  rs.getString(2) );
      }

      rs.close();
      st.close();
      dbcon.close();
    } catch ( SQLException e ) {
      e.printStackTrace();
    }
  }
}
```

Kopieren Sie den Quelltext in eine Datei mit dem Namen “C:\Lab\TestSelect.java“ (Achtung: Beachten Sie die Groß-/Kleinschreibung beim Dateinamen).
Anschliessend kompilieren Sie den Quelltext mit dem Befehl “javac TestSelect.java“.

Damit stehen uns nun auch die Testapplikationen zur Verfügung.

Die Testumgebung

Eine stark vereinfachte Darstellung der bis hierher eingerichteten Umgebung soll Ihnen ansatzweise verständlich machen, welche Rolle Sequoia in unserem Beispiel übernimmt.



Der Sequoia Controller nimmt alle Befehle (insert, select, etc.) anstelle eines JDBC-Treibers entgegen. Mehrstufige Verarbeitungsschritte innerhalb von Sequoia übernehmen die Authentifizierung und Authorisierung, das Scheduling, die weitere verteilte Übergabe der Befehle an die backend Datenbanken (die backend Datenbanken hier blau dargestellt) sowie die Sicherung der Anfragen via Recovery Log (die Recovery Datenbank hier orange dargestellt).

Sequoia Starten

Allem voran muß der Sequoia Controller gestartet werden. Dies erreichen wir, indem wir in einem Befehlszeilenfenster die Datei “C:\Lab\sequoia-2.4-bin\bin\controller.bat“ ausführen.
Es besteht die Möglichkeit hierfür einen Dienst einzurichten. Eine allgemein gültige Anleitung finden Sie bei tanukisoftware unter http://wrapper.tanukisoftware.org/doc/english/integrate-simple-win.html. Eine Vorlage für eine Konfigurationsdatei finden Sie nach der Installation hier im Unterverzeichnis C:\Lab\sequoia-2.4-bin\doc\examples\WindowsService. Der Dateiname lautet wrapper.conf.

Nachdem Sie die controller.bat ausgeführt haben erhalten Sie mehrere Ausgabezeilen, beginnend mit der Angabe des aktuellen Datums und der Uhrzeit, gefolgt von dem Ausgabetyp (INFO, WARN, ERROR) und weiteren Detailinformationen. Der Start ist abgeschlossen, wenn die letzte Zeile ähnlich der im folgenden dargestellten lautet:

2005-12-10 20:51:27,359 INFO controller.core.Controller Controller 192.168.0.1:25322 ready, listening to requests ...

Da der Controller zum ersten mal gestartet wurde, gibt es keinen bekannten Status der backends. Diese müssen nun initialisiert und gesichert werden.

Der Einsatz der Administrations-Console ist nun möglich. In unserem Fall wurden automatisch während des Startens die notwendigen Tabellen in der Recovery Datenbank erstellt. Was zu beweisen wäre – nun gut, dann setzten wir in einem weiteren Befehlsfenster den entsprechenden PostgreSQL Befehl ab:

```
testrec=# \dt
          Liste der Relationen
 Schema |     Name     |   Typ   | Eigentmer
--------+--------------+---------+-----------
 public | backend      | Tabelle | dbuser
 public | checkpoint   | Tabelle | dbuser
 public | dump         | Tabelle | dbuser
 public | pg_ts_cfg    | Tabelle | dbuser
 public | pg_ts_cfgmap | Tabelle | dbuser
 public | pg_ts_dict   | Tabelle | dbuser
 public | pg_ts_parser | Tabelle | dbuser
 public | recovery     | Tabelle | dbuser
(8 Zeilen)
```

An der Ausgabe erkennen Sie, daß neben den Basistabellen vier weitere mit den Namen “backend“, “checkpoint“, “dump“ und “recovery“ erstellt wurden. Wir brauchen uns hierum nicht weiter zu kümmern. Sollten Sie beim ersten Starten jedoch ERROR Meldungen erhalten haben, kann eine Ursache dafür sein, daß diese Tabellen nicht erstellt werden konnten. Prüfen Sie daher, ob diese vorhanden sind. Wenn nicht, könnte dies an fehlenden Berechtigungen des hinterlegten Users liegen.

Die ersten Administrationstätigkeiten

Die Administrations-Console wird gestartet, indem in einem Befehlszeilenfenster die Datei "C:\Lab\sequoia-2.4-bin\bin\console.bat" ausgeführt wird. Daraufhin sollten Sie folgende Ausgabe erhalten:

```
C:\Lab\sequoia-2.4-bin\bin\>console.bat
Launching the Sequoia controller console
Initializing Controller module...
Initializing VirtualDatabase Administration module...
Initializing SQL Console module...
Sequoia driver (Sequoia core v2.4) successfully loaded.
localhost:1090 >
```

Sie haben nun die Möglichkeit allgemeine Sequoia Administrationsbefehle auszuführen. Eine Liste der verfügbaren Befehle erhalten Sie, wenn Sie an diesem Prompt den Befehl "help" absetzen. Beachten Sie die Groß-/Kleinschreibung bei der Verwendung der Befehle.
Um die Administration der virtuellen Datenbanken durchführen zu können, müssen Sie am Consolen Prompt den Befehl "admin vDBtest" absetzen. Anschliessend werden Sie zunächst aufgefordert, die User-ID und dann das Paßwort dieses Users (in unserem Beispiel "adsecret") einzugeben:

```
localhost:1090 > admin vDBtest
Virtual database Administrator Login > aduser
Virtual database Administrator Password >                (wird während der Eingabe nicht angezeigt)
Ready to administrate virtual database vDBtest
vDBtest(aduser) >
```

Sie befinden sich nun auf einer Ebene der Console, auf der Ihnen spezielle Befehle zur Administration der virtuellen Datenbank zur Verfügung stehen. Dies erkennen Sie an dem veränderten Prompt, der als Name der virtuellen Datenbank, gefolgt von der verwendten User-ID in Klammern, dargestellt wird. Die hier verfügbaren Befehle erhalten Sie, wenn Sie an diesem Prompt den Befehl "help" absetzen. Sie verlassen diesen Modus, in dem Sie den Befehl "quit" am Prompt eingeben.

Da aufgrund des Wertes "false" des Attributs "autoEnableBackends" der Konfigurationsdatei "controller.xml" beim Starten des Controllers die Initialisierung nicht automatisch statt gefunden hat, sollten wir dies nun manuell ausführen:

```
vDBtest(aduser) > initialize testNode1
Virtual Database test has been successfully initialized from backend testNode1
vDBtest(aduser) >
```

Während der Initialisierung wurde das angegebene backend (testNode1) als Referenz backend gesetzt.

Abschließend müssen wir nur noch die backends aktivieren, bevor wir das erste mal die Beispielapplikation verwenden können. Sie können entweder alle backends gleichzeitig aktivieren...

```
vDBtest(aduser) > enable *
Enabling all backends from their last known checkpoints    (das dauert jetzt einen kleinen Moment)
vDBtest(aduser) >
```

...oder Sie setzen für jedes einzeln den Befehl "enable testNode1" und dann "enable testNode2" ab.

Der erste Applikationszugriff

Ab diesem Status ist es möglich unsere Beispielapplikationen zu verwenden. Öffnen Sie ein weiteres Befehlsfenster, wechseln Sie in das Verzeichnis C:\Lab und setzen Sie dort folgenden Befehl ab:

```
C:\Lab\>%JAVA_HOME%\bin\java TestInsert Einstein
```

Wenn keine Meldung erscheint, wurde der Befehl erfolgreich ausgeführt. Sollten Sie vergessen haben den Aufrufparameter “Einstein“ anzugeben, erhalten Sie folgende Fehlermeldung:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at pgdbSequoiaInsertTest.main(TestInsert.java:18)
```

Die Applikation übergibt im Erfolgsfall den Befehl:
“insert into test_person (person_name) values ('Einstein')“
an den Sequoia Treiber.
Dieser wiederum übergibt ihn einerseits an die beiden backend Datenbanken (über deren JDBC-Treiber)...

```
testdb1=# select * from test_person;
 person_name | person_id
-----------------+-----------
 Einstein        |         1
(1 Zeile)
```

```
testdb2=# select * from test_person;
 person_name | person_id
-----------------+-----------
 Einstein        |         1
(1 Zeile)
```

...und andererseits wird der Befehl in der Recovery Datenbank gespeichert:

```
testrec=# select * from recovery;
 id  | vlogin |                                  sql                                  | transaction_id
----+--------+-------------------------------------------------------------------------+----------------
  1  | apuser | insert into test_person (person_name) values ('Einstein') |        0
(1 Zeile)
```

Die zweite Beispielapplikation soll uns auch noch beweisen, daß Sequoia ebenso gut mit einem SELECT Befehl umgehen kann:

```
C:\Lab\>%JAVA_HOME%\bin\java TestSelect
person_name     person_id
Einstein                 1
```

Testszenario – Ausfall einer backend Datenbank

Tabelle löschen

Dann wollen wir Sequoia mal auf eine etwas härtere Probe stellen. Wir simulieren den Ausfall einer backend Datenbank. Dies erreichen wir, indem wir die einzige Tabelle der Datenbank testdb2 löschen.

```
testdb2=# DROP TABLE test_person;
DROP TABLE
testdb2=# DROP SEQUENCE test_person_person_id_seq;
DROP SEQUENCE
```

Applikationszugriff testen

Anschließend setzen wir anhand unserer Beispielapplikation einen weiteren INSERT Befehl ab. Hiermit verifizieren wir, ob das weitere Arbeiten mit der Applikation möglich ist, da die Anwender nicht bemerken sollen, daß eine Datenbank nicht mehr zur Verfügung steht (sonst hätten wir ja keine Hochverfügbarkeit). Nach dem INSERT folgt der SELECT um zu prüfen, ob die letzte Eingabe in der Datenbank gespeichert wurde.

```
C:\Lab\>%JAVA_HOME%\bin\java TestInsert Curie

C:\Lab\>%JAVA_HOME%\bin\java TestSelect
person_name     person_id
Einstein                1
Curie           2
```

Die Arbeit ist durch den Ausfall der zweiten Datenbank nicht beeinträchtigt. Das liegt daran, daß die erste Datenbank weiterhin verfügbar ist und Sequoia alle Befehle nun an diese weiterleitet.

Tabelle wiederherstellen

Wie auch in einer professionellen Umgebung sollte die Verfügbarkeit der ausgefallene Datenbank nun wieder hergestellt werden. Hierzu wird die Tabelle die wir vorher gelöscht haben erneut erstellt:

```
testdb2=# CREATE TABLE test_person
testdb2=# (
testdb2=#   person_name varchar(50) NOT NULL,
testdb2=#   person_id int4 NOT NULL
testdb2=# );
CREATE TABLE
testdb2=# ALTER TABLE test_person ADD CONSTRAINT pk_test_person_nr
testdb2=#    PRIMARY KEY(person_id);
ALTER TABLE
testdb2=# CREATE SEQUENCE test_person_person_id_seq
testdb2=#   INCREMENT 1
testdb2=#   MINVALUE 1
testdb2=#   MAXVALUE 9223372036854775807
testdb2=#   START 1
testdb2=#   CACHE 1;
CREATE SEQUENCE
testdb2=# ALTER TABLE test_person ALTER COLUMN person_id
testdb2=#   SET DEFAULT nextval ('test_person_person_id_seq'::regclass);
ALTER TABLE
```

Konsistenten Zustand der Sequoia-Umgebung herstellen

Wir versetzen nun die wiederhergestellte Datenbank in einen konsistenten Zustand. Hierzu wechseln wir zurück zur Console. Da wir diese nicht beendet hatten befinden wir uns immer noch an dem Prompt “vDBtest(aduser) >“. Um sicher zu stellen, daß das entsprechende backend deaktiviert ist, setzen wir den entsprechenden Befehl ab:

```
vDBtest(aduser) > disable testNode2
vDBtest(aduser) >
```

Daraufhin erhalten Sie eine von zwei möglichen Ausgaben. Entweder “Disabling backend testNode2 with automatic checkpoint“, wenn dieses backend noch aktiviert war, oder “There was an exception of type class org.continuent.sequoia.common.exceptions.VirtualDatabaseException with message Backend testNode2 is already disabled“, wenn das backend bereits deaktiviert wurde.

Um die weiterhin notwendigen Administrationstätigkeiten durchführen zu können, müssen wir in den sogenannten Experten Modus wechseln:

```
vDBtest(aduser) > expert on
Expert mode on
vDBtest(aduser) >
```

Da keinerlei Daten in der neu erstellten Datenbank enthalten sind, müssen wir einen Weg finden, alle durch die Applikation bis dato eingegebenen Daten wieder herzustellen. Hier kommt uns Sequoia wieder zu Hilfe, da alle schreibenden SQL-Befehle in der Recovery Datenbank gespeichert wurden. Um nun diese Befehle auszulesen, auf der neu erstellten Datenbank auszuführen und das zweite backend anschliessend wieder zu aktivieren, setzen wir die foglenden Befehl ab:

```
vDBtest(aduser) > force checkpoint testNode2 Initial_empty_recovery_log
Forcing last know checkpoint of backend testNode2 to Initial_empty_recovery_log
vDBtest(aduser) > enable testNode2
Enabling backend testNode2 from its last known checkpoint
vDBtest(aduser) >
```

Im ersten Schritt wird der Checkpoint für das backend testNode2 definiert, ab dem eine Wiederherstellung notwendig ist (in diesem Fall der Checkpoint “Initial_empty_recovery_log“, welcher bereits in der Konfigurationsdatei “controller.xml“ definiert wurde und den ersten Checkpoint seit Einrichten der Beispielumgebung darstellt.). Eigentlich ist dieser Befehl nicht zwigend notwendig. Er stellt aber sicher, daß genau dieser checkpoint verwendet wird auch dann, wenn zwischenzeitlich weitere checkpoints erzeugt wurden (was dazu führen würde, daß beim Ausführen des Befehls “enable testNode2“ lediglich die Daten wiederhergestellt würden, die seit dem letzten checkpoint via Sequoia in die Datenbank geschrieben wurden).
Der zweite Schritt aktiviert das backend “testNode2“. Im Verlauf der Aktivierung werden alle schreibenden SQL-Befehle auf diesem backend ausgeführt. Dies führt wiederum dazu, daß sich nach Abschluß der Aktivierung die Umgebung wieder in einem konsistenten Zustand befindet und das zweite backend im vollen Umfang zur Verfügung steht. Der Abschluß der Aktivierung des backend “testNode2“ wird vom Controller mit einer Ausgabe ähnlich der folgenden dargestellt:

```
2005-12-10 21:21:54,890 INFO  controller.recoverylog.RecoverThread Database backend testNode2 is
now enabled
```

Sofern Sie den neuen Zustand und die Verfügbarkeit verifizieren möchten, führen Sie die folgenden Schritte durch. Zunächst führen Sie einen neuen INSERT und anschliessenden SELECT aus:

```
C:\Lab\>%JAVA_HOME%\bin\java TestInsert Diaz
C:\Lab\>%JAVA_HOME%\bin\java TestSelect
person_name     person_id
Einstein                1
Curie           2
Diaz            3
```

Dies bestätigt uns, daß alle eingegebenen Daten verarbeitet wurden. Es ist damit aber noch nicht sichergestellt, daß diese Daten auch in beiden backend Datenbanken enthalten sind. Um auch diesen Zustand sicherzustellen, führen wir jeweils einen SELECT auf den einzelnen backend Datenbanken über die Kommandozeilenumgebung der verwendeten Datenbanksoftware aus:

```
testdb1=# select * from test_person;
 person_name | person_id
-----------------+-----------
 Einstein        |         1
 Curie           |         2
 Diaz            |         3
(3 Zeilen)
```

```
testdb2=# select * from test_person;
 person_name | person_id
-----------------+-----------
 Einstein        |         1
 Curie           |         2
 Diaz            |         3
(3 Zeilen)
```

Damit ist sichergestellt, daß der gewünschte Zustand hergestellt wurde. Die Arbeit des Applikationsanwenders wurde weder durch den Ausfall, noch durch die administrativen Tätigkeiten zur Wiederherstellung beeinträchtigt.

Fazit

Durch den Einsatz von Sequoia haben wir eine Hochverfügbarkeit erreicht, welche beim Ausfall einer backend Datenbank sicherstellt, daß keine Beeinträchtigung der Applikationsfunktionalität besteht.
In einem produktiven Umfeld empfiehlt es sich, die Recovery Datenbank durch eine weitere Instanz oder weitere Mechanismen abzusichern (bei der Verwendung einer PostgreSQL- oder DB2-Datenbanksoftware könnten hierzu zum Beispiel write ahaed logs bzw. Redologs verwendet werden, welche auf einem Bandlaufwerk gespeichert werden).
Je nach Anspruch an Performance und Verfügbarkeit bzw. entsprechend der Applikationsarchitektur können mehrere Controller und somit auch weitere, ggf. verteilte backend Datenbanken verschiedenster Hersteller eingesetzt werden, um größtmögliche Ausfallsicherheit zu erreichen.
Der Einsatz von Sequoia ersetzt jedoch nicht das Monitoring (der Ausfall eines backend sollte immer zeitnah festgestellt werden, um die notwendigen Maßnahmen einzuleiten und schnellstmöglich wieder einen konsistenten Zustand herzustellen).
Ein weiterer positiver Effekt beim Einsatz von Sequoia ist, daß Offline-Zeiten für Tätigkeiten wie Datenbank-Sicherungen (dump) nicht mehr notwendig sein werden.

Testszenario – Datenbanksicherung erstellen

Ein paar Worte zu Octopus

Mit Sequoia wird ein generisches Sicherungswerkzeug mitgeliefert, basierend auf Enhydra Octopus (http://octopus.enhydra.org/). Das Anwendungsgebiet von Octopus im Rahmen von Sequoia ist der Export einer Datenbank via jdbc nach xml um anschließend anhand dieser xml-Datei via jdbc-import die Datenbank wiederherzustellen.

Pro:

- Der Im- und Export funktioniert (theoretisch) für alle Datenbankhersteller, für die ein jdbc Treiber existiert.
- Sequoia kann mit Octopus Sicherungen erzeugen, Sicherungen wieder zurück spielen und Datenbanken auch von einem Server auf einen anderen verschieben/kopieren.
- Für den Benutzer kann der Prozess transparent gemacht werden, er muss nur einen Befehl, wie zum Beispiel 'Transfer' eintippen und Sequoia ruft im Hintergrund Octopus auf um die Datenbank zu exportieren und auf einem anderen Server neu anzulegen

Contra:

- Es können nur jene Datenbankobjekte exportiert werden, welche via jdbc ablesbar sind. Man riskiert hier unter Umständen, daß Informationen 'verloren' gehen (Grants, Triggers, Stored Procedures, Parameters für sequences, exotische Datentypen, primary keys, foreign keys, etc). Die meisten Datenbankhersteller arbeiten mit Features, die von jdbc nicht abgedeckt werden und somit bei einem Export via jdbc zwangsläufig verloren gehen.
- Das vom jeweiligen Datenbank-Hersteller gelieferte Backuptool ist wahrscheinlich aufgrund der längeren Entwicklungsarbeit hieran stabiler als Octopus.
- Der Export/Import via jdbc und xml ist sicherlich wesentlich langsamer als das vom Datenbank-Hersteller gelieferte Backuptool.

Obwohl Octopus mit Sequoia mitgeliefert wird, ist die Verwendung von Octopus nicht die einzige Möglichkeit für die Sicherung der backend Datenbanken. Die Verwendung von Octopus kann die Betriebstätigkeiten optimieren, da zum Beispiel bei Verwendung von backend Datenbanksoftware verschiedenster Hersteller nur ein Werkzeug zur Sicherung und Wiederherstellung benötigt wird. Dennoch sollten in jedem Fall die bereits erwähnten Pro- und Contra-Argumente mit einbezogen werden um sich für oder gegen Octopus zu entscheiden.
Aufgrund der Verfügbarkeit des Quelltextes und der Steuerung per JMX besteht auch die Möglichkeit, eine Schnittstelle zur Sicherung und Wiederherstellung zu programmieren, welche auf Ihre Anforderungen zugeschnitten ist (beachten Sie in diesem Fall beim Vertrieb Ihres Produktes die jeweiligen Lizenzbedingungen).
In unserem Beispiel verwenden wir die vom Datenbankhersteller mitgelieferten Werkzeuge zur Sicherung (in diesem Fall pg_dump) und Wiederherstellung (pg_restore) um möglichen Negativ-effekten vorzubeugen.

Vorbereitung der backend Datenbanksicherung

Im Rahmen der Administrationstätigkeiten beim Datenbankbetrieb sind auch bei der Verwendung von Sequoia regelmäßige Datenbanksicherungen anzuraten. Unter anderem um im Fall des Ausfall einer backend Datenbank die Wiederherstellung so schnell wie möglich durchzuführen. Zudem wird hierdurch erreicht, daß die Recovery Datenbank nur soviele Daten wie nötig enthält (theoretisch müssen nur die SQL-Befehle seit dem letzten checkpoint enthalten sein).

Als erstes benötigen wir einen neuen checkpoint. Hierzu setzen wir den Befehl “disable testNode1“ auf der Console ab:

```
vDBtest(aduser) > disable testNode1
Disabling backend testNode1 with automatic checkpoint.
```

Anhand der Ausgabe ist zu erkennen, daß hierbei automatisch ein neuer checkpoint gesetzt wurde. Dies können wir beim Betrachten der checkpoint Tabelle nachvollziehen:

```
testrec=# select * from checkpoint;
                                 name                                 | request_id
----------------------------------------------------------------------+------------
 Initial_empty_recovery_log                                           |          0
 disable_backend_testNode2_Sat Dec 10 21:21:54 CET 2005 |          1
 disable_backend_testNode1_Sat Dec 10 21:47:37 CET 2005 |          2
(3 Zeilen)
```

Wir können die vorhanden checkpoints aber auch über die Sequoia Console abfragen:

```
vDBtest(aduser) > show checkpoints
Checkpoints...
[0]    Initial_empty_recovery_log
[1]    disable_backend_testNode2_Sat Dec 10 21:21:54 CET 2005
[2]    disable_backend_testNode1_Sat Dec 10 21:47:37 CET 2005
```

Datenbanksicherung durchführen

Das backend “testNode1“ wurde deaktiviert. Nun können wir einen dump der entsprechenden backend Datenbank “testdb1“ durchführen:

```
C:\>md C:\Lab\DBDump
C:\>pg_dump.exe -h localhost -p 5432 -U dbuser -f C:\Lab\DBDump\200512102150.sql testdb1
Paßwort:                              (wird während der Eingabe nicht angezeigt)
```

Um annähernd realistische Vorraussetzungen nachzustellen, werden wir einmal mehr unsere kleinen Programme ausführen:

```
C:\Lab\>%JAVA_HOME%\bin\java TestInsert Newton
C:\Lab\>%JAVA_HOME%\bin\java TestSelect
person_name    person_id
Einstein               1
Curie          2
Diaz           3
Newton                 4
```

Backend reaktivieren

Nachdem der Datenbank-Dump durchgeführt wurde stellen wir das backend wieder der Sequoia-umgebung zur Verfügung:

```
vDBtest(aduser) > enable testNode2
Enabling backend testNode2 from its last known checkpoint
vDBtest(aduser) >
```

Ein SELECT auf den einzelnen backend Datenbanken über die Kommandozeilenumgebung der verwendeten Datenbanksoftware zeigt uns einmal mehr, ob wir wieder einen konsistenten Zustand erreicht haben:

```
testdb1=# select * from test_person;
 person_name | person_id
-----------------+-----------
 Einstein        |         1
 Curie           |         2
 Diaz            |         3
 Newton          |         4
(4 Zeilen)

testdb2=# select * from test_person;
 person_name | person_id
-----------------+-----------
 Einstein        |         1
 Curie           |         2
 Diaz            |         3
 Newton          |         4
(4 Zeilen)
```

Fazit

Dieses Testszenario zeigt wie einfach und schnell auch Betriebstätigkeiten, wie das Erstellen eines Datenbank dump mit Sequoia möglich sind, ohne daß die Anwendung der Applikation hierdurch beeinflußt wird.
Beim Ausfall einer backend Datenbank können wir diese Datenbanksicherungen verwenden, um die Datenkonsistenz bis zum letzten checkpoint wieder herzustellen um anschließend die fehlenden Datensätze über den Befehl “enable <backendname>“ automatisch durch Sequoia wiederherstellen zu lassen.

Eclipse als grafisches Administrationsfrontend

Für die Freunde der grafischen Benutzeroberflächen gibt es ein Eclipse-Plugin: Oak. Die Kommunikation basiert, ebenso wie bei der Console, auf JMX. Beachten Sie, daß sich dieses Plugin aber noch im Beta-Status befindet und somit seitens der Entwickler nicht als stabil genug für einen Produktiveinsatz betrachtet wird. In experimentellen Umgebungen ist es jedoch interessant festzustellen, in welche Richtung die Entwicklung auch für Administrationstools geht.

Sofern Sie dieses Plugin zu Testzwecken installieren möchten empfehle ich Ihnen, hierzu Eclipse in der aktuellen Version 3.1.1 (http://www.eclipse.org/downloads/) zu verwenden.

Oak kann direkt über den Update Manager von Eclipse installiert werden. Eine Anleitung in Englisch finden Sie unter: http://oak.continuent.org/HowToInstallOakFromEclipse.

Sequoia Controller als Dienst unter Verwendung des Java Service Wrapper

Viele von Ihnen werden es sicherlich vorziehen, den Sequoia Controller unter Microsoft Windows als Dienst im Hintergrund laufen zu lassen. Da der Controller nur in einer JRE ausführbar ist, stellt das scheinbar ein Problem dar. Jedoch gibt es hierfür eine Lösung und zwar den “Java Service Wrapper“.

Der Java Service Wrapper ist im Entwicklungsstatus Produktion/Stabil und kann von folgender URL heruntergeladen werden: http://sourceforge.net/project/showfiles.php?group_id=39428

Laden Sie das Zip-Archiv (wrapper_win32_3.1.2.zip) herunter und entpacken Sie es in einem Verzeichnis Ihrer Wahl. Kopieren Sie alle Dateien aus dem Unterverzeichnis “bin“ in das Unterverzeichnis “bin“ von Sequoia. Benennen Sie die Datei “MyApp.bat“ nach “Sequoia.bat“ um. Kopieren Sie alle Dateien aus dem Unterverzeichnis “lib“ in das Unterverzeichnis “lib“ von Sequoia. Unter “Anhang A“ dieser Anleitung finden Sie den Inhalt der benötigten Konfigurationsdatei “wrapper.conf“. Diese wird zwar im Unterverzeichnis “doc\examples\WindowsService“ mitgeliefert allerdings sind sowohl in der Version 2.3, als auch in 2.4 fehlerhafte Zeilenumbrüche enthalten, welche dazu führen, daß der Dienst nicht startet. Erstellen Sie daher im Sequoia Unterverzeichnis “config“ eine Datei mit dem Namen “wrapper.conf“ und kopieren Sie die Konfiguration aus dem Anhang in diese Datei.

Ersetzen Sie die Zeile: set _WRAPPER_CONF="%_REALPATH%..\conf\wrapper.conf"
gegen diesen Eintrag: set _WRAPPER_CONF="%_REALPATH%..\config\wrapper.conf"
in allen Batchscripten des Wrapper ausser “UninstallTestWrapper-NT.bat“.

Nachdem der Dienst eingerichtet wurde sollten Sie an dieser Konfigurationsdatei keine Änderungen vornehmen. Sollten Anpassungen notwendig sein, deinstallieren Sie zunächst den Dienst, führen Sie dann die Anpassungen durch um anschließend den Dienst erneut einzurichten.
Um den Java Service Wrapper für den Sequoia Controller verwenden zu können ist es unbedingt notwendig, daß Sie die globalen Umgebungsvariablen JAVA_HOME und SEQUOIA_HOME setzen. Den Dienst installieren Sie, indem Sie die Datei “InstallTestWrapper-NT.bat“ ausführen. In der Liste der Dienste erhalten Sie einen neuen Eintrag: “Sequoia DB Controller“. Zur Deinstallation führen Sie die Datei “UninstallTestWrapper-NT.bat“ aus (evtl. ist im Anschluß ein Reboot notwendig).
Da nach dem Einrichten des Controllers als Dienst keine Ausgaben mehr in einem Kommandozeilenfenster angezeigt werden, ist der jeweilig aktuelle Status nicht gleich erkennbar. Im Unterverzeichnis “log“ der Sequoia-Installation finden Sie jedoch unter anderem die Protokolldatei “sequoia.log“, in welcher alle relevanten Informationen Protokolliert werden.

Anhang A – Inhalt der Datei wrapper.conf

Speichern Sie die folgenden Informationen unter %SEQUOIA_HOME%\config\wrapper.conf

```
#*****************************************************************
# Wrapper Properties
# Example contributed by: Bernard.McGourty@aspect.com
#*****************************************************************
# Java Application
#
wrapper.java.command=%JAVA_HOME%\bin\java

# Java Main class.  This class must implement the WrapperListenerinterface
#  or guarantee that the WrapperManager class is initialized.  Helper
#  classes are provided to do this for you.  See the Integration section
#  of the documentation for details.
#
wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperSimpleApp

# Java Classpath (include wrapper.jar)
#
# Add class path elements as needed starting from 1
#
wrapper.java.classpath.1=%SEQUOIA_HOME%\lib\wrapper.jar
wrapper.java.classpath.2=%JAVA_HOME%\lib\tools.jar
wrapper.java.classpath.3=%SEQUOIA_HOME%\lib\octopus\Octopus.jar
wrapper.java.classpath.4=%SEQUOIA_HOME%\lib\octopus\OctopusGenerator.jar
wrapper.java.classpath.5=%SEQUOIA_HOME%\lib\octopus\csvjdbc.jar
wrapper.java.classpath.6=%SEQUOIA_HOME%\lib\sequoia-controller.jar
wrapper.java.classpath.7=%SEQUOIA_HOME%\lib\sequoia-backend.jar
wrapper.java.classpath.8=%SEQUOIA_HOME%\lib\sequoia-backupers.jar
wrapper.java.classpath.9=%SEQUOIA_HOME%\lib\sequoia-cache.jar
wrapper.java.classpath.10=%SEQUOIA_HOME%\lib\sequoia-commons.jar
wrapper.java.classpath.11=%SEQUOIA_HOME%\lib\sequoia-jmx.jar
wrapper.java.classpath.12=%SEQUOIA_HOME%\lib\sequoia-sql.jar
wrapper.java.classpath.13=%SEQUOIA_HOME%\lib\octopus
wrapper.java.classpath.14=%SEQUOIA_HOME%\drivers
wrapper.java.classpath.15=%SEQUOIA_HOME%\lib\hedera-commons.jar
wrapper.java.classpath.16=%SEQUOIA_HOME%\lib\hedera-jgroups.jar
wrapper.java.classpath.17=%SEQUOIA_HOME%\lib\jgroups-core.jar
wrapper.java.classpath.18=%SEQUOIA_HOME%\lib\commons-logging.jar
wrapper.java.classpath.19=%SEQUOIA_HOME%\lib\dom4j-1.5-beta-2.jar
wrapper.java.classpath.20=%SEQUOIA_HOME%\lib\jaxen-1.1-beta-2.jar
wrapper.java.classpath.21=%SEQUOIA_HOME%\lib\log4j.jar
wrapper.java.classpath.22=%SEQUOIA_HOME%\lib\commons-cli.jar
wrapper.java.classpath.23=%SEQUOIA_HOME%\lib\jmx\mx4j.jar
wrapper.java.classpath.24=%SEQUOIA_HOME%\lib\jmx\mx4j-remote.jar
wrapper.java.classpath.25=%SEQUOIA_HOME%\lib\jmx\mx4j-tools.jar
wrapper.java.classpath.26=%SEQUOIA_HOME%\xml
wrapper.java.classpath.27=%SEQUOIA_HOME%\lib\crimson.jar
wrapper.java.classpath.28=%SEQUOIA_HOME%\lib\xml-apis.jar
wrapper.java.classpath.29=%SEQUOIA_HOME%\config\language
```

```
wrapper.java.classpath.30=%SEQUOIA_HOME%\config\controller
wrapper.java.classpath.31=%SEQUOIA_HOME%\config\virtualdatabase
wrapper.java.classpath.32=%SEQUOIA_HOME%\config\

# Java Library Path (location of Wrapper.DLL or libwrapper.so)
wrapper.java.library.path.1=%SEQUOIA_HOME%\lib

# Java Additional Parameters
wrapper.java.additional.1=-Dsequoia.home=%SEQUOIA_HOME%
wrapper.java.additional.2=-Dsequoia.log=%SEQUOIA_HOME%\log
wrapper.java.additional.3=-Djava.security.policy=%SEQUOIA_HOME%\config\java.policy
wrapper.java.additional.4=-Djava.net.preferIPv4Stack^=true

# Initial Java Heap Size (in MB)
wrapper.java.initmemory=256

# Maximum Java Heap Size (in MB)
wrapper.java.maxmemory=256

# Application parameters.  Add parameters as needed starting from 1
wrapper.app.parameter.1=org.continuent.sequoia.controller.core.Controller
wrapper.app.parameter.2=-f %SEQUOIA_HOME%\config\controller\controller.xml

# File that contains the Java process ID
wrapper.java.pidfile=%SEQUOIA_HOME%\log\sequoia.java.pid
wrapper.java.additional.2=-Dsequoia.log=%SEQUOIA_HOME%\log

# Timeouts - set longer than the default
wrapper.ping.timeout=60
wrapper.cpu.timeout=25

# Request a JVM Thread Dump when JVM does not exit. Note this will also allow
# a user to send a CTRL-Break to the JVM to request a Thread Dump
wrapper.request_thread_dump_on_failed_jvm_exit=true

#*********************************************************************
# Wrapper Logging Properties
#*********************************************************************
# Format of output for the console.  (See docs for formats)
wrapper.console.format=PM

# Log Level for console output.  (See docs for log levels)
wrapper.console.loglevel=INFO

# Log file to use for wrapper output logging.
wrapper.logfile=%SEQUOIA_HOME%\log\sequoia.service.log

# Format of output for the log file.  (See docs for formats)
wrapper.logfile.format=LPTM

# Log Level for log file output.  (See docs for log levels)
wrapper.logfile.loglevel=INFO
```

```
# Maximum size that the log file will be allowed to grow to before
#  the log is rolled. Size is specified in bytes.  The default value
#  of 0, disables log rolling.  May abbreviate with the 'k' (kb) or
#  'm' (mb) suffix.  For example: 10m = 10 megabytes.
wrapper.logfile.maxsize=10m

# Maximum number of rolled log files which will be allowed before old
#  files are deleted.  The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10

# Log Level for sys/event log output.  (See docs for log levels)
wrapper.syslog.loglevel=NONE

#*********************************************************************
# Wrapper NT Service Properties
#*********************************************************************
# WARNING - Do not modify any of these properties when an application
#  using this configuration file has been installed as a service.
#  Please uninstall the service before modifying this section.  The
#  service can then be reinstalled.

# Name of the service
wrapper.ntservice.name=Sequoia

# Display name of the service
wrapper.ntservice.displayname=Sequoia DB Controller

# Description of the service
wrapper.ntservice.description=Sequoia Database Controller

# Service dependencies.  Add dependencies as needed starting from 1
wrapper.ntservice.dependency.1=

# Mode in which the service is installed.  AUTO_START or DEMAND_START
wrapper.ntservice.starttype=AUTO_START

# Allow the service to interact with the desktop.
wrapper.ntservice.interactive=false

wrapper.working.dir=%SEQUOIA_HOME%
```

Epilog, Danksagung und Copyright-Hinweis

Das Open Source Projekt Sequoia stellt eine interessante Alternative und/oder Ergänzung bei der Einrichtung einer hochverfügbaren JDBC-Umgebung bereit. Meiner Ansicht nach befindet es sich in einem Status, der es erlaubt, die Software auch im Produktivumfeld einzusetzen. Im derzeitigen Stadium empfiehlt sich jedoch, für jedes Projekt eine Analyse der Gesamtarchitektur (inklusive Ihrer Applikations- und Datenbankkomponenten) in einer Abnahmeumgebung durchzuführen, um später keine bösen Überraschungen zu erleben.
In diesem Zusammenhang könnten die Informationen, welche Sie über die Mailingliste (Archiv: https://forge.continuent.org/pipermail/sequoia/) und der Projektstatusseite erhalten, von Interesse für Sie sein (https://forge.continuent.org/jira/browse/SEQUOIA).

Derzeit werden folgende Funktionalitäten vom Sequoia Treiber NICHT unterstützt:
- java.sql.Array und java.sql.Ref Typen,
- Custom type mapping using java.sql.Connection.setTypeMap(java.util.Map map),
- XAConnections (Besuchen Sie das XAPool Projekt (http://xapool.experlog.com) für XA Unterstützung für Sequoia),
- Aufrufbare Statements mit OUT Parameter oder der Rückgabe von mehrfachen ResultSets,
- Streamable ResultSets funktionieren nicht im autocommit Modus.

Folgende Sequoia Controller Funktionalitäten werden derzeit NICHT unterstützt:
- GRANT/REVOKE Befehle werden zwar an die Datenbank-Engine übertragen, aber es werden hierbei keine Benutzer vom Authentisierungs Manager der virtuellen Datenbank entfernt oder hinzugefügt.
- Netzwerk partition/reconciliation wird nicht unterstützt,
- Verteilte joins werden nicht unterstützt. Das bedeutet, daß Sie sicherstellen müssen, daß jeder query von mindestens einem einzelnen backend ausgeführt werden kann,
- RAIDb-1ec und RAIDb-2ec Level werden nicht unterstützt.

Entgegen einigen Dokumentationen enthält Sequoia nicht die grafische SQL-Console “SQuirreL“. SQuirreL wurde inzwischen durch “iSQL“ (http://www.isqlviewer.com/) ersetzt. Sollten Sie daran interessiert sein SQuirreL kennenzulernen, können Sie es unter der URL http://squirrel-sql.sourceforge.net/ herunterladen.

Zögern Sie nicht sich bezüglich Lob, konstruktiver Kritik, Korrekturvorschläge und Anregungen an mich zu wenden. Schreiben Sie hierzu an rainer.brinkmoeller@web.de. Bitte beginnen Sie den Eintrag in der Betreffzeile mit [Sequoia_2.4_deAnleitungRB] da mich Ihre Nachricht anderenfalls wahrscheinlich nicht erreichen wird.

Danksagung
An dieser Stelle möchte ich Marc Wick (C-JDBC Commiter) für seine Unterstützung in Form von Empfehlungen, guten Ratschlägen und verständlichen Erläuterungen danken. Ohne seine Hilfe hätte ich diese Anleitung nicht so schnell fertigstellen können.