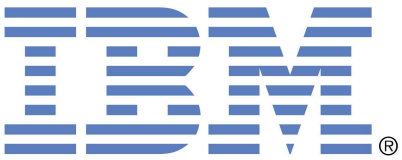
Eberhard Karls
Universität
Tübingen



Fakultät für Informations- und Kognitionswissenschaften
Wilhelm-Schickard-Institut

Workload Prediction für den WLM

Diplomarbeit

Verfasser:

Semmo Bakircioglu

31. Januar 2009

Betreuer:

Prof. Dr. Wilhelm G. Spruth
Prof. Dr. Martin Bogdan
Dipl.-Math. Peter Bäuerle (IBM)

Bakircioglu, Semmo:
Workload Prediction
Diplomarbeit Informatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 01.05.2008 - 31.01.2009

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Tübingen, den 25. Januar 2009 Semmo Bakircioglu

Kurzfassung

Diese Diplomarbeit führt das *Joint Research Project* „Workload Prediction auf Mainframes“ der Universität Tübingen mit dem IBM Labor in Böblingen weiter. Im Rahmen dieses Projekts soll eine Machbarkeitsstudie zur möglichen Lastvorhersage mit Hilfe von künstlichen neuronalen Netzen erstellt werden. Als Ergebnis dieser Bemühungen entstand mit dem *WLP-Framework* ein erster Prototyp, der die gewünschte Funktionalität erstmals implementierte.
In der vorliegenden Arbeit wird die Weiterentwicklung dieses Prototypen thematisiert. Der besondere Fokus liegt dabei auf vier Punkten:
Als erstes wurde eine präzise und umfangreiche „Bedienungsanleitung“ für das komplexe *WLP-Framework* erarbeitet, damit der Einstieg in das Thema nicht nur für die eigentliche Benutzung, sondern auch im Hinblick auf mögliche Weiterentwicklungen möglichst schnell und einfach erfolgen kann.
Im Anschluss wurde die ursprünglich verwendete *Apache Derby* Datenbank durch die von IBM entwickelte Datenbank *DB2 for z/OS* ersetzt. Eine zentrale Rolle spielt dabei die seit Version 9 eingeführte und so genannte *pureXML*-Technologie.
Durch diese Datenbankumstellung war es möglich den größten Nachteil des *WLP-Frameworks*, das *generische Tabellenformat*, zu beseitigen. Diese Aufgabe zog eine umfangreiche und aufwendige Refaktorisierung des Prototypen nach sich, in deren Zuge auch andere allgemeine Verbesserungen und Fehlerbehebungen am Quellcode vorgenommen wurden.
Nachdem mit der Datenbankumstellung ein erster Schritt weg vom Client unternommen wurde, folgte als letzter Schritt die Migration des für die Aufzeichnung zuständigen *RMFRecorders* und der für die Vorhersage zuständigen *WLPrediction* auf den Großrechner. Dazu wurden alle benötigten Dateien auf die *UNIX System Services* (USS) übertragen, und nach einigen notwendigen Anpassungen waren beide Anwendungen erfolgreich auf dem Großrechner ausführbar.
Somit ist mit Abschluss der vorliegenden Arbeit das komplette *WLP-Framework* explizit auch auf dem Großrechner lauffähig.

Danksagung

An dieser Stelle möchte ich all jenen danken, die eine Erstellung dieser Diplomarbeit erst ermöglicht haben. In erster Linie wären das Herr Prof. Dr. Spruth und Herr Prof. Dr. Bogdan als universitäre Betreuer sowie Herr Peter Bäuerle als Betreuer auf Seiten der IBM. Ohne sie wäre diese Diplomarbeit nie entstanden.
Weiterhin danke ich dem kompletten Böblinger WLM-Team von Frau Jutta Brenner, die mir alle jederzeit und aufgeschlossen mit Rat und Hilfestellungen zur Seite standen. Vielen Dank Adrian, Hakan, Thomas, ... um nur ein paar wenige Namen zu nennen. Großer Dank gebührt auch Emil Wolf, der mir auf Nachfrage in Rekordzeit die benötigte Datenbank aufgesetzt und freundlicherweise auch die Administration übernommen hat.
Vor allem aber danke ich meiner Freundin Rine, die sich mit mir die Nächte um die Ohren geschlagen hat, um Korrektur zu lesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Um den Umfang dieser Arbeit zu beschränken wurde bewusst darauf verzichtet die theoretischen Grundlagen zu sehr zu vertiefen, da dies in den vorhergehenden Diplomarbeiten dieses Projekts ausführlich erledigt wurde.
Falls Interesse an tiefer gehenden Informationen zum Thema künstliche neuronale Netze besteht, wird auf die Arbeit von Kleeberg [Kle06] verwiesen. Für Informationen zur Datengewinnung und -verarbeitung wird Gebhard [Geb06] empfohlen. Alternativ kann auch die Arbeit von Hagmann [Hag07] als komprimierte Zusammenfassung der vorher genannten angesehen werden.
In allen drei Arbeiten finden sich weitere Quellenangaben für zusätzliche Informationen.

1.1. Motivation

Zentrales Thema dieser Arbeit ist die Lastvorhersage auf IBM Großrechnern. Die Verwaltung von Betriebsmitteln solcher Systeme erfolgt über den *Workload Manager* (WLM), der integraler Bestandteil des Betriebssystems z/OS ist. Bisher wird diese Verwaltung dynamisch nach benutzerdefinierten Zielvorgaben durchgeführt. Die Basis dafür bilden aktuell gemessene Systemdaten sowie Daten aus der Vergangenheit. Der WLM ist eine zentrale Komponente in der alltäglichen Arbeit eines Großrechner, da ein priorisierter Zugang zu Betriebsmitteln zwingend erforderlich ist. Diese Priorisierung ist notwendig, weil Großrechner darauf ausgelegt sind eine Vielzahl an unterschiedlichen Anwendungen von verschiedenen Benutzern zeitgleich und ohne spürbare Geschwindigkeitseinbußen auszuführen.
Durch seine zielorientierte Arbeitsweise sind dem WLM aber Grenzen gesetzt. Es kann immer nur zeitverzögert auf Ressourcen-Engpässe reagiert werden, da sie erst

nach ihrem tatsächlichen Eintreten bemerkt werden können. Mit dieser Arbeitsweise sind Engpässe unvermeidbar. Um diesen Schwachpunkt mittelfristig zu beseitigen und Engpässe möglichst im voraus erkennen zu können, wurde dieses Projekt mit dem Ziel einer Lastvorhersage mit Hilfe von künstlichen neuronalen Netzen gestartet.
Durch die Einführung des *Capacity Provisionings* (CP) mit z/OS 1.9 gewann dieser Ansatz an zusätzlichem Gewicht. Der CP-Manager hat Zugriff auf die Systemdaten der *Resource Measurement Facility* (RMF). Sobald klar wird, dass eine Anwendung ihr festgelegtes WLM-Ziel verfehlt weil ein Ressourcen-Engpass besteht, kann der CP-Manager entweder angemessene Maßnahmen vorschlagen oder automatisch zusätzliche Kapazitäten aktivieren [IBM08d].

Ziel dieses Projekts ist eine Anwendung zur Lastvorhersage mit Hilfe von künstlichen neuronalen Netzen, die in die bestehende WLM- und CP-Umgebung integriert werden soll. Mit Hilfe der vorhergesagten Werte sollen mögliche Engpässe bereits im Voraus erkannt und an das CP weitergegeben werden, damit es je nach Bedarf vorhandene stillgelegte Betriebsmittel zur richtigen Zeit aktivieren kann.

1.2. Gliederung der Arbeit

Kapitel 1 dient als Motivation und Einstieg in das Thema. Zusätzlich werden die Ziele und Schwerpunkte dieser Arbeit erläutert und es wird auf die bisherigen Diplomarbeiten jeweils kurz eingegangen um einen groben chronologischen Überblick über das ganze Projekt zu bekommen. Kapitel 2 stellt die Arbeit von Michael Hagmann [Hag07] detaillierter vor, welche mit dem *WLP-Framework* einen ersten Software-Prototypen für das *Joint Research Project* umfasst. Es bildet die Grundlagen für die anschließenden Weiterentwicklungen in Kapitel 3, dem eigentlichen Kern dieser Arbeit. Hier wird dann, beginnend mit dem Erstellen einer Bedienungsanleitung und einer Datenbankumstellung, die Migration auf den Großrechner durchgeführt. Zudem erfolgt vor der Migration eine Refaktorisierung, die strukturelle Optimierungen zur Folge hat. Anschließend werden in Kapitel 4 die erzielten Ergebnisse rekapituliert und diskutiert, bevor in Kapitel 5 eine kurze Zusammenfassung mit Ausblick auf zukünftige Forschungsarbeiten im Rahmen dieses Projekts die Arbeit abschließt.

1.3. Ziele und Schwerpunkte dieser Arbeit

Ziel dieser Diplomarbeit ist die Weiterentwicklung der Machbarkeitsstudie von Hagmann [Hag07]. Dieser hat einen ersten Prototypen zur Lastvorhersage mit künstlichen neuronalen Netzen für IBMs Großrechner *System Z* implementiert. Im Verlauf der vorliegenden Arbeit liegt der Fokus vor allem auf vier Punkten:

- Erstellen einer präzisen und umfassenden Bedienungsanleitung zur Inbetriebnahme und zum allgemeinen Verständnis des Prototypen
- Ersetzen der aktuell benutzten *Apache Derby* Datenbank durch IBM *DB2 for z/OS* 9.1
- Elimination des *generische Tabellenformats* mit Hilfe der neuen DB2 *pureXML*-Technologie und eine dadurch bedingte umfangreiche Refaktorisierung großer Teile des *WLP-Frameworks*
- Migration des *WLP-Frameworks* vom Windows-Client auf den Großrechner

1.4. Bisherige Arbeiten

Dies ist die vierte Diplomarbeit im Rahmen des *Joint Research Projects* „Workload Prediction auf Mainframes“. Nach anfangs theoretischen Arbeiten über „Datengewinnung und Parameterbestimmung zur Lastvorhersage in z/OS“ von Gebhard [Geb06] und zum Thema „Neuronale Netze und Maschinelles Lernen zur Lastvorhersage in z/OS“ von Kleeberg [Kle06] erfolgte im Anschluss der Versuch einer ersten lauffähigen Umsetzung der Ergebnisse in der Arbeit „Architektur und Integration der Workload-Vorhersage auf Basis neuronaler Netze in z/OS“ von Hagmann [Hag07]. Auf die Themen dieser Diplomarbeiten wird im Folgenden kurz eingegangen.

1.4.1. Diplomarbeit von Clemens Gebhard

In der Arbeit von Clemens Gebhard [Geb06] wurde in erster Linie die Beschaffung von Systemdaten untersucht: Zum einen die Gewinnung von realen Daten und

zum anderen die Generierung von künstlichen Daten. Ziel war es möglichst realitätsnahe Daten für die Durchführung einer Lastvorhersage mit Hilfe von künstlichen neuronalen Netzen zu erhalten.
Ein weiterer wichtiger Aspekt war die Aufzeichnung und Weiterverarbeitung der Daten. Hierzu wurde erstens die Abfrage über die RMF und zweitens über die *System Management Facility* (SMF) betrachtet. Abschließend wurden verschiedene Techniken der Vorverarbeitung für die extrahierten Daten erörtert, da die Daten in der Regel Ausreißer und fehlende Werte beinhalten, welche entsprechend skaliert, gefiltert oder anderweitig bearbeitet werden müssen.
Zudem wurden erste Ansätze zur Implementierung der erarbeiteten Ergebnisse präsentiert, mit denen die Daten graphisch aufbereitet und analysiert werden konnten.
Die Ergebnisse dieser Arbeit bilden den theoretischen Hintergrund des implementierten *WLP-Frameworks* zum Thema Datengewinnung und -verarbeitung.

1.4.2. Diplomarbeit von Sarah Dorothea Kleeberg

Die Arbeit von Sarah Kleeberg befasst sich schwerpunktmäßig damit, die Leistungsfähigkeit von unterschiedlichen Vorhersagemodellen zu betrachten, um in der Folge abwägen zu können welche Modelle für welchen Vorhersagezeitraum geeignet wären. Im Detail wurden die künstlichen neuronalen Netztypen *Feed-Forward* und *FlexNet* sowie der maschinelle Lernalgorithmus *Support Vector Regression* bezüglich ihrer Leistung durch mehrere Testläufe unter verschiedenen Voraussetzungen verglichen. Die Modell-Bewertung lief dabei über die Betrachtung von verschiedenen Fehlermaßen. Darunter auch der im Prototyp verwendete *Mean Squared Error with Regularization* (MSEREG).
Als Ergebnis dieser Testläufe wurde festgehalten, dass für längerfristige Vorhersagen *Feed-Forward* Netze und *FlexNet* die besten Ergebnisse liefern. *Support Vector Regression* war nur bei kürzeren Zeiträumen mit ungeglätteten Datensätzen die überlegenere Methode.
Als Schlussfolgerung aus dem oben genannten Verhalten und der detaillierten Betrachtung der restlichen Ergebnisse wurde die Erkenntnis gewonnen, dass für unterschiedliche Problemstellungen auch unterschiedliche Vorhersagemodelle aus einem „Pool“ zur Auswahl stehen sollten.

Die Vorhersage-Anwendung, die die Komponente mit den künstlichen neuronalen Netzen des *WLP-Frameworks* implementiert wurde nach den Ergebnissen dieser theoretischen Arbeit modelliert.

1.4.3. Diplomarbeit von Michael Hagmann

Nachdem in den beiden vorherigen Arbeiten die theoretischen Grundlagen geschaffen wurden, war der Schwerpunkt der Arbeit von Hagmann [Hag07] das Herausarbeiten einer Architektur, die diese gewonnen Erkenntnisse zu einer lauffähigen Anwendung kombiniert und möglichst ohne Nachteile für das Gesamtsystem implementiert.
Der erste Schritt dazu war die Datengewinnung und -speicherung. Es musste ein Datenzugriff bei laufender Anwendung ermöglicht werden und für die weitere Bearbeitung musste ein Weg gefunden werden die Daten möglichst vorteilhaft zwischenzuspeichern. Die einzige plausible und zeitgemäße Lösung für diese Aufgabe war eine relationale Datenbank.
Ein weiterer entscheidender Aspekt betraf die automatische Merkmalsselektion. Es sollte eine möglichst große Auswahl an Metriken (Merkmalen) zur Verfügung stehen, um je nach Vorhersagemodell die am besten geeigneten Merkmale auch selektieren zu können. Um je nach Bedarf trotzdem eine Vorauswahl der Metriken zu treffen, wurde eine Konfigurationsdatei eingeführt, in der mit *Include/Exclude*-Anweisungen Metriken ein-/ausgeschlossen werden können.
Um das *WLP-Framework* möglichst flexibel zu gestalten, musste eine Möglichkeit gefunden werden mehrere Vorhersagemodelle erstens unabhängig voneinander und zweitens parallel zu betreiben und zu verwalten. Damit war gewährleistet, dass man während des Trainierens eines neu angelegten Netzes gleichzeitig eine Vorhersage mit einem schon trainierten Netz durchführen kann. In der Folge wurden der *RMFRecorder* zur Datenaufzeichnung und die *WLPrediction* zur Vorhersageberechnung als komplett eigenständige und unabhängige Anwendungen implementiert.
Als letzten Schritt wurde zudem noch mit Hilfe eines Webservers eine einfach gehaltene Schnittstelle implementiert, die die Vorhersagewerte in graphischer Form anschaulich präsentiert, und die zukünftig dazu beitragen könnte die Anwendung zu steuern und zu konfigurieren.

2. Grundlagen: Die Komponenten des *WLP-Frameworks*

Als Grundlage für die vorliegende Arbeit dient das in [Hag07] von Hagmann erstellte *WLP-Framework*. Bevor auf die eigentlichen Aufgaben dieser Arbeit eingegangen wird, soll deswegen zuerst dieses als Ausgangspunkt dienende Framework im Detail vorgestellt werden.
Das *WLP-Framework* basiert auf einer verteilten Architektur, die in Abbildung 2.1 schematisch dargestellt wird. Sie kann grob in drei Hauptkomponenten aufgeteilt werden: Den *RMFRecorder* für die Datenaufzeichnung, die Datenbank für die Datenhaltung und die *WLPrediction* für die eigentliche Vorhersage. Alle drei Komponenten werden auf einem Windows-Client ausgeführt.
Die Architektur wurde auf möglichst viel Flexibilität ausgelegt. Es ist möglich, die Datenaufzeichnung und die Vorhersage getrennt voneinander durchzuführen, da sie beide als unabhängige und eigenständige Anwendungen konzipiert wurden. Auch ein Parallelbetrieb ist jederzeit möglich. Zudem ist das in Java implementierte *WLP-Framework* theoretisch plattformunabhängig, da es aber auf der Windows-Plattform erstellt wurde, ist es momentan auch nur darauf getestet worden.
Im Folgenden werden die Komponenten jeweils kurz betrachtet.

2.1. Datenaufzeichnung: *RMFRecorder*

Wie bereits erwähnt, ist der *RMFRecorder* von der Vorhersage-Anwendung unabhängig, da er in einem eigenen Prozess läuft. Der Vorteil dabei besteht in der Möglichkeit erst einmal nur Daten aufzuzeichnen und zu sammeln, während die eigentliche Vorhersage u. U. erst viel später erfolgt. Darüber hinaus kann eine Vorhersage auch parallel zu einer laufenden Datenaufzeichnung durchgeführt werden.

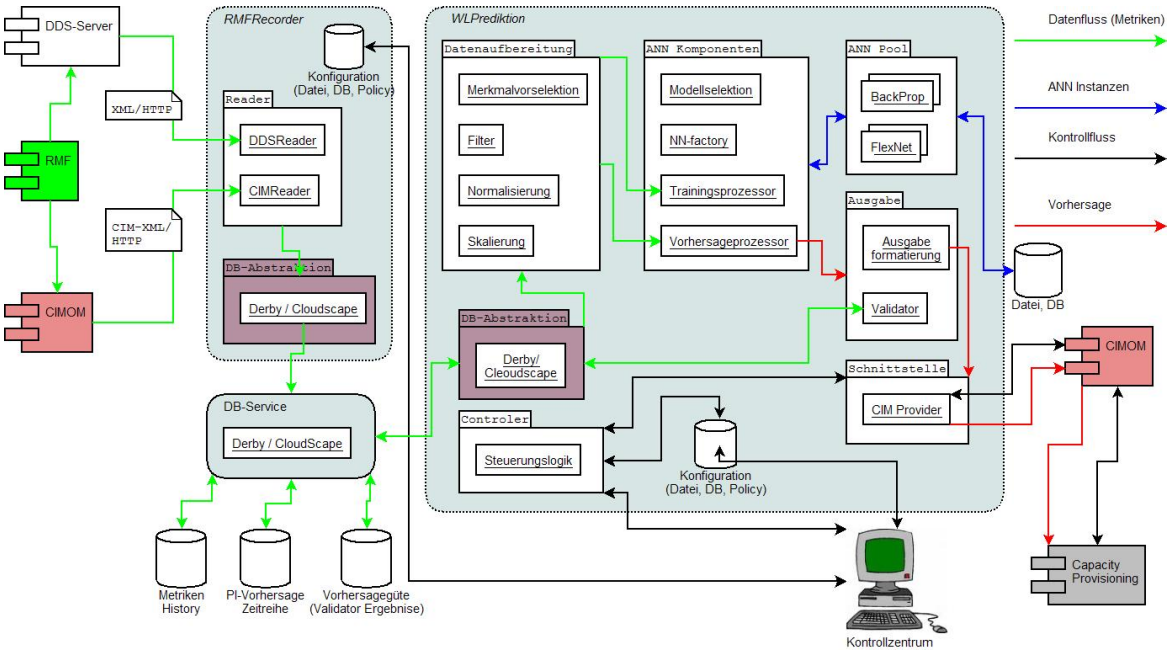


Abbildung 2.1.: Schematische Übersicht über die Komponenten des *WLP-Frameworks* (Abbildung aus [Hag07])

Als Datenquelle für den *RMFRecorder* dient RMF, da sich herausgestellt hat, dass sämtliche für die Vorhersage benötigten Daten im RMF-Datenbestand enthalten sind [Hag07]. Das Aufzeichnen der Daten erfolgt über so genannte *Reader*-Module. Dabei ist es möglich mehrere verschiedene solcher *Reader* zu implementieren und je nach Wunsch den jeweiligen zum Aufzeichnen zu nutzen. Diese Auswahl muss über die Haupt-Konfigurationsdatei `WLPDefaults.properties` vorgenommen werden.

Aktuell ist aber nur der *CIMReader* implementiert, mit dem die Aufzeichnung der Metriken über das von der *Distributed Management Task Force* standardisierte *Common Information Model* (CIM) erfolgt. Bei CIM handelt es sich um einen offenen Standard, der Definitionen für das Management von IT-Systemen, Netzwerken, Anwendungen und Services bereitstellt. Mit diesen standardisierten Definitionen wird der Austausch von einheitlichen Informationen über Management-Schnittstellen zwischen verschiedenen Systemen anbieter- und plattformunabhängig ermöglicht [Dis08a]. CIM ist weitverbreitet und wird u. a. auch von CP verwendet, welches in Zukunft idealerweise mit dem *WLP-Framework* kommunizieren soll. Auch deswegen fiel in [Hag07] die Wahl auf den CIM-Standard.

Um die RMF-Metriken in CIM abzubilden, wurden die CIM-Standardtypen

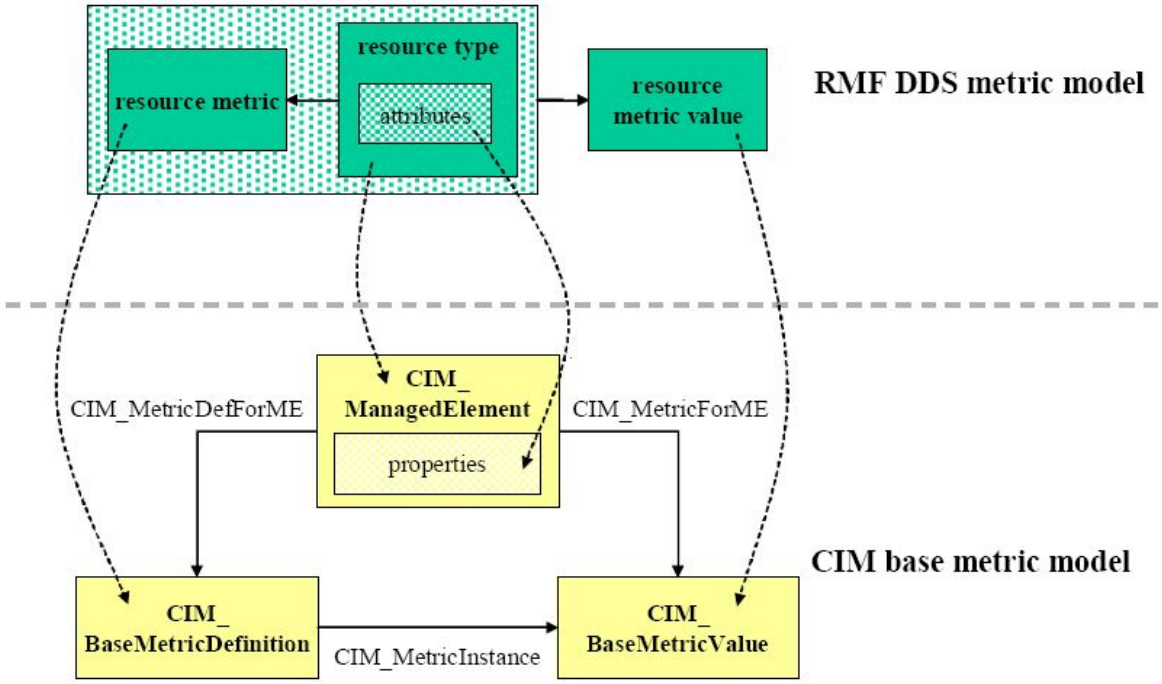


Abbildung 2.2.: Abbildung von RMF Metrik-Elementen auf CIM-Elemente (Abbildung aus [Hag07])

`CIM_BaseMetricDefinition` und `CIM_BaseMetricValue` zu `IBMzOS_BaseMetricDefinition` und `IBMzOS_BaseMetricValue` erweitert. Die Beziehung zwischen Metrik-Definition und Metrik-Wert wird dabei über so genannte `Associations` definiert. Zusätzlich sollten die Definition-Werte-Paare auch mit einer Instanz von `CIM_ManagedElement` verknüpft sein. Dieser Zusammenhang wird in Abbildung 2.2 nochmals veranschaulicht.

Die Bereitstellung von Daten mit Hilfe des CIM-Datenmodells erfolgt über eine *Client/Server*-Architektur, in welcher der Server Daten bereitstellt auf die der Client jederzeit durch Anfragen zugreifen kann.
Auf dem Server kommt *OpenPegasus* zum Einsatz. Es handelt sich dabei um ein Open Source Projekt der *Open Group* das sich aus CIM-Server, *Object Broker* (CIMOM) und CIM-Provider für die wichtigsten Klassen zusammen setzt. Es ist in C++ implementiert um möglichst schnell und effizient zu sein. Zusätzlich ist es darauf ausgelegt möglichst modular und portabel zu sein [Ope08]. *OpenPegasus* ist in den neueren z/OS-Releases bereits integriert.
Auf dem Client findet der aus dem Open Source Projekt *Standards Based Linux Instrumentation for Manageability* (SBLIM) stammende *SBLIM CIM Client for Java*

Verwendung. Auch der Client wurde auf Geschwindigkeit und Effizienz optimiert, bietet zusätzlich aber auch Vorteile beim Debuggen durch Bereitstellen von *Logfiles* [Sta08].
Als Übertragungsprotokoll für den Austausch zwischen Client und Server wird das ebenfalls von der DMTF standardisierte CIM-XML verwendet. Es kapselt die Daten plattformunabhängig in XML und benutzt zur Übertragung eine TCP/HTTP-Verbindung [Dis08b].
Die über CIM abgefragten Metriken speichert der *RMFRecorder* zur weiteren Verarbeitung durch die Vorhersage-Anwendung in einer relationalen Datenbank ab. Da selten alle vorhandenen Metriken benötigt werden und eine komplette Speicherung über Wochen oder Monate hinweg zu viel Speicherplatz in Anspruch nehmen könnte, besteht die Möglichkeit einer Merkmalsselektion über die Konfigurationsdatei *cimmetrics.xml*. Diese spezielle Auswahl der gewünschten Metriken erfolgt dabei mit Hilfe von *Include/Exclude*-Anweisungen, die bestimmte Komponenten modular von vornherein ein- oder ausschließen. Ein mögliches Beispiel für eine solche Konfigurationsdatei findet sich in Abbildung 2.3.

Weitere Konfigurationsmöglichkeiten für den *RMFRecorder* lassen sich in der Haupt-Konfigurationsdatei `WLPDefaults.properties` vornehmen. Neben der Möglichkeit die Stützstellen der Zeitreihen durch *Downsampling* zu reduzieren und dadurch weiter Speicherplatz zu sparen wird dort u. a. die Datenbank näher spezifiziert, festgelegt in welchem Zeit-Intervall Aufzeichnungen erfolgen sollen und das oben erwähnte Modul des *Readers* ausgewählt.
Der schematische Aufbau des *RMFRecorders* wird in Abbildung 2.4 dargestellt. Hier wird neben dem aktuell implementierten *CIMReader*-Modul auch das optionale *DDSReader*-Modul abgebildet, welches die RMF-Metriken mit Hilfe des *Dedicated Data Server* (DDS) aufzeichnet.

2.2. Datenhaltung: *Apache Derby*

Ein wichtiger Aspekt bei der Vorhersage mit künstlichen neuronalen Netzen ist die Datenhaltung. Je mehr Daten für das Training des Netzes zur Verfügung stehen, desto höher die Chancen auf gute Vorhersage-Ergebnisse. Da diese Daten optimalerweise auch über einen längeren Zeitraum verfügbar sein müssen, wurden sie in einer relationalen Datenbank gespeichert.

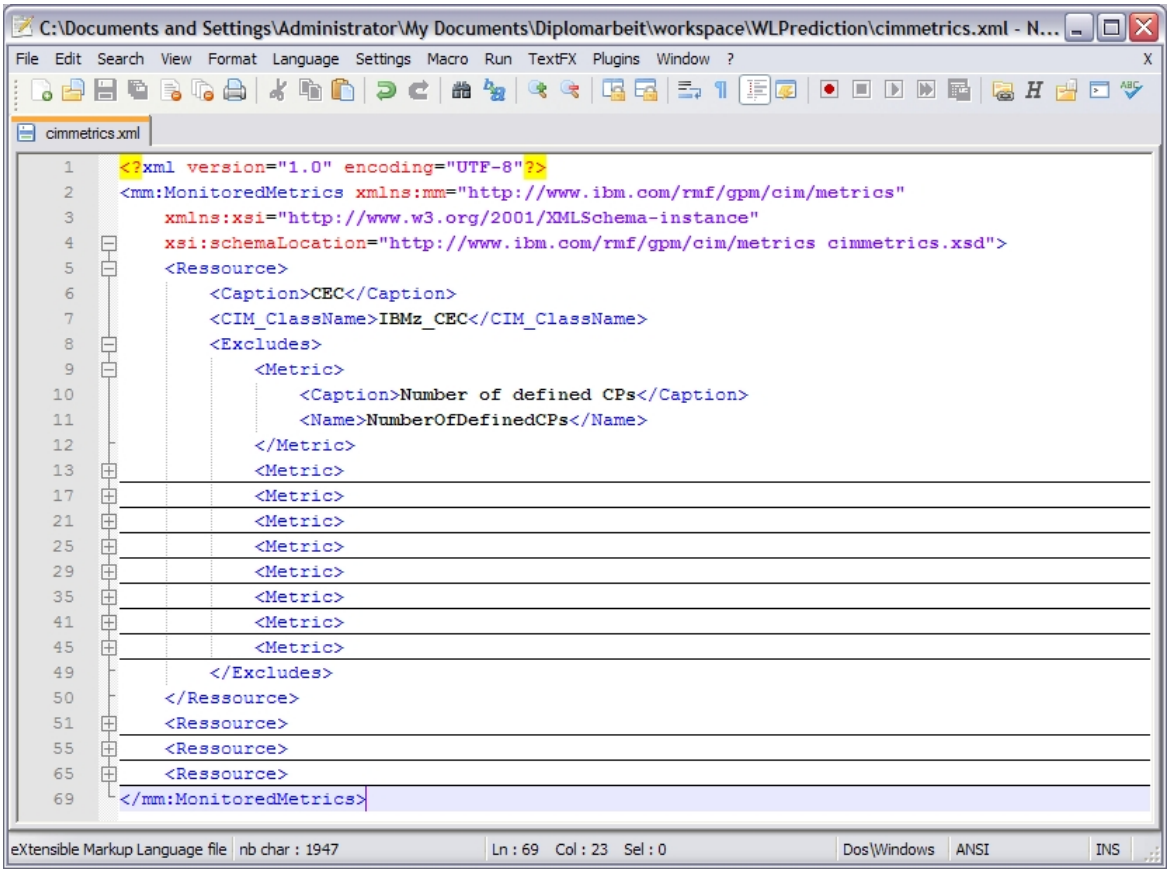


Abbildung 2.3.: *CIMReader* Konfigurationsdatei (`cimmetrics.xml`)

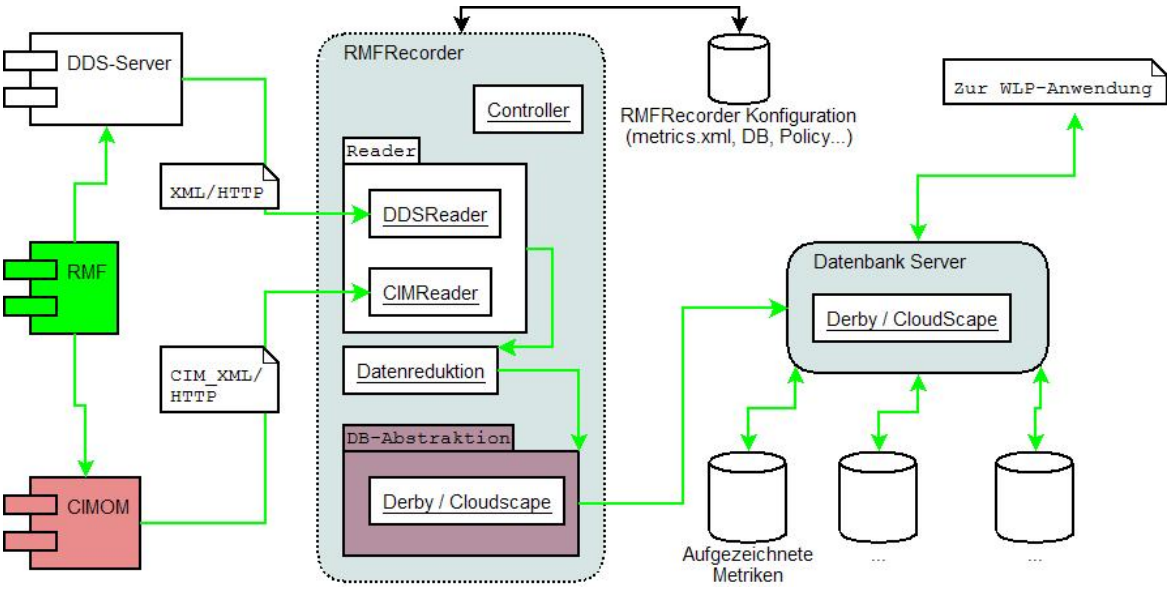


Abbildung 2.4.: Schematische Übersicht des *RMFRecorder* (Abbildung aus [Hag07])

Für diese Zwecke fiel die Wahl auf die *Apache Derby* Datenbank, der Open Source Variante der mittlerweile eingestellten *IBM Cloudscape* Datenbank. Genauso wie der *RMFRecorder* läuft auch diese auf dem Client. Vorteile von *Apache Derby* sind die leichtgewichtige Implementation, die freie Verfügbarkeit und der eingebettete Treiber für *Java Database Connectivity* (JDBC). Da sie zusätzlich in Java implementiert ist und durch ihre Architektur – alle Daten werden in einer Verzeichnisstruktur und in *dat*-Dateien gespeichert – betriebssystemunabhängig ist, stellt sie eine hoch portable und flexible Datenbank dar, die relativ schnell in jede beliebige Java-Anwendung eingebettet werden kann. Schließlich besitzt *Apache Derby* auch noch einen Netzwerk-JDBC-Modus der im vorliegenden Fall unentbehrlich ist, da die benötigten Metriken über eine *Client/Server*-Architektur direkt vom Großrechner über das Netzwerk auf den Client zurückgeschrieben werden [Apa08].

Die erste große Hürde bei der Implementierung stellte das Datenbankdesign dar. Nach dem relationalen Modell hätte jede aufgezeichnete Metrik jeweils eine eigene Spalte bekommen. Durch die festgelegte Vorgabe nach maximaler Flexibilität besaß dieser Ansatz aber einen bedeutenden Nachteil. In der Folge hätte dies dazu führen können, dass bei jeder Aufzeichnung eine unterschiedliche Anzahl von Spalten verwendet worden wäre. Zudem könnte die selbe Metrik in verschiedenen Aufzeichnungen jeweils in unterschiedlichen Spalten gespeichert worden sein.
Unter den gegebenen Voraussetzungen war dieser Ansatz nicht realisierbar, da sämtliche in der *WLPrediction*-Anwendung eingebetteten SQL-Abfragen je nach vorliegendem Fall angepasst werden müssten. Um dieses Problem zu lösen, hat sich Hagmann in [Hag07] eines „Tricks“ bedient und das so genannte *generische Tabellenformat* eingeführt: Die Metrik-Bezeichnungen werden als Teil des Primärschlüssels verwendet und alle Metriken kommen in die selbe Spalte. Somit ist die Anzahl der Spalten fix, die Werte dieser Metrik-Spalte sind aber variabel und jederzeit problemlos erweiterbar. Abbildungen 2.5 und 2.6 erläutern diese Idee anschaulich. Dieses *generische Tabellenformat* ermöglicht es weiterhin den Plan einer flexiblen Architektur mit automatischer Merkmalsselektion zu realisieren. Damit ist es theoretisch möglich, eine größere Anzahl an Metriken zu speichern und für den jeweils vorliegenden Fall das optimale Vorhersagemodell mit den dafür benötigten Metriken auszuwählen.
Die Metriken werden dazu in der Tabelle `metrics` gespeichert, die wie in [Hag07] beschrieben mit dem folgenden SQL-Befehl erstellt wurde:

```
CREATE TABLE metrics
(
  TimeStamp TIMESTAMP NOT NULL,
  InstanceId VARCHAR (50) NOT NULL,
  MetricId VARCHAR (20) NOT NULL,
  MetricValue VARCHAR (20) NOT NULL,
  DataType SMALLINT NOT NULL,
  PRIMARY KEY (TimeStamp, InstanceId)
);
CREATE INDEX met_id ON metrics (MetricId);
```

Der Schlüssel setzt sich aus dem Zeitstempel und der Instanz-ID zusammen. Da aber auch eine Suche nach der Metrik in manchem Fall sinnvoll sein könnte wurde darauf zusätzlich ein Index erstellt um potentielle Abfragen darauf zu beschleunigen.

Abschließend betrachtet, stellte sich das *generische Tabellenformat* als sehr großer Nachteil bei der Implementierung der *WLPrediction*-Anwendung heraus. Hauptproblem war die Reihenfolge der in der Datenbank gespeicherten Metriken. Da der Primärschlüssel aus einer Kombination der Spalten *TimeStamp* und *InstanceId* bestand, konnte diese Reihenfolge nicht garantiert werden. Bei künstlichen neuronalen Netzen ist dieser Punkt aber zwingende Voraussetzung für eine korrekte Funktionsweise, da die Metriken immer in der selben Reihenfolge auf den Eingangsneuronen anliegen müssen. Diese Tatsache führte bei der Implementierung zu großen Problemen und hatte die Entwicklung eines Hilfskonstrukt zur Folge, welches die Komplexität des *WLP-Frameworks* deutlich erhöhte.
Ein Teil dieser Hilfkonstruktion war z. B. für die Konvertierung vom *generischen* in

ID	Spalte 1	Spalte 2	Spalte 3
1	Wert (1,1)	Wert (1,2)	Wert (1,3)
2	Wert (2,1)	Wert (2,2)	Wert (2,3)
3	Wert (3,1)	Wert (3,2)	Wert (3,3)
...			

Abbildung 2.5.: Relationales Tabellenformat (Abbildung aus [Hag07])

ID	Spalte	Wert
1	Spalte 1	Wert (1,1)
1	Spalte 2	Wert (1,2)
1	Spalte 3	Wert (1,3)
2	Spalte 1	Wert (2,1)
2	Spalte 2	Wert (2,2)
2	Spalte 3	Wert (2,3)
3	Spalte 1	Wert (3,1)
3	Spalte 2	Wert (3,2)
3	Spalte 3	Wert (3,3)

Abbildung 2.6.: *Generisches Tabellenformat* (Abbildung aus [Hag07])

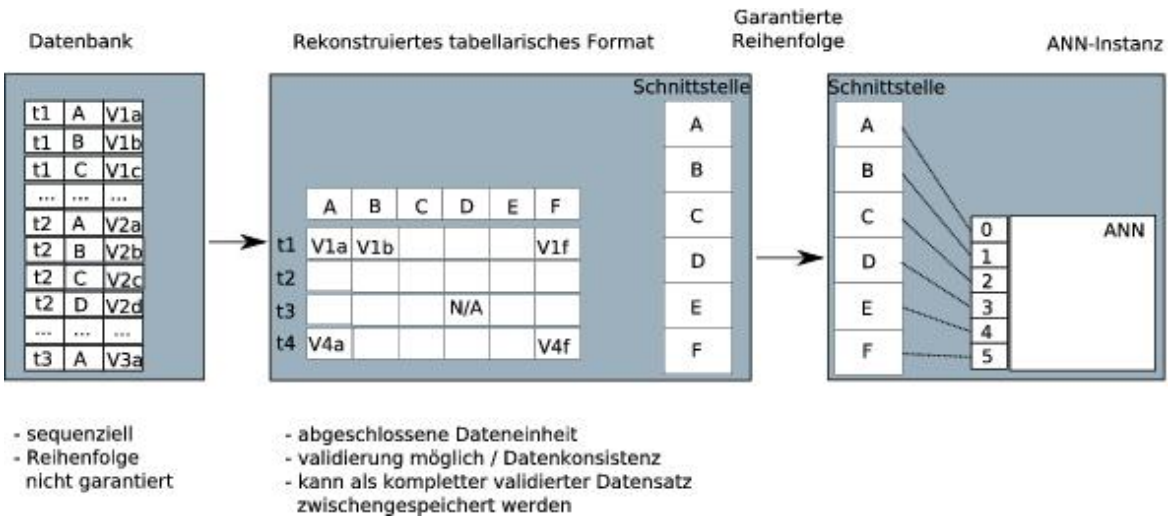


Abbildung 2.7.: Konvertierung vom *generischen* in das von der *WLPrediction*-Anwendung benötigte relationale Tabellenformat (Abbildung aus [Hag07])

das relationale Tabellenformat zuständig. Abbildung 2.7 stellt diesen Teil schematisch dar.
Auf die Beseitigung dieses gravierenden Nachteils wird in Kapitel 3 dieser Arbeit detailliert eingegangen.

2.3. Datenverarbeitung: *WLPrediction*

Die dritte und letzte Komponente des *WLP-Frameworks* bildet die *WLPrediction*-Anwendung. Aufgabe dieser Komponente ist die Generierung von Vorhersagen mit den vorher durch den *RMFRecorder* aufgezeichneten Metriken. Diese Vorhersagen werden mit Hilfe von künstlichen neuronalen Netzen berechnet. Wie die beiden Komponenten davor, läuft auch *WLPrediction* auf dem Client-Rechner.
Kernstück der Anwendung ist das *Agent Building and Learning Environment* (ABLE), welches für die Umsetzung des künstlichen neuronalen Netzwerks zuständig ist. Dabei handelt es sich um ein Projekt im Rahmen des *IBM alphaWorks* Programm das eine Java-Entwicklungsumgebung und Komponentenbibliothek für die Entwicklung von Anwendungen und so genannten „Agenten“ basierend auf Elementen des maschinellen Lernens und der Künstlichen Intelligenz bereitstellt [IBM08a]. Diese Funktionen wurden in ABLE-Beans realisiert und setzen sich laut [IBM08a] im Wesentlichen wie folgt zusammen:

- Data Beans
- Learning Beans
- Rule Beans
- ABLE Agents

Durch das Beans-Konzept ist es möglich mehrere ABLE-Beans zu Agenten zusammenzufassen oder gar mehrere Agenten zu einem neuen Agenten zu verknüpfen. Zusätzlich zu den schon vorhandenen ABLE-Beans ist es möglich eigene Beans mit speziell benötigten Funktionen zu implementieren und in das *WLP-Framework* einzubinden. Dies ist auch in der *WLPrediction* des vorliegenden Prototypen so geschehen. Der implementierte *WlpAgent* besteht aus den folgenden Beans:

- `TrainingImport` (Trainingsdaten)
- `TestImport` (Testdaten)
- `OnlineImport` (Onlinedaten)
- `InFilter` (Skalierung der Eingangsdaten)
- `MovingAverageFilter` (Glätten der Daten mit gleitendem Mittelwert)
- `TimeSeriesFilter` (Mapping der Metriken auf die Eingangsneuronen)
- `BackPropagation` (Lernalgorithmus für das *Feed-Forward* Netzwerk)
- `OutFilter` (Rückskalierung der Ausgangsdaten)

Sämtliche Beans wurden entweder neu implementiert oder sind aus den schon vorhandenen ABLE-Beans durch Erweiterung und Anpassung entstanden. Dabei kamen die Ergebnisse aus den theoretischen Arbeiten [Kle06] und [Geb06] zum Einsatz.

WLPrediction benötigt für die Vorhersagen einen trainierten *WlpAgent*, den es aus einem vorher festgelegten `AgentPool` bezieht. Das Erstellen, Trainieren und Speichern des jeweiligen Agenten ist Aufgabe des Benutzers und muss vor der Ausführung von *WLPrediction* erfolgen. Diese Aufgabe wird mit dem *ABLE-Editor* erledigt, der Teil des *ABLE-Frameworks* ist und im *Eclipse-Workspace* als *ablegui*-Projekt zu finden ist. Ein im *ABLE-Editor* frisch angelegter *WlpAgent* wird in Abbildung 2.8 dargestellt:

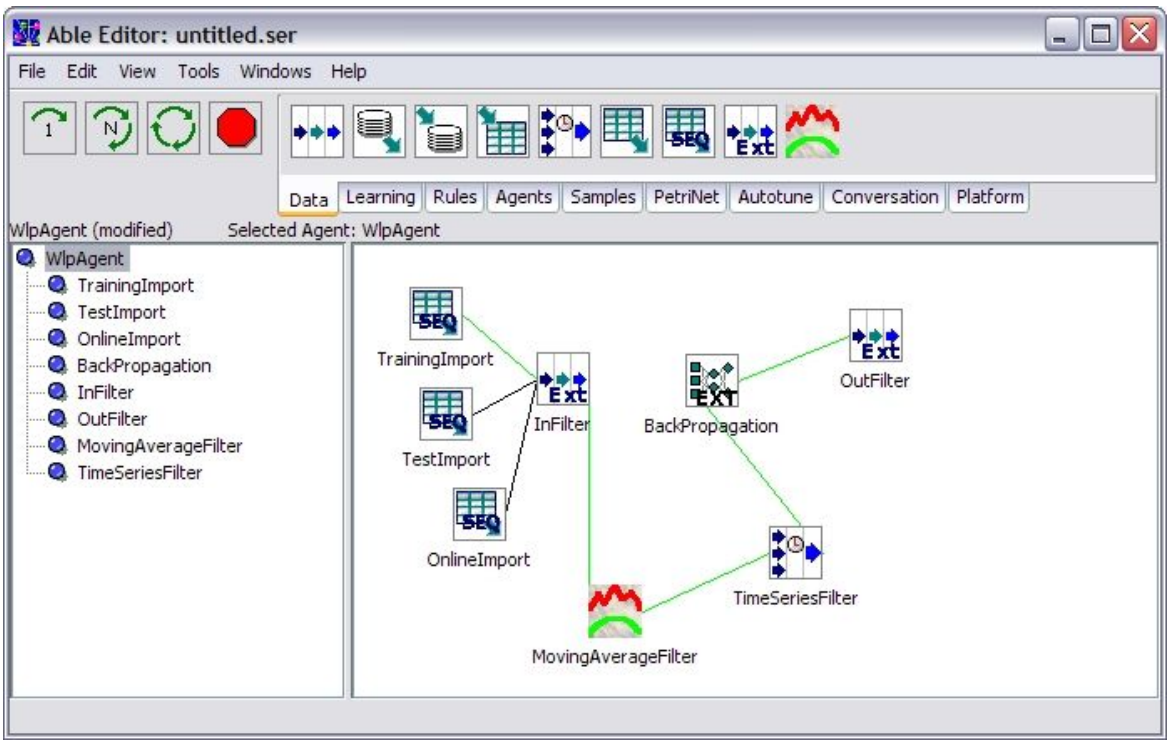


Abbildung 2.8.: *ABLE-Editor* mit erstelltem *WlpAgent*

Bei der Erstellung des *WlpAgents* werden neben den Angaben zur Datenbank und den Tabellen unter anderem auch die gewünschten Metriken, Intervalle und Vorhersageschritte angegeben. Mehr Details zur Erstellung und zum Trainieren von Agenten mit Hilfe des *ABLE-Editors* folgen in Kapitel 3.1 und in der Bedienungsanleitung in Anhang A.

Die *WLPrediction*-Anwendung speichert als Ausgabe drei Werte in die Datenbank, genauer in die Tabelle `prediction`. Neben dem tatsächlichen Wert der einem bestimmten Zeitstempel zugeordnet ist, wird für jeden Zeitstempel auch noch der mit dem gleitenden Mittelwert geglättete Wert zurückgeschrieben. Dieser geglättete Wert dient dann auch als Zielwert für den dritten und letzten Wert. Dabei handelt es sich um den Vorhersagewert, der um die im *WlpAgent* festgelegten Zeitschritte verschoben wird.
Durch das Speichern aller drei Werte ist es möglich die Vorhersagen zu validieren und nach dem Eintreten der tatsächlichen Werte je nach Qualität der Vorhersagen bestimmte Aktionen manuell oder automatisiert durchzuführen. Dies ist bisher aber nur angedacht und noch nicht implementiert worden. Denkbar wären z. B. folgende Aktionen: Ein Nachtrainieren des Agenten, das Ändern der verwendeten

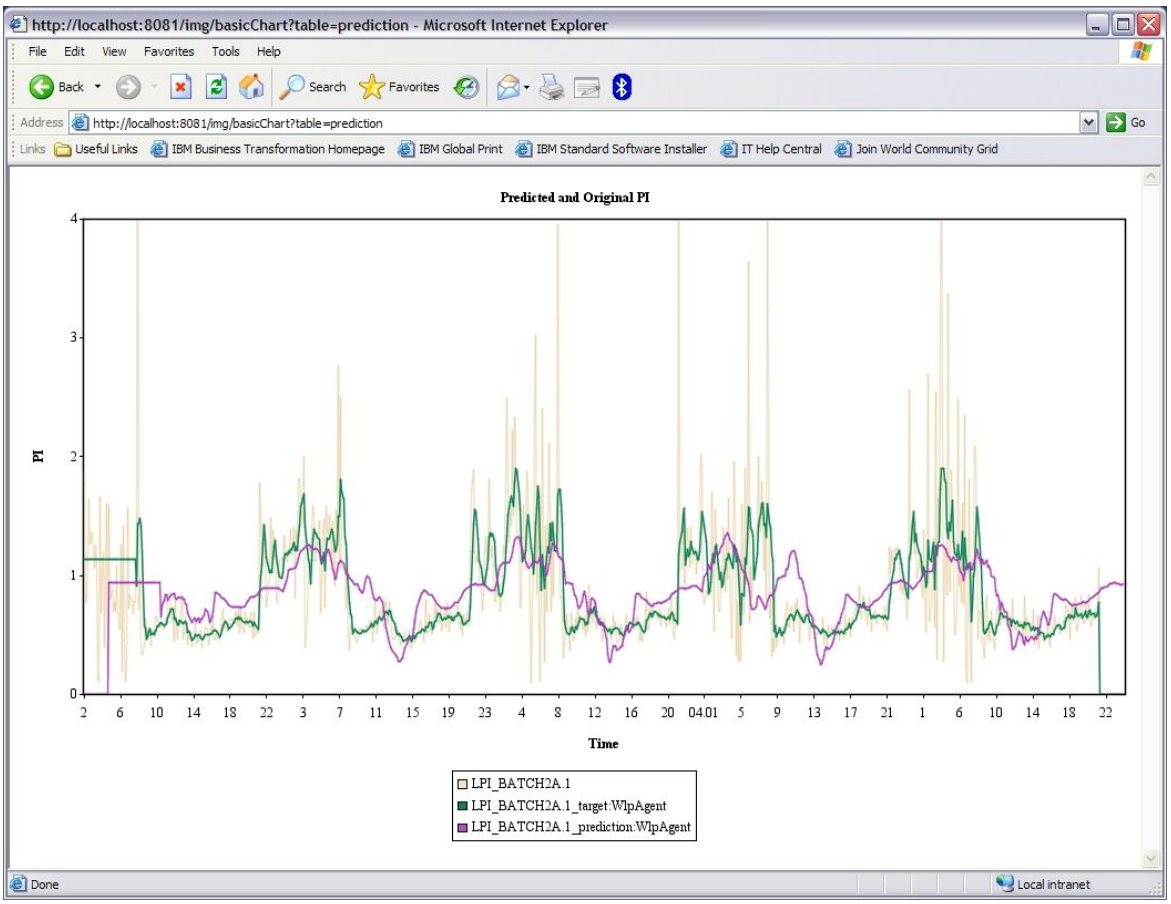


Abbildung 2.9.: Graphische Darstellung der Vorhersagewerte mit Hilfe von *Jetty* und *JCharts*

Metriken oder ein Wechsel des benutzten neuronalen Netzes, falls mehrere zur Verfügung stehen sollten.
Der Anwender hat zudem die Möglichkeit sich diese drei Werte graphisch anzeigen zu lassen. Zu diesem Zwecke wurde in die *WLPrediction*-Anwendung der Open Source Webserver *Jetty* eingebettet. Durch seine schlanke und einfach gehaltene Art bietet er eine effiziente Lösung. Er ist dafür ausgelegt in anderen Anwendungen als Komponente eingebettet zu werden, und da er in Java programmiert wurde erleichterte dies den Einsatz im *WLP-Framework* [Mor08]. Das eigentliche Schaubild, das alle drei Werte als Diagramm abbildet wird mit Hilfe des Java-Pakets *JCharts* erstellt [Kry08]. Diese graphische Darstellung im Browser ist um ein Vielfaches aussagekräftiger als eine Serie gespeicherter Werte in einer Datenbank. Abbildung 2.9 zeigt eine derart erzeugte graphische Darstellung der Vorhersage-Werte.

Nachdem nun mit dem Aufbau und den einzelnen Komponenten des *WLP-Frameworks* die Grundlagen für diese Arbeit detailliert betrachtet wurden, soll mit der Weiterentwicklung dieses Frameworks im nächsten Kapitel die eigentliche Hauptaufgabe begonnen werden.

3. Weiterentwicklung des *WLP-Frameworks*

Im Rahmen dieser Arbeit soll das *WLP-Framework*, das in der Machbarkeitsstudie von Hagmann [Hag07] entstanden ist und als Prototyp vorliegt, entscheidend weiterentwickelt werden. Dieser Prototyp basiert auf den vorliegenden im größten Teil theoretischen Ergebnissen der ersten beiden Arbeiten von Kleeberg [Kle06] und Gebhard [Geb06] des *Joint Research Projects* „Workload Prediction auf Mainframes“.
Um einen Einstieg in das komplexe Thema zu finden, besteht die erste Aufgabe darin, eine Art „Bedienungsanleitung“ zu erstellen, da in diese Richtung bisher keine Dokumentation vorhanden ist und das *WLP-Framework* sehr umfangreich und komplex in seiner Bedienung ist.
Durch das dadurch angeeignete Wissen wird anschließend die vorhandene *Apache Derby* Datenbank gegen die IBM *DB2 for z/OS* Datenbank getauscht um damit die Umstellung auf das XML-Speicherformat vorzubereiten. Dadurch werden für jeden Zeitstempel sämtliche Metriken in einem eigenen XML-Dokument abgelegt.
Diese Umstellung ist der erste Schritt zur Elimination des *generischen Tabellenformats*. Im Anschluss wird die eigentliche Refaktorisierung durchgeführt. Bei diesem weitreichenden strukturellen Umbau werden mehrere Klassen editiert bzw. entfernt um als Ziel die Nachteile des *generischen Tabellenformats* zu beseitigen und das *WLP-Framework* von unnötiger Komplexität zu befreien.
Nach dem die Datenbank schon auf dem Großrechner läuft wird als letztes auch der *RMFRecorder* sowie die *WLPrediction* darauf migriert, indem sie auf den Großrechner übertragen und für die dortige Ausführung konfiguriert werden.

3.1. Erstellen einer Bedienungsanleitung

Das Erstellen einer Bedienungsanleitung ist Teil dieser Diplomarbeit. Da sie stellenweise detaillierte Schritt-für-Schritt Anweisungen mit Screenshots enthält und dadurch einen großen Umfang besitzt, wird sie an dieser Stelle nur kurz angeschnitten Sie befindet sich aber in vollständigem Umfang in Anhang A dieser Arbeit.
Aktuell ist diese Anleitung in sieben Kapitel unterteilt und als „work-in-progress“ deklariert. Mit der Weiterführung dieses Projekts sollte auch die Bedienungsanleitung ergänzt werden. Zu Beginn werden die Voraussetzungen für den Betrieb des *WLP-Frameworks* erläutert. Dies ist ein wichtiges Kapitel, da für das erfolgreiche Ausführen der Anwendungen einiges beachtet werden muss. Um neben der Bedienungsanleitung auch eine Art „Checkliste“ dieser Voraussetzungen zur Hand zu haben, wird ein Ablaufdiagramm für beide Anwendungen erstellt. Abbildung 3.1 zeigt dieses Diagramm für den *RMFRecorder* und Abbildung 3.2 analog für die *WLPrediction*.
Da für das *WLP-Framework* ein *Vicom*-System zum Einsatz kommt, wird in diesem Kapitel der *Initial Program Load* (IPL) bei Verwendung eines solchen Systems beschrieben. *Vicom* ist ein Virtualisierungssystem, das oftmals beim Entwickeln von Anwendungen für den Großrechner zum Einsatz kommt. Damit ist es beispielsweise möglich für ein System die gewünschte Betriebssystemversion aus einer Liste von vorhandenen Versionen hochzufahren.
Anschließend werden im zweiten Kapitel die Möglichkeiten zur Herstellung einer Verbindung mit der verwendeten Datenbank *DB2 for z/OS* beschrieben. Zu diesem Zweck werden die für das Verbinden nötigen Schritte mit den Programmen *Data Studio Developer* und *SQuirreL* erklärt. Detailliertere Informationen zu diesen beiden Programmen folgen in den Kapiteln 3.2.3.1 und 3.2.3.2.
Im nächsten Kapitel wird die Benutzung des *ABLE-Editors* erläutert. Dabei werden vom Erstellen eines *WlpAgents* über dessen Training bis hin zum abschließenden Speichern im `AgentPool` sämtliche Schritte im Detail erklärt.
Kapitel vier und fünf behandeln das Starten des *RMFRecorders* und der *WLPrediction*, bevor in Kapitel sechs die Vorgehensweise zur Migration des *WLP-Frameworks* auf den Großrechner angegeben wird.
Das letzte Kapitel umfasst momentan nur eine unsortierte Liste von wichtigen Hinweisen und generellen Tipps zu dem vorliegenden Thema.

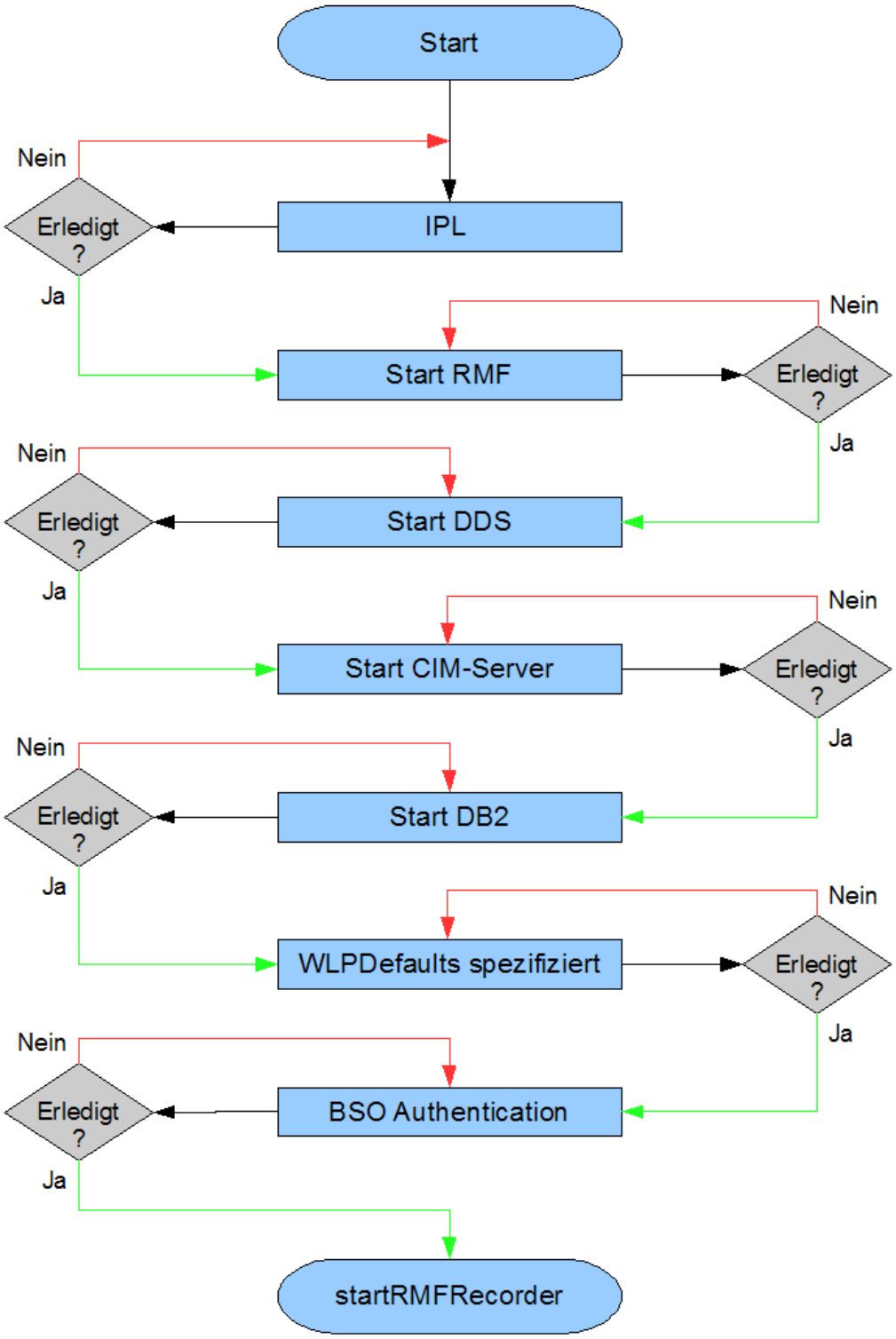


Abbildung 3.1.: Ablaufdiagramm für die Inbetriebnahme des *RMFRecorder*

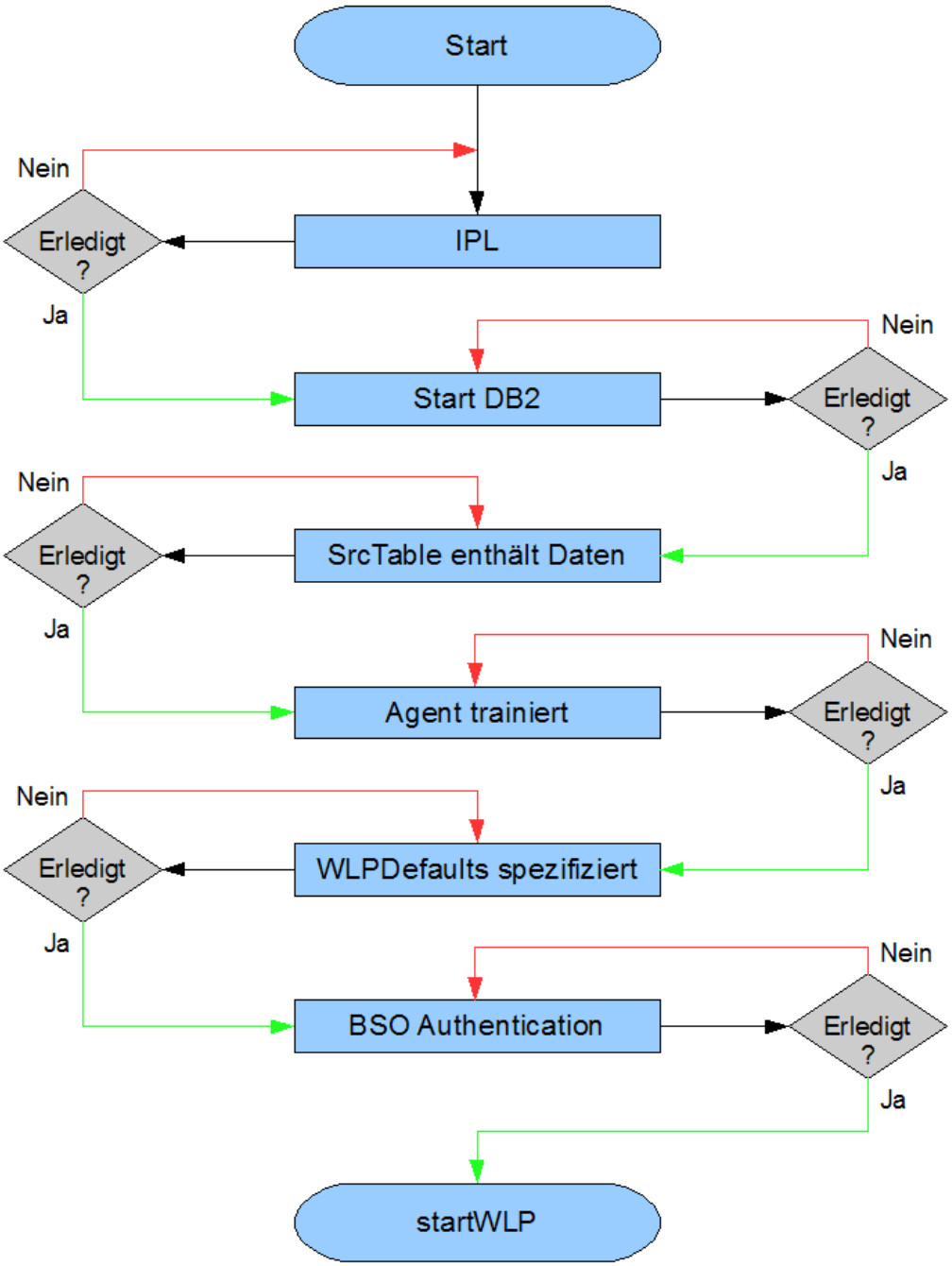


Abbildung 3.2.: Ablaufdiagramm für die Inbetriebnahme der *WLPrediction*

3.2. Umstellen der Datenbank von *Apache Derby* auf *DB2 for z/OS*

Obwohl *Apache Derby* eine solide und gut funktionierende Datenbank ist, gibt es mindestens zwei Gründe, die für eine Umstellung auf DB2 sprechen.
Zum einen ist DB2 eine neue Art hybrider Datenbanken, die die Vorteile einer relationalen Datenbank mit denen der XML-Speichertechnologie kombiniert. Dies bietet die Möglichkeit zur Elimination des *generischen Tabellenformats*.
Zum anderen gibt es innerhalb der DB2-Produktfamilie eine als *DB2 for z/OS* bezeichnete Version für den Großrechner. Somit wird durch die Datenbankumstellung gleichzeitig die erste Komponente des *WLP-Frameworks* auf den Großrechner migriert.

3.2.1. DB2 *pureXML*-Technologie

IBMs DB2 Datenbank wurde 1983 auf den Markt gebracht und hat seither viele Evolutionsstufen hinter sich, wobei in den letzten zwei Dekaden kaum eine so gravierend und herausragend war wie der Sprung auf die Version 9. Neben den üblichen relationalen Daten können in dieser Version auch Dokumente, Audio- und Videodateien, Bilder, Webseiten und digital signierte XML-Transaktionen durch das System verwaltet werden. Laut IBM-Angaben ist DB2 9 branchenweit die erste relationale Datenbank, die neben den oben genannten Daten zusätzlich auch XML nativ speichern kann. Dies wird durch die speziellen XML-Spalten einer relationalen Tabelle realisiert und ermöglicht somit einen nahtlosen Informationsfluss zwischen relationalen Daten und XML-Daten. Diese Eigenschaft wird *pureXML*-Technologie genannt und ist mit ein Grund dafür, dass IBM ihre Datenbanken seither als Daten-Server bezeichnen, die sämtliche Arten von Daten verwalten und zur Verfügung stellen können [IBM06].
Neben der neuen *pureXML*-Technologie wurden zwei weitere signifikante Neuerungen eingeführt, eine als *Venom* bezeichnete Speicherkomprimierung und eine verbesserte autonome Speicherverwaltung. Eine weitere ebenfalls interessante Eigenschaft ist der neuerdings ermöglichte Zugriff auf *Oracle*- und *MySQL*-Datenbanken. Im Gegenzug ist mit Datenbanken dieser beiden Hersteller jedoch kein Zugriff auf DB2 möglich [IBM06].

Da für diese Arbeit nur von der *pureXML*-Technologie intensiv Gebrauch gemacht wurde, beschränkt sich die detailliertere Ausführung nur auf dieses Thema und lässt die anderen Themen außer acht. Für tiefer gehende Informationen zu diesem kontinuierlich in der Entwicklung stehenden IBM-Produkt wird auf [IBM08b] verwiesen.

Laut [IBM07] gab es vor DB2 9 für XML-Daten folgende traditionelle Speichermethoden:

- XML-Dokumente in Dateisystemen speichern
- XML-Daten in relationale Datenbanken als *Large Objects* (LOBs) speichern
- Aufsplitten der XML-Daten in mehrere relationale Tabellen und Spalten
- Daten isoliert in reinen XML-Datenbanken speichern

Da alle oben genannten Methoden gravierende Nachteile besitzen ist IBM mit DB2 9 einen komplett neuen Weg gegangen. Die relationalen Services und die XML-Services wurden im Datenbank-Kern eng miteinander verzahnt und bilden somit die Grundlage für den branchenweit ersten hybriden Daten-Server für relationale Daten und XML-Daten. Diese neue Art hybrider Datenbanken wird in Abbildung 3.3 veranschaulicht.
Die *pureXML*-Technologie umfasst laut [IBM07] u. a. folgende Eigenschaften:

- Einführung eines neuen *pureXML*-Datentyps und neuer Speichertechniken um hierarchische Strukturen, die in XML üblich sind, effizient zu verwalten
- *pureXML* Index-Technologien um Abfragen auf bestimmte Teile eines XML-Dokumentes zu beschleunigen
- Einführen der standardisierten Abfragesprachen *XML Query Language* (XQuery) und SQL/XML
- Die Möglichkeit XML-Schemata zu verwalten, zu validieren und weiterzuentwickeln
- Funktionen zum Aufsplitten und Zusammenbauen von XML-Dokumenten aus bestehenden relationalen Daten

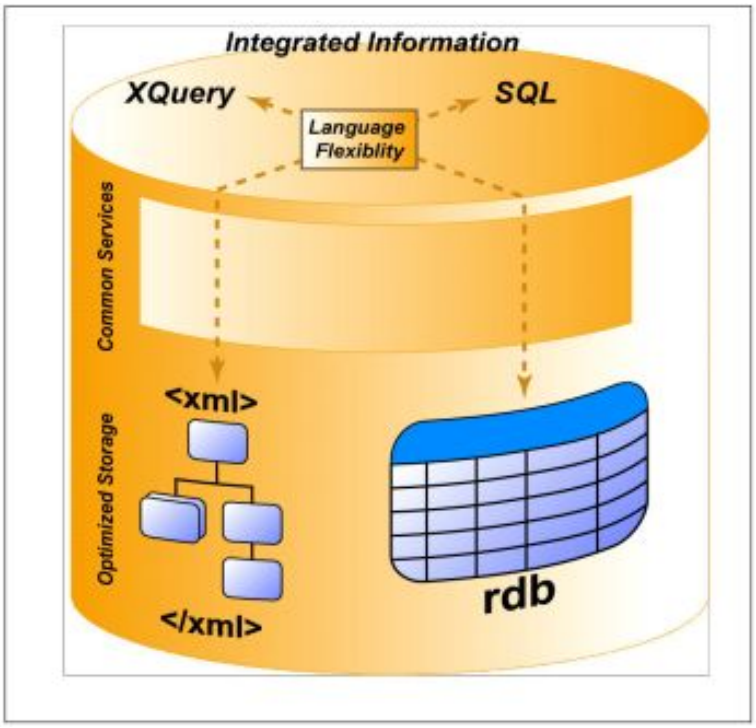


Abbildung 3.3.: Hybride Datenbank DB2 9: Kombinierte Speicherung von XML-Daten und relationalen Daten (Abbildung aus [IBM07])

Im Gegensatz zu anderen relationalen Datenbanken, die XML in LOBs speichern, hat DB2 9 extra einen neuen XML-Datentypen eingeführt. Dieser Datentyp speichert die eingelesenen XML-Dokumente in einer hierarchischen Baum-Struktur, wodurch die aufwendige Umwandlung in eine relationale Form entfällt. Durch das Wegfallen dieses Mehraufwands wird das Ablegen und Abfragen von XML-Daten entsprechend effizient ermöglicht. Dieser entscheidende Vorteil wird ebenfalls in der erwähnten Abbildung 3.3 dargestellt.

Mit den in DB2 9 integrierten Möglichkeiten zur hybriden Datenverwaltung erhält man einen führenden Daten-Server für relationale Daten und XML-Daten, der einfach zu bedienen ist, auf Standards basiert und in der Industrie schon jahrelang bewährt ist. Gerade in Zeiten in denen XML in immer mehr Bereichen Einzug hält, ist die effiziente Speicherung und Abfrage ohne zeitaufwendiges Umwandeln in das relationale Modell ein wesentlicher Faktor um Geschäftsprozesse leichter, schneller und billiger zu implementieren.
Im vorliegenden Fall wird ein wesentlicher Nachteil des *WLP-Frameworks*, das *generische Tabellenformat*, mit Hilfe der *pureXML*-Technologie beseitigt.

3.2.2. Durchführung der Datenbankumstellung

Die Umstellung von *Apache Derby* auf *DB2 for z/OS* ist nicht ganz trivial. Die DB2 Version für den Großrechner hat einen historisch gewachsenen Sonderstatus innerhalb der DB2-Produktfamilie. Um die Problematik zu verdeutlichen, müssen zuerst einige Informationen über die DB2-Produktfamilie erläutert werden.
Die wohl wichtigste und älteste DB2-Produktlinie besteht seit 1983 für die Großrechner und wird heute *DB2 for z/OS* genannt. In den 90er-Jahren kamen dann auch Versionen für OS/2, UNIX und Windows dazu. Zu dieser weiteren wichtigen Produktlinie kam inzwischen Linux als Betriebssystem hinzu, und da OS/2 schon lange vom Markt verschwunden ist, wird diese Produktlinie als *DB2 for Linux, UNIX and Windows* oder kurz als *DB2 LUW* bezeichnet. Daneben gibt es noch zwei weitere Produktlinien für *System i* und z/VSE respektive z/VM, die praktisch keine Rolle mehr spielen und auf die deswegen nicht weiter eingegangen wird [Wik08].
Zwischen *DB2 for z/OS* und *DB2 LUW* bestehen entscheidende Unterschiede, angefangen mit der Tatsache, dass Ersteres in PL/S und Letzteres in C++ implementiert wurde.
Zudem ist *DB2 for z/OS* für Kunden mit geschäftskritischen, komplexen und schwierigen Ansprüchen, denen *DB2 LUW* nicht gerecht werden kann weil es darauf nicht ausgelegt ist.
Ein weiterer Unterschied ist die Verknüpfung zwischen Hard- und Software auf dem Großrechner. So besteht bei der Weiterentwicklung von *DB2 for z/OS* als Software wie auch von *System Z* als Hardware eine beidseitige enge Zusammenarbeit. Dies wird beispielsweise dadurch deutlich, dass DB2 auf dem Großrechner ein eigenes Subsystem von z/OS darstellt.
Mit diesem Hintergrund ist es nicht verwunderlich, dass spezielle Eigenschaften, die auf dem Großrechner nicht wegzudenken sind auf *DB2 LUW* nicht existieren und auch gar nicht realisierbar sind, weil die verwendete Hardware dies gar nicht unterstützen bzw. anbieten kann. Als Beispiel soll hier stellvertretend das *Parallel Sysplex Data Sharing* genannt werden, welches exklusiv auf Großrechner beschränkt ist. Dennoch sind beide Versionen in den Kernfunktionen, die ein *Database Management System* (DBMS) ausmachen größtenteils identisch. Die oben aufgeführten Unterschiede stellen nur einen Auszug dar, weitere Unterschiede und allgemeine Informationen zur DB2-Produktfamilie finden sich in [Wan06].
Es soll damit zum Ausdruck gebracht werden, dass *DB2 for z/OS* in Installati-

on und Administration von vergleichbaren Produkten auf „gewöhnlichen“ x86-Architekturen wesentlich abweicht. Da im Rahmen dieser Diplomarbeit eine Einarbeitung in benötigtem Maße nicht möglich war, wurde das Aufsetzen der Datenbank von einem erfahrenen Datenbank-Administrator der IBM durchgeführt. Sämtliche für den Betrieb benötigten Daten und Angaben wurden von diesem nach der Installation zur Verfügung gestellt. Da die Verwendung der *pureXML*-Technologie, welche mit der Version 9 eingeführt wurde, eine zentrale Rolle in dieser Arbeit spielt, muss mindestens diese oder eine aktuellere Version verwendet werden. Im vorliegenden Fall wird *DB2 for z/OS* 9.1 eingesetzt.

Nachdem die Datenbank *WLMD941* angelegt wurde und die Daten für den Host *BOEWLM4* und den Port *5941* bekannt sind, kann mit diesen Daten schließlich die Verbindungs-URL zusammengesetzt werden:
jdbc:db2://BOEWLM4:5941/WLMD941.
Der Austausch der Datenbank ist somit in der Theorie vollzogen. Abschließend müssen dem verwendeten Benutzer *PRAK1* ausreichend Rechte erteilt werden und zudem muss die Umstellung der Verbindungs-URL im Quellcode der Anwendungen *RMFRecorder* und *WLPrediction* erfolgen. Dies wird am besten mit der *Search/Replace*-Funktion von *Eclipse* durchgeführt. Jedes Vorkommen der alten URL *jdbc:derby://localhost:1527/WlpDatabase* muss durch die neue Verbindungs-URL *jdbc:db2://BOEWLM4:5941/WLMD941* ersetzt werden.
Die Datenbankumstellung ist damit erfolgreich abgeschlossen.

3.2.3. Herstellen einer Datenbank-Verbindung

Nachdem die Datenbank aufgesetzt wurde, sollten als nächstes die benötigten Tabellen angelegt werden, da diese zur Laufzeit der Anwendungen *RMFRecorder* und *WLPrediction* schon vorhanden sein müssen. Zu diesem Zweck gibt es je nach Aufgabenbereich und persönlichen Vorlieben viele verschiedene Programme. Im Anschluss wird zum einen der *IBM Data Studio Developer* (im Folgenden als *Data Studio* bezeichnet) und zum anderen die Open Source Anwendung *SQuirreL* betrachtet.
Selbstverständlich kann auch jedes andere Programm, welches sich durch einen JDBC-Treiber mit einer Datenbank verbinden lässt verwendet werden, jedoch kann eine reibungslose Funktionsweise nicht gewährleistet werden. Eine weitere Alternative, die hier allerdings nicht näher beschrieben werden soll, bietet die *Interactive*

System Productivity Facility(ISPF) mit *SQL Processing Using File Input* (SPUFI) über eine 3270-Hostsession.

Bezüglich der Verbindung zu *DB2 for z/OS* mit dem JDBC-Treiber muss auf Folgendes hingewiesen werden:
Beim *Data Studio* wird der benötigte Treiber durch die Installation gleich mitgeliefert, dies ist bei *SQuirreL* nicht der Fall. Das liegt daran, dass der Treiber aus lizenzrechtlichen Gründen einzeln nicht frei verfügbar ist und somit auch nicht in beliebige Anwendungen integriert werden kann. Falls *SQuirreL* als SQL-Client verwendet wird, muss entweder das *Data Studio* zusätzlich installiert werden oder der im Verzeichnis `DB2z-Treiber` der beigelegten DVD bereitgestellte Treiber eingebunden werden. Die Datei `db2jcc.jar` enthält den Treiber und die Datei `db2jcc_licence_cisuz.jar` die benötigte Lizenz, wobei beide gleichermaßen benötigt werden.
Informationen zur Konfiguration der Verbindungsherstellung bzw. zum Einbinden des JDBC-Treibers finden sich in der Bedienungsanleitung in Anhang A.

3.2.3.1. *IBM Data Studio Developer*

Das *Data Studio* bietet eine integrierte Datenbank-Entwicklungsumgebung die auf *Eclipse* basiert und somit ein vertrautes und vielfach bewährtes Werkzeug liefert. Es ist sowohl für Entwickler als auch für Administratoren gedacht und unterstützt beide gleichermaßen mit Funktionen zur Umsetzung von SQL Skripten, *Stored Procedures* oder administrativen Aufgaben [Seu08].
Neben den üblichen Standardfunktionen zum Erstellen und Pflegen von Datenbanken und Tabellen per SQL gibt es für Datenbank-Administratoren laut [Seu08] u. a. folgende Funktionen:

- Erstellen physischer Datendiagramme
- Anzeigen der Datenverteilung
- Objekt-Verwaltung
- Daten-Verwaltung

Für Datenbank-Entwickler hingegen, heben [Seu08] und [IBM08c] u. a. folgende Funktionen hervor:

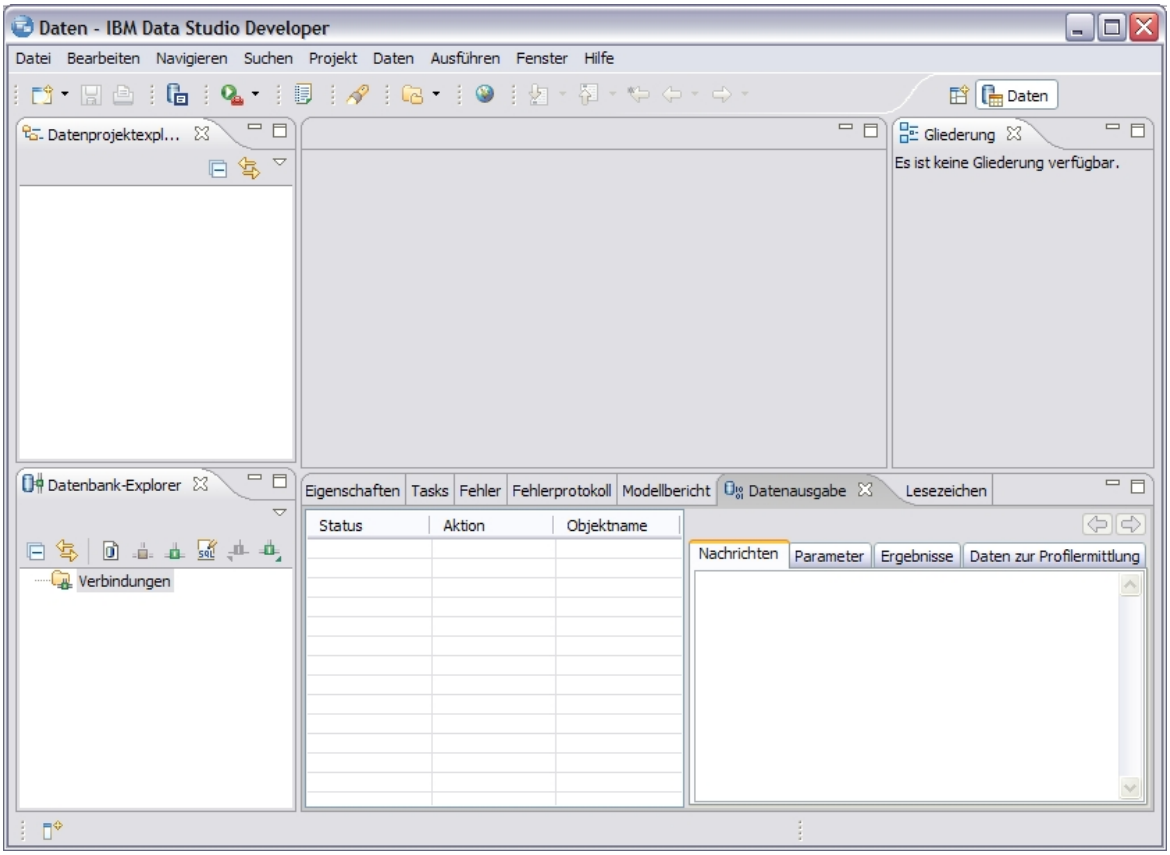


Abbildung 3.4.: *Data Studio* Startbildschirm

- Entwicklung von Datenbank- und Java-Anwendungen (JDBC)
- Möglichkeit zur schnelleren Entwicklung von Datenbank-Anwendungen mit einem integrierten Abfrage-Editor für SQL und XQuery
- *SQL Builder*
- XML (Schema-) Editor
- Erstellen und Testen von *Stored Procedures* (Java und SQL)
- Optimierung von existierenden JDBC-Anwendungen ohne Änderungen am Quellcode

Abbildung 3.4 zeigt einen Screenshot des *Data Studios* in der aktuell vorliegenden Version 1.2.

Durch seine umfassenden Funktionen bietet das *Data Studio* somit viele Vorteile für den längerfristigen Gebrauch. Die Datenbank-Administration sowie die Entwick-

lung und Pflege von Datenbank-Anwendungen werden in großem Maße erleichtert. Häufig verwendete SQL-Befehle können als Skripte abgespeichert und mit einem Klick ausgeführt werden. Außerdem unterstützt es als von IBM entwickeltes Produkt die neue *pureXML*-Technologie am umfassendsten mit zum Teil extra darauf zugeschnittenen Werkzeugen.
Demgegenüber steht aber die Schwerfälligkeit und Komplexität, die solch eine Funktionsfülle mit sich bringt. Es kann einige Zeit dauern bis man sich mit den neuen Funktionen vertraut macht und alles auf die persönlichen Bedürfnisse zugeschnitten wurde. Da es gleichzeitig Entwickler und Administratoren ansprechen soll, wird zudem eine Vielzahl der vorhandenen Funktionen wahrscheinlich nie zum Einsatz kommen.
Bei dieser Arbeit konnte nur auf eine zeitlich beschränkte Demo-Version zurückgegriffen werden, die im Funktionsumfang aber keinerlei Einschränkungen aufwies. Diese kann jederzeit kostenlos von der IBM Webseite herunterladen werden.

3.2.3.2. *SQuirreL SQL*

Im Gegensatz zum *Data Studio* ist *SQuirreL* eine Open Source Anwendung deren Verwendung kostenlos ist. *SQuirreL* ist ein universeller graphischer SQL-Client für relationale Datenbanken, der auf jede Datenbank zugreifen kann, für die ein JDBC-Treiber verfügbar ist. Außerdem ist es in Java geschrieben und somit auf jeder Java-fähigen Plattform lauffähig. Im Vergleich zum *Data Studio* ist *SQuirreL* um ein Vielfaches kleiner und kann wahlweise als installierbare Version oder als gepacktes Archiv bezogen werden. Letzteres liegt sofort nach dem Entpacken in einer lauffähigen Form vor.
Die aktuelle Version ist 2.6.8, wobei sich im Verlauf dieser Arbeit herausgestellt hat, dass in dieser Version eine entscheidende Eigenschaft noch nicht implementiert wurde: Das Anzeigen des nativen XML-Datentyps. Dieses Manko ist in den so genannten *Weekly Snapshots* behoben und erfolgreich getestet worden. Da es stabil läuft kann damit gerechnet werden, dass es in eine der nächsten offiziellen Versionen implementiert wird. Für die vorliegende Arbeit wird einer dieser *Weekly Snapshots* verwendet.
Trotz seiner Kompaktheit bietet *SQuirreL* eine beachtliche Funktionsvielfalt. Der folgende Auszug ist aus [WG08]:

- Betrachten und Editieren von Daten

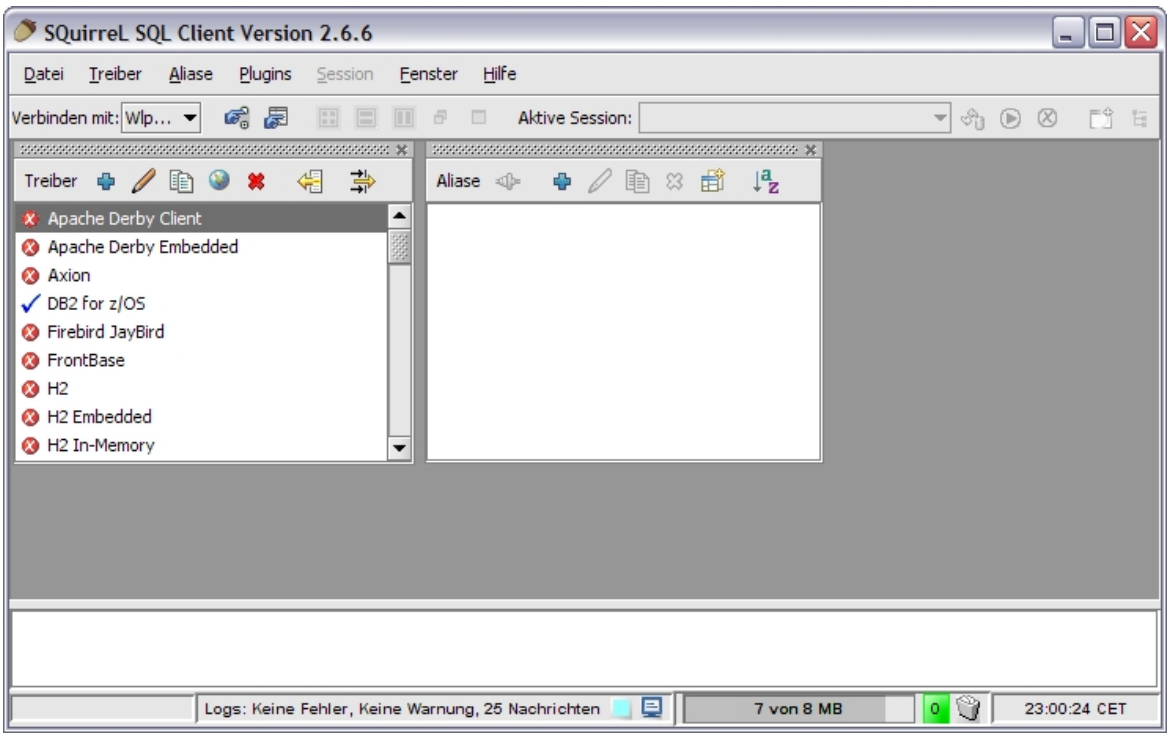


Abbildung 3.5.: *SQuirreL* Startbildschirm

- Betrachten von Datenbank-Metadaten
- Paralleles Arbeiten mit mehreren Datenbanken auf beliebigen Servern
- Arbeiten mit verschiedenen Datenbanksysteme über eine einheitlichen Oberfläche
- Einbinden von speziellen Funktionen des jeweiligen Datenbanksystems durch produktspezifische Plugins
- Bearbeiten von Daten wahlweise direkt in Datenbanktabellen oder per SQL-Befehl

In Abbildung 3.5 ist ein Screenshot von *SQuirreL* zu sehen.

Mit seiner Flexibilität, seiner Leichtigkeit und der Möglichkeit auch heterogene Datenbank-Landschaften mit einem einzigen Werkzeug bearbeiten zu können, bietet *SQuirreL* dem Nutzer gleich mehrere Vorteile.
Für die Entwicklung von Datenbank-Anwendungen ist es hingegen eher ungeeignet und auch in der Administration besitzt es Schwächen, da *SQuirreL* allgemein gehalten wurde, um möglichst viele Datenbanken unterstützen zu können. Dieser

Punkt fällt aber kaum ins Gewicht, da die Möglichkeit besteht Plugins zu entwickeln und einzubinden, die spezielle produktspezifische Funktionen implementieren. Durch die aktiven Benutzer und Entwickler dieses Werkzeugs können zu den bereits vorhandenen Plugins jederzeit neue hinzukommen.
Ein weiterer Nachteil sind die benötigten JDBC-Treiber, die man selbst beschaffen und einbinden muss. Aber auch für dieses Problem wurde in Kapitel 3.2.3 eine Lösung vorgestellt. Entweder man installiert das *Data Studio*, da dieses den benötigten Treiber gleich mitinstalliert, oder es wird der auf DVD bereitgestellte Treiber eingesetzt.
Da sich der Zugriff auf die Datenbank in überschaubaren Grenzen hält, wurde im Rahmen dieser Arbeit hauptsächlich *SQuirreL* verwendet. Das *Data Studio* kam nur zu Beginn zum Einsatz um einige Abfragen durchzuführen.

3.3. Elimination des *generischen Tabellenformats*

Nachdem in den ersten Schritten die Datenbank auf *DB2 for z/OS* 9.1 umgestellt wurde, werden nun das *generische Tabellenformat* und die damit verbundenen Nachteile eliminiert. Aus logischen Gesichtspunkten wird diese Aufgabe im *RMFRecorder* begonnen. Bevor aber mit der Refaktorisierung begonnen werden kann, muss zuerst eine passende Möglichkeit gefunden werden XML mit Java zu verarbeiten. Dies umfasst das Erstellen, Editieren und Einlesen von XML-Dokumenten.

3.3.1. Möglichkeiten der XML-Verarbeitung in Java

Es gibt mittlerweile verschiedene alternative Methoden um XML mit Java zu verarbeiten, je nachdem welches Anwendungsgebiet angesprochen werden soll. Im vorliegenden Fall stellt sich die Frage, wie die Metrik-Daten, die über den CIM-Client abgegriffen werden, für die Weiterverarbeitung in das XML-Format überführt werden können. Dafür gibt es laut [Ull08b] drei verschiedene Verarbeitungstypen:

- DOM-orientierte Schnittstellen (repräsentieren den XML-Baum im Speicher): W3C-DOM, *JDOM*, *dom4j*, *XOM*, ...

- Push-Schnittstellen (nach dem *Callback*-Prinzip ruft der Parser Methoden auf und meldet Element-Vorkommen): *Simple API for XML* (SAX) ist ein populärer Repräsentant

- Pull-Schnittstellen (es wird wie ein *Tokenizer* über die Knoten gegangen): Dazu gehören *XML Pull Parser* (XPP), wie sie die *Streaming API for XML* (StAX) definiert

Da die letzten beiden Schnittstellen ereignisorientiert arbeiten und dadurch das Dokument nur in aktuell verwendeten Stücken geladen wird, eignet sich die Verarbeitung mit Pull- und Push-Schnittstelle eher für Darstellungen von XML-Dateien im Browser oder für die Suche nach bestimmten Inhalten und ähnlichen Aufgaben. Deswegen sind sie für die Lösung des vorliegenden Problems nicht zu empfehlen.
Im Gegensatz dazu verfolgt die vom *World Wide Web Consortium* (W3C) standardisierte Schnittstelle des *Document Object Models* (DOM) einen anderen Ansatz: Sie lädt die gesamte XML-Struktur als Baum in den Speicher. Nur so ist die interaktive Verarbeitung eines XML-Dokumentes möglich. Da bei der Datenaufzeichnung im Rahmen dieser Arbeit die interaktive XML-Verarbeitung im Speicher essentiell ist, kommt als Verarbeitungsmöglichkeit nur DOM in Frage.
Ziel der DOM-Spezifikation ist eine Schnittstelle zur Verfügung zu stellen, die es Programmen ermöglicht dynamisch auf ein strukturiertes Dokument zuzugreifen und dessen Inhalt samt Struktur und Stil zu verändern, um es aktualisiert wieder darzustellen. Der Zugriff soll dabei nicht nur möglichst einfach und einheitlich, sondern auch programmiersprachen- und plattformunabhängig erfolgen [Wor08a]. So würde beispielsweise die in Abbildung 3.6 dargestellte XHTML-Tabelle als DOM-Baum der Abbildung 3.7 entsprechen.

Bevor ein bestehendes Dokument verarbeiten werden kann, muss es zuerst durch die Anwendung eingelesen und im Arbeitsspeicher als navigierbarer DOM-Baum angelegt werden. Dann ist es möglich durch das Dokument zu navigieren, Knoten zu erzeugen, löschen und zu verschieben. Es ist auch möglich Attribute und Textinhalte von Knoten auszulesen, zu ändern oder zu löschen. Am Ende der Verarbeitung kann dann aus dem Dokument-Objekt durch Serialisierung ein neues Dokument erstellt werden.

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

Abbildung 3.6.: XHTML-Tabelle (Abbildung aus [Wor08b])

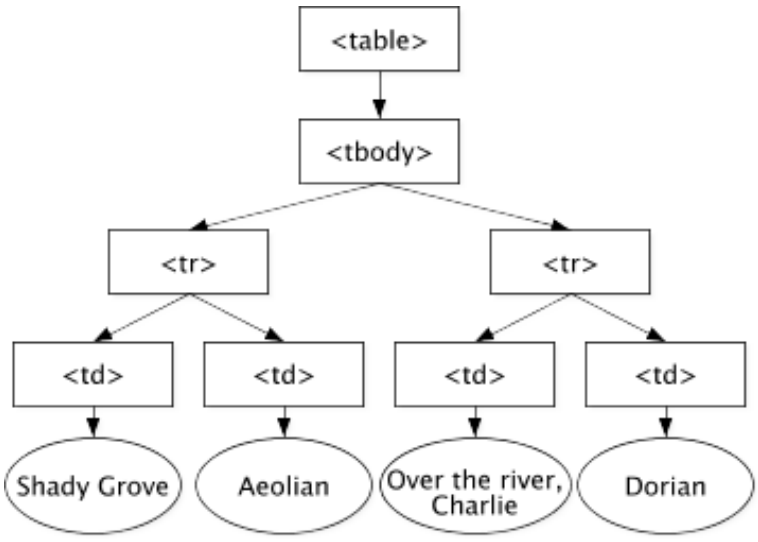


Abbildung 3.7.: Entsprechender DOM-Baum für XHTML-Tabelle aus Abbildung 3.6 (Abbildung aus [Wor08b])

3.3.2. XML-Verarbeitung mit *JDOM*

Auch bei den Implementierungen der DOM-Schnittstelle für Java sind mehrere Alternativen verfügbar. Betrachtet werden hier *JDOM*, *dom4j* und *XOM*.
Alle drei Projekte sind als Open Source verfügbar, wobei die ersten beiden am weitesten verbreitet sind und *JDOM* die älteste Implementierung darstellt. Das *XOM*-Projekt wurde ursprünglich als Weiterentwicklung von *JDOM* gestartet, bildet mittlerweile aber ein eigenständiges Projekt. Dadurch bestehen mehr Gemeinsamkeiten als Unterschiede zwischen den beiden Implementierungen [Har08]. Diese Tatsache trifft auch auf *dom4j* und *JDOM* zu. Durch die Orientierung an der DOM-Spezifikation unterscheiden sich beide Projekte nur marginal und unwesentlich voneinander [Met08].
Wegen der weiten Verbreitung und der Verwendung samt Tutorial in dem renommierten Werk [Ull08a] fällt die Wahl hier auf die *JDOM*-Schnittstelle. Für die vorliegende Aufgabe werden im Wesentlichen nur die Standard-Funktionen der XML-Verarbeitung benötigt. Da diese Funktionen von allen drei Schnittstellen zur Verfügung gestellt werden, kann die vorliegende Aufgabe genauso gut mit den anderen Schnittstellen gelöst werden.
Durch das Ziel DOM möglichst allgemein zu halten – es muss mit HTML- und XML-Dokumenten arbeiten und mit möglichst allen Browsern kompatibel sein – ist es unhandlich in der Benutzung, ineffizient und vor allem speicherlastig.
JDOM dagegen versucht diese Mängel einzuschränken oder ganz zu beseitigen. Es ist eine Implementierung der DOM-Schnittstelle, die speziell auf Java zugeschnitten ist und somit einige Optimierungen und Verbesserungen beinhaltet. *JDOM* ist nur für XML ausgelegt, setzt *Java Collections* ein und arbeitet praktischerweise mit anderen Standards wie z. B. der *Simple API for XML*(SAX) zusammen.
Als Resultat ist es einfach und sauber implementiert, was wiederum zu einer intuitiveren Schnittstelle führt und eine schnelle fehlerfreie Integration in die eigene Anwendung begünstigt. *JDOM* ist freie Software und daher ein kostengünstiger Einstieg in die XML-Verarbeitung mit Java [JDO08].

Das Erstellen und Bearbeiten eines Dokuments ist denkbar einfach. So sind beispielsweise für das Hinzufügen oder Löschen eines Knotens die Funktionen `addContent()` und `removeContent()` zuständig. Wenn bevorzugt mit den Standard Java-Klassen gearbeitet wird, dann können durch die Funktion `getChildren()` sämtliche Unterknoten als `java.util.List` übergeben wer-

den, um diese anschließend mit den bekannten Methoden der `List`-Schnittstelle zu bearbeiten. Alle an der Liste durchgeführten Änderungen werden direkt auf den DOM-Baum im Speicher übertragen. Somit bleibt es dem Entwickler überlassen, wie er das Dokument bearbeiten will.
Neben den üblichen `get`-Methoden, um durch den Baum zu navigieren, unterstützt *JDOM* auch die ebenfalls vom W3C standardisierte *XML Path Language* (XPath). Dabei handelt es sich um eine Abfragesprache mit der man Teile eines XML-Dokuments adressieren kann [Wor08c]. Die von *JDOM* verwendete XPath-Schnittstelle ist *Jaxen*, eine pure Java-Schnittstelle, die auch auf DOM oder *dom4j* zugeschnitten werden kann [Cod08]. XPath dient auch als Grundlage für eine Reihe weitere Standards, wie z. B. *Extensible Stylesheet Language Transformation* (XSLT), *XML Pointer Language* (XPointer) oder der in Kapitel 3.2.1 im Rahmen der *pureXML*-Technologie kurz erwähnten Abfragesprache XQuery.
XPath betrachtet die XML-Datenstruktur als Baum und mit „/“ als XPath-Wurzel. Die zur Navigation verwendete vereinfachte Dateisystem-Notation ist an die Regeln des bekannten UNIX-Dateisystems angelehnt. Analog dazu kann man jeden XML-Knoten von der Wurzel aus adressieren, als ob ein Verzeichnis im UNIX-Dateisystem adressiert wird. Oftmals ist es deutlich einfacher den Pfad zu einem XML-Knoten mit einem XPath-Ausdruck anzugeben als mit einzelnen Methodenaufrufen.
Im Rahmen dieser Arbeit wurden bevorzugt XPath-Ausdrücke eingesetzt.

3.3.3. Refaktorisierung des *RMFRecorders*

Nachdem die theoretischen Probleme der Refaktorisierung bezüglich XML geklärt wurden, kann nun dazu übergegangen werden sämtliche vom *RMFRecorder* aufgezeichneten Metriken in einer hierarchischen XML-Struktur aufzuzeichnen. Durch diese strukturellen Veränderungen wird aus der ursprünglichen Tabelle `metrics` aus Kapitel 2.2 die Tabelle `metrics_xml` mit den Spalten *Timestamp* und *MetricsXML*. Der SQL-Befehl für diese neue auf XML umgestellte Tabelle sieht wie folgt aus:

```
CREATE TABLE metrics_xml
(
  Timestamp  TIMESTAMP NOT NULL,
  MetricsXML XML       NOT NULL,
  PRIMARY KEY (Timestamp)
);
```

Durch die neue XML-Spalte ist auch eine Anpassung des Index notwendig. DB2 *pureXML* führt dazu eine neue Index-Technologie ein, um die Abfrage auf XML-Dokumenten zu beschleunigen. Dazu muss zum gewöhnlichen SQL-Befehl `CREATE INDEX` zusätzlich noch ein so genanntes *xmlpattern* angegeben werden. Dieses *xmlpattern* besteht aus einem XPath-Ausdruck, welcher eine Teilmenge eines XML-Dokuments spezifiziert [IBM07]. Der Index im vorliegenden Fall soll auf die XML-Knoten *Instance* und *Metric* angewendet werden, genauer auf das Attribut *ID* dieser Knoten. Im Gegensatz zur alten Tabelle muss der Index auf die Instanz-ID einer Metrik explizit angelegt werden. Dort war die Instanz-ID Teil des Schlüssels, wodurch automatisch ein Index darauf angelegt wurde. Da zusätzlich auch noch die Metrik-IDs von Interesse sein könnten, wird mit den folgenden SQL-Befehlen auf beide XML-Teilmengen ein Index angelegt:

```
CREATE INDEX instID_idx on metrics_xml(metricsXML)
  GENERATE KEY USING XMLPATTERN
  '/MonitoredMetrics/Metric/Instance/@ID'
  AS SQL VARCHAR(50);

CREATE INDEX metrID_idx on metrics_xml(metricsXML)
  GENERATE KEY USING XMLPATTERN
  '/MonitoredMetrics/Metric/@ID'
  AS SQL VARCHAR(50);
```

Durch die neue Tabellen-Struktur mit nur zwei Spalten existiert für jeden Zeitstempel genau ein Tupel in der Tabelle und nicht wie zuvor für jede aufgezeichnete Instanz-ID noch ein weiteres zusätzlich Tupel pro Zeitstempel. Diese Tatsache ist der Schlüssel zur Elimination des *generischen Tabellenformats*. Jedes Tupel wird eindeutig durch einen Zeitstempel definiert und somit ist die Reihenfolge in der Tabelle garantiert und eindeutig. Dies konnte im alten Prototypen des *WLP-Frameworks*

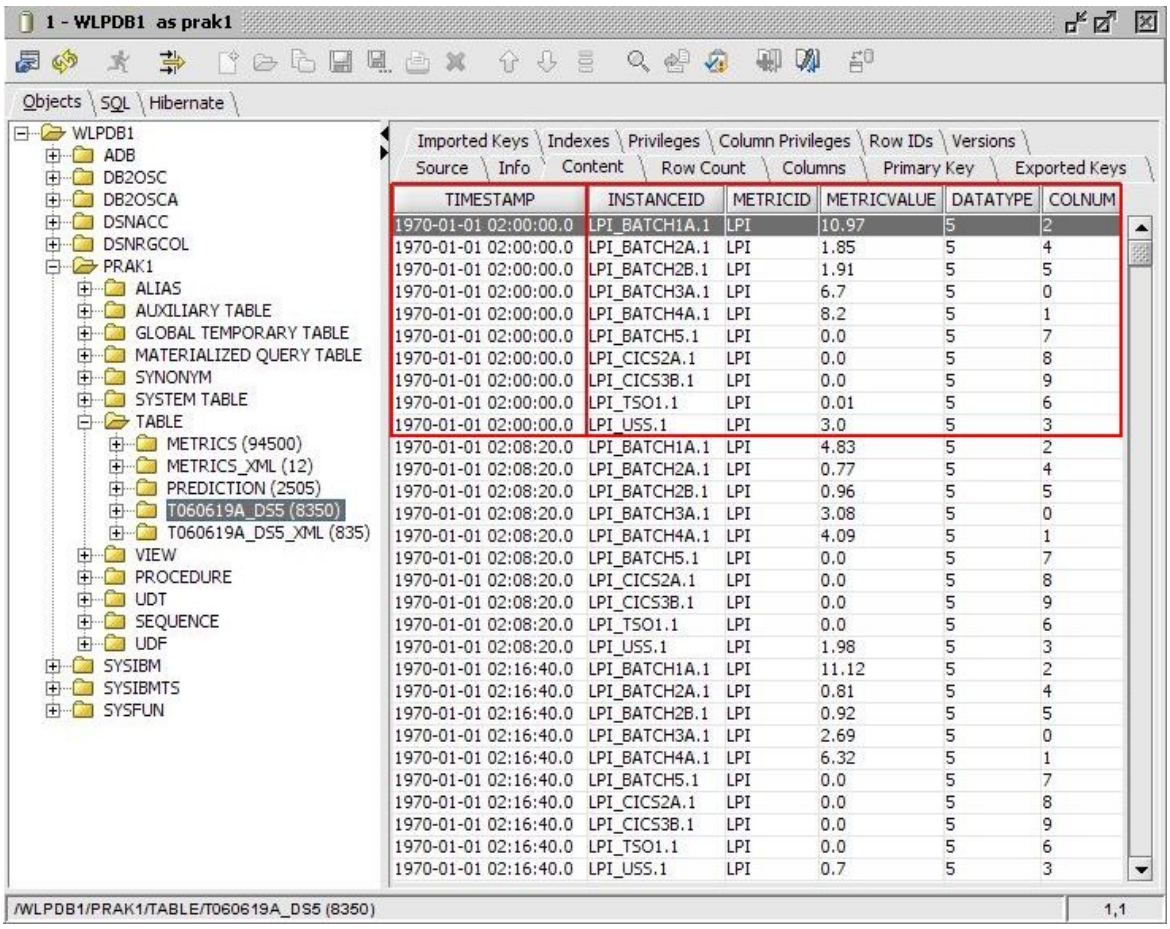


Abbildung 3.8.: Das alte *generische Tabellenformat* (Screenshot aus *SQuirreL*)

nicht garantiert werden, ist aber für die Verwendung von künstlichen neuronalen Netzwerken zwingend erforderlich, wie in Kapitel 2.2 ausführlich beschrieben wurde. Durch die Elimination des alten Tabellenformats ist es möglich das komplette *WLP-Framework* entscheidend zu vereinfachen, da das bis dato verwendete komplizierte Hilfskonstrukt damit überflüssig geworden ist.
Abbildungen 3.8 und 3.9 zeigen Screenshots aus *SQuirreL*, jeweils mit altem und neuem Tabellenformat.

Nach der Anpassung der Tabellen müssen als nächstes die durch den CIM-Client bereitgestellten Daten mit Hilfe von *JDOM* zu einem XML-Dokument zusammengebaut werden. Dazu wird in der Klasse `Database.java` die Funktion `insertRecords()` editiert. Dabei muss beachtet werden, dass jede Metrik-ID auch mehrere Instanz-IDs haben kann. Deswegen wird ein Algorithmus implementiert, der für jede Metrik-ID erst einmal prüft, ob diese im *JDOM*-Baum bereits exis-

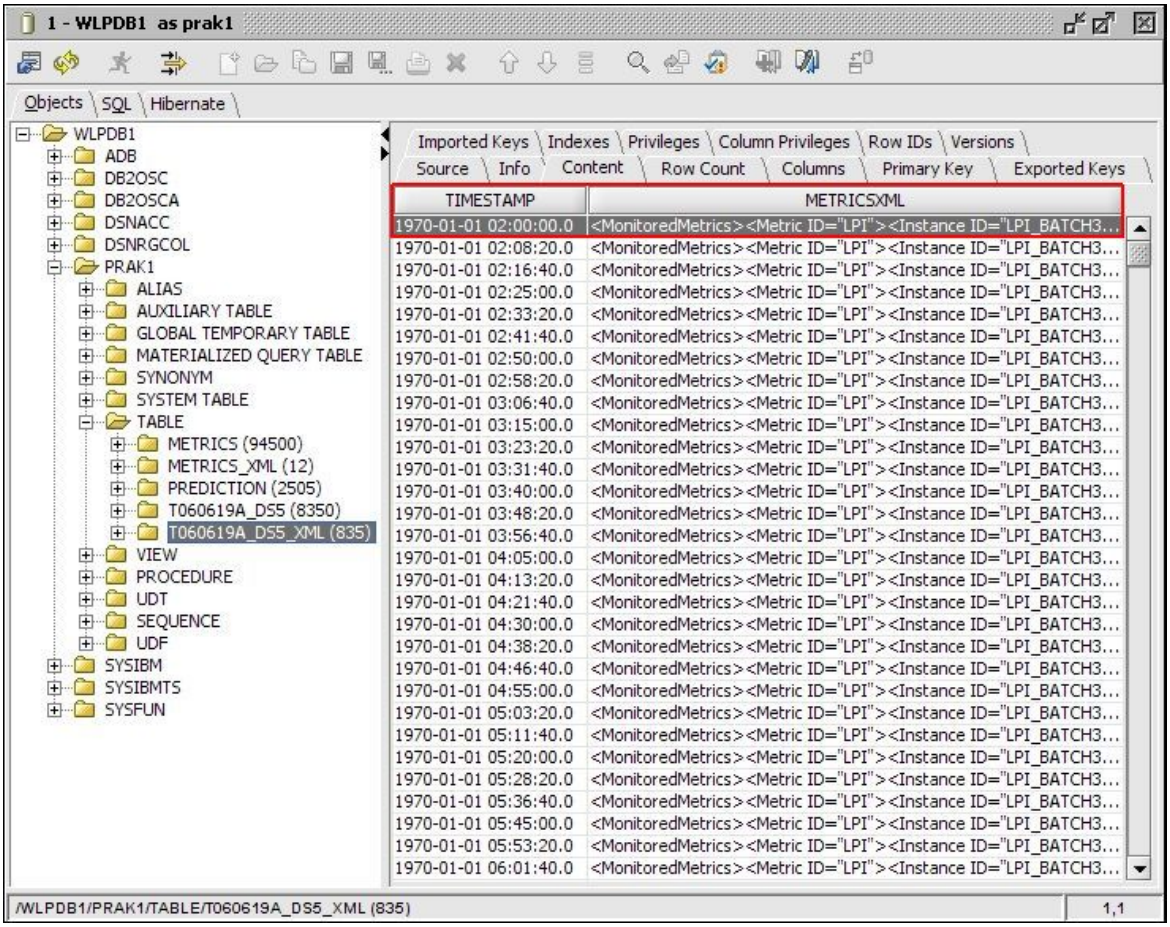


Abbildung 3.9.: Das neue XML-Tabellenformat (Screenshot aus *SQuirreL*)

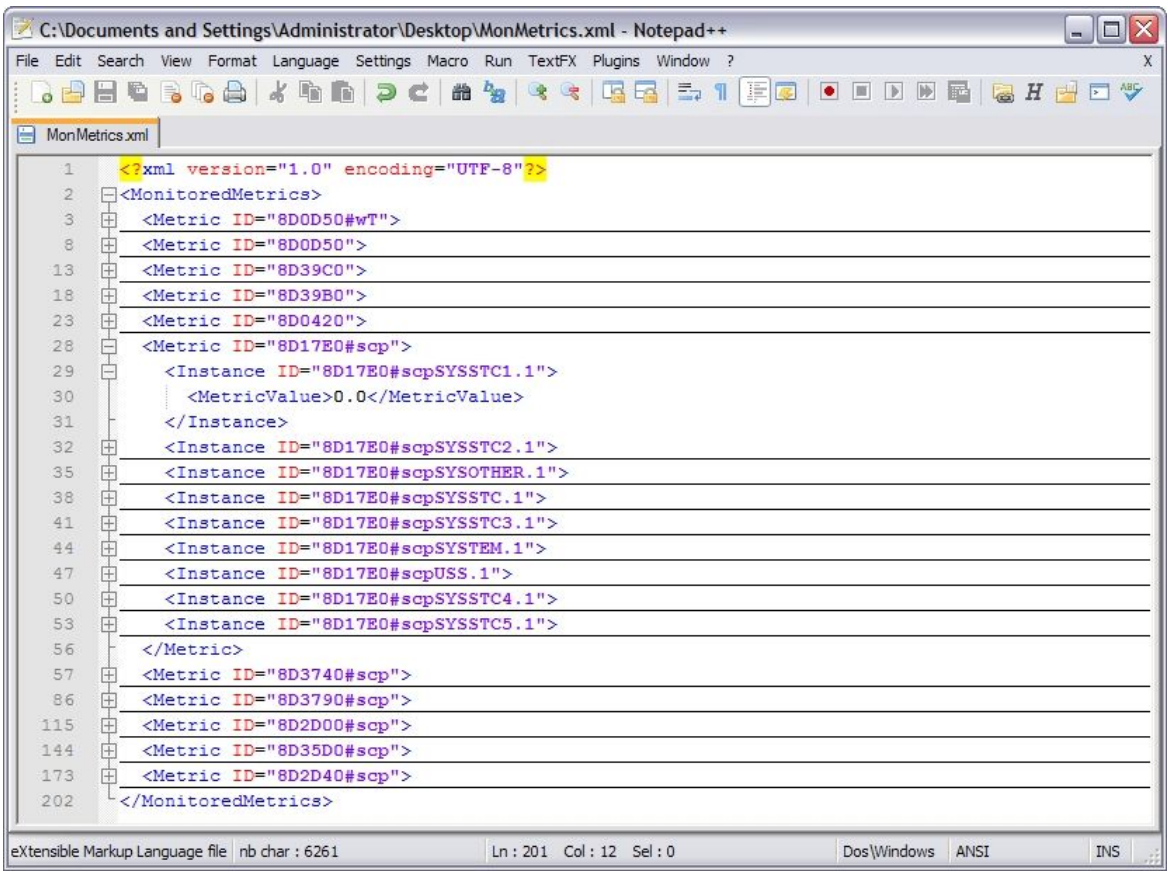


Abbildung 3.10.: XML-Dokument mit aufgezeichneten Metriken (erstellt mit Hilfe von *JDOM*)

tiert. Ist dies nicht der Fall, wird mit der Funktion `buildMetricNode()` ein neuer XML-Knoten mit dieser Metrik-ID und den entsprechenden Unterknoten angelegt. Falls diese jedoch schon existiert, werden nur die Unterknoten an den schon existierenden Knoten angehängt. Ein Unterknoten umfasst in diesem Fall immer eine Instanz-ID und den eigentlich gesuchten Wert *MetricValue*. Ein so erstelltes XML-Dokument wird in Abbildung 3.10 dargestellt.
Somit werden alle Metriken in einem XML-Dokument pro Zeitstempel gespeichert und der erste Schritt zur Elimination des *generischen Tabellenformats* ist gemacht.

3.3.4. Refaktorisierung der *WLPrediction*

Um die Qualität der Vorhersage-Ergebnisse vergleichen zu können, muss auf die in [Geb06] generierten Metrik-Daten zurückgegriffen werden. Bevor al-

so die Vorhersage-Anwendung refaktorisiert werden kann, sollten als erstes diese Referenzdaten in das neue XML-Speicherformat überführt werden. Zu diesem Zweck ist das Konvertierungsprogramm *Converter* entstanden. Es ist als Teil des *WLPrediction*-Projekts angelegt und besteht nur aus der Klasse `com.ibm.wlp.converter`. Die einzige Aufgabe dieses Programms ist die einmalige Konvertierung von relational gespeicherten Metrik-Daten in das notwendige XML-Speicherformat. Dazu sind folgende Angaben nötig: Zugangsdaten zur Datenbank, die zu konvertierende Tabelle sowie die zu befüllende XML-Tabelle. Im Fall der verwendeten Referenzdaten sind dies die Tabellen `T060619A_DS5` und `T060619A_DS5_XML`. Die 8350 Referenzdatensätze reduzieren sich um den Faktor 10 auf 835, da in der ursprünglichen Tabelle für jeden Zeitstempel 10 Tupel existieren während nach der Konvertierung jeweils nur ein einziger übrig bleibt.
Dieses Konvertierungsprogramm erhebt keinerlei Ansprüche auf korrekte Funktionsweise, es ist als provisorische Lösung anzusehen. Allerdings konnte die Konvertierung der vorliegenden Daten damit erfolgreich durchgeführt werden.

Bei der Refaktorisierung der *WLPrediction* wird systematisch an das Problem herangegangen und bei der ersten Funktion angesetzt, welche das Tabellenformat involviert. Um außerdem das Fehlerrisiko zu verringern, werden die alten Funktionen und die alte Datenstruktur parallel zu der neuen beibehalten und erst nach erfolgreicher Umstellung auskommentiert. Nach jedem Schritt der Umstellung wird die Anwendung erst einmal ausführlich getestet und die Vorhersage-Ergebnisse mit den Ergebnissen der Referenzdaten verglichen, um eventuelle Fehler zu lokalisieren und sofort zu beseitigen bevor sie sich ansammeln. Erst wenn garantiert ist, dass die Anwendung fehlerfrei durchläuft und die gleichen Ergebnisse wie bei den Referenzdaten ausgegeben werden, wird der alte Quellcode komplett auskommentiert, alte Verweise angepasst oder entfernt und unnötige *Import*-Anweisungen oder ähnliche Spuren beseitigt. Nachdem dies erledigt wurde, kann der nächste Vereinfachungsschritt begonnen werden.
Die oben erwähnten ausführlichen Tests werden nach dem folgenden Schema durchgeführt:
Als erstes wird der *ABLE-Editor* im Debug-Modus von *Eclipse* mit dem Ziel gestartet einen neuen *WlpAgent* anzulegen. Der eigentliche Test beginnt dann mit einem Klick auf den Button *Generate Beans* und dem damit verbundenen Initialisieren des *WlpAgents* (vgl. Anleitung in Anhang A).
Dabei muss unbedingt darauf geachtet werden, dass nach jeder Änderung im

Quellcode das Projekt *WLPrediction* neu erstellt wird. Der Grund liegt darin, dass der *ABLE-Editor* auf Ressourcen des *WLPrediction*-Projekts zugreift. Dies wiederum sind diejenigen Ressourcen, die refaktorisiert werden. Mit dem Ausführen des *Ant*-Skripts `build.xml` im *WLPrediction*-Projekt werden die im Code vorgenommenen Änderungen kompiliert und in die *jar*-Dateien `wlpcore.jar` und `wlprediction.jar` gepackt. Wenn im Anschluss der *ABLE-Editor* gestartet wird, dann greift er auf die aktualisierten Funktionen in diesen zwei *jar*-Dateien zu und die vorgenommenen Änderungen können validiert werden.

Im Prototypen von Hagmann [Hag07] erfolgt die Konvertierung vom *generischen* in das für die künstlichen neuronalen Netze benötigte relationale Format in der Klasse `TableWrapper`. Diese greift wiederum auf den speziellen Datentyp einer assoziativen Tabelle `AssocTable` zu, welcher eigens für den Prototypen erstellt wurde.
Um die zwei Elemente `TableWrapper` und `AssocTable` wurde das umfangreiche Hilfskonstrukt gebaut, welches am Ende entfernt wird.
Die Refaktorisierung wird mit dem `TableWrapper` begonnen. Dazu wird das Laden der zu verarbeiteten Daten im Debug-Modus schrittweise verfolgt, um genau festzustellen wann welche Funktion auf die `TableWrapper`-Klasse zugreift. `TableWrapper` ist als Superklasse von `SeqDBTable` angelegt und Letztere wiederum ist eine an das *generische Tabellenformat* angepasste Version der `AbleDBTable`-Klasse aus dem originalen *ABLE-Framework*.
Das Präfix `Seq` in den Klassen-Namen soll darauf hinweisen, dass die Klasse an das *generische Tabellenformat* mit seiner sequentiellen Tabellen-Struktur angepasst wurde. Als Ziel für die Refaktorisierung kann folglich festgehalten werden, dass sämtliche Klassen die ein `Seq` im Namen tragen an das neue Format angepasst werden müssen. Zum Beseitigen der Klasse `TableWrapper` wird diese zunächst mit allen Funktionen in die Klasse `SeqDBTable` verschoben, um sie anschließend an das neue XML-Tabellenformat anzupassen und wieder in `AbleDBTable` umzubenennen.
Als nächstes wird die Datenstruktur der assoziativen Tabelle ersetzt. Dafür muss zunächst geklärt werden, wie die aus der Datenbank ausgelesene Tabelle der Metrik-Daten java-intern verarbeitet werden soll. In Java gibt es aktuell keinen Datentypen und keine Datenstruktur, die explizit für das Abbilden einer Tabelle konzipiert wurde. In Microsofts *.NET* beispielsweise gibt es für solche Fälle die Datentypen `DataTable`, `DataColumn` und `DataRow`, um entweder eine komplette Tabelle, ihre Spalten oder ihre Zeilen in eine Anwendung zu laden.

Eine gängige Lösung um Tabellen in Java zu implementieren ist ein zweidimensionaler Array. Da dies die einfachste Methode ist und sie für die vorliegenden Aufgaben ausreichend ist, wird sie hier als Lösung gewählt. Dadurch kann verhältnismäßig einfach eine Tabellen-Struktur mit Spalten und Zeilen abgebildet werden, und so über Ansteuerung der einzelnen Elemente mit Hilfe zweidimensionaler Koordinaten eine transparente Verarbeitung der Daten erreicht werden. Während der Umsetzung hat sich der Datentyp in manchen Fällen aber als zu „starr“ erwiesen, weil beispielsweise nach der Initialisierung eines Arrays seine Größe nicht mehr verändert werden kann. So ist in der Folge eine bei Tabellen oft gebrauchte Aktion wie das Entfernen einer Spalte oder Zeile nicht ohne weiteres möglich. Da diese Arbeit eine reine Machbarkeitsstudie darstellt und eine optimale Implementierung nur zweitrangig ist, kann dieser kleine Makel vernachlässigt werden.
Bei der Ablösung der `AssocTable`-Datenstruktur wird systematisch nach jedem Vorkommen dieser Datenstruktur gesucht, parallel dazu an der selben Stelle die neue aufgebaut und die Werte von alter und neuer Datenstruktur auf Gleichheit überprüft. Beim Auftreten eines Fehlers kann somit genau verfolgt werden ab welchem Punkt die Werte voneinander abweichen.
Von diesem Austausch ist nicht nur die Klasse `SeqDBTable` betroffen, sondern auch `MyDataSetDefinition`, eine angepasste Version der Klasse `AbleDataSetDefinition` aus dem originalen *ABLE-Framework*, welche die von ABLE benötigten Metadaten der SQL-Abfrage liefert.
Größere Probleme treten erstmals bei der Funktion `validate()` in `SeqDBTable` auf. Diese Funktion regelt die Vorgehensweise für Ausreißer, falsch skalierte und fehlende Werte einer für die Vorhersage zu verwendenden Instanz-ID. Vor allem im Fall einer kompletten Streichung der Instanz-ID, stößt die Array-Datenstruktur auf oben schon angesprochene Grenzen: Die betroffene Instanz-ID, die intern als Spalte des zweidimensionalen Arrays implementiert ist, kann nicht komplett entfernt werden, da der Java-Array in seiner Größe nicht verändert werden kann. Dieses Problem ist aber mit relativ wenig Aufwand über die Benutzung eines zweiten temporären Arrays lösbar.
Nachdem die neue Struktur vollständig parallel zur alten implementiert ist und auch die Vorhersagen mit beiden Strukturen in unterschiedlichen Datenbank-Tabellen gleiche Werte zurückschreiben, kann die alte Datenstruktur schließlich auskommentiert und alles auf die neue Struktur zugeschnitten werden.
Bei der Überprüfung der graphischen Darstellung der Ergebnisse mit Hilfe von

Jetty, taucht dann ein weiteres Problem auf. Die Funktion `basicChart()` in der Klasse `ImagesServlet`, die für das Darstellen des Diagramms zuständig ist, funktioniert nicht fehlerfrei. Die Vorhersage beginnt nicht wie erwartet um die Anzahl von Vorhersageschritten verzögert, sondern zum Zeitpunkt Null.
In der alten Datenstruktur wurden die Vorhersageschritte im *WlpAgent* festgelegt und über das komplizierte Hilfskonstrukt der Funktion `basicChart()` bereitgestellt. Obwohl diese Information weiterhin im *WlpAgent* festgelegt wird, steht sie durch die Beseitigung des Hilfskonstrukts der Funktion `basicChart()` nicht mehr zur Verfügung. Deswegen wird mit relativ wenig Aufwand ein Algorithmus implementiert, der den gewünschten Wert ermittelt und der Funktion bereitstellt.

Nachdem nun auch die Vorhersage-Ergebnisse korrekt sind, muss als letzter Schritt der Quellcode „aufgeräumt“ werden. Sämtliche Klassen mit dem Präfix `Seq` werden in ihre Original-Namen aus dem *ABLE-Framework* umbenannt. So wird beispielsweise aus `SeqDBTable` wieder `AbleDBTable` und aus `SeqDBImportImpl` wird `AbleDBImportImpl`. Im Anschluss müssen auch sämtliche Verweise darauf angepasst oder falls nicht mehr benötigt entfernt werden.
Im Rahmen dieser umfangreichen Refaktorisierung werden zahlreiche Funktionen in verschiedenen Klassen angepasst, in der `SeqDBTable`-Klasse z. B. die Funktionen `getRowCount()`, `getColumnCount()`, `getValueAt()`, `getColumnName()`, `getRowKey()`, `isLoaded()`, `loadDataFile()`, `validate()`. Um die erfolgten Änderungen auch zu einem späteren Zeitpunkt noch nachvollziehen zu können, werden die ursprünglichen von Hagmann [Hag07] erstellten Klassen nicht aus dem *Eclipse-Workspace* entfernt, sondern nur vollständig auskommentiert und mit einem Unterstrich als erstes Zeichen im Namen markiert. Dabei handelt es sich um die folgenden Klassen: `RemoteSeqDBImport`, `SeqDBImport`, `SeqDBImportBeanInfo`, `SeqDBImportCustomizer`, `SeqDBImportDataBean`, `SeqDBImportImpl`, `SeqDBImportPanel`, `SeqDBImportProxy`, `SeqDBTable`, `TableWrapper`, `AssocLinkedList`, `AssocListMap` und `AssocTable`.
Abgesehen davon wird die aufwendige Refaktorisierung genutzt, um allgemein den Quellcode zu optimieren und Fehler zu bereinigen. So werden die Start-Skripte angepasst, Ausgaben auf die Konsole kosmetisch und funktional optimiert, unnötige Funktionen wie das in der *WLPrediction* nicht mehr verwendete *Downsampling* entfernt, und wo nötig der Quellcode kommentiert. Als konkretes Beispiel soll hier die Benennung der drei Ausgabewerte der

WLPrediction-Anwendung angeführt werden. Da die bisherige Benennung des geglätteten (*LPI_BATCH2A.1_target:WlpAgent*) und des vorhergesagten Wertes (*LPI_BATCH2A.1:WlpAgent*) recht verwirrend ist, wurde die Bezeichnung für den Vorhersagewert in *LPI_BATCH2A.1_prediction:WlpAgent* abgeändert.

3.4. Migration des *WLP-Frameworks* auf den Großrechner

Mit Abschluss der letzten Aufgabe dieser Arbeit soll es möglich sein, das *WLP-Framework* komplett vom Client loszulösen und auf den Großrechner zu migrieren. Im ersten Schritt wurde die Datenbank von *Apache Derby* auf *DB2 for z/OS* umgestellt und läuft seither auf dem Großrechner. Die beiden anderen Komponenten, *RMFRecorder* und *WLPrediction*, müssen für die Migration auf den Großrechner erst vorbereitet werden. Im Verzeichnis des *WLPrediction*-Projekts befindet sich ein Unterverzeichnis mit dem Namen `dist`, welches alle zur Ausführung benötigten Dateien enthält. Eine Übersicht über den Inhalt dieses Verzeichnisses findet sich in Abbildung 3.11.
Im Unterverzeichnis `lib` befinden sich alle zur Ausführung benötigten Dateien, darunter die für ABLE benötigten *jar*-Dateien, der JDBC-Treiber und alle anderen beim Start der Anwendungen referenzierte *jar*-Dateien.
In einem weiteren Unterverzeichnis mit dem Namen `AgentPool` befinden sich die erstellten und trainierten Agenten, die jederzeit erweitert werden können.
Die eigentlichen Anwendungen befinden sich im Verzeichnis `wlp`. Bei Änderungen und Weiterentwicklungen am Quellcode muss jedes Mal durch Ausführen des *Ant*-Skripts `build.xml` neu kompiliert werden. Dieses Skript kopiert dann automatisch die neue Version der Anwendungen in das `dist`-Verzeichnis an die entsprechende Stelle.
Im `dist`-Verzeichnis befinden sich auch die Konfigurationsdateien für das *WLP-Framework*: `WLPDefaults.properties` für die allgemeine Konfiguration und `cimmetrics.xml` mit zugehöriger Schema-Datei `cimmetrics.xsd` für die Merkmalsselektion. Die für die Ausführung der beiden Anwendungen benötigten *Shell*-Skripte befinden sich im selben Verzeichnis. Für den *RMFRecorder* ist das die Datei `startRMFRecorder` und für die *WLPrediction* die Datei `startWLP`.

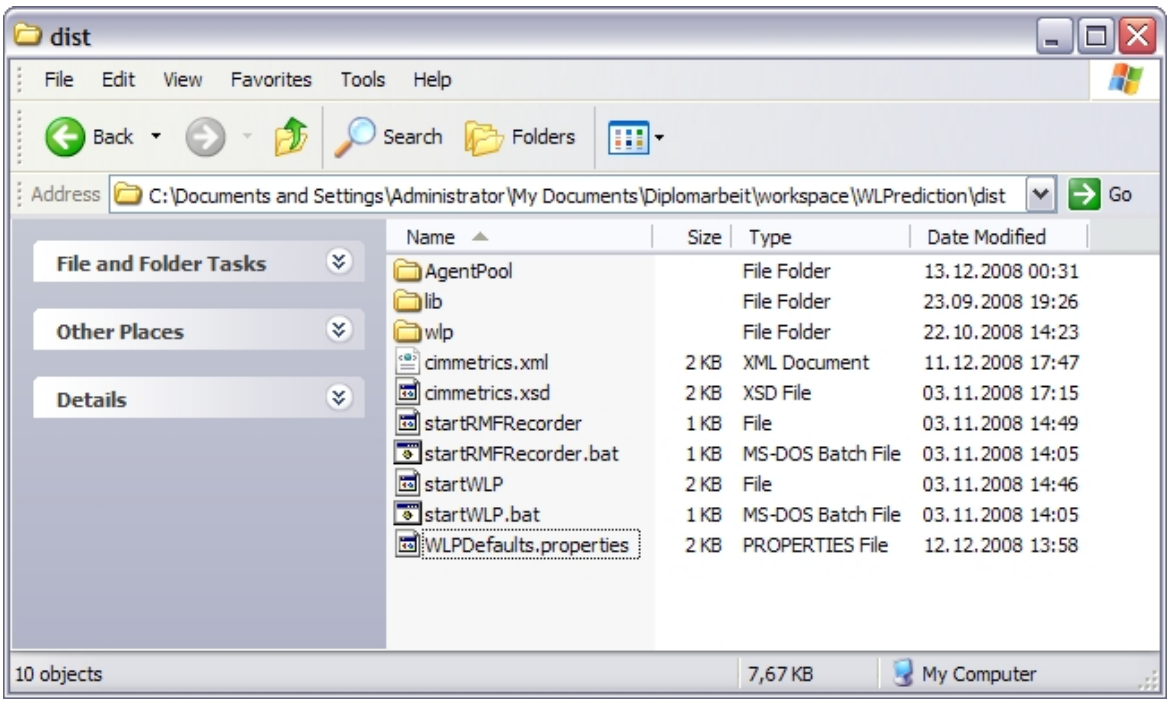


Abbildung 3.11.: Übersicht über den Inhalt des `dist`-Verzeichnisses

Das `dist`-Verzeichnis muss komplett mit Inhalt auf den Großrechner übertragen werden. Gegebenenfalls muss es nach vorgenommenen Änderungen am Quellcode erneut übertragen werden. Dies ist an sich eine triviale Aufgabe, die aber bei mangelnder Erfahrung und wegen der speziellen Architektur der Großrechner dennoch Probleme bereiten kann.
Erstens verwenden Großrechner die spezielle Zeichenkodierung *Extended Binary Coded Decimals Interchange Code* (EBCDIC). Deswegen muss bei der Übertragung der Dateien auf den Großrechner darauf geachtet werden, welches Format die jeweilige Datei besitzt. Gewöhnliche Text-Dateien, dazu gehören auch HTML- und Skript-Dateien, müssen im ASCII-Modus übertragen werden. Alle anderen Dateien, die proprietäre Dateiformatierungen besitzen, müssen im Binär-Modus übertragen werden. Zu Letzteren gehören beispielsweise *doc*-, *exe*-, *zip*- und *jar*-Dateien.
Die Umwandlung von ASCII zu EBCDIC geschieht während der Übertragung automatisch im Hintergrund. Es wird dabei nicht wie sonst bei einer ASCII-Übertragung üblich nur eine Konvertierung zwischen den Zeilenumbruchsvarianten der verschiedenen Betriebssystemen vorgenommen, sondern eine „echte“ Konvertierung der Zeichenkodierung. Diese Übertragung findet üblicherweise per

File Transfer Protocol (FTP) statt, wofür ein FTP-Client benötigt wird. Im Internet gibt es zahlreiche dieser Clients oftmals als kostenlose Software, z. B. *Filezilla*, *WinSCP*, oder auch das *Firefox*-Plugin *FireFTP*.
Interessanterweise müssen die XML-Dateien `cimmetrics.xml` und `cimmetrics.xsd` im binären Modus übertragen werden, auch wenn sie nach der Übertragung auf dem Großrechner nicht korrekt darstellbar sind, da ansonsten der *RMFRecorder* nicht funktioniert. Dies kann damit zusammenhängen, dass das Projekt auf dem Client für ASCII kompiliert wird und der *RMFRecorder* deswegen auch eine in ASCII kodierte Datei zum Einlesen erwartet. Dies ist ein unerwartetes Verhalten und sollte in anschließenden Arbeiten nach Möglichkeit genauer untersucht werden.
Zweitens muss geklärt werden wohin die Dateien übertragen werden müssen. Auf Großrechnern gibt es ein extra Subsystem, die *UNIX System Services* (USS), die es ermöglichen UNIX-Anwendungen auf dem Großrechner auszuführen. Dies gilt auch für Java-Anwendungen wie im vorliegenden Fall. Wie bei UNIX üblich müssen die Dateien in das jeweilige Benutzerverzeichnis kopiert werden, da man nur für diese Destination ausreichend Rechte besitzt. In der Regel ist dies das Verzeichnis `/u/"Benutzername"`. Ausführlichere Details zur Datei-Übertragung auf den Großrechner finden sich in der angehängten Anleitung in Anhang A.
Drittens muss darauf geachtet werden, welche Java-Version benutzt wird. Auf dem hier verwendeten System *BOEWLM4* ist die *Java Runtime Environment* (JRE) in Version 1.5 installiert. Eine Kompilierung des Projekts mit Java 1.6 ist auf dem System somit nicht lauffähig und muss zwingend mit einer älteren Java-Version kompiliert werden.
Schließlich müssen nach erfolgreicher Übertragung die entsprechenden Dateirechte mit dem UNIX-Befehl `chmod` gesetzt werden, um die Anwendung ausführen zu können. Dazu reicht es wenn die *Shell*-Skripte, die zum Starten der Anwendungen erstellt wurden mit den benötigten Rechten ausgestattet werden. Für den *RMFRecorder* ist dies die Datei `startRMFRecorder` und für die *WLPrediction* ist es die Datei `startWLP` im Verzeichnis `dist`.

Abschließend soll hier bezüglich der möglichen Fehlerquellen nochmals auf die Bedienungsanleitung in Anhang A verwiesen werden. Zu sämtlichen oben aufgeführten Punkten finden sich dort detailliertere Hilfestellungen.

4. Ergebnisse

Dieses Kapitel dient der Auflistung, der im Rahmen dieser Arbeit erbrachten Ergebnisse. Dabei wurden zu Beginn vier Aufgaben festgelegt, zu denen nun die jeweiligen Ergebnisse präsentiert werden.
Zur generellen Erleichterung der Bedienbarkeit wurde damit begonnen eine Bedienungsanleitung zu erstellen, die kontinuierlich ergänzt wurde. Im nächsten Schritt erfolgte die Umstellung der Datenbank von *Apache Derby* auf *DB2 for z/OS*, bevor in einem weiteren Schritt das bisher verwendete *generische Tabellenformat* eliminiert wurde. Schließlich erfolgte noch die Migration des *WLP-Frameworks* vom Windows-Client auf den Großrechner.

4.1. Bedienungsanleitung

Als erstes Ergebnis ist eine ausführliche Bedienungsanleitung für das *WLP-Framework* entstanden, welche der Diplomarbeit angehängt wurde und in Anhang A zu finden ist. Bisher war keinerlei Dokumentation vorhanden, aber aufgrund der Komplexität des *WLP-Frameworks* erschien eine solche Bedienungsanleitung notwendig und sinnvoll. Sie stellt eine Art Sammlung aller aufgetretenen Erfahrungen und Probleme beim Betrieb des *WLP-Frameworks* dar. Dennoch ist diese Bedienungsanleitung nicht vollständig und sollte in weiteren Arbeiten parallel zur Fortführung des *Joint Research Projects* „Workload Prediction auf Mainframes“ ergänzt werden. Um diese Ergänzungen zu erleichtern, ist die Bedienungsanleitung im international genormten *Open Document Format* (ODF) erstellt worden. Ein gängiges Office-Paket, welches dieses Format unterstützt und auch bei dieser Arbeit eingesetzt wurde ist das frei verfügbare *OpenOffice*.

4.2. Datenbankumstellung auf *DB2 for z/OS*

Die Umstellung der Datenbank von *Apache Derby* auf *DB2 for z/OS* hat sich als wichtiger Schritt erwiesen, der gleich zwei Vorteile mit sich bringt. Zum einen erfolgte der Austausch, um durch die Einführung des XML-Speicherformats die Beseitigung des *generischen Tabellenformats* vorzubereiten. Zum anderen war dies ein erster Schritt zur Migration des *WLP-Frameworks* auf den Großrechner. Die Wahl fiel deswegen bewusst auf die z/OS-Version und nicht auf die in Kapitel 3.2.2 ebenfalls erwähnte LUW-Version von DB2.
Das neue XML-Speicherformat, welches relationale Tabellen mit XML-Dokumenten kombiniert, ist exklusiv an die DB2-Datenbank gebunden. Dieses Speicherformat wurde erst durch die Einführung der *pureXML*-Technologie in DB2 Version 9.1 ermöglicht. Als weiterer Vorteil wurde durch diese neue Technologie die Elimination des *generischen Tabellenformats* möglich.
Den einzigen Nachteil stellten die in Kapitel 3.2.2 angesprochenen Besonderheiten von *DB2 for z/OS* dar, die durch die spezielle Architektur von *System Z* bedingt sind. Da aber in aller Regel nur Datenbank-Administratoren mit dem Aufsetzen und der Administration einer Datenbank auf einem Großrechner beauftragt werden, relativiert sich dieses Problem für den eigentlichen Benutzer.
Vor dem Hintergrund der oben aufgeführten gewichtigen Vorteile ist dieser Nachteil irrelevant und unbedeutend. Es bleibt festzuhalten, dass die im Rahmen der Arbeit festgelegten Ziele ausschließlich mit *DB2 for z/OS* zu erreichen waren.

4.3. Elimination des *generischen Tabellenformats*

Die Beseitigung des *generischen Tabellenformats* war zentraler Punkt dieser Arbeit. Als erstes wurde mit der Refaktorisierung des *RMFRecorders* begonnen, bei der die Umstellung des Speicherformats für die Metrik-Daten mit Hilfe der *JDOM*-Schnittstelle durchgeführt wurde. Die vorher relational gespeicherten Metrik-Daten werden nun pro Zeitstempel komplett in ein eigenes XML-Dokument gespeichert. Dieses Dokument wiederum wird in die Spalte einer relationalen Tabelle abgelegt, welche über den jeweiligen Zeitstempel als eindeutigen Schlüssel abgefragt werden kann. Laufzeiteinbußen durch die Umstellung auf das XML-Speicherformat sind nur marginal feststellbar und fallen nicht ins Gewicht. Bei ent-

sprechender Optimierung der Abfragen können sie sogar ganz verschwinden.
JDOM als DOM-orientierte Schnittstelle zur XML-Verarbeitung erwies sich als äußerst unkompliziert und einfach zu integrieren. Trotz der Einfachheit, ist die Schnittstelle auch für kompliziertere Aufgaben geeignet und folglich auch für künftige Weiterentwicklungen gerüstet. Zudem zeichnet sich *JDOM* durch freie Verfügbarkeit und weite Verbreitung aus.

Da die generierten Daten aus [Geb06] auch in dieser Arbeit als Referenzdaten verwendet wurden, musste ein Weg gefunden werden die vorhandenen relational gespeicherten Metrik-Daten in das XML-Format zu überführen. Als Ergebnis entstand das Konvertierungsprogramm *Converter*, mit dem die 8350 relationalen Referenzdatensätze aus Tabelle `T060619A_DS5` in 835 Datensätze in Tabelle `T060619A_DS5_XML` konvertiert wurden.
Im Gegensatz zur Refaktorisierung des *RMFRecorder* bei dem „nur“ die Konstruktion eines XML-Dokuments umgesetzt werden musste, gestaltete sich die Refaktorisierung der *WLPrediction* schwieriger. Es waren gleich mehrere Klassen und Funktionen von den Änderungen betroffen und der Aufwand war somit relativ hoch. Trotzdem hatte die Beseitigung des *generischen Tabellenformats* einige Vorteile zur Folge.
Zunächst einmal wurde dadurch die umfangreiche und komplexe Hilfskonstruktion beseitigt (vgl. 2.2), die für die Datenrekonvertierung aus dem *generischen* in das relationale Format zuständig war. Dieses Konstrukt erschwerte eine mögliche Weiterentwicklung aufgrund der Komplexität und des resultierenden Programmieraufwands in erheblichem Maße.
Die dieser Datenrekonvertierung zu Grunde liegende Datenstruktur einer assoziativen Tabelle wurde in diesem Zusammenhang durch einen einfachen zweidimensionalen Array ersetzt. Dies brachte nicht nur Vorteile in der Laufzeit, sondern auch in der Transparenz und dem Aufbau des gesamten *WLP-Frameworks*.
Das einzige während der Umsetzung aufgetretene Problem entstand durch die mangelnde Flexibilität des verwendeten Datentyps. Ein initialisierter Java-Array ist in seiner Größe nicht mehr veränderbar. Im vorliegenden Fall betraf dies das Löschen einer Tabellenspalte, die deswegen nur über den Umweg eines zweiten Arrays möglich war. Dieser Nachteil ist aber vernachlässigbar, weil er kein alltägliches Szenario darstellt und darüber hinaus ohne spürbare Laufzeiteinbußen umgangen werden konnte. Es ist allerdings nicht vollkommen ausgeschlossen, dass bei der zukünftigen Weiterentwicklung der Array-Datentyp an seine Grenzen stößt

und ein besserer gefunden werden muss.
Nach der erfolgten Refaktorisierung wurde in der graphischen Darstellung ein weiterer Fehler gefunden: Sämtliche Vorhersagewerte waren nicht um die Anzahl der Vorhersageschritte zeitlich verzögert, da die Anzahl der Vorhersageschritte nicht an die Funktion, die für die graphische Darstellung zuständig ist, übergeben wurde. Letztendlich konnte aber auch dieses Problem behoben werden.

4.4. Migration auf den Großrechner

Die Durchführung der Migration stellte keinen großen Aufwand dar. Alle benötigten Dateien wurden automatisiert per *Ant*-Skript in das vorgefertigte `dist`-Verzeichnis kopiert. Dieses Verzeichnis musste im Anschluss auf das USS-Subsystem übertragen und minimal konfiguriert werden.
Durch die Migration des *WLP-Frameworks* auf den Großrechner ist auch die geographische Nähe zum WLM und zur CP, mit denen es idealerweise eng zusammenarbeiten soll, hergestellt worden. Ein Vorteil dieser Migration waren die resultierenden kürzeren Wege für das Transportieren der Daten und die Loslösung vom Client-Rechner. Darüber hinaus ermöglichte dieser Schritt eine zukünftige engere Verzahnung der drei oben aufgeführten Anwendungen, die zu neuen interessanten Möglichkeiten mit weiteren Vorteilen führen könnte.
Als nachteilig erwies sich jedoch die fehleranfällige Übertragung auf den Großrechner. Es gab einige mögliche Fehlerquellen für in diesem Gebiet unerfahrene Benutzer, die als Konsequenz in der Anleitung festgehalten wurden sowie ein paar offene Fragen. So ist z. B. das Problem mit der Wahl des Übertragungsmodus für die Datei `cimmetrics.xml` noch nicht abschließend gelöst worden und sollte näher untersucht werden. Zudem muss darauf hingewiesen werden, dass trotz der erfolgten Migration nicht alle zum Betrieb erforderlichen Komponenten auf dem Großrechner liefen. Der für das Erstellen und Trainieren von Agenten zuständige *ABLE-Editor* war durch die Bedienung über eine graphische Benutzeroberfläche auf dem USS-Subsystem nicht lauffähig. Als Folge musste diese Aufgabe auch weiterhin auf dem Windows-Client erledigt werden, wobei diese Tatsache kaum ins Gewicht fällt, da das Erstellen, Trainieren und Speichern eines Agenten in der Regel einmalig durchgeführt wird. Zusätzlich könnte das rechenintensive Trainieren eines Agenten auf dem Großrechner andere laufende Prozesse negativ beeinflußen.

Schließlich lieferte die Laufzeit der *WLPrediction*-Anwendung auf den ersten Blick nur unbefriedigende Ergebnisse ab. Durch die oben erwähnten kürzeren Wege hätte man in der Folge eine kürzere Laufzeit erwartet, das Resultat war jedoch trotz identischem Quellcode eine eher längere Laufzeit. Dieses untypische Verhalten liegt möglicherweise zum einen an der WLM-Konfiguration und der dadurch bereitgestellten Betriebsmittel und zum anderen an der vorliegenden Systemkonfiguration. Das verwendete Testsystem *BOEWLM4* war ein per *Vicom* virtualisiertes System, das in erster Linie zum Testen von entwickelten Anwendungen dient und somit bezüglich Laufzeiten nicht repräsentativ war. Zu diesem Verhalten sollten in folgenden Arbeiten weitere Untersuchungen durchgeführt werden.
Trotz dieser bisher ungelösten Probleme überwiegen dennoch die Vorteile. Durch die erfolgreiche Migration des *WLP-Frameworks* auf den Großrechner kann zukünftig eine weitere Optimierung durch die Integration in das z/OS-Gesamtsystem mit WLM und CP erfolgen. Dies ist Voraussetzung für eine sinnvolle Fortführung des *Joint Research Projects* „Workload Prediction auf Mainframes“.

5. Zusammenfassung und Ausblick

Die Fortsetzung des *Joint Research Projects* „Workload Prediction auf Mainframes“ hat entscheidende Fortschritte erbracht und zu neuen richtungsweisenden Erkenntnissen geführt. Das *WLP-Framework*, welches in der Arbeit von Hagmann [Hag07] erstellt wurde und auf den theoretischen Grundlagen von Gebhard [Geb06] und Kleeberg [Kle06] basiert, konnte in wesentlichem Umfang weiterentwickelt und verbessert werden.
Durch die Erstellung einer Bedienungsanleitung wurde die Bedienung des *WLP-Frameworks* vereinfacht. Eine weitere Vereinfachung auf struktureller Ebene erfolgte durch die Umstellung auf die Datenbank *DB2 for z/OS* 9.1 mit deren neu eingeführter *pureXML*-Technologie. In der Folge wurde das *generische Tabellenformat* und alle damit verbundenen Nachteile eliminiert und durch eine Kombination aus relationalem Speicherformat und XML-Speicherformat ersetzt. Im Zuge der Refaktorisierung der beiden Anwendungen *RMFRecorder* und *WLPrediction* wurde der verwendete Datentyp einer assoziativen Tabelle durch den Datentyp eines einfachen zweidimensionalen Arrays abgelöst. Auch das Hilfskonstrukt zur Datenrekonvertierung, welches durch das Einführen des *generischen Tabellenformats* notwendig geworden war, konnte entfernt werden. Nachdem die Refaktorisierung abgeschlossen war, wurde mit der Migration des *WLP-Frameworks* auf den Großrechner begonnen. Durch die Datenbankumstellung lief die Datenbank als erste der drei Komponenten schon auf dem Großrechner. Mit der Übertragung der beiden anderen Komponenten auf die *UNIX System Services* (USS) wurde die Migration vervollständigt. Die Vorgehensweise dieser eigentlich trivialen Aufgabe und die dabei aufgetretenen Probleme wurden in der Bedienungsanleitung detailliert dokumentiert.

Diese Arbeit untermauert die schon aus den vergangenen Arbeiten hervorgegangenen Argumente zur Umsetzung einer Lastvorhersage auf dem Großrechner. Sie hat aber auch neue Fragen und Problemstellungen aufgeworfen und interessante Erkenntnisse geliefert.

Das mittelfristige Ziel sollte eine Integration des Prototypen in die Umgebung des *Workload Managers*(WLM) und des *Capacity Provisionings* (CP) sein, da beide auf die berechneten Vorhersagewerte mit eventuell nötigen Betriebsmitteln reagieren können. Durch die Migration auf den Großrechner hat das Projekt eine wichtige Entwicklungsstufe auf dem Weg zur Produktreife erreicht und rückt diesem Ziel noch ein Stück näher. Gleichzeitig entstehen neue Aufgabenstellungen, wie z. B. eine engere Verknüpfung zwischen WLM, *WLP-Framework* und CP konkret aussehen könnte. So ist zwar eine Schnittstelle zwischen WLM und *WLP-Framework* implementiert, aber eine entsprechende Schnittstelle, um dem CP die vorhergesagten Daten zur Verfügung zu stellen, ist bislang nicht vorhanden. Da das CP ohnehin das *Common Information Model* (CIM) benutzt, um RMF-Daten abzufragen, könnte eine sinnvolle Lösung in Form eines CIM-Providers als Ausgabe-Schnittstelle in die *WLPrediction*-Anwendung eingebaut werden.
Weiterhin muss erörtert werden, ob und wie das Erstellen und Trainieren von Agenten auf dem Großrechner ermöglicht werden kann ohne dadurch andere Prozesse einzuschränken oder zu behindern. Dabei kann man zum einen versuchen den aktuell dafür eingesetzten *ABLE-Editor* so anzupassen und zu erweitern, dass er auf dem Großrechner lauffähig ist. Zum anderen kann der eingebetteten Webserver *Jetty* so erweitert werden, dass er diese Funktionalität über den Browser zur Verfügung stellt. Da ein Browser heutzutage auf jeglicher Art Rechner verfügbar ist, sollte generell die Möglichkeit einer erweiterten Nutzung von *Jetty* als eine Art Kontrollzentrale in Betracht gezogen werden. Eine zentralisierte Oberfläche im Browser, in der man von der Konfiguration des kompletten *WLP-Frameworks* über das Starten der einzelnen Anwendungen bis hin zur graphischen Darstellung der Ergebnisse alles erledigen kann, würde immense Vorteile in der Benutzung und Weiterentwicklung des *WLP-Frameworks* mit sich bringen.
Darüber hinaus sind noch nicht alle theoretischen Ergebnisse aus [Geb06] und [Kle06] in das *WLP-Framework* implementiert worden. So verwendet das eingesetzte *Feed-Forward* Netz im Trainingsalgorithmus nicht das *Konjugierte Gradienten-Verfahren mit Polak-Ribiere Updates*, sondern den *Backpropagation*-Algorithmus, und auch das zweite untersuchte neuronale Netz, das *FlexNet*, wurde noch nicht als ABLE-Bean implementiert.
Ein weiteres interessantes Gebiet betrifft die mögliche Validierung der Vorhersage-Ergebnisse, durch die automatisch Funktionen ausgelöst werden können. Die Möglichkeiten reichen, abhängig von der Qualität der berechneten Ergebnisse, von

Nachtrainieren des Agenten über Ändern der verwendeten Metriken bis hin zum Wechsel des benutzten neuronalen Netzes. Dabei kann auch eine automatische Merkmals- und Modellselektion nützlich sein, die je nach vorliegendem Fall das passende Netz und die passenden Metriken auswählt. Beides ist noch nicht implementiert, kann aber eine große Wertsteigerung mit sich bringen.
Abschließend muss als einer der nächsten Schritte das *WLP-Framework* beim Kunden in einer z/OS-Installation unter realen Bedingungen getestet werden, um die Alltagstauglichkeit unter Beweis zu stellen.

A. Bedienungsanleitung für das *WLP-Framework*

Bedienungsanleitung für das WLP-Framework

Diese Bedienungsanleitung soll den Betrieb des WLP-Frameworks erleichtern. Es werden keine Hinweise zur Bedienung eines Großrechners gemacht, dies wird explizit vorausgesetzt! Das selbe gilt für die Bedienung von Eclipse als IDE, die hier vorgestellten SQL-Clients und die Bedienung eines FTP-Clients.
Da das verwendete System mit Vicom virtualisiert wurde und dies nicht zur allgemeinen Wissensbasis über Großrechner zählt, wird zu Beginn auch der IPL eines Vicom-Systems ausführlich beschrieben.

Diese Bedienungsanleitung ist “**work-in-progress**” und entstand während der Nutzung des WLP-Frameworks im Rahmen der Diplomarbeit “Workload Prediction für den WLM” von Semmo Bakircioglu. Sie wurde nach bestem Wissen und Gewissen angefertigt, erhebt aber dennoch keinen Anspruch auf Vollständigkeit und Korrektheit!

Inhaltsverzeichnis

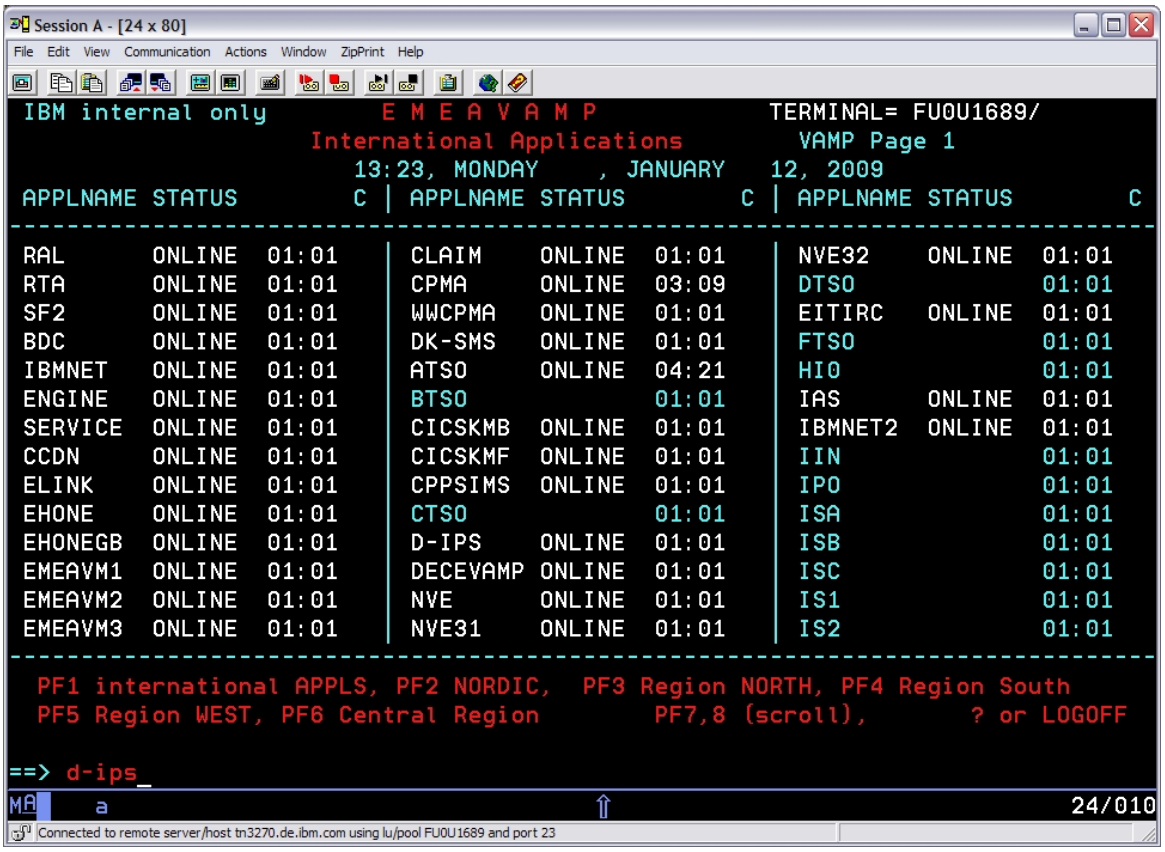
1 Voraussetzungen für das Starten der WLP-Anwendungen

Diese Voraussetzungen gelten für den Betrieb der beiden Anwendungen **RMFRecorder** und **WLPrediction**, wobei Kapitel 1.2 für *WLPrediction* keine Rolle spielt.

1.1 Initial Program Load (IPL)

Als erstes wird ein Host Account benötigt. Als Betriebssystem sollte mindestens z/OS 1.9 hochgefahren werden. Und selbstverständlich werden für die meisten Aufgaben Admin-Rechte benötigt.

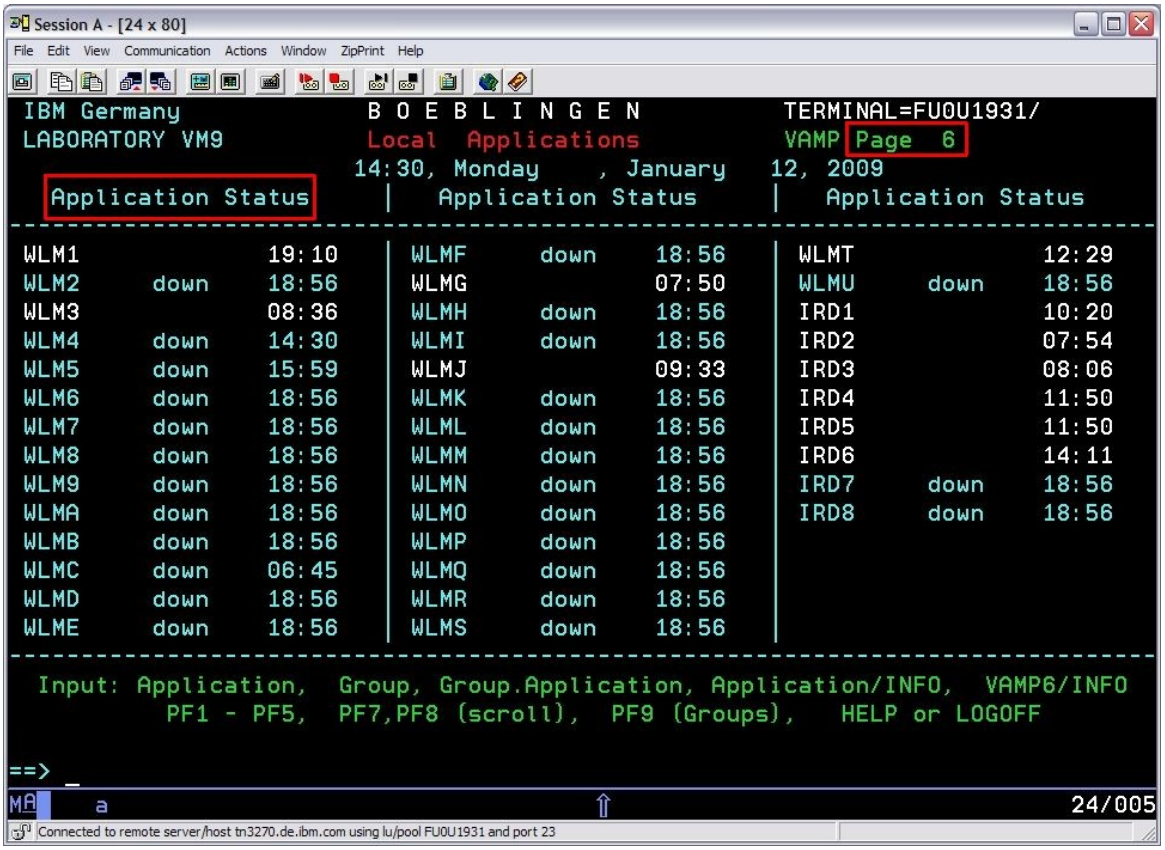
1. Als erstes muss eine **tn3270 Host Session** gestartet und im geöffneten Fenster der Befehl '**d-ips**' eingegeben werden, um die verfügbaren Systeme auf Deutschland einzuschränken.



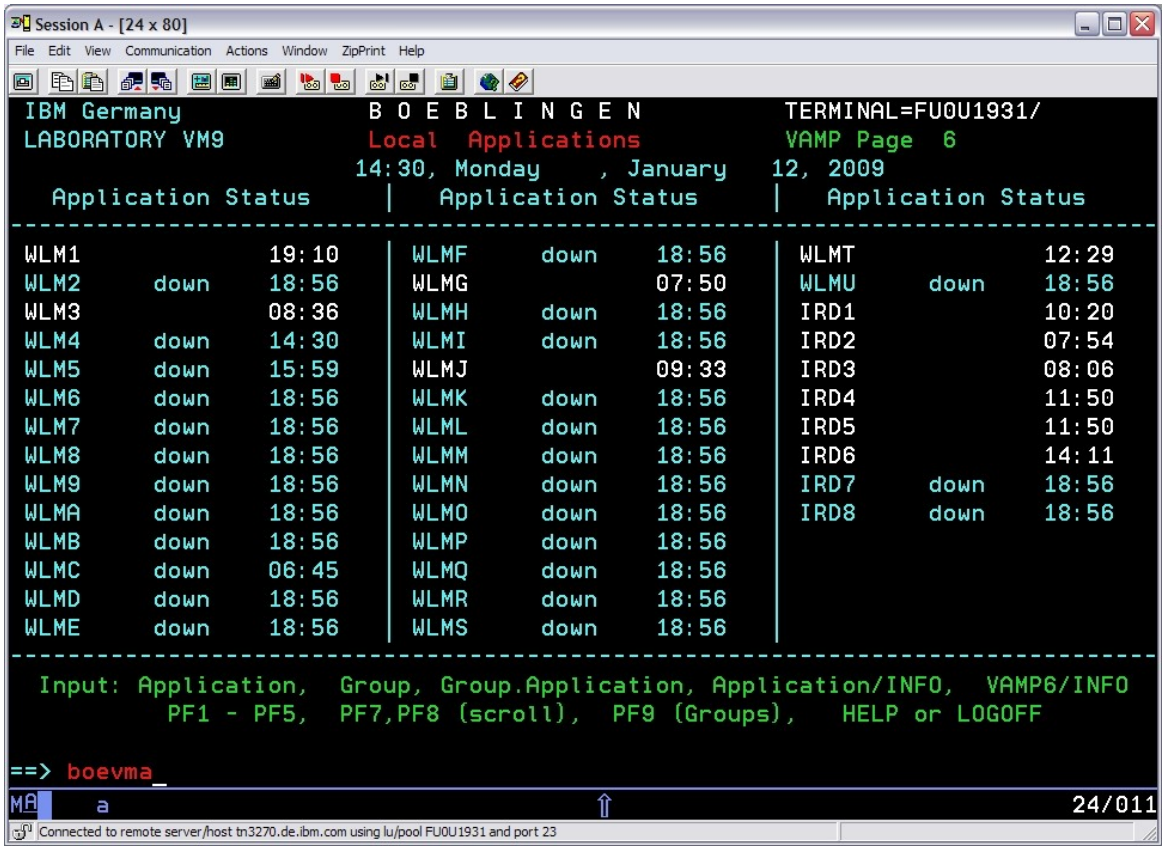
(**Note:** Der Befehl wird mainframe-typisch mit der **Strg**-Taste abgeschickt, nicht mit **Enter**!)

2. Nun kann man schauen ob das gewünschte System schon online ist. Dies kann man am **Application Status** ablesen. Wenn hinter dem Systemnamen **down** steht, dann ist der Host offline, ansonsten ist er online. Zusätzlich besteht auch noch eine

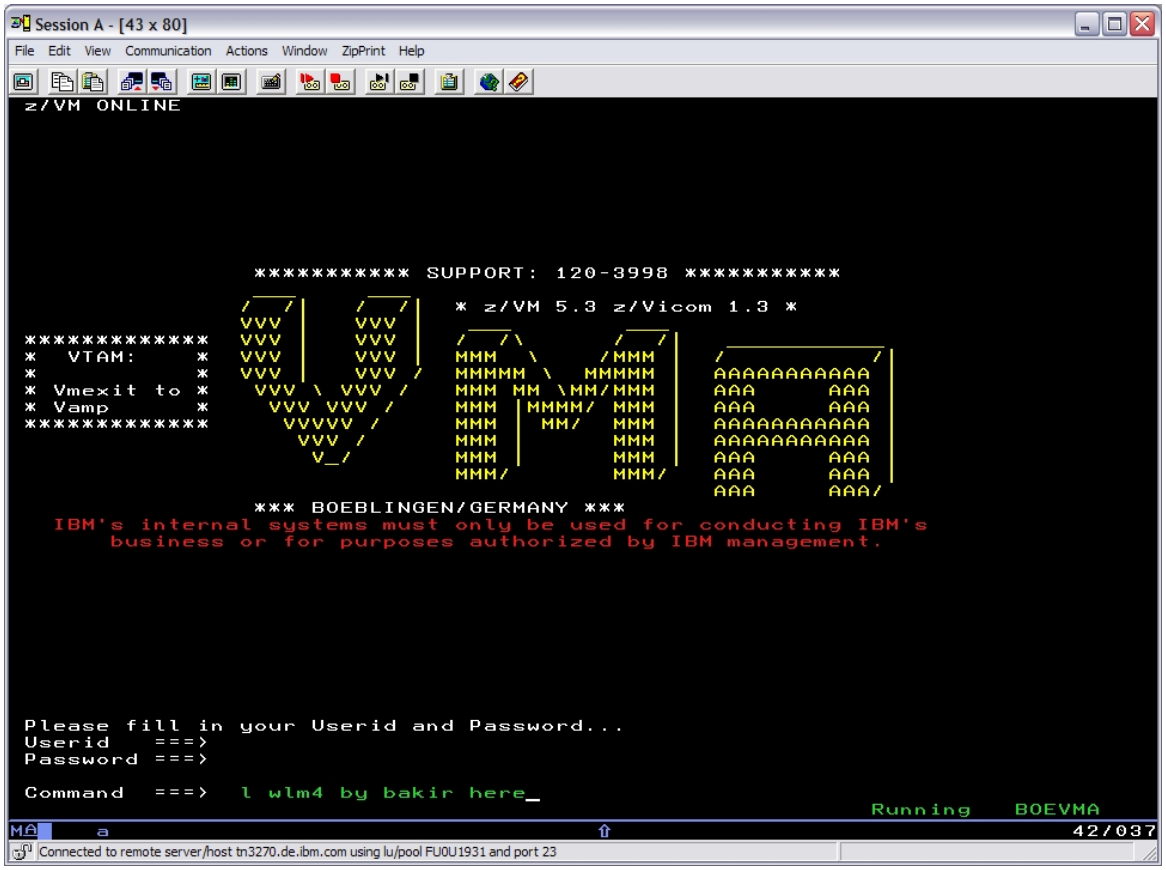
farbliche Unterscheidung zwischen online und down.
Das hier benutzte Testsystem **WLM4** findet man, indem man erst die **F6**-Taste drückt um auf **Page 6** zu wechseln.
Falls das gewünschte System online ist, bitte mit Schritt 10 weitermachen.



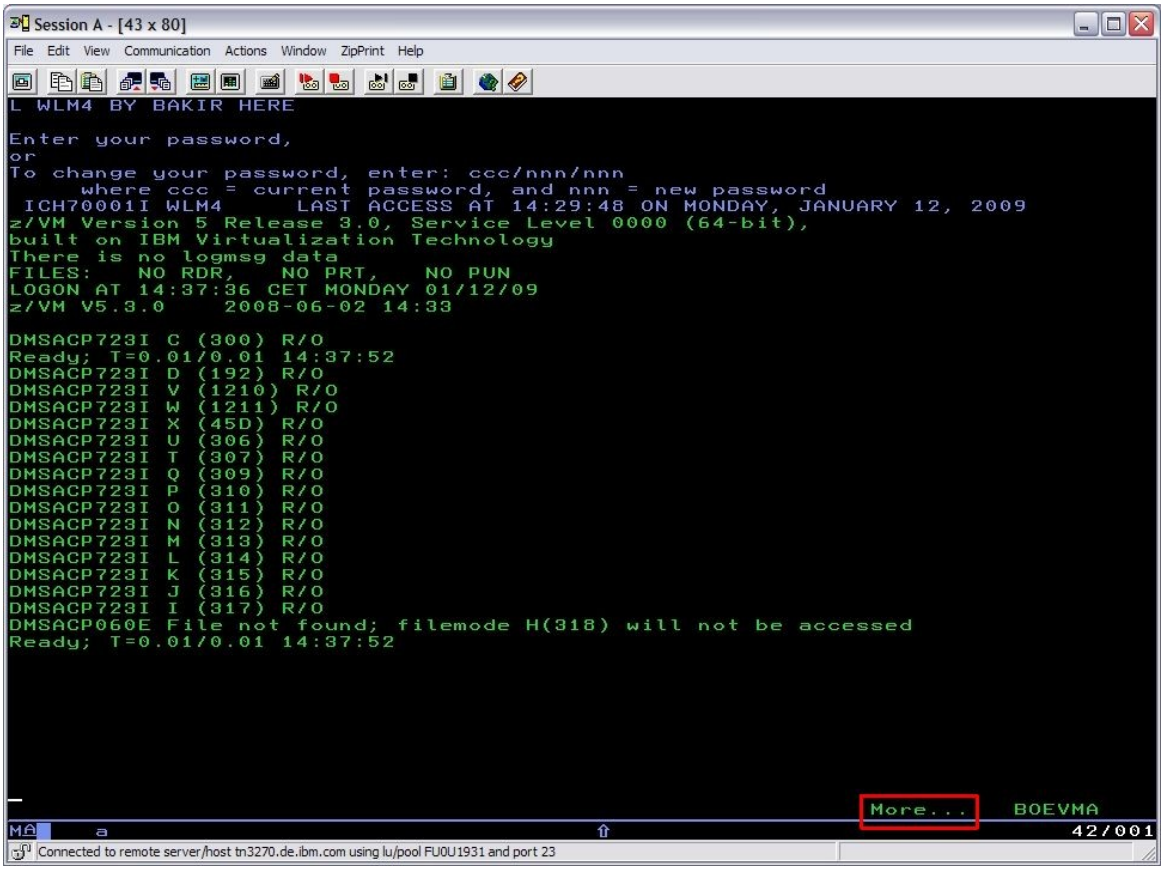
3. Wenn das System down ist, dann muss man es zunächst hochfahren, bei Mainframes heisst das **IPL**. Das hier benutzte Testsystem ist ein virtualisiertes System (**z/Vicom**) in welches man sich erst einloggen muss, bevor man es hochfahren kann.
   Mit dem Befehl '**boevma'** kommen wir in die Konsole.



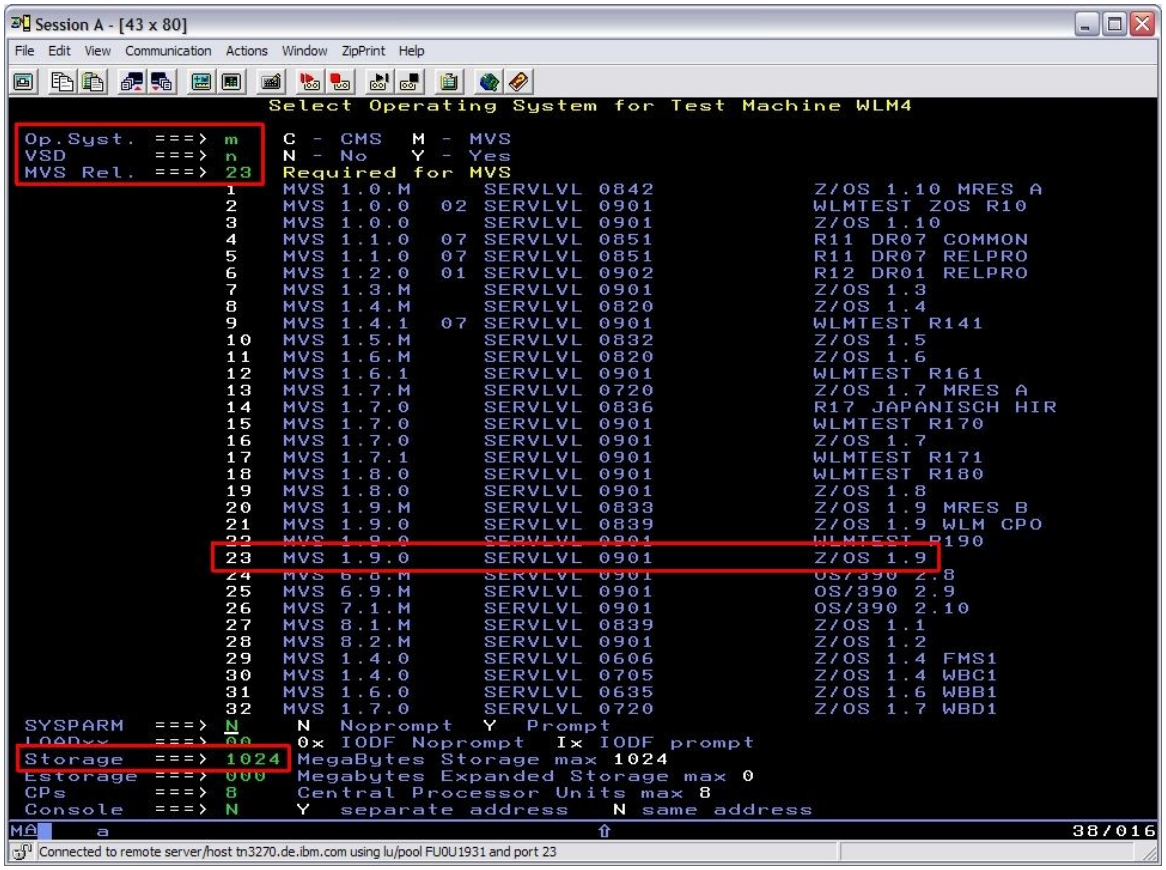
4. Dort wechselt man auf die Zeile **Command** ganz unten und gibt folgenden Befehl ein: **'l Hostname by UserID here'**.
   In den meisten Fällen entspricht die UserID für die Konsole **NICHT** der oben erwähnten UserID, die man für das einloggen auf dem Host verwendet, da Erstere erweiterte Rechte benötigt.
   Im vorliegenden Fall würde der Befehl so aussehen:
   'l wlm4 by bakir (here)'
   Das ist **'here'** optional, für den Fall eines **Reconnects**. Im Anschluss wird wie gewöhnlich ein Passwort abgefragt.



5. Unter Umständen muss nach der Passwortbestätigung wiederholt **Strg** gedrückt werden. Falls am rechten unteren Rand nach Drücken der Strg-Taste **More...** stehen sollte, dann muss zum weiterkommen die **Pause**-Taste gedrückt werden. Mit Hilfe der Kombination beider Tasten kommt man dann zum nächsten Screen

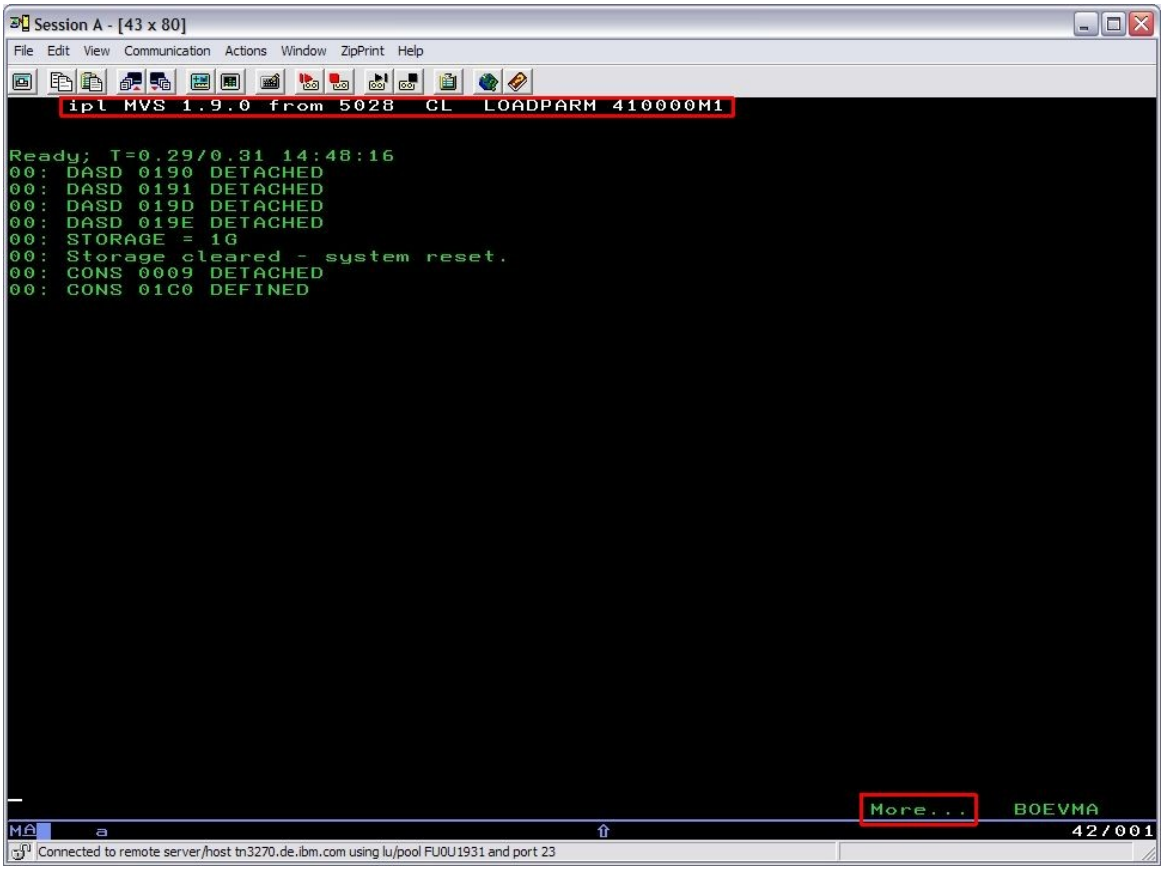


6. Im Screen **Select Operating System for Test Machine** muss man unter Op. Syst. **'m'** angeben, unter VSD **'n'** und und abschließend unter MVS Rel. die Nummer für das gewünschte Release. Per Voraussetzung sollte mindestens **z/OS 1.9** oder höher ausgewählt werden. Im vorliegenden Fall wird **z/OS 1.9** mit Release **'23'** gewählt
   Bei den restlichen Einstellungen kann man die Default-Werte übernehmen und mit der **Strg**-Taste bestätigen.



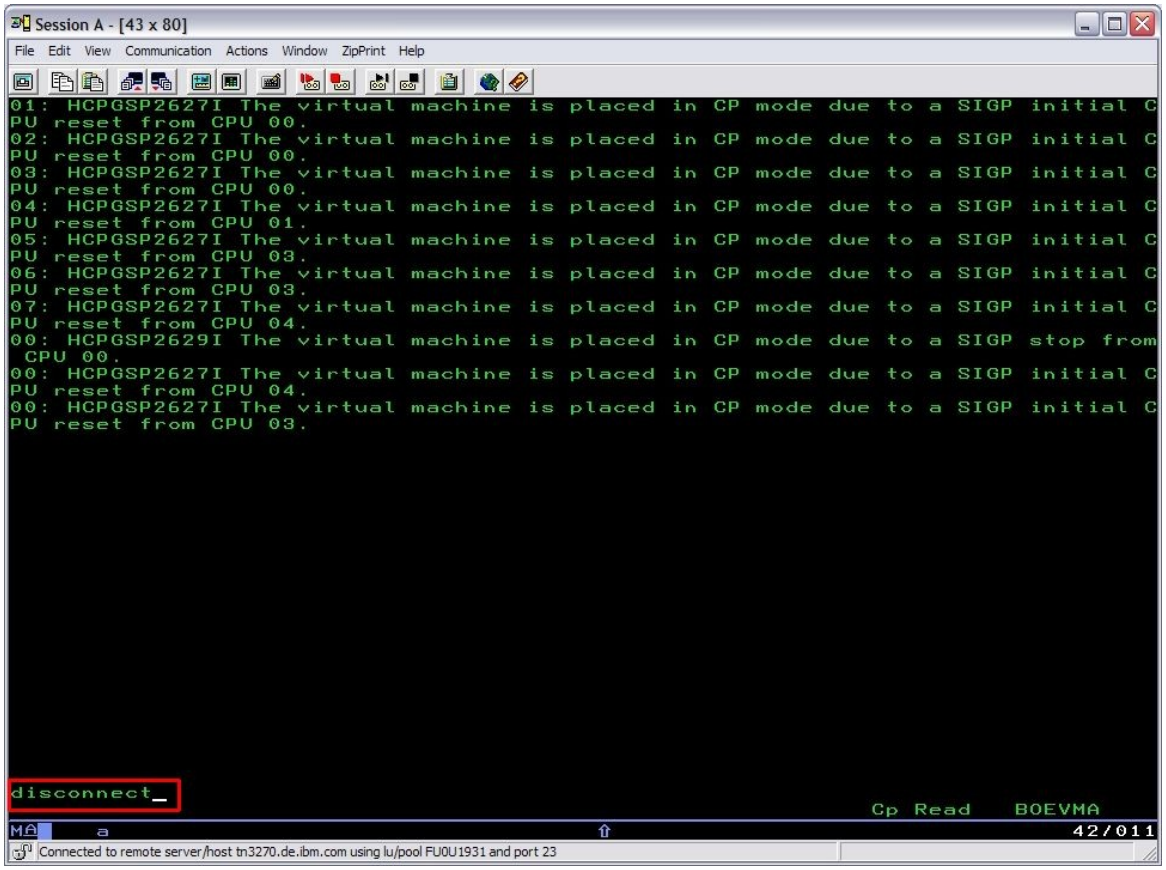
(**Note:** Im Zuge der Umstellung von **Apache Derby** auf **DB2**, sollte der Wert für Storage per Default mindestens auf **1024 MB** stehen!)

7. Nun kommt man durch wiederholtes Drücken der **Strg-** und **Pause-**Tasten zum unten angezeigten **ipl MVS** Screen. Nach kurzer Zeit sollte unten rechts statt **Running** wieder **More...** stehen. Durch erneutes Drücken von **Strg** startet dann der eigentliche IPL, der einige Minuten in Anspruch nehmen kann.

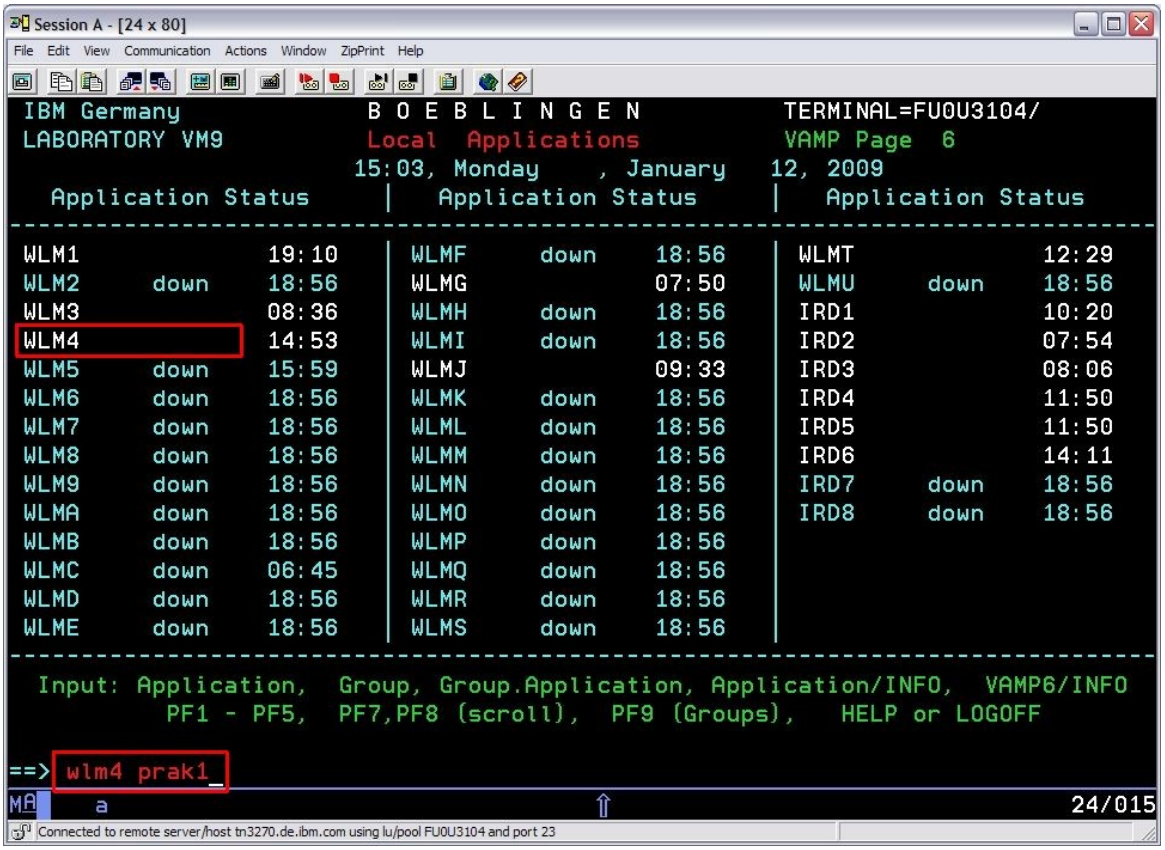


8. Wenn man sicher gehen will, dass der IPL nun abgeschlossen ist, kann ein weiteres Terminal-Fenster geöffnet werden und durch wiederholen der Schritte 1-3 der Status des Systems wiederholt werden.

9. Bevor nun die Konsole geschlossen wird, sollte eine Abmeldung erfolgen. Das wird durch Drücken der **Escape-**Taste gefolgt von der Eingabe des Befehls **'disconnect'** und abschließender Bestätigung durch drücken der **Strg**-Taste erledigt.
   Die Konsole kann nun geschlossen werden.



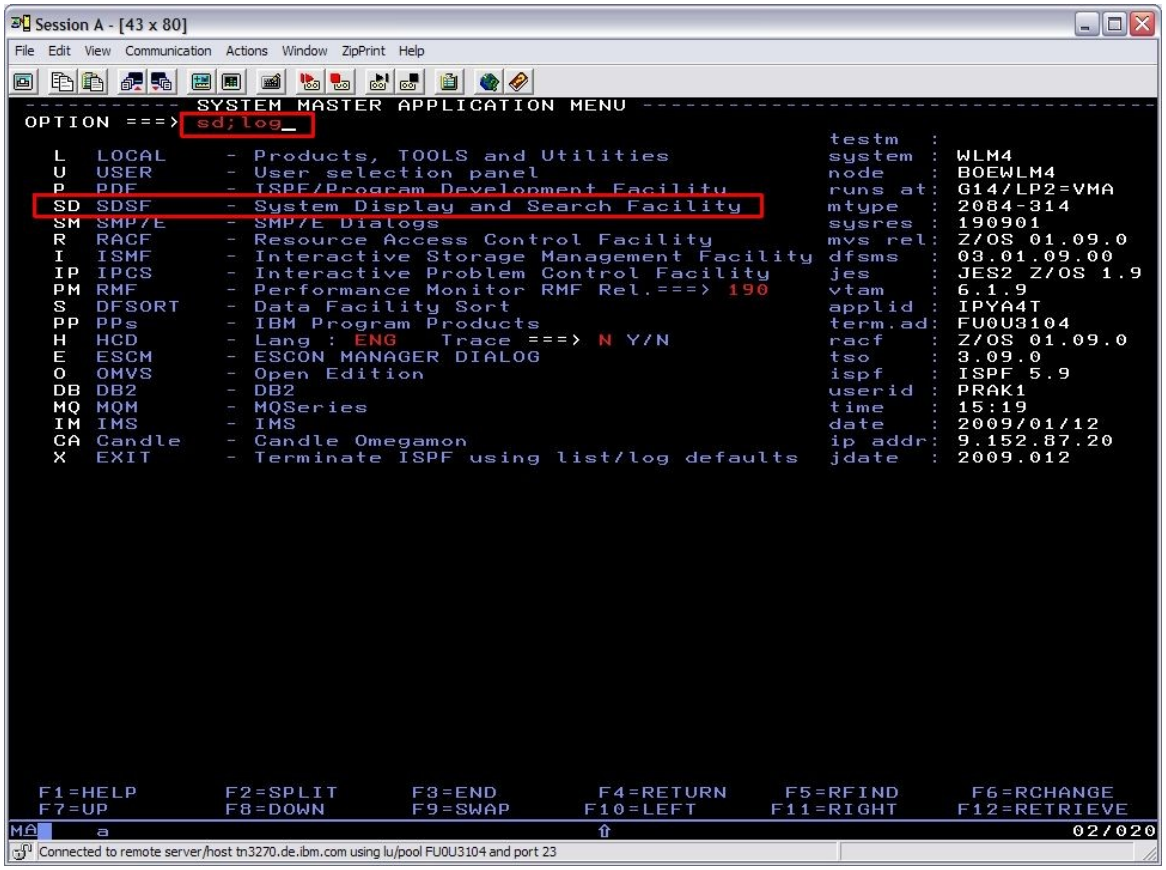
10. Wenn das System schließlich online ist, kann der Login mit dem Befehl **'Hostname UserID'** erfolgen. Für den vorliegenden Fall wäre das **'wlm4 prak1'**.
    Nach dem Login-Befehl erfolgt die obligatorische Passwort-Abfrage.



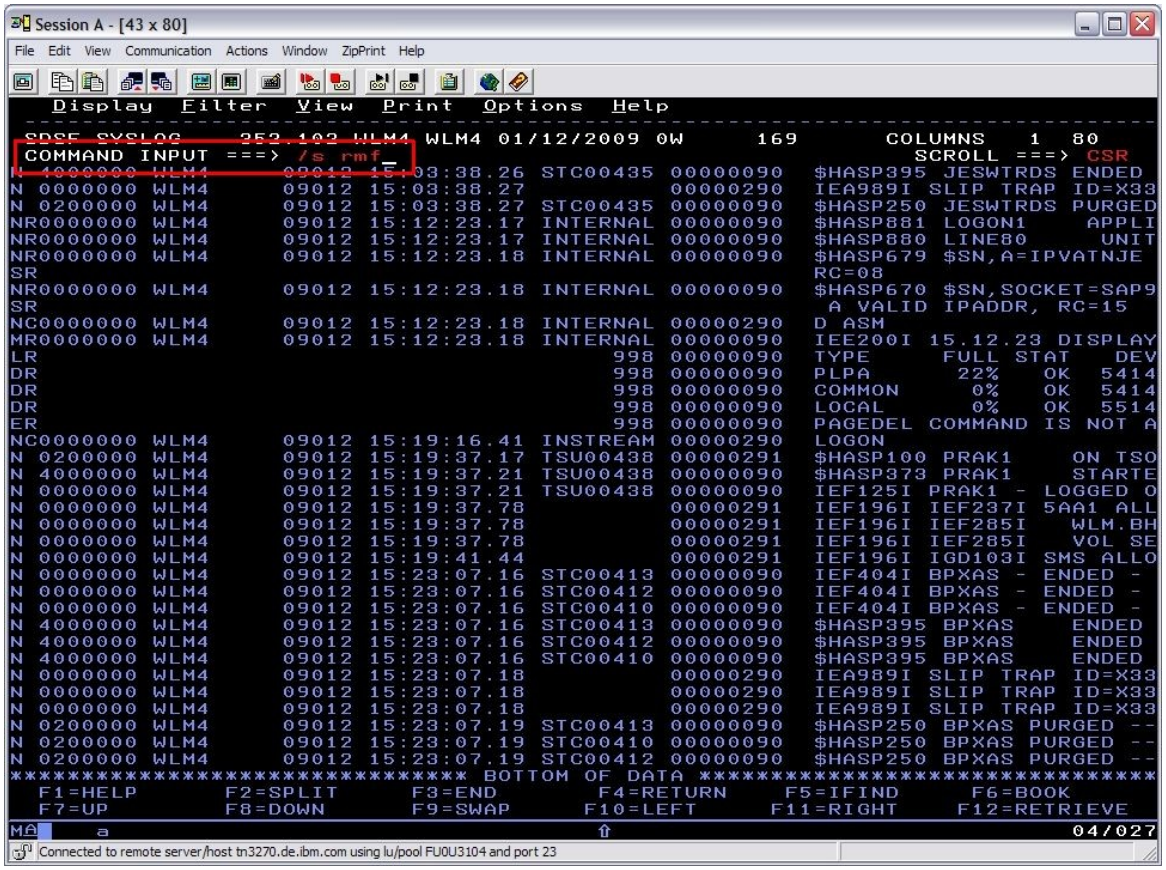
1.2 RMF-Monitor, DD-Server und CIM-Server starten

Falls nicht schon geschehen, sollten nacheinander die Komponenten **RMF**, **DDS** und **CIM** gestartet werden:

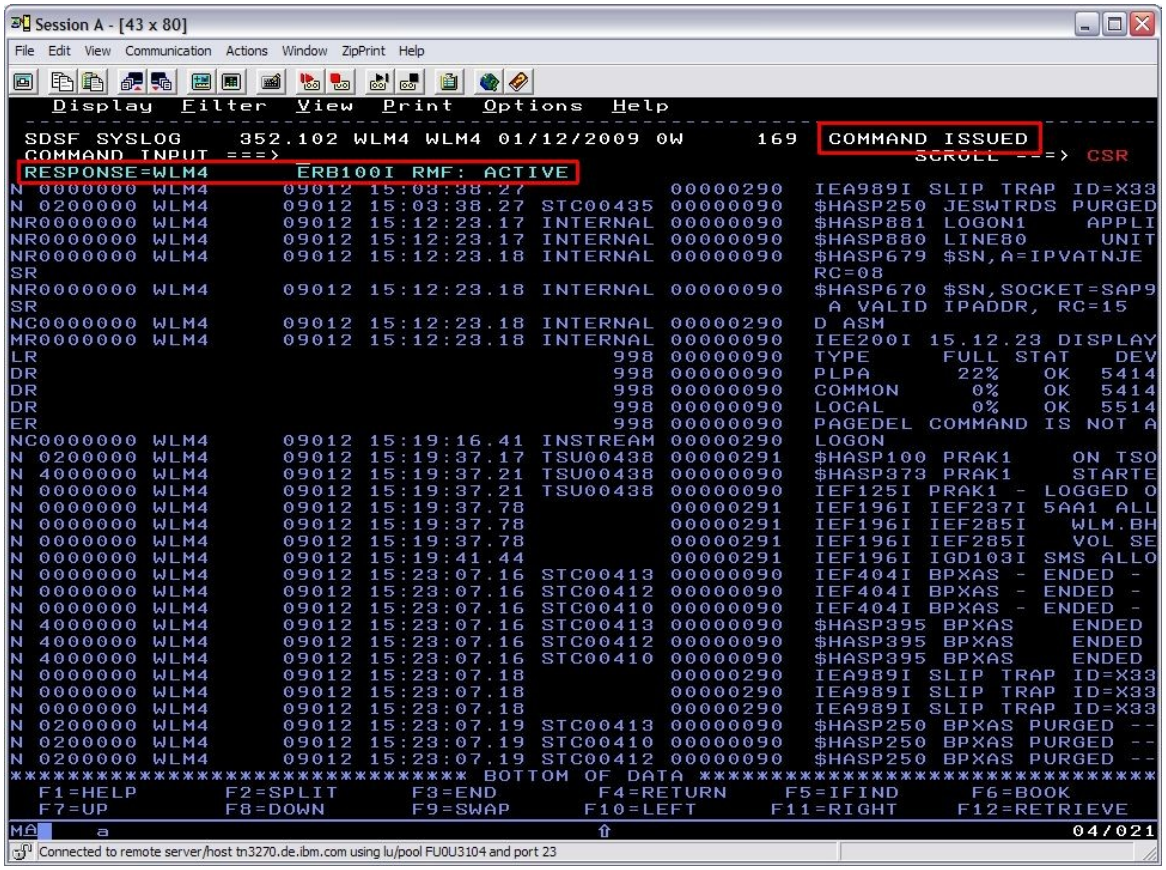
11. Es hat sich als überaus nützlich erwiesen, Befehle direkt im SDSF Log auszuführen, denn es kann in der Folge das Resultat direkt überprüft werden. Um in das **SDSF** zu kommen muss man im System Master Application Menu **'sd'** eingeben und im folgenden Untermenü **'log'** oder direkt im ersten Screen **'sd;log'**.



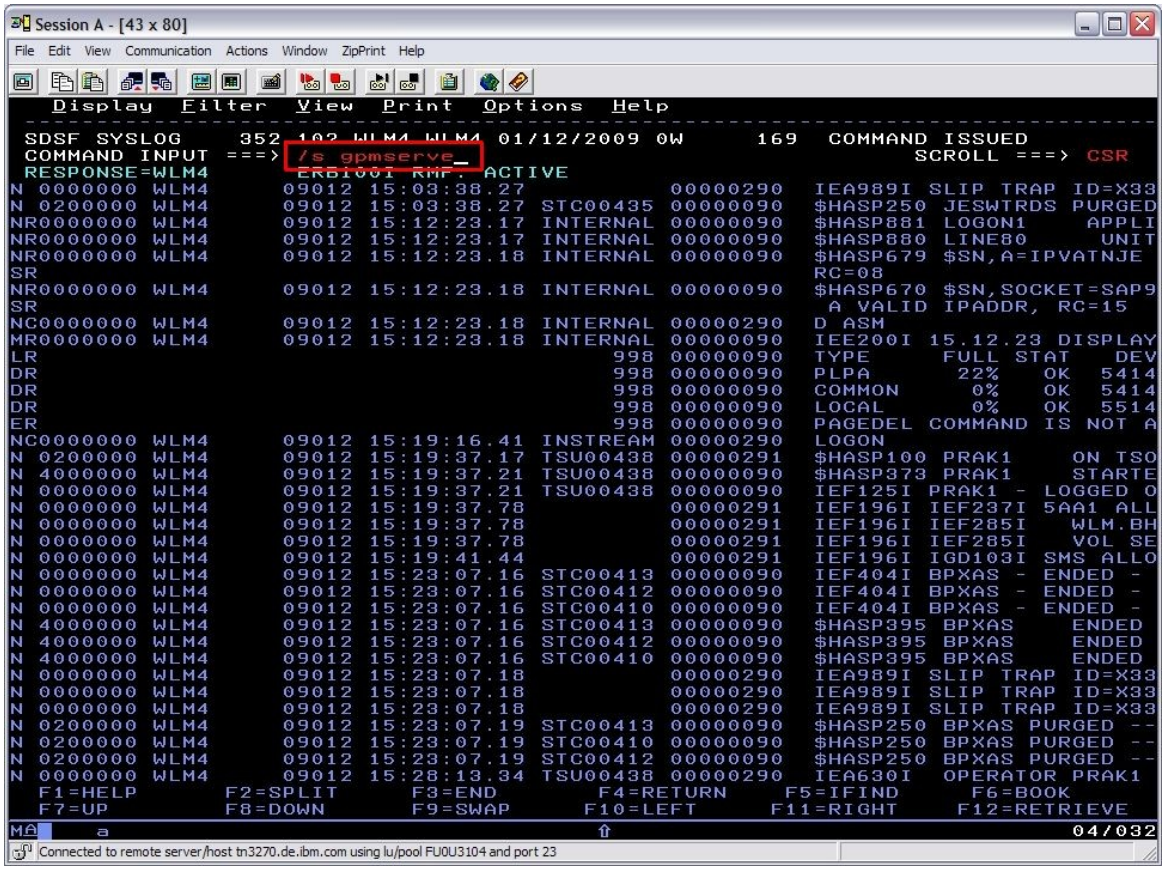
12. Starten des **RMF Monitor (III)** mit dem Befehl **'/s(tart) rmf'**.



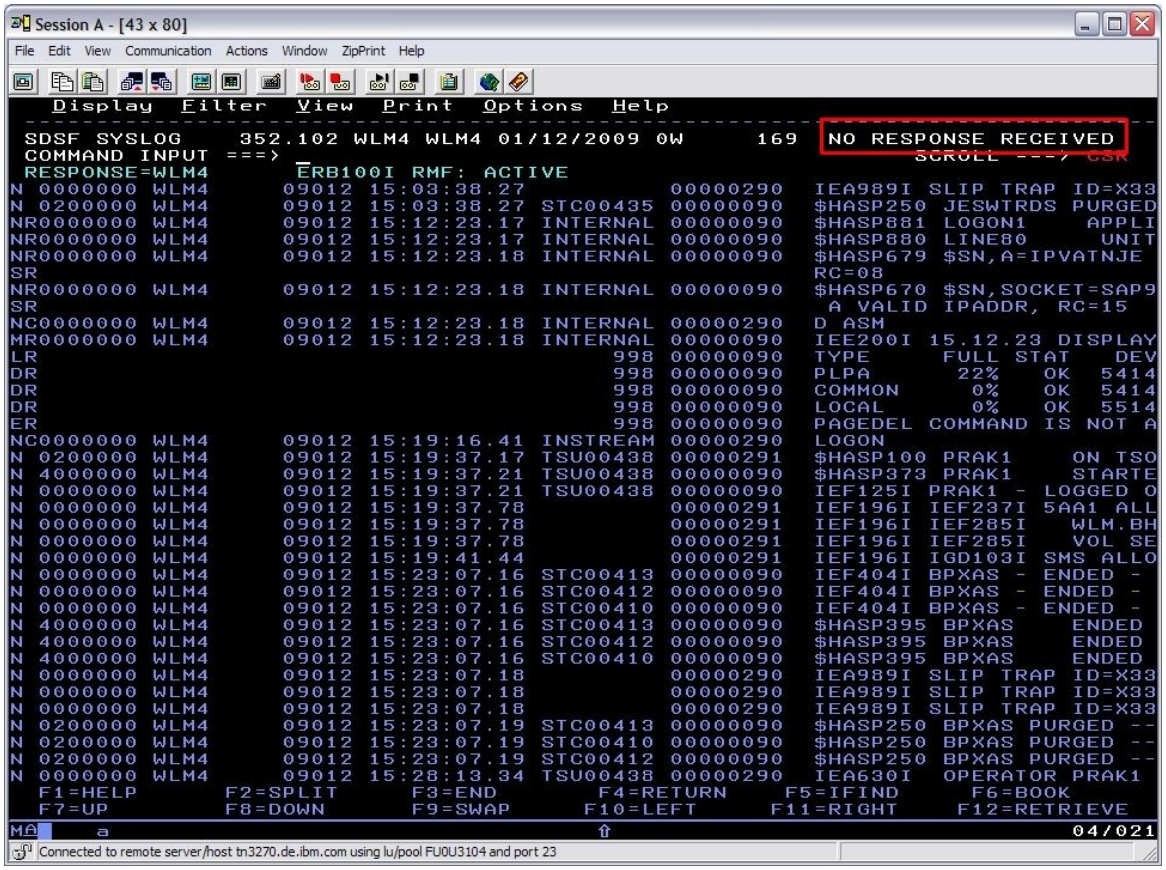
13. Nach der Ausführung des Befehls sollte eine **Command Issued** Meldung und als **Response** eine **RMF: Active** Meldung erscheinen.



14. Als nächstes sollte der **RMF Distributed Data Server (GPMSERVE)** gestartet werden. Dies wird mit Hilfe des Befehls **'/s gpmserve'** erledigt.

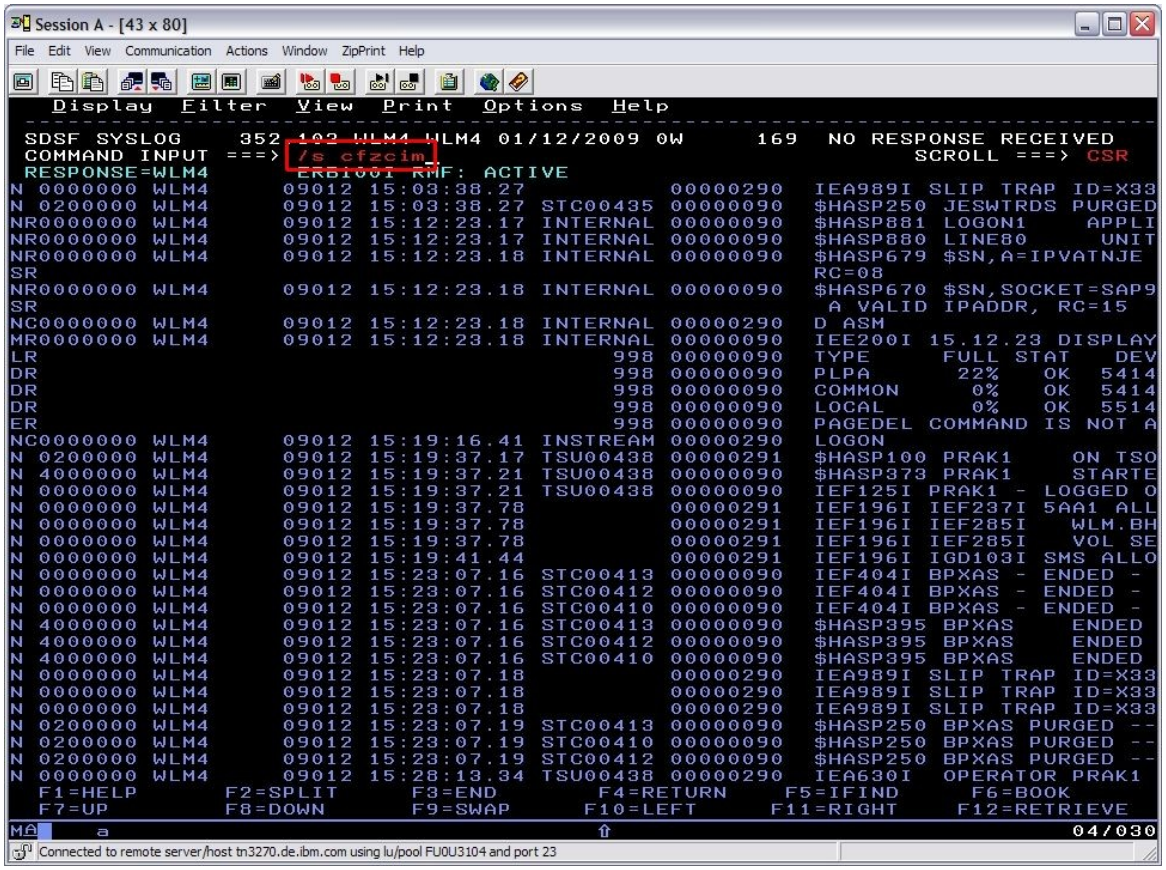


15. Es passier offenbar nichts und es erscheint die Meldung **No Response Received**. Das ist normal und sollte nicht beunruhigen.



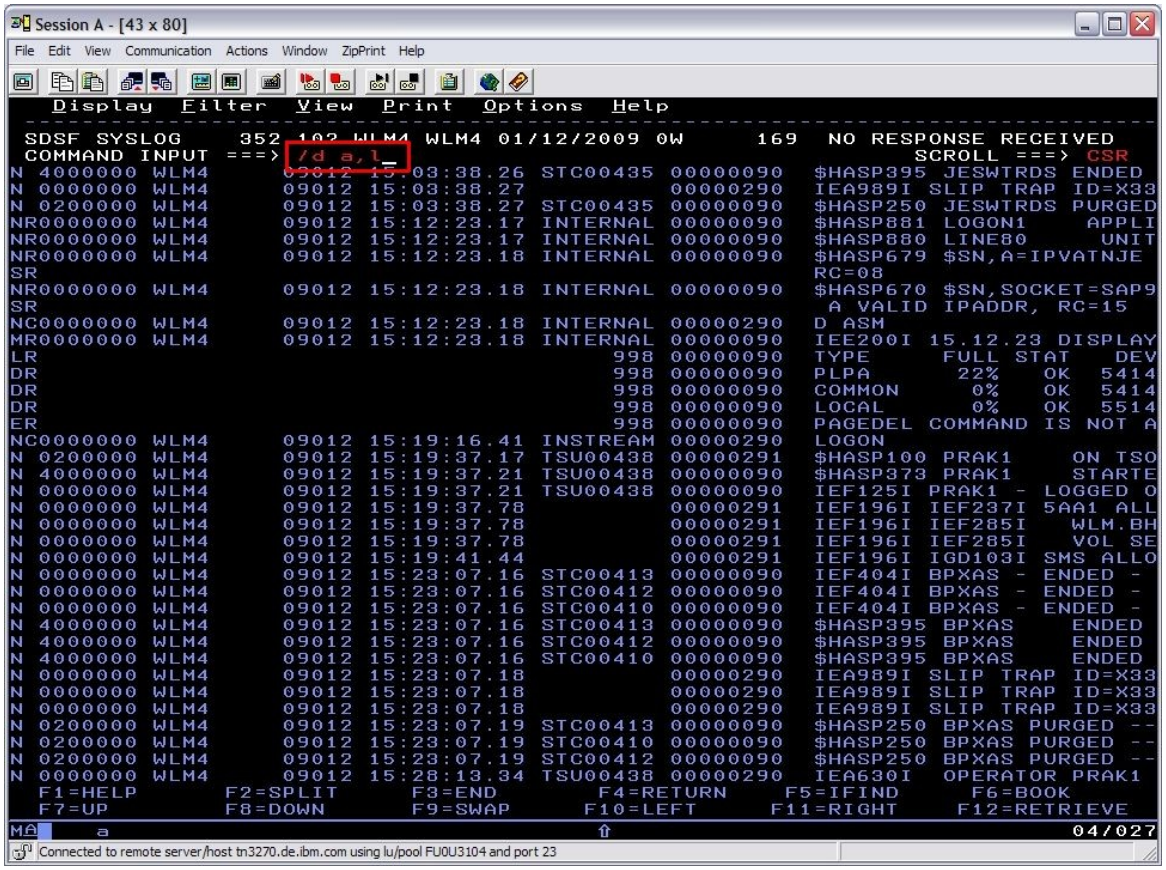
(**Zu beachten:** Die UserID, mit der man den Server startet benötigt zwingend ein **OMVS Segment** in RACF und ein **Home Directory** im USS Filesystem. Eventuelle Fehlermeldungen bitte darauf überprüfen!)

16. Schließlich muss als Letztes der **Pegasus CIM-Server** mit dem Befehl **'/s cfzcim'** gestartet werden.

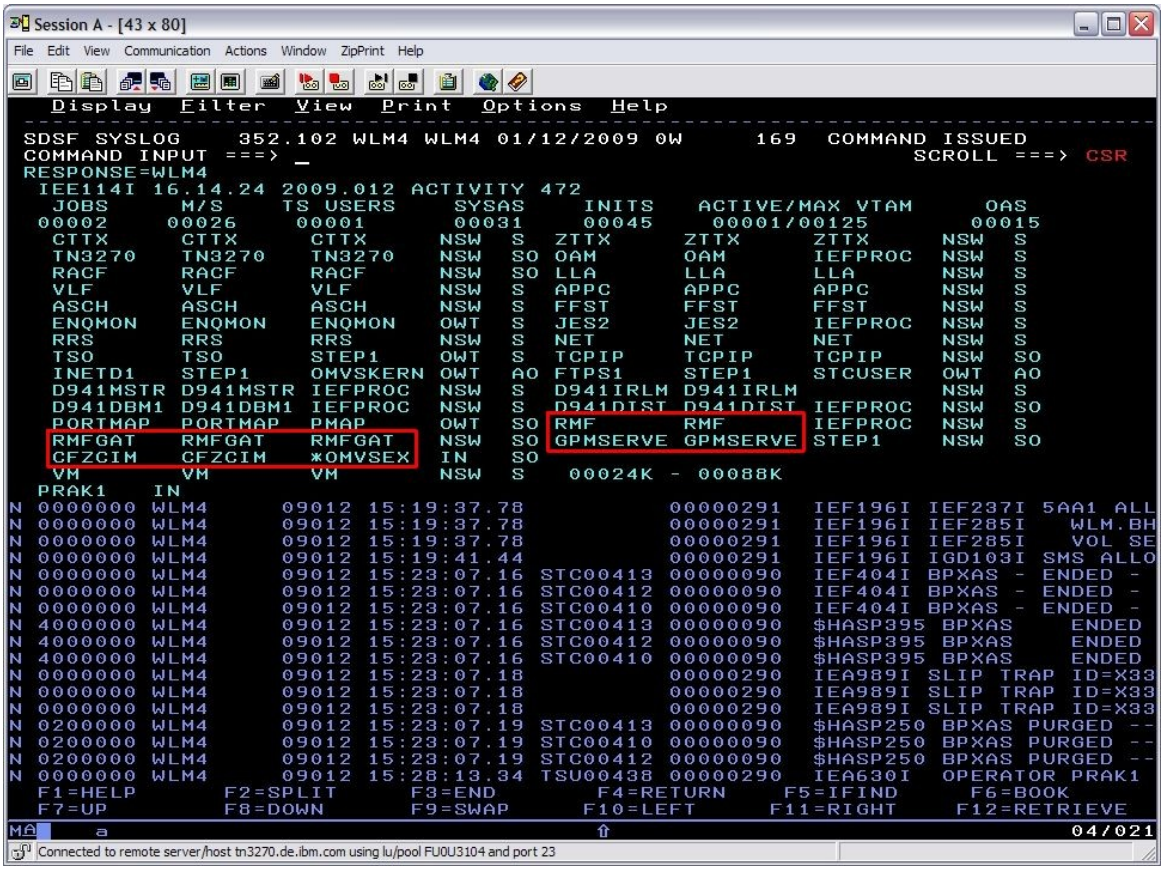


17. Wiederum kommt eine **No Response Received** Meldung und scheinbar geschieht nichts.

18. Bevor man nun weitermacht, sollte vorher auf jeden Fall verifiziert werden ob die einzelnen Komponenten auch erfolgreich gestartet wurden. Dies wird mit dem Befehl **'/d(isplay) a,l'** erledigt.



19. Falls alles erfolgreich gestartet wurde, sollten Einträge wie **RMF**, **RMFGAT**, **GPMSERVE** und **CFZCIM** existieren.



Wenn die einzelnen Komponenten zum ersten Mal gestartet werden sollen, dann könnte es vorkommen, dass die UserID keine ausreichenden Rechte besitzt. Dies kann an den entsprechenden Fehlermeldungen im LOG erkannt werden und muss von einem Admin im RACF gelöst werden.

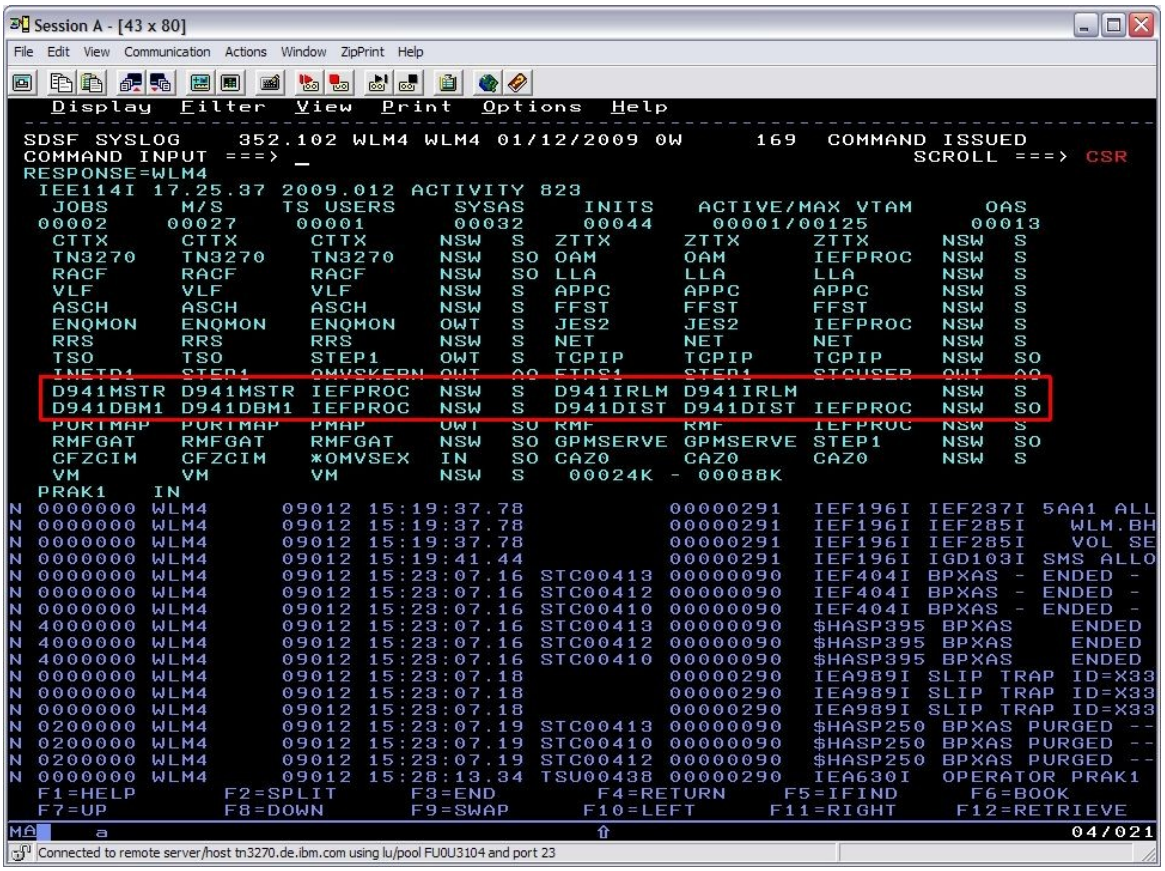
1.3 Starten der DB2-Datenbank

Als nächstes sollte man verifizieren, dass die Datenbank läuft. Das **WLP-Framework** benötigt wegen den neu eingeführten **PureXML**-Technologie IBMs **DB2 for z/OS** in der Version 9.1 oder neuer.

In der Regel wird ein Datenbank-Administrator sich um das Aufsetzen der Datenbank kümmern. Falls er nach Details für die Installation fragen sollte: Die Verbindung zur Datenbank wird über JDBC hergestellt, eine Standard-Installation sollte vollkommen ausreichen (Die benötigten XML-Features werden per Default installiert). Da Tabellen angelegt werden müssen, benötigt man auch Admin-Rechte und zusätzlich ist es äußerst praktisch, wenn die **DB2 automatisch beim IPL gestartet** wird. All dies kann ein Datenbank-Administrator recht schnell einrichten.

20. Mit dem bekannten Befehl **'/d a,l'** kann im Log überprüft werden ob DB2 läuft. Es sollten Einträge mit dem Namen der DB-Instanz vorhanden sein. Diesen erhaltet ihr vom Datenbank-Administrator, ihr werdet ihn auch später noch für die Verbindung zur Datenbank benötigen.

Im vorliegenden Fall existieren Einträge für die DB-Instanz **'D941DBM1'**.



Damit der **RMFRecorder** und die **WLPrediction** problemlos funktionieren, müssen die in der Datei **WLPDefaults.properties** (s. Kapitel 1.4) angegebenen Tabellen vor dem Ausführen der WLP-Anwendungen schon vorhanden sein.

Auf die Datenbank kann je nach Präferenz über ISPF und **SPUFI** oder mit Tools wie dem **IBM Data Studio Developer** und **SQuirreL** zugegriffen werden. Details zu den letzten beiden Tools folgen in Kapitel 2. Sobald eine Verbindung mit einem der Tools hergestellt wurde, sollten als erstes sämtliche benötigten Tabellen angelegt werden.

1.4 WLPDefault.properties

Die Konfiguration des **WLP-Frameworks** wird in erster Linie über die Datei **WLPDefaults.properties** im Root-Verzeichnis des WLPrediction-Projekts vorgenommen. Bevor also die Anwendungen gestartet werden können, sollten zuerst die benötigten Angaben in der Konfigurations-Datei gemacht werden.
Diese werden nachfolgen möglichst allgemein angegeben. Die in der Diplomarbeit verwendeten Daten werden als Beispiel in Klammern angegeben.

- Database configuration:
  - dbTrainTestURL = **jdbc:db2://Host:Port/Db2-Instanz (jdbc:db2://BOEWLM4:5941/WLMD941)**

- dbTrainTestUser = **Host-Login (prak1)**
- dbTrainTestPasswd = **Host-Passwort (weihn8en)**
- dbOnlineURL = **jdbc:db2://Host:Port/Db2-Instanz (jdbc:db2://BOEWLM4:5941/WLMD941)**
- dbOnlineUser = **Host-Login (prak1)**
- dbOnlinePasswd = **Host-Passwort (weihn8en)**

- Which reader type to use:
  - ReaderType = **CIMReader**
- CIMReader settings:
  - cimomHost = **host.boeblingen.de.ibm.com (boewlm4.boeblingen.de.ibm.com)**
  - cimomPort = **Port (5988)**
  - cimomUser = **Host-Login (prak1)**
  - cimomPass = **Host-Passwort (weihn8en)**
  - cimMetricsFile = **cimmetrics.xml**
- Set query interval length:
  - queryIntervalLength = **100000**
- Specify if RmfInterval Length is used:
  - useRmfIntervalLength = **true**
- Set query lag time:
  - queryLagTime = **10000**
- Downsampling factor:
  - downSamplingFactor = **5**
- WLP Main Settings:
  - wlpAgentRepositoryPath = **AgentPool**
  - asynchronProcessing = **true**
  - srcTable = **metrics (t060619a_ds5_xml)**
  - dstTable = **prediction**
- Web interface:
  - wlpServerPort = **8081**

Zu manchen Parametern finden sich in der Datei genauere Informationen.

1.5 BSO Authentication

Es muss beachtet werden, dass für den Zugriff innerhalb des IBM-Netzwerkes eine **BSO HTTPS Authentication** mit der **IntranetID** durchgeführt werden muss, da ansonsten kein Zugriff auf die Datenbank und den CIM-Server möglich ist. Es erscheint dann eine

entsprechende Fehlermeldung.
Dies muss in einem bestimmten Zeitintervall (mehrere Stunden) wiederholt werden!

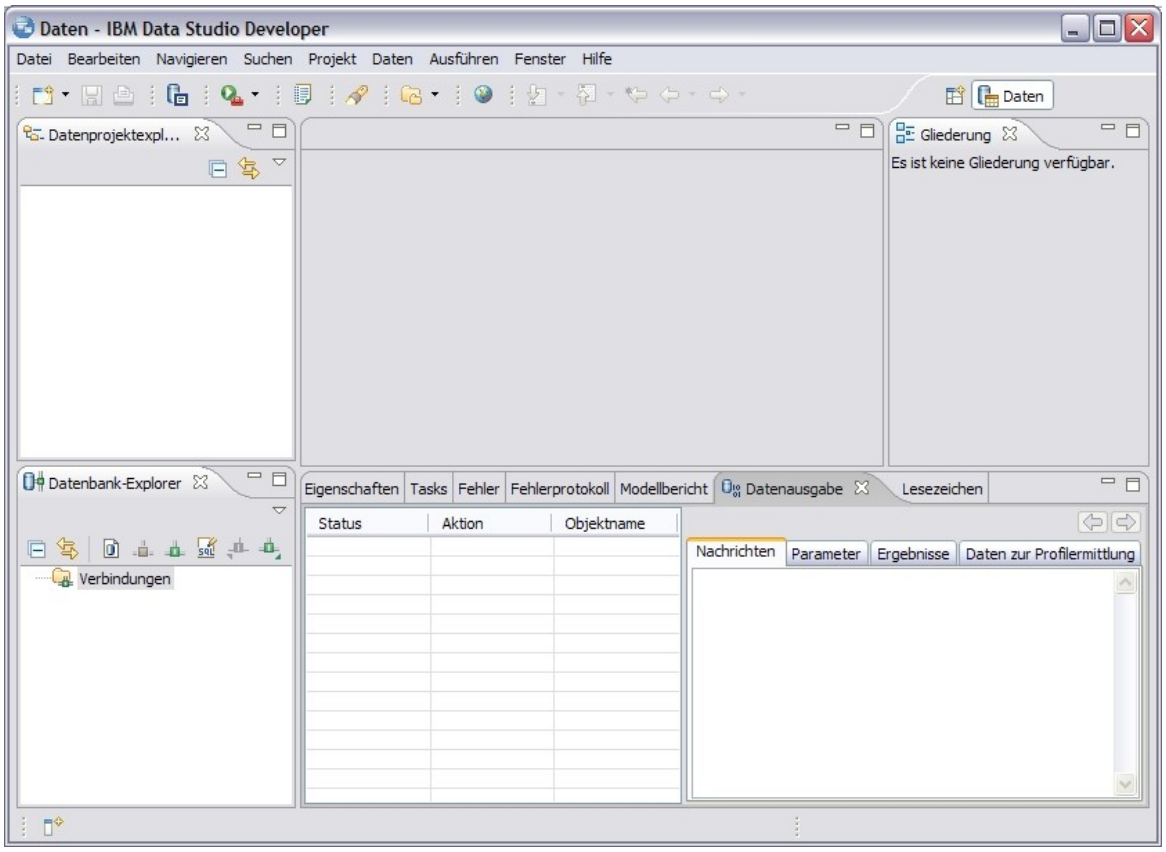
2 Herstellen einer Datenbankverbindung

Vor der Ausführung der WLP-Anwendungen müssen zunächst die benötigten Tabellen in der Datenbank erstellt werden. Das kann wahlweise über ISPF und SPUFI erfolgen oder bequem über einen graphischen SQL-Client. Im Anschluss werden zwei mögliche Werzeuge vorgestellt und alle zur Verbindungseinrichtung nötigen Schritte erläutert.

Ein Problem stellt der JDBC-Treiber für DB2 for z/OS dar. Es ist nicht möglich ihn isoliert zu beziehen. Er wird beispielsweise mit der Installation des im nächsten Kapitel vorgestellten Data Studio Developer mitinstalliert, welches als kostenlose Trial-Version öffentlich zugänglich ist.

2.1 IBM Data Studio Developer

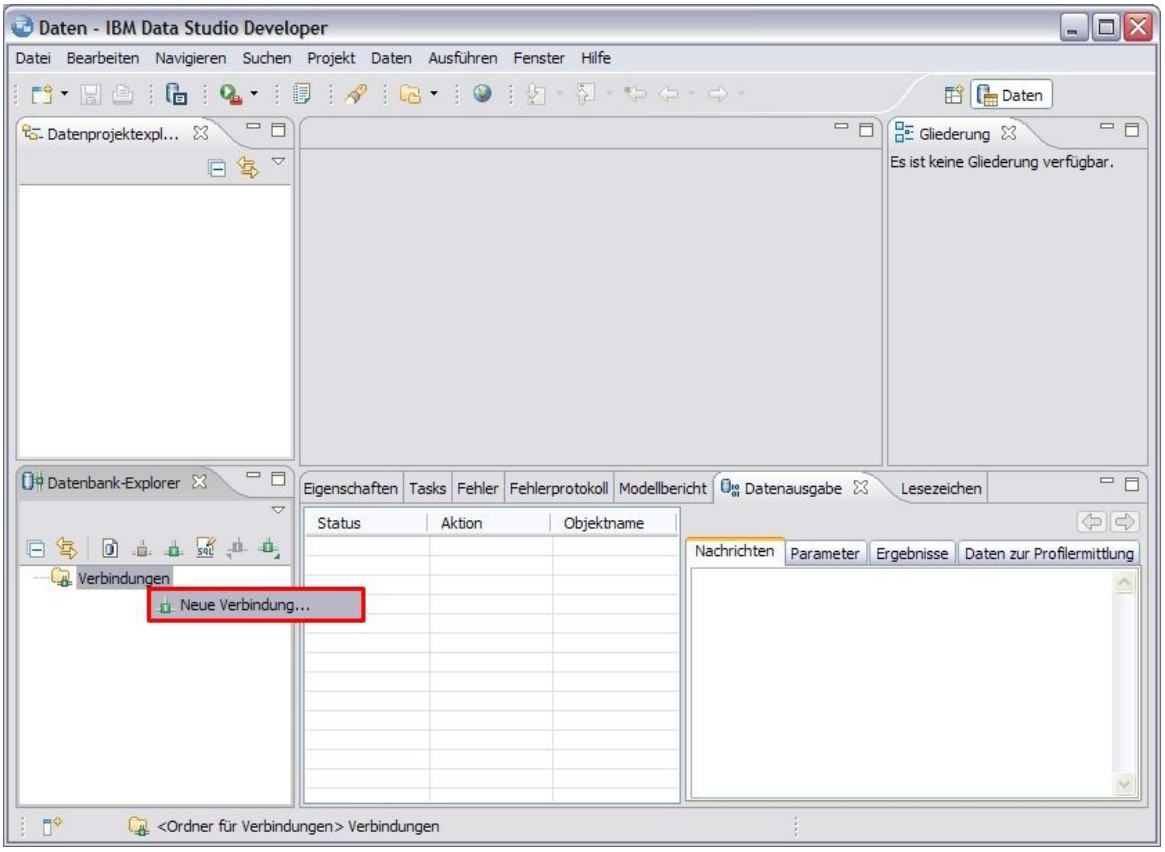
IBMs Data Studio Developer (im Folgenden als **Data Studio** bezeichnet) ist eine IDE für Datenbankentwickler, die auf Eclipse basiert. Sie ist für Entwickler und Administratoren gleichermaßen konzipiert und bietet für beide Gruppen entsprechende Funktionalität.



2.1.1 Einrichten der Verbindung

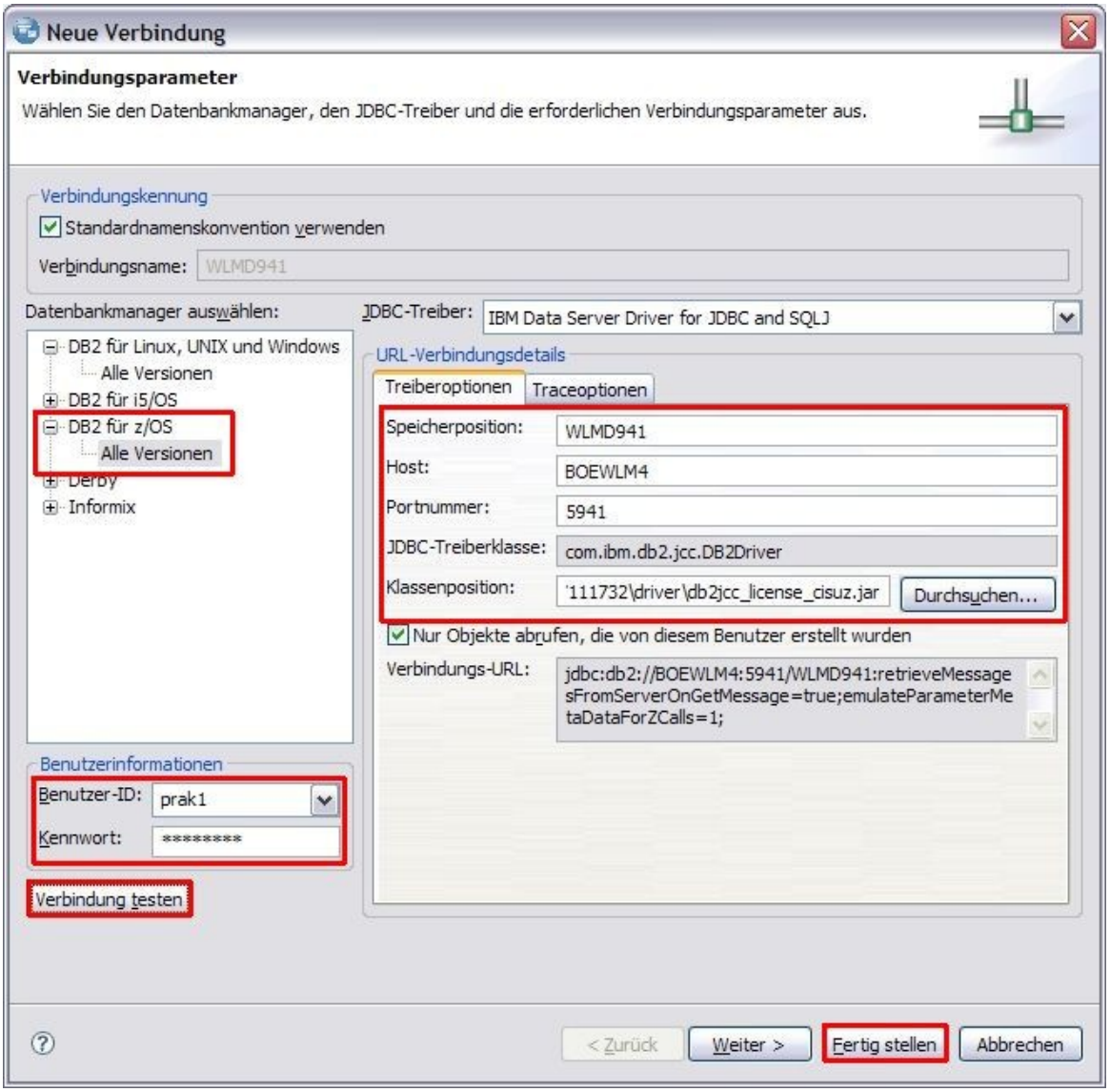
Als erstes sollte mit dem Einrichten der Verbindung begonnen werden.

21. Im geöffneten Data Studio rechtsklick auf **Verbindungen** in der **Datenbank-Explorer** View und **Neue Verbindung...** auswählen



22. Es öffnet sich ein neues Fenster mit dem Titel **Neue Verbindung**.
Hier unter Verbindungskennung den Haken bei **Standardnamenskonvention verwenden** belassen oder den Namen der Datenbank-Instanz als **Verbindungsnamen** eingeben. Als nächstes unter **Datenbankmanager auswählen** den Eintrag **DB2 für z/OS** aufklappen und **Alle Versionen** auswählen.
Unter **JDBC-Treiber** den per Default ausgewählten **IBM Data Server Driver for JDBC and SQLJ** nicht verändern und als nächstes die URL-Verbindungsdetails spezifizieren:
Als **Speicherposition** Namen der **Datenbank-Instanz** festlegen,
den **Host** und die **Portnummer** spezifizieren und als letztes unter **Klassenposition** den Pfad zum Jar-File des Treibers inklusive der Lizenz-Datei. In der Regel sollte das der Pfad **C:\Program Files\IBM\DS12Shared\plugins\com.ibm.datatools.db2_1.0.201.v200807111732\driver** sein und die benötigten Jars sind **db2jcc.jar** und **db2jcc_license_cisuz.jar**. Zur Kontrolle muss im Feld **JDBC-Treiberklasse** der Eintrag

com.ibm.db2.jcc.DB2Driver stehen.
Abschließend noch die benötigten Benutzerinformationen **Benutzer-ID** und **Kennwort** eingeben und mit Klick auf den Button **Verbindung testen** die eingetragenen Angaben auf Korrektheit verifizieren.

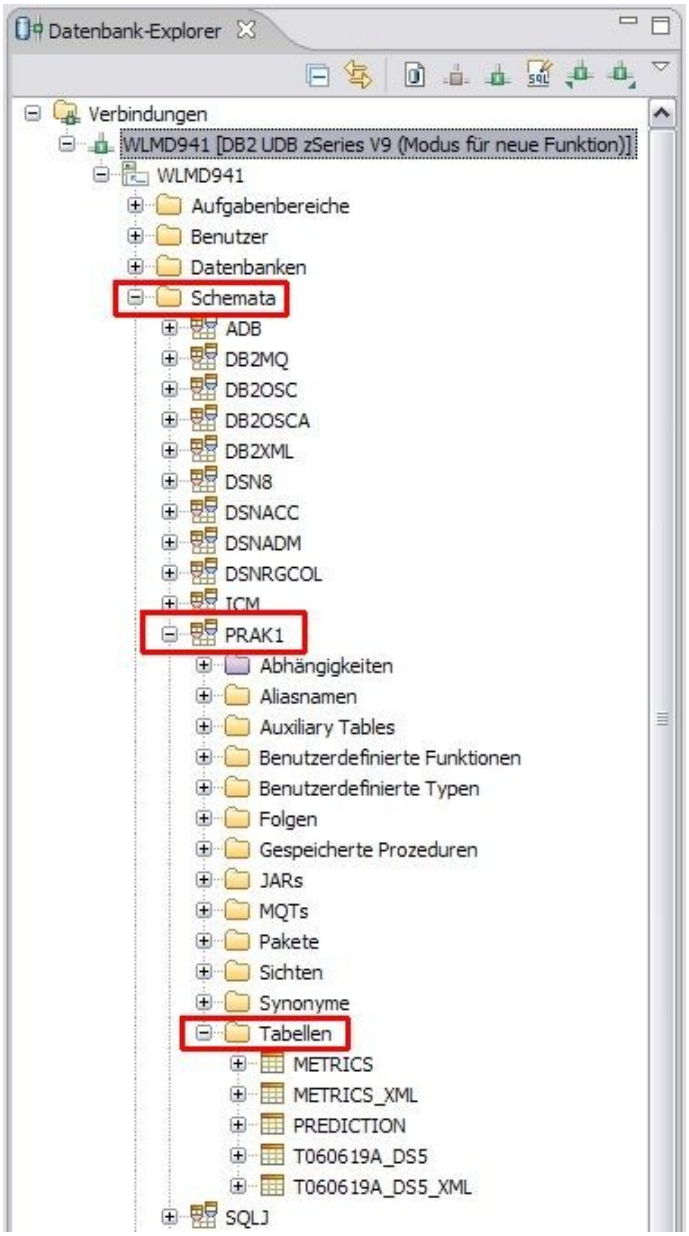


23. Falls mit einem Popup eine erfolgreiche Verbindung bestätigt wurde, auf **Fertigstellen** klicken um das Erstellen zu beenden.

2.1.2 Herstellen der Verbindung

Die neu eingerichtete Verbindung ist nun in der **Datenbank-Explorer** View verfügbar und kann eingesetzt werden.

24. Um eine Verbindung mit der Datenbank herzustellen muss man die Verbindung nur doppelklicken.
    Über eine aufklappende Baum-Struktur sind dann alle vorhanden Daten wie Benutzer, Schemata, etc.
    Unter **Schemata** befindet sich auch der jeweilige **Benutzer** und die von ihm erstellten **Tabellen**.



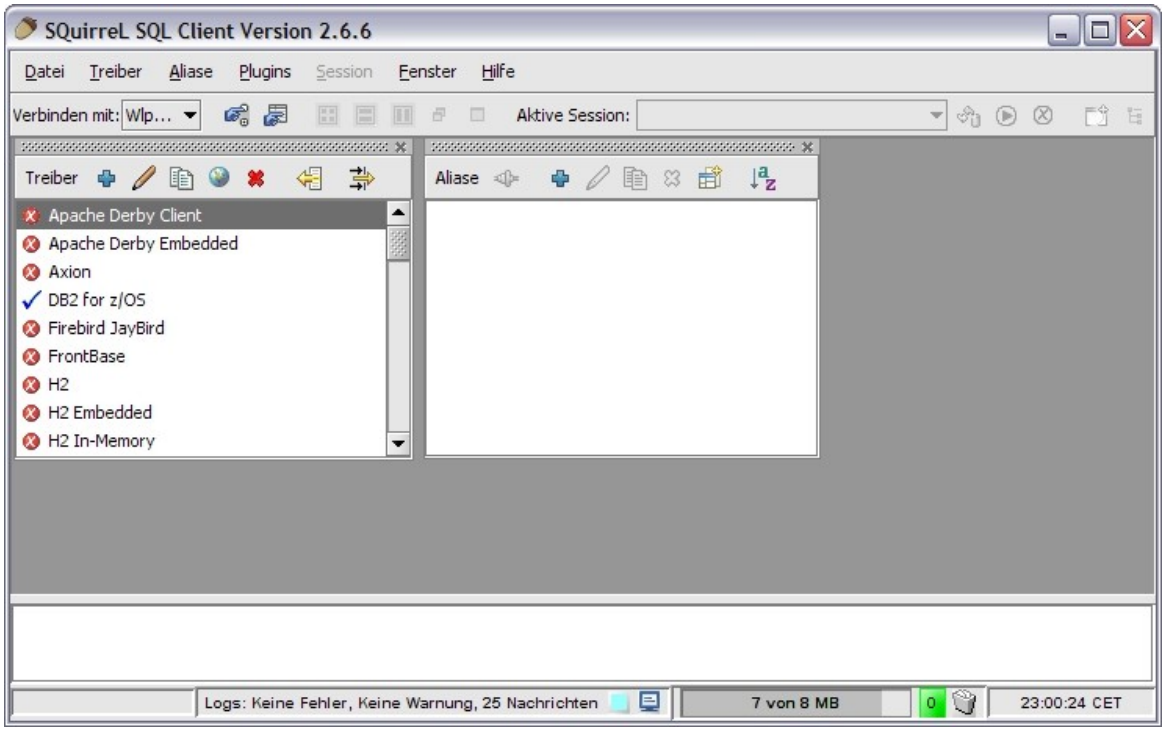
25. Nachdem die Verbindung hergestellt wurde, können sämtliche von Datenbanken bekannte Funktionen durchgeführt werden.

26. Entwickler können dann auch wie bei Eclipse üblich in der **Datenprojektexplorer** View verschiedene Projekte, so genannte **Datenentwicklungsprojekte** anlegen und komfortabel alles dafür benötigte (SQL-Skripte,...) anlegen

2.2 SQuirreL SQL-Client

SQuirreL ist ein universeller graphischer SQL-Client, der frei verfügbar ist. Mit SQuirreL ist

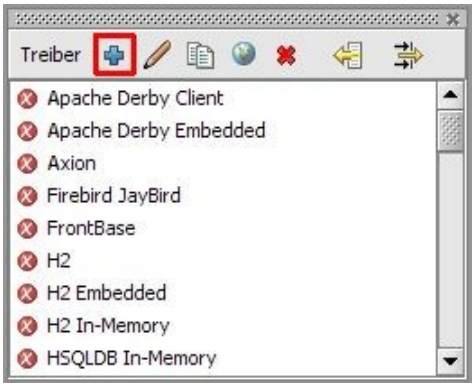
es möglich auf jede relationale Datenbank zuzugreifen, für die ein JDBC-Treiber verfügbar ist.



2.2.1 Hinzufügen und Laden des JDBC-Treibers für *DB2 for z/OS*

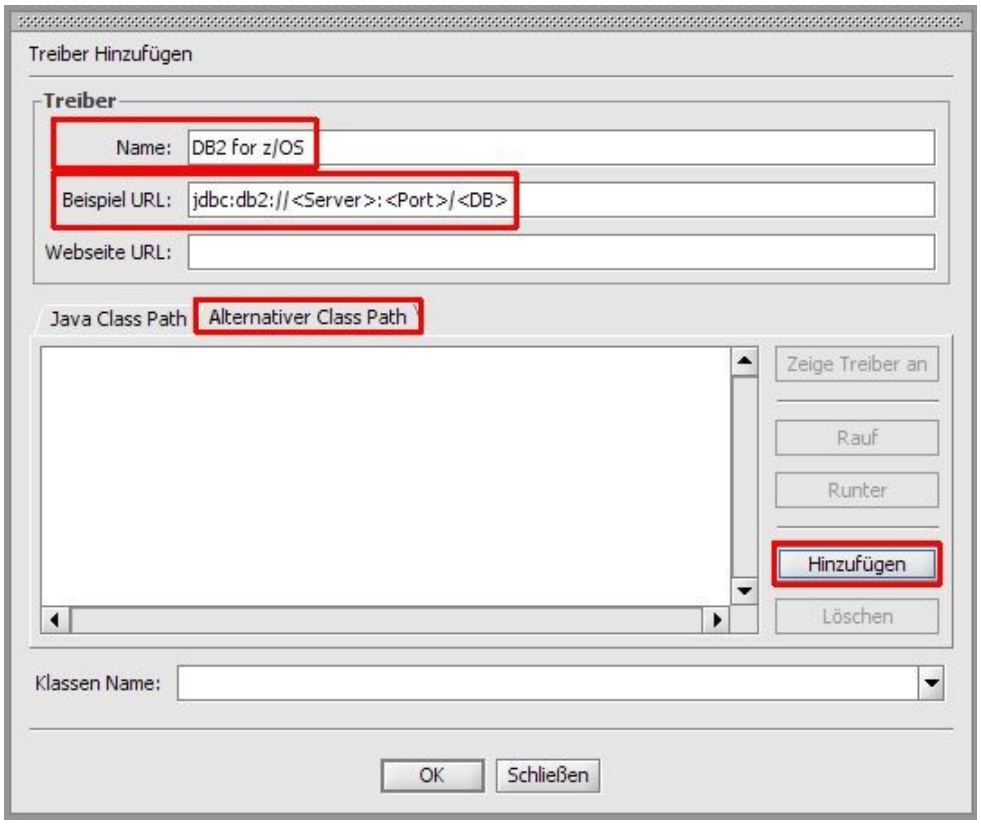
Für die erstmalige Benutzung muss der benötigte JDBC-Treiber **einmalig** hinzugefügt und geladen werden.

27. Im geöffneten SQuirreL-Fenster gibt es auf der linken Seite ein kleineres Fenster mit dem Titel **Treiber**. Um einen neuen Treiber hinzuzufügen, auf das **+**-Icon in der Titelleiste klicken.

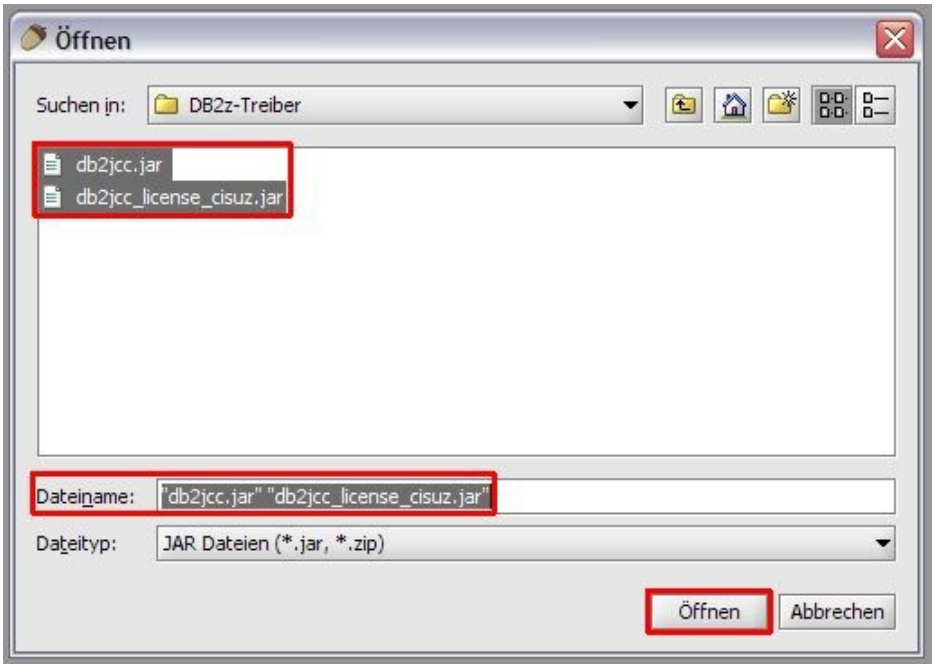


28. Es öffnet sich ein neues Fenster mit dem Titel **Treiber Hinzufügen**. Hier als Name **DB2 for z/OS** und als Beispiel URL

jdbc:db2://<Server>:<Port>/<DB> spezifizieren.
Danach auf den Reiter **Alternativer Class Path** wechseln und auf den Button **Hinzufügen** klicken.



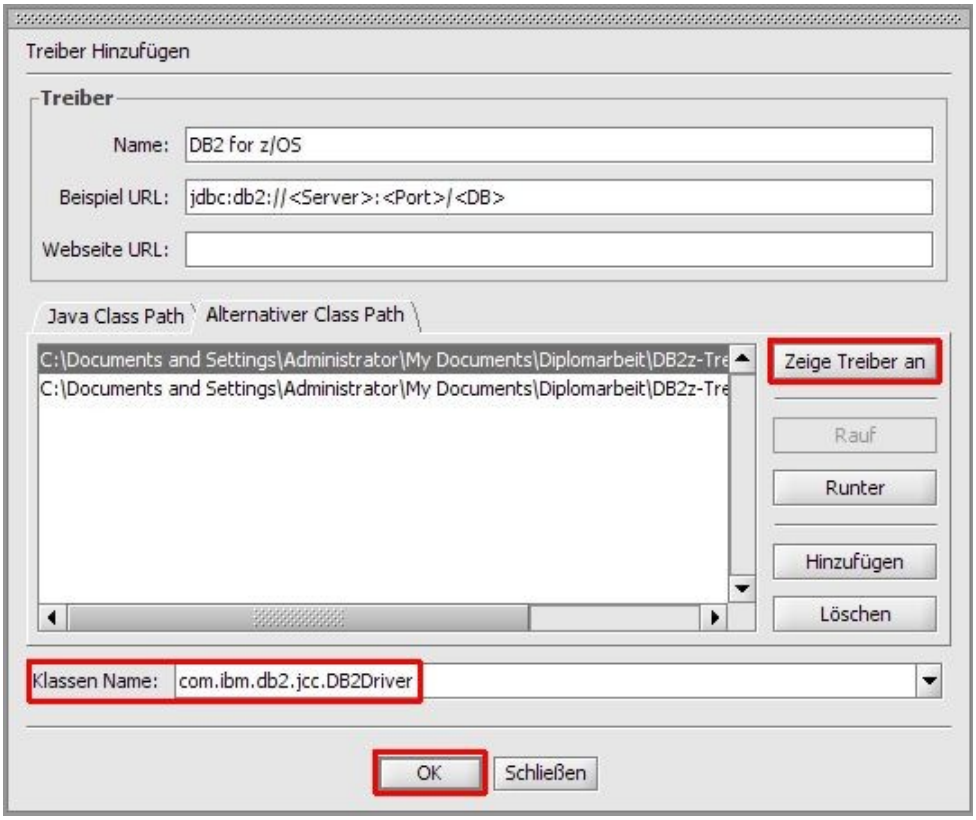
29. Es öffnet sich ein weiteres Fenster in dem man den Pfad zum JDBC-Treiber angeben muss. Der mitgelieferte treiber befindet sich im Verzeichnis **DB2z-Treiber**.
Dort mit Strg den Treiber **db2jcc.jar** und die Lizenz-Datei **db2jcc_license_cisuz.jar** auswählen und auf **Öffnen** klicken.



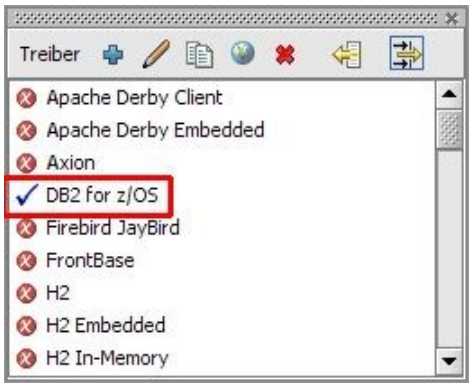
Note: Falls das **Data Studio Developer** installiert wurde, dann wurden die benötigten DB2-Treiber mitinstalliert. Sie sie sollten sich im Verzeichnis **c:\Program Files\IBM\DS12Shared\plugins\com.ibm.datatools.db2_1.0.201.v200807111732\driver** befinden

TODO: Schauen ob bei DB2 Connect die Treiber auch installiert werden

30. Zurück im **Treiber Hinzufügen** Fenster zuerst auf den Button **Zeige Treiber an** klicken, damit im Feld **Klassen Name** der Name **com.ibm.db2.jcc.DB2Driver** des JDBC-Treibers angezeigt wird.
    Zum Bestätigen auf den Button **OK** klicken.

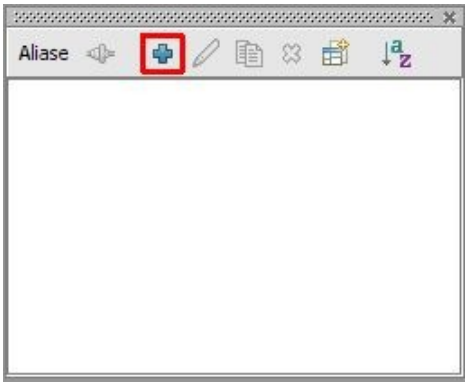


31. Der benötigte JDBC-Treiber ist hiermit hinzugefügt und geladen. Wenn das Hinzufügen erfolgreich war, dann ist vor dem namen des hinzugefügten Treibers ein blauer Haken zu sehen.



2.2.2 Einrichten der Verbindung

Nachdem der Treiber hinzugefügt und geladen ist, kann im nächsten Schritt die Verbindung zur Datenbank eingerichtet und im Anschluss hergestellt werden.



32. Im geöffneten SquirreL-Fenster gibt es auf der rechten Seite ein weiteres kleines Fenster mit dem Titel **Aliase**. Um einen neue Verbindung hinzuzufügen, auf das **+**-Icon in der Titelleiste klicken.

33. Es öffnet sich ein neues Fenster mit dem Titel **Alias Hinzufügen**.
Hier als Name z.B. **WLPDB1** spezifizieren. Im Dropdown-Menü **Treiber** den hinzugefügten JDBC-Treiber **DB2 for z/OS** auswählen. Im Feld URL das vorgegebene Template mit den entsprechenden Daten anpassen, im vorliegenden Fall wäre das **jdbc:db2://BOEWLM4:5941/WLMD941**.
Optional können auch noch weitere Angaben gemacht werden, z.B. zum Benutzernamen und Passwort.
Bevor die Verbindung gespeichert wird, sollte sie getestet werden, dazu muss auf den Button **Test** geklickt werden

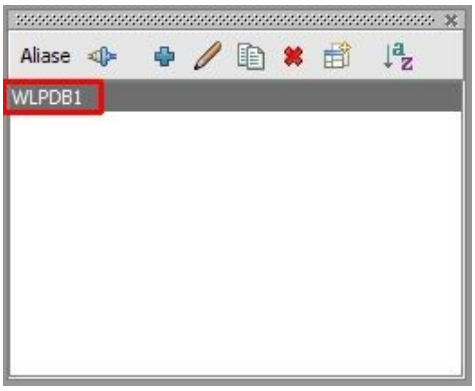
34. Es öffnet sich ein weiteres Fenster mit dem Titel **Verbinden mit: WLPDB1**. Nachdem Benutzernamen und Passwort eingegeben wurden auf den Button **Verbinden** klicken.

Bei erfolgreicher Herstellung der Verbindung wird dies mit einer entsprechenden Meldung angezeigt.

(**Zu beachten**: Wenn die Herstellung der Verbindung nicht gelingt, dann kann das erneut an den Voraussetzungen aus Kapitel 1 liegen, vor allem 1.1, 1.3 und 1.5.)

35. Nach erfolgreichem Test das Status-Fenster mit **OK** schließen und die Verbindung mit Klick auf den **OK** Button speichern.

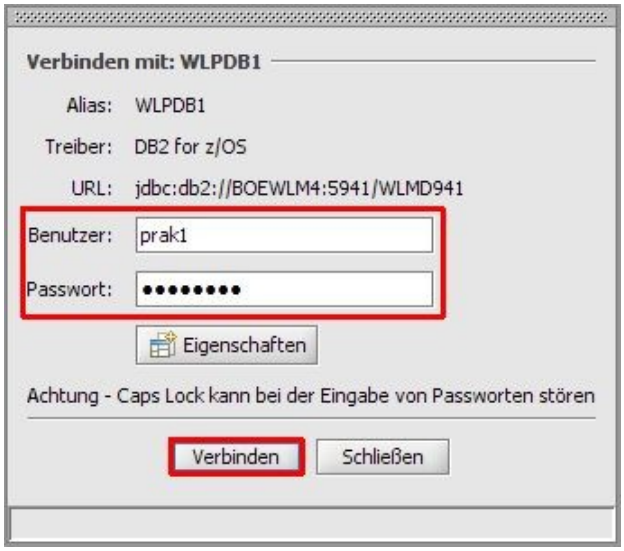
36. Die Verbindung ist nun eingerichtet und steht im Fenster **Aliase** zur Verfügung.



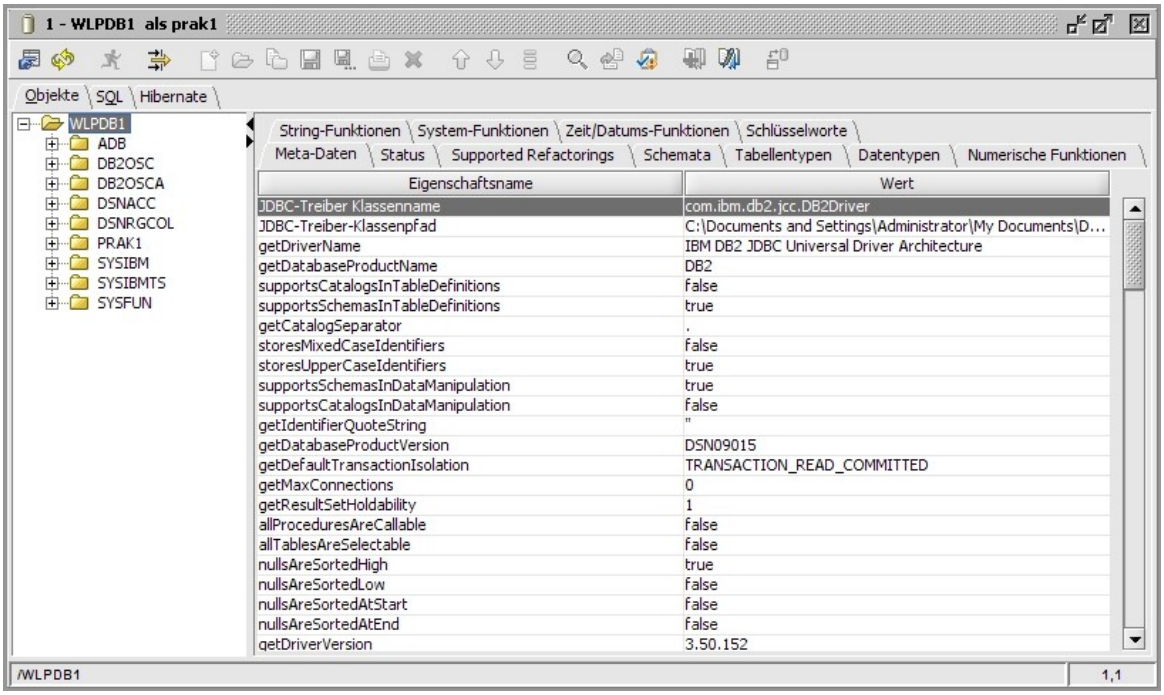
2.2.3 Herstellen der Verbindung und kurze Einführung in die Benutzung von SQuirreL

Die neu eingerichtete Verbindung ist nun im Fenster **Aliase** verfügbar und kann eingesetzt werden.

37. Im Fenster **Aliase** auf die Verbindung **WLPDB1** doppelklicken.
    Im sich öffnenden Fenster **Verbinden mit: WLPDB1** die nötigen Angaben für Benutzer und Passwort eingeben und auf den Button **Verbinden** klicken um die Verbindung herzustellen.

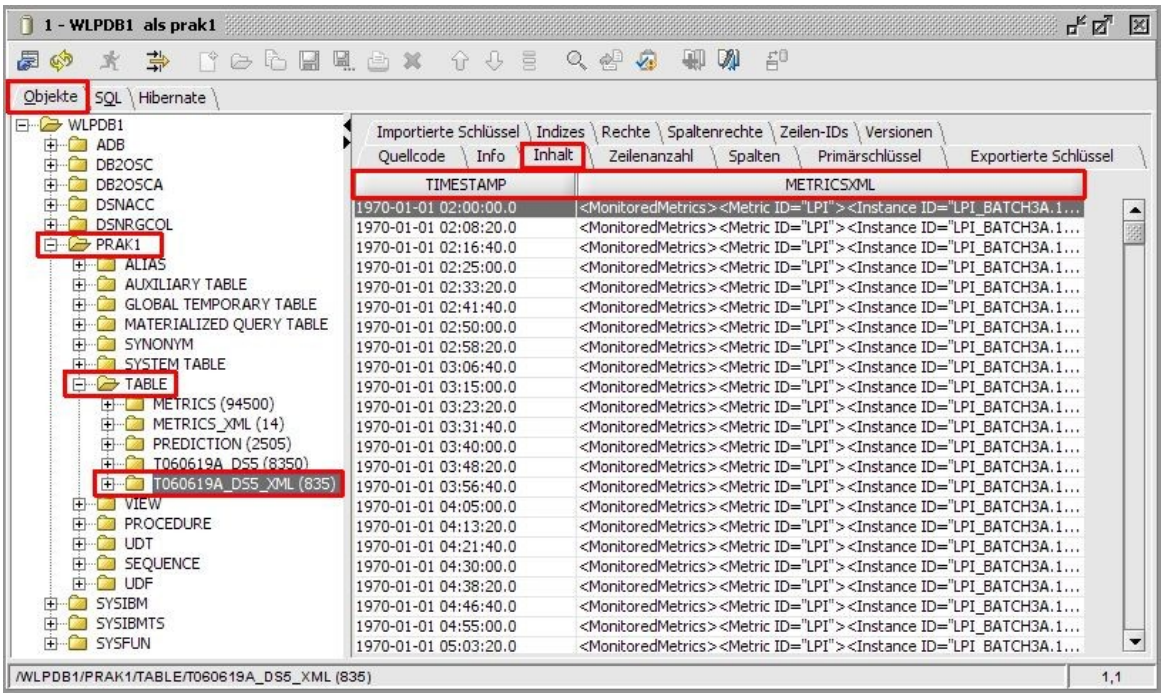


38. Es öffnet sich ein neues Fenster mit der Datenbankverbindung für den gewählten Benutzer.

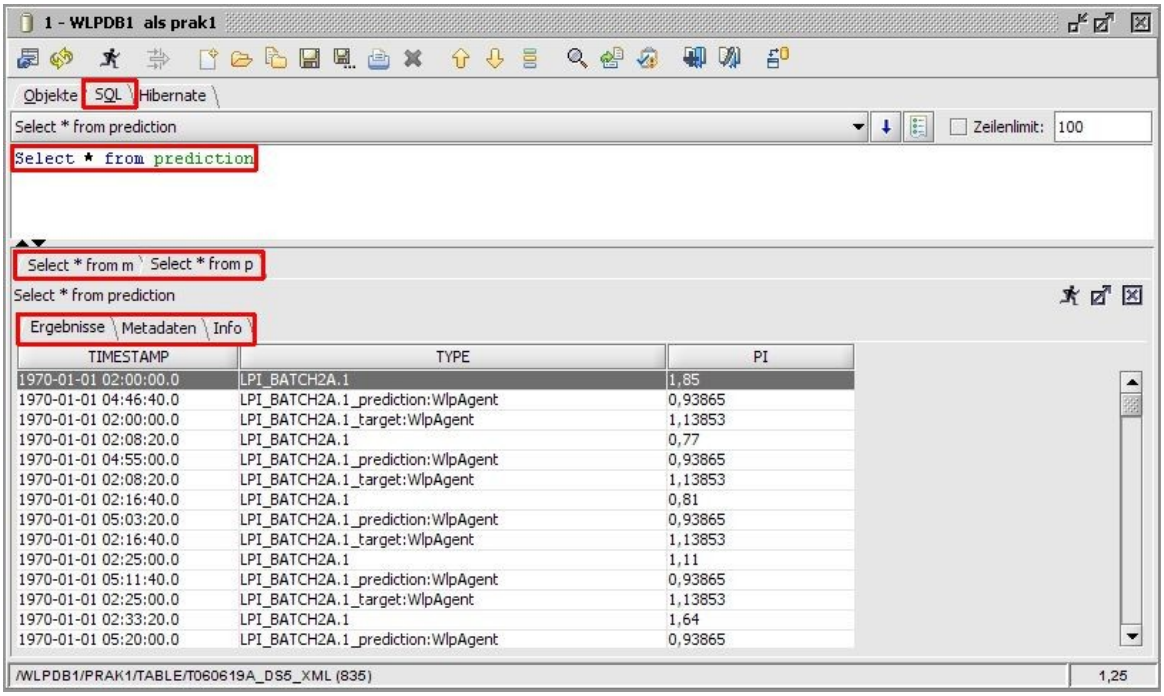


39. Mit Hilfe des Reiters **Objekte** in der linken Hälfte kann man durch die verschiedenen Datenbank-Objekte navigieren. Von Interesse ist aber nur das Verzeichnis mit dem Benutzer, im vorliegenden Fall **PRAK1**. Im Unterverzeichnis **Table** finden sich alle angelegten Tabellen.
    Wenn eine Tabelle ausgewählt wurde, hat man auf der rechten Seite mehrere Reiter

mit verschiedenen Funktionen. Über den Reiter **Inhalt** kann man sich den Inhalt der Tabelle ansehen. Es finden sich auch allgemeine Informationen und Meta-Daten in den verschiedenen Reitern auf der rechten Seite.



40. Mit Hilfe des Reiter **SQL** kann man SQL-Abfragen auf die Datenbank ausführen. Dabei kann man mit Hilfe von verschiedenen Reitern in der unteren Hälfte auch die Ergebnisse mehrerer Abfragen parallel vergleichen.
Für jede Abfrage stehen neben den Ergebnissen auch Meta-Daten und weitere Informationen zur Verfügung.

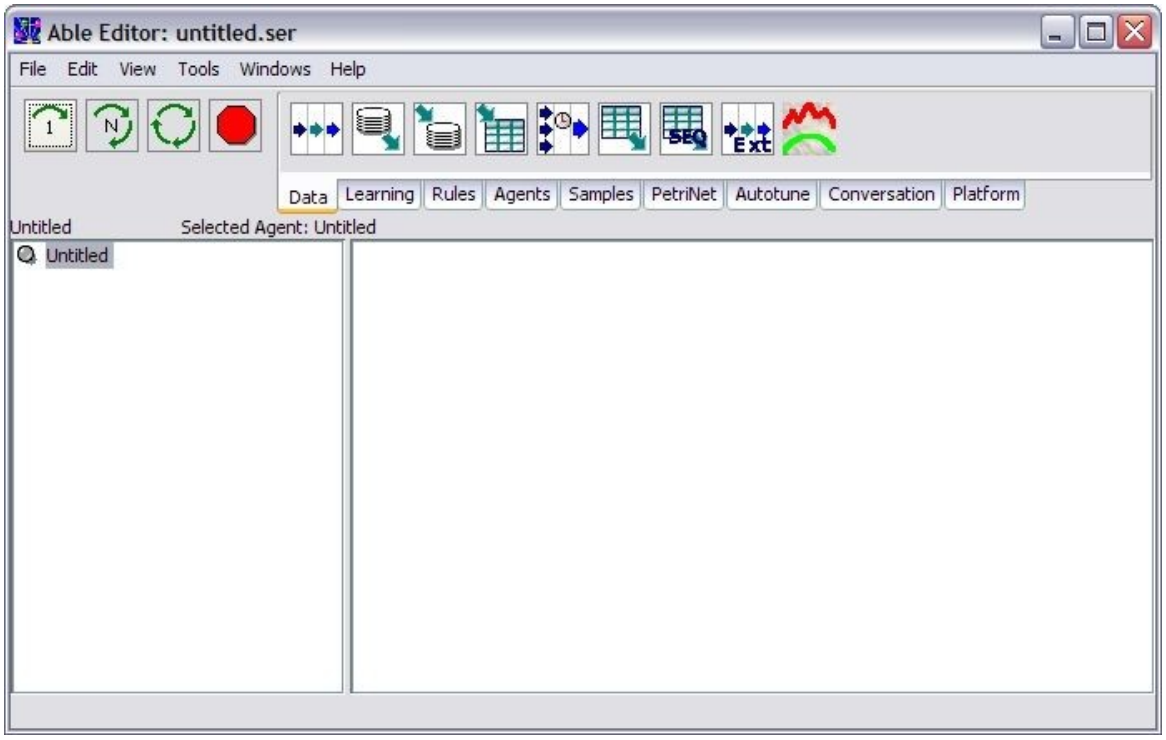

1 - WLPDB1 als prak1
Objekte
SQL
Hibernate
Select * from prediction
Zeilenlimit:
100
Select * from prediction
Select * from m
Select * from p
Select * from prediction
Ergebnisse
Metadaten
Info
TIMESTAMP
TYPE
PI
1970-01-01 02:00:00.0 LPI_BATCH2A.1 1,85
1970-01-01 04:46:40.0 LPI_BATCH2A.1_prediction:WlpAgent 0,93865
1970-01-01 02:00:00.0 LPI_BATCH2A.1_target:WlpAgent 1,13853
1970-01-01 02:08:20.0 LPI_BATCH2A.1 0,77
1970-01-01 04:55:00.0 LPI_BATCH2A.1_prediction:WlpAgent 0,93865
1970-01-01 02:08:20.0 LPI_BATCH2A.1_target:WlpAgent 1,13853
1970-01-01 02:16:40.0 LPI_BATCH2A.1 0,81
1970-01-01 05:03:20.0 LPI_BATCH2A.1_prediction:WlpAgent 0,93865
1970-01-01 02:16:40.0 LPI_BATCH2A.1_target:WlpAgent 1,13853
1970-01-01 02:25:00.0 LPI_BATCH2A.1 1,11
1970-01-01 05:11:40.0 LPI_BATCH2A.1_prediction:WlpAgent 0,93865
1970-01-01 02:25:00.0 LPI_BATCH2A.1_target:WlpAgent 1,13853
1970-01-01 02:33:20.0 LPI_BATCH2A.1 1,64
1970-01-01 05:20:00.0 LPI_BATCH2A.1_prediction:WlpAgent 0,93865
/WLPDB1/PRAK1/TABLE/T060619A_DS5_XML (835)
1,25

3 Benutzung des ABLE-Editors (Projekt Ablegui)

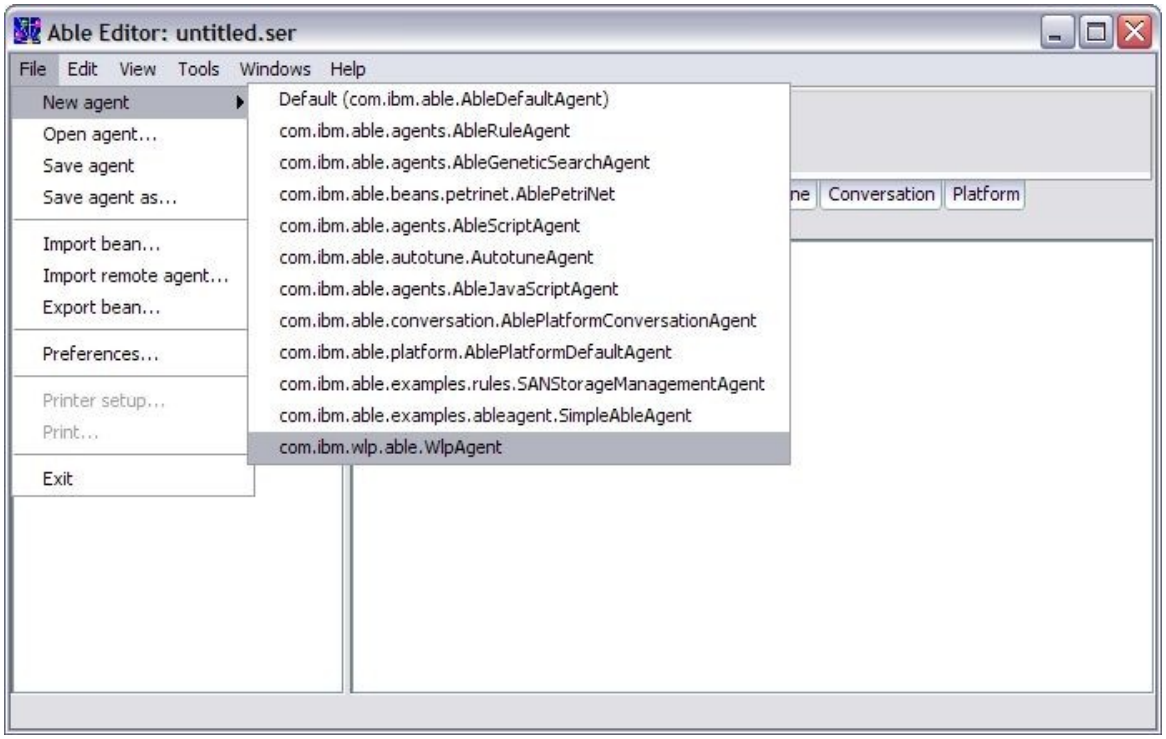
Der ABLE-Editor ist eine Anwendung mit graphischer Benutzeroberfläche, um Agenten für die Vorhersage zu erstellen, zu konfigurieren und zu Trainieren bevor man sie in der eigentlichen WLPrediction-Anwendung einsetzen kann.

3.1 WlpAgent erstellen

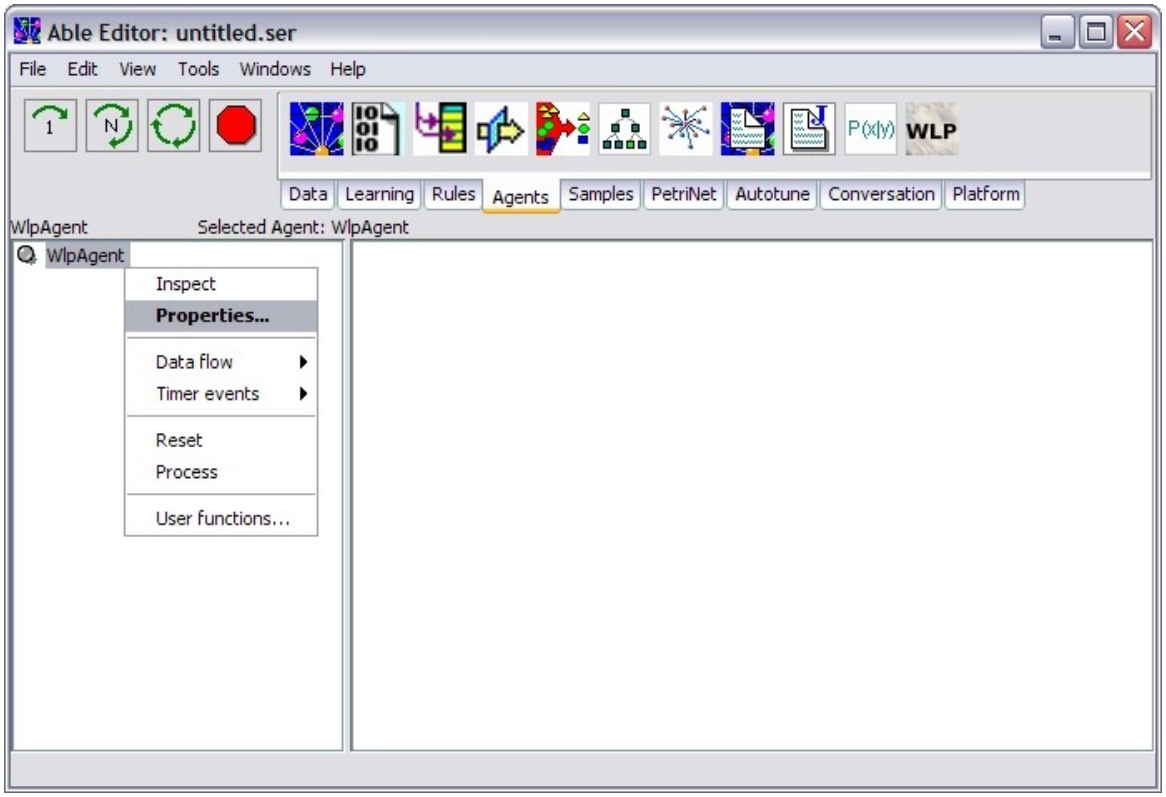
41. Der ABLE-Editor wird mit der Datei **startAbleGui.bat** im Root-Verzeichnis des WLPrediction-Projektes gestartet. Nach dem Start öffnet sich folgendes Fenster:



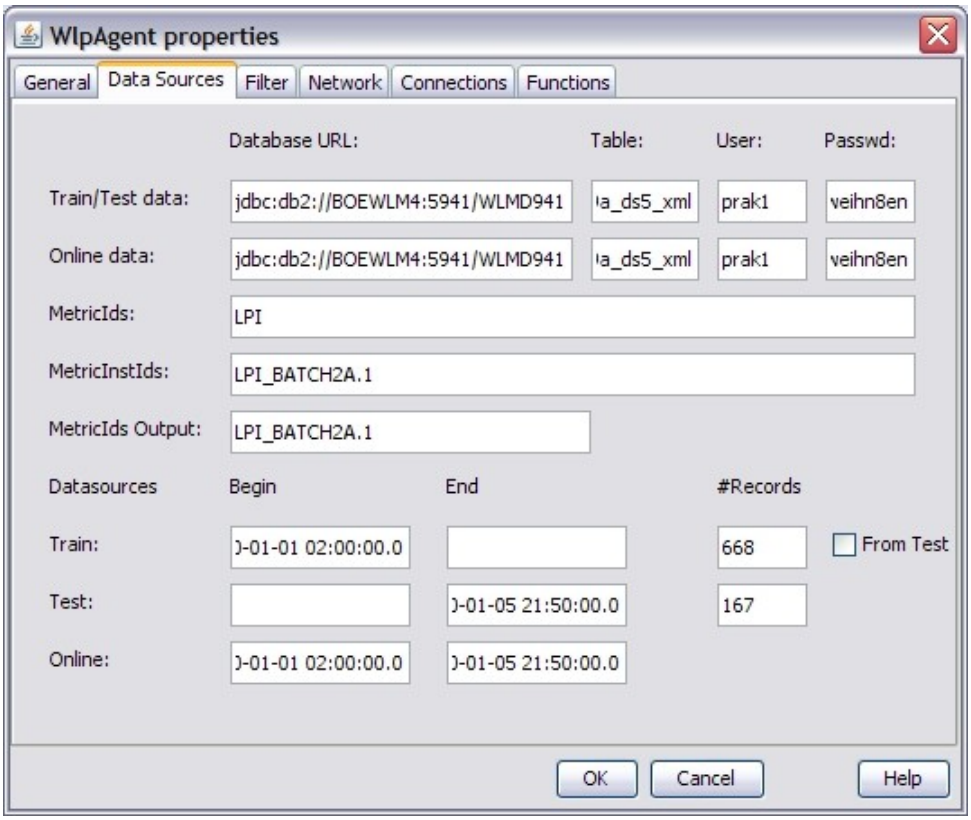
42. Für das WLP-Framework wurde ABLE um einige Beans erweitert. Aus diesen Beans wurde ein neuer Agent zusammengesetzt, der **WlpAgent**. Dieser Agent beinhaltet automatisch alle benötigten Beans. Man kann ihn durch einen Linksklick auf
File → New agent → com.ibm.wlp.able.WlpAgent
erstellen.



43. Im linken Bereich sollte nun ein **WlpAgent** vorhanden sein. Diesen muss man nun rechts-klicken und **Properties...** auswählen.



44. Es öffnet sich das Fenster **WlpAgent properties**, in welchem man den WlpAgent konfigurieren kann. Um das Erstellen des WlpAgents fertigzustellen müssen zuerst die restlichen benötigten Beans generiert werden.
Dazu wird auf den Reiter **Data Sources** gewechselt um dort sämtliche benötigten Daten zu spezifizieren.



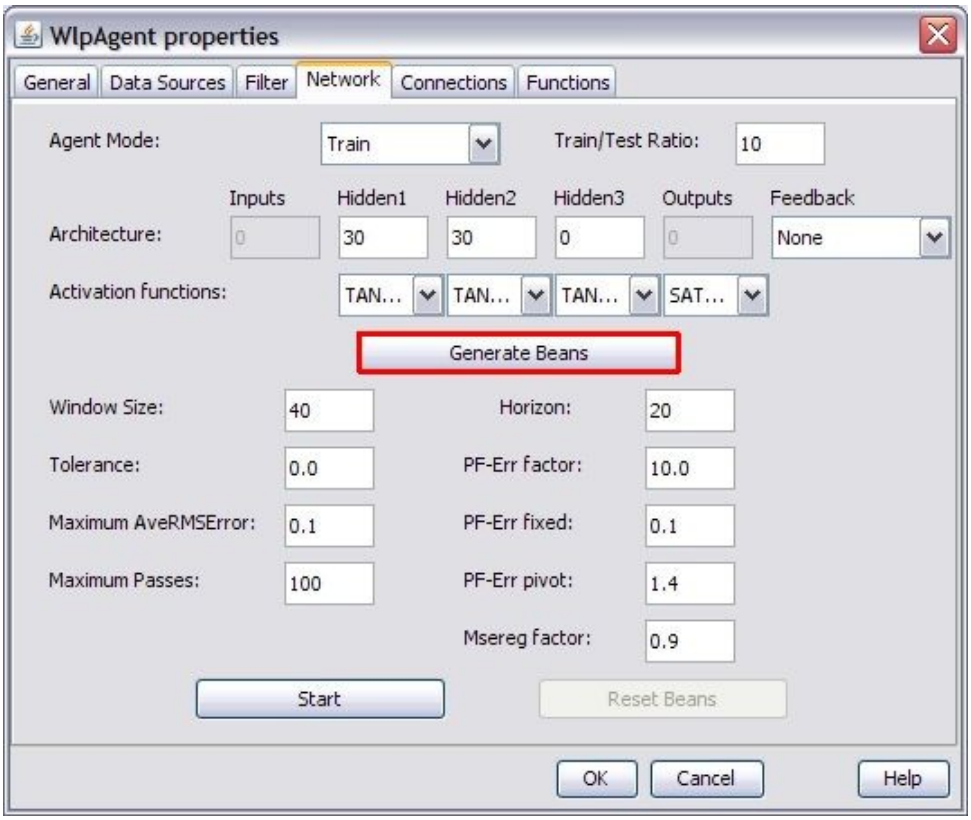
Für die verwendeten Referenzdaten von Clemens Gebhard wären das:

- Train/Test/Online Data:
  - Database URL: **jdbc:db2://BOEWLM4:5941/WLMD941**
  - Table: **t060619a_ds5_xml**
  - User: **Host-Login (prak1)**
  - Passwd: **Host-Passwort (weihn8en)**
- MetricIds: **LPI**
- MetricInstIds: **LPI_BATCH2A.1**
- MetricIds Output: **LPI_BATCH2A.1**
- Train
  - Begin: **1970-01-01 02:00:00.0**
  - End: **LEER LASSEN!**
  - #Records: **668**

- Test
  - Begin: **LEER LASSEN!**
  - End: **1970-01-05 21:50:00.0**
  - #Records: **167**
- Online
  - Begin: **1970-01-01 02:00:00.0**
  - End: **1970-01-05 21:50:00.0**
  - #Records: **LEER LASSEN!**

(**Note:** Bei den Datasources **Train**/**Test**/**Online** kann man drei Variablen angeben, aber es reichen auch jeweils zwei Stück aus, denn der WlpAgent kann aus den zwei angegebenen Werten den dritten errechnen)

45. Wenn alle Daten erfolgreich angegeben wurden, muss als nächstes auf den Reiter **Network** gewechselt werden. Sämtliche benötigten Voreinstellungen werden dem WlpAgent automatisch vorgegeben. Jetzt muss nur noch auf den Button **Generate Beans** geklickt werden, um sämtliche Beans des WlpAgents automatisch zu generieren.



(**Note:** Da beim Generieren auf die Datenbank und somit auf das Intranet zugegriffen wird, müssen die **Voraussetzungen** unter Kapitel 1 erfüllt sein!)

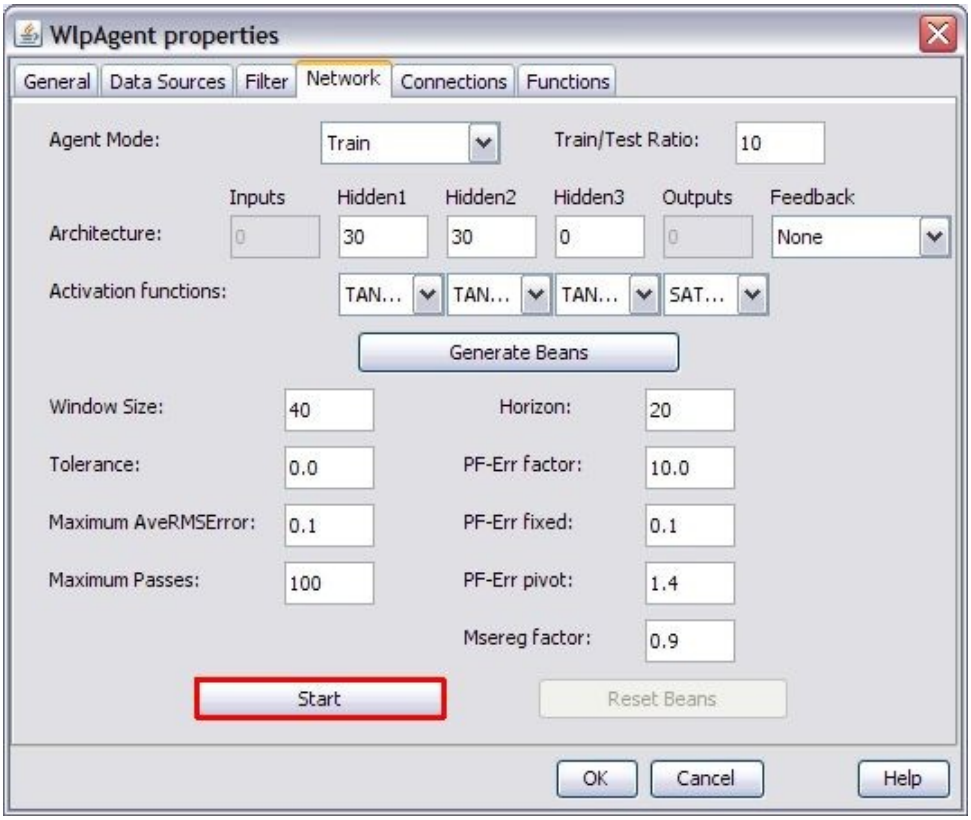
46. Mit den angegebenen Daten wurde nun ein vollständiger, aber noch **untrainierter WlpAgent** erstellt. Dieser sollte aus den folgenden Komponenten bestehen:

    - TrainingImport
    - TestImport
    - OnlineImport
    - Backpropagation
    - InFilter
    - OutFilter
    - MovingAverageFilter
    - TimeSeriesFilter

3.2 WlpAgent trainieren

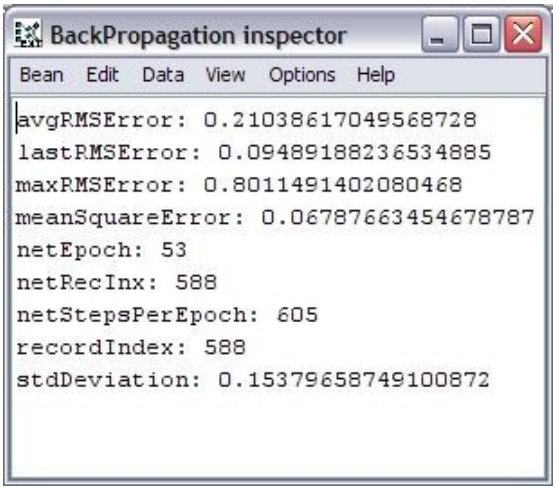
Der erstellte Agent muss nun für die Generierung von Vorhersagen trainiert werden. Auch dies geschieht mit Hilfe des ABLE-Editors.

47. Dazu wird der erstellte WlpAgent rechts-geklickt und **Properties...** ausgewählt.
    Anschließend muss erneut in den Reiter **Network** gewechselt werden.
    Wenn ein WlpAgent neu erstellt wird, ist er zu Beginn automatisch im **Train Mode**.
    Um das Trainieren nun zu starten, muss man auf den Button **Start** links unten klicken.



Der Agent läuft jetzt über den angegebenen **Trainingsdatensatz**. Das kann je nach Datensatz mehrere Minuten dauern. Mit dem Datensatz von Clemens Gebhard dauerte es knapp über 15 Minuten. Wenn der Agent brauchbare Ergebnisse liefern soll, dann sollte er einen **avgRMSError** von **0,21** oder weniger haben.

48. Der ABLE-Editor stellt für die Überwachung und Kontrolle des Trainings so genannte **Inspectors** zur Verfügung. Diese Inspektoren gibt es sowohl in Textform als auch in graphischer Form. Ein Beispiel für einen sinnvollen **Inspector** auf das **Backpropagation Bean** könnte so aussehen:

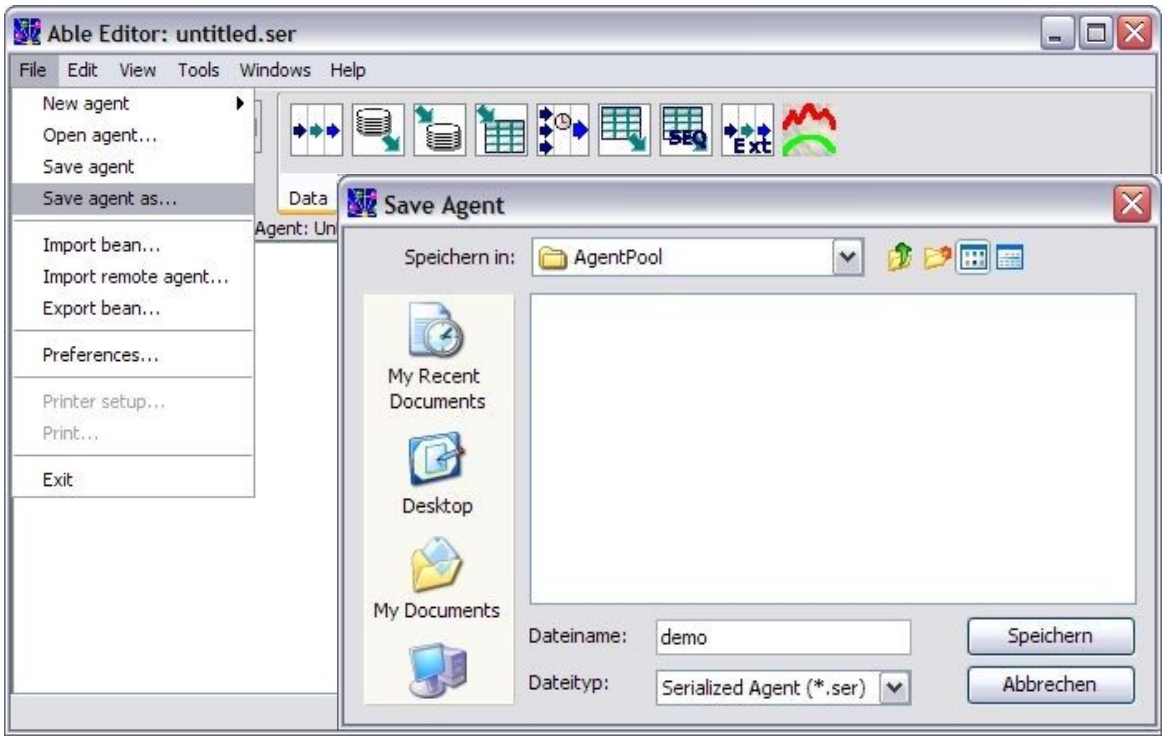


Es wird u.a. der **durchschnittliche Fehler**, die **Standardabweichung**, der **aktuell bearbeitete Datensatz** und die **aktuell bearbeitete Epoche** angezeigt.

49. Wenn der WlpAgent seine voreingestellte Anzahl von Epochen (per Default 100) abgearbeitet hat, ist das Training beendet und der Agent wird in den **Run Mode** gesetzt.
    Das noch offene Properties Fenster kann jetzt mit Klick auf **OK** geschlossen werden.

3.3 WlpAgent speichern

50. Als letztes muss der Agent noch zur weiteren Verwendung in der WLPrediction gespeichert werden.
    Dazu wählt man unter **File → Save agent as...** einen Namen und den Speicherplatz für den serialisierten Agenten aus. Als Speicherplatz bietet sich der in der Konfigurations-Datei spezifizierte Pfad zum **AgentPool** an.



Damit der WlpAgent zur Vorhersage benutzt werden kann, muss er erstens im **Run Mode** sein und zweitens muss er im **AgentPool** liegen. Beide Voraussetzungen müssen erfüllt sein!

4 Starten des RMFRecorders

Zur Ausführung des RMFRecorders müssen sämtliche Voraussetzungen aus Kapitel 1 erfüllt sein. Zusätzlich müssen die zur Verwendung festgelegten **Tabellen angelegt** sein. Der RMFRecorder wird mit dem Shell-Skript **startRMFRecorder.bat** im Root-Verzeichnis des WLPrediction-Projektes gestartet (U.U. muss man im Batch-Skript den Pfad zum Workspace und zur java.exe anpassen).

Der Aufruf in der .bat-Datei muss mit den folgenden Jar-Dateien erfolgen:

java.exe -cp
".;bin;lib/sblim/sblimCIMClient.jar;lib/db2/db2jcc.jar;lib/db2/db2jcc_license_cisuz.jar
;lib/jdom/jdom.jar;lib/jdom/jaxen-core.jar;lib/jdom/jaxen-
jdom.jar;lib/jdom/saxpath.jar" com.ibm.wlp.rmfrecorder.RMFRecorder

Man sollte darauf achten, dass alle benötigten, oben angegebenen Jar-Archive beim Aufruf angegeben werden. Beim fehlen eines Jar-Archivs wird eine entsprechende Fehlermeldung angezeigt.

Der RMFRecorder ist für das Aufzeichnen der Metrik-Daten zuständig. Dazu holt sich die Anwendung die RMF-Metriken in einem festgelegten Zeitintervall mit Hilfe eines CIM-Clients vom CIM-Server ab und speichert sie in einem XML-Dokument pro Zeitstempel in der Tabelle **srcTable** (Referenzdatensätze: **metrics**).

5 Starten der WLPrediction

Zur Ausführung der WLPrediction müssen sämtliche Voraussetzungen aus Kapitel 1 erfüllt sein (wobei Kapitel 1.2 hier keine Rolle spielt). Zusätzlich müssen die zur Verwendung festgelegten **Tabellen angelegt** sein, das **srcTable** Daten enthalten, die verarbeitet werden können und für das berechnen von Voraussagen muss der benutzte WlpAgent auch im **Run Mode** arbeiten.
Die WLPrediction wird mit dem Shell-Skript **startWLP.bat** im Root-Verzeichnis des WLPrediction-Projektes gestartet (U.U. muss man im Batch-Skript den Pfad zum Workspace und zur java.exe anpassen).

Der Aufruf in der .bat-Datei muss mit den folgenden Jar-Dateien erfolgen:

java.exe -cp ".;bin;lib/able/able.jar;lib/able/ablebeans.jar;lib/able/JLog.jar;lib/jetty/jetty-6.1.4.jar;lib/jetty/jetty-util-6.1.4.jar;lib/jetty/servlet-api-2.5-6.1.4.jar;lib/charting/jCharts-0.7.5.jar;lib/db2/db2jcc.jar;lib/db2/db2jcc_license_cisuz.jar;lib/jdom/jdom.jar;lib/jdom/jaxen-core.jar;lib/jdom/jaxen-jdom.jar;lib/jdom/saxpath.jar" com.ibm.wlp.prediction.WLPrediction

Man sollte darauf achten, dass alle benötigten, oben angegebenen Jar-Archive beim Aufruf angegeben werden. Beim fehlen eines Jar-Archivs wird eine entsprechende Fehlermeldung angezeigt.

Die WLPrediction berechnet die eigentliche Voraussage. Dazu lädt die Anwendung zunächst alle Datensätze (Train, Test und Online) aus der Tabelle **srcTable** (Referenzdatensätze: **t060619a_ds5_xml**) und lädt den gespeicherten **trainierten WlpAgent** aus dem **AgentPool**. Danach arbeitet sie jeden Eintrag des angegebenen Zeitintervalls einzeln ab und schreibt die Vorhersagewerte in die festgelegte Tabelle **destTable** (Referenzdatensätze: **prediction**) der Datenbank zurück.

6 Migration auf den Großrechner

Das WLP-Framework sollte mit **Eclipse/Rational** bearbeitet werden, da die folgenden Schritte auch darauf ausgelegt sind. Selbstverständlich kann die Entwicklung mit jedem beliebigen Werkzeug angegangen werden.

6.1 Vorgehensweise

In diesem kapitel sollen die nötigen Schritte zur Migration auf den Großrechner erklärt werden.

51. Als erstes muss das Projekt **WLPrediction** im Eclipse-Workspace neue Erstellt werden, damit alle Änderungen im Quellcode übernommen werden.
    Dazu das im Projekt-Root vorhandene Ant-Skript **build.xml** ausführen
    WICHTIG: Es sollte darauf geachtet werden, welche JRE auf dem verwendeten Großrechner installiert ist! Die installierte JRE sollte mindestens die selbe Version besitzen wie der benutzte Java-Compiler, d.h. bei einer installierten JRE 1.5 darf der Java-Compiler maximal in der Version 1.5 vorliegen, 1.6 würde nicht laufen.

52. Wenn alle Änderungen kompiliert wurden, dann kann das Verzeichnis **dist**, ein Unterverzeichnis im **WLPrediction**-Projekt, mit einem FTP-Client auf die USS des Großrechners übertragen werden. Dazu müssen zwei Punkte beachtet werden:
    1. Die Adresse für den Host muss bekannt sein (im vorliegenden Fall **boewlm4**) und man muss seinem FTP-Client mitteilen, dass das **Startverzeichnis** auf dem Host das **eigene Benutzer-Verzeichnis** ist (im vorliegenden Fall **/u/prak1**), denn nur dafür werden ausreichend Rechte besessen. Dieses Verzeichnis muss selbstverständlich auch existieren.
    2. Beim Übertragen der Dateien muss man penibel darauf achten welche Datei in welchem **Übertragungsmodus** hochgeladen wird.
    Gewöhnliche Textdateien wie HTML- und Skript-Dateien **müssen** im **ASCII-Modus** übertragen werden. Dabei wird dann automatisch eine echte Konvertierung in die von Großrechnern verwendete **EBCDIC-Zeichenkodierung** vorgenommen, und nicht nur eine Konvertierung der Zeilenumbruchsvarianten der verschiedenen Betriebssysteme.

    Im Klartext heisst das für den vorliegenden Fall:
    Alle Dateien (größtenteils sind das binäre Jar-Dateien) müssen **binär übertragen** werden ausser die Shell-Skripte **startRMFRecorder** und **startWLP**, sowie die Konfigurationsdatei **WLPDefaults.properties**.
    Die XML-Dateien **cimmetrics.xml** und **cimmetrics.xsd** müssen **explizit** (obwohl sie Textdateien sind!) im **binären Modus übertragen** werden, da sonst der RMFRecorder nicht funktioniert. Diese tatsache mag verwundern, aber das liegt daran, dass das komplette **Projekt auf ASCII kompiliert** wurde und der **RMFRecorder** beim Einlesen der **cimmetrics.xml** deswegen eine **ASCII-Datei erwartet**.
    Diese These wurde aber noch nicht durch entsprechende Tests bestätigt! Das sollte nachgeholt werden.

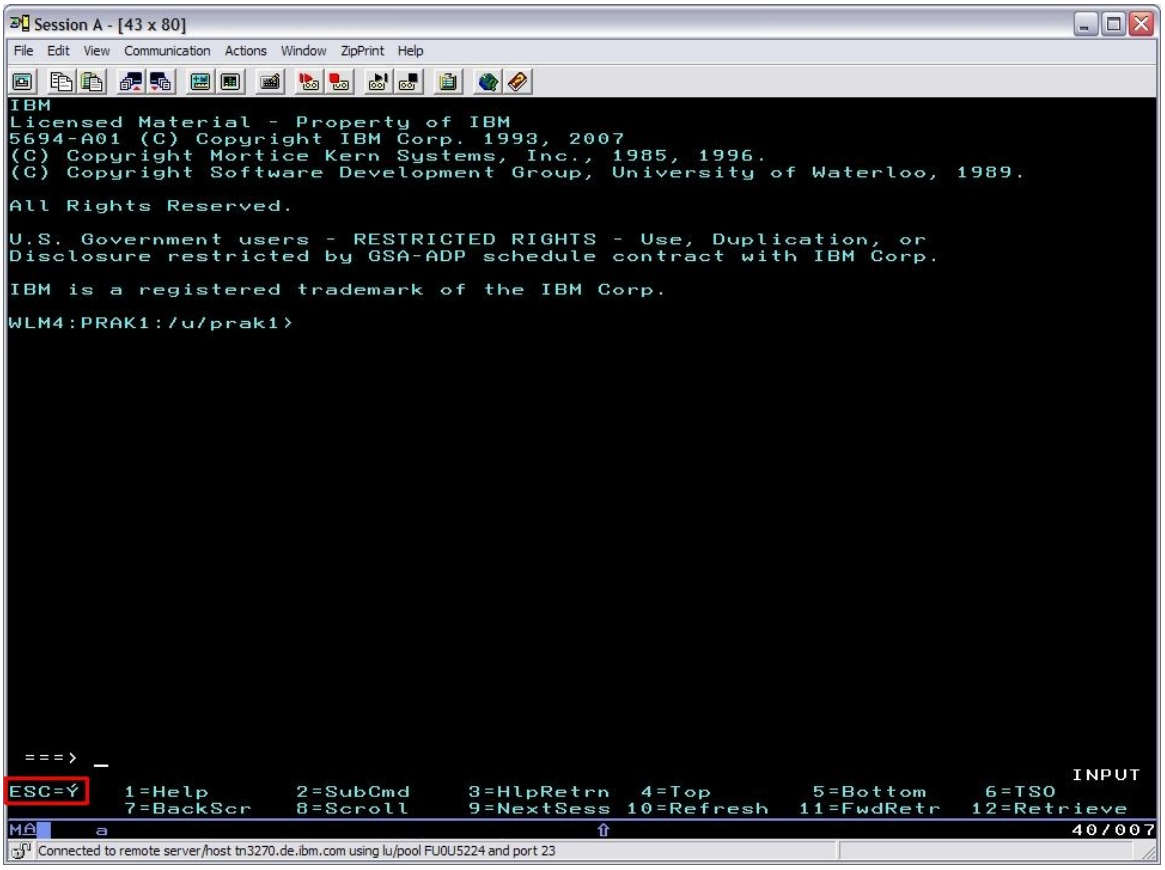
53. Wenn dann alle benötigten Dateien korrekt übertragen wurden müssen nur noch die entsprechenden Rechte zum Ausführen der Start-Skripte **startRMFRecorder** und **startWLP** vergeben werden. Das erledigt man mit dem Befehl **chmod 755 startRMFRecorder** und **chmod 755 startWLP** respektive.

54. Schließlich sind der **RMFRecorder** sowie die **WLPrediction** auf ihre Ausführung vorbereitet und können mit den entsprechenden Start-Skripten gestartet werden. Genau wie bei der Ausführung auf dem Client, muss auch hier wieder auf die Voraussetzungen aus Kapitel 1 verwiesen werden.
Wichtig: Durch ihre Architektur, laufen die Anwendungen in einer Endlosschleife, um sie zu beenden muss man das **SIGINT-Signal** abschicken, wie das auf dem Großrechner bewerkstelligt wird, wird im folgenden kapitel kurz beschrieben.

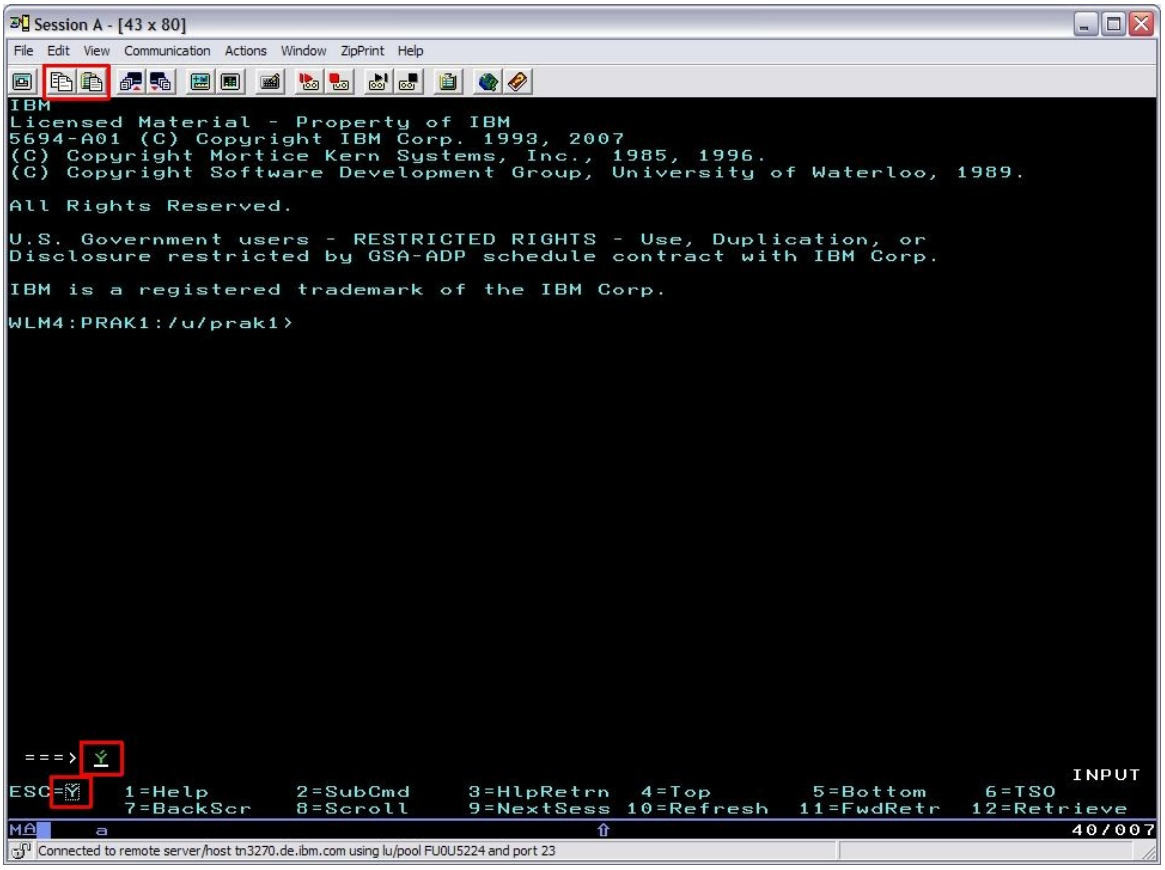
6.2 Beenden der WLP-Anwendungen auf dem Großrechner (SIGINT)

Auf dem Windows-Client wird das **SIGINT-Signal** mit der **Tastenkombination Strg + C** abgeschickt. Das wird auf dem Großrechner anderst erledigt. Eine mögliche Lösung wird im folgenen erläutert.

55. Wenn man sich in der **OpenMVS Shell** befindet, dann befindet sich links unten im Eck die Information **ESC=Ý**.

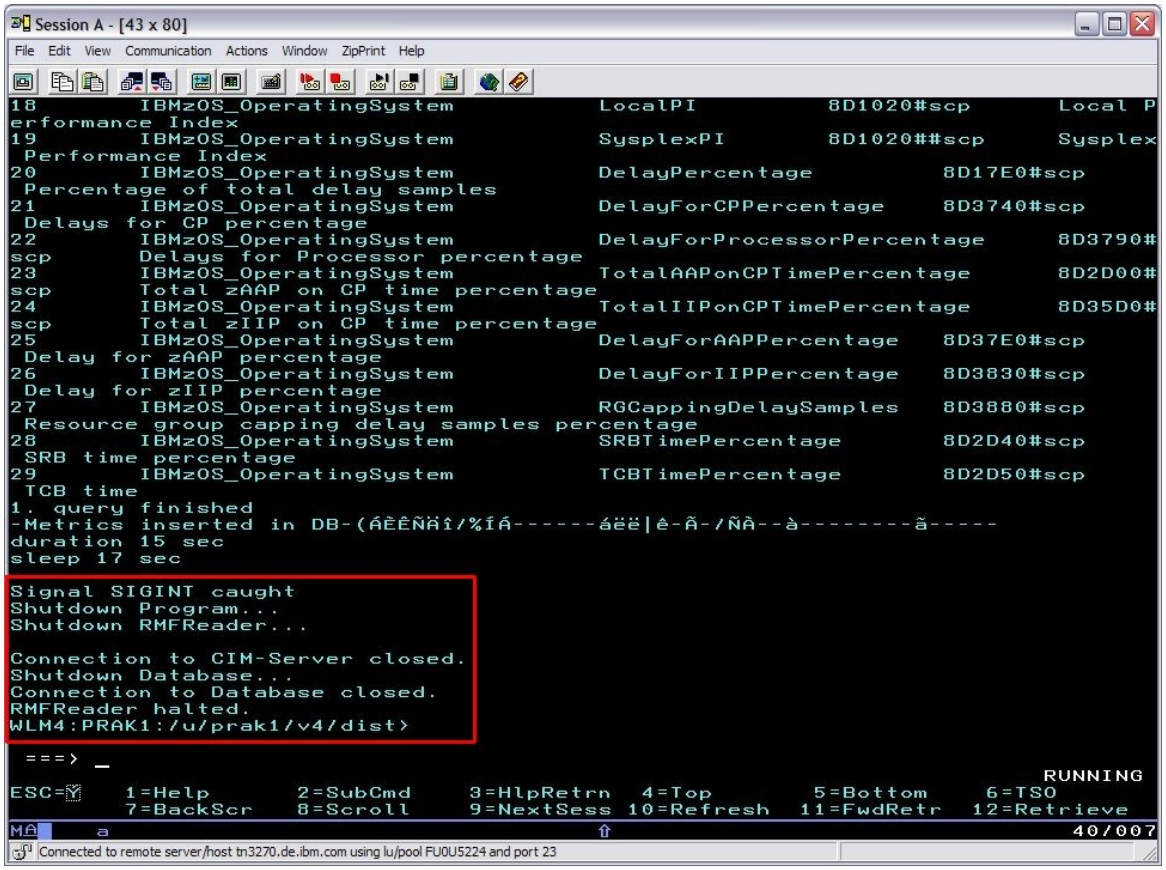


56. Da man dieses kryptische Zeichen auf der tastatur nur schwer findet, wird es mit Hilfe des Terminal Emulator einfach mit der Maus markiert und per **copy&paste** auf die Kommandozeile kopiert. Das geht wahlweise über das Menü oder die **zwei Buttons links oben**. Der **linke kopiert** und der **rechte fügt ein**.



57. Zusätzlich zu dem **kopierten Zeichen** wird auch noch ein anschließendes **c** benötigt. Zum bestätigen die **Strg**-Taste drücken.

58. Wenn es geklappt hat, dann beendet er die aktuell laufende Anwendung und zeigt den Pfad zum aktuellen Verzeichnis an.
Im Beispiel unten wurde der laufende RMFRecorder beendet.



59. Praktischerweise besitzt der hier verwendete Terminal-Emulator eine **History-Funktion**. Mit den Tasten **F11** und **F12** kann durch die schon verwendeten Befehle gescrollt werden. Somit muss man den befehl nicht jedes mal aufs Neue zusammenbauen

7 Sonstige wichtige Hinweise und Tips

Dieses Kapitel behandelt sonstige Tips, Tricks und andere Hinweise, die die Bedienung erleichtern oder Helfen Fehler zu vermeiden. Momentan ist keine Struktur vorhanden, es ist einfach nur eine Liste die möglichst kurz die Fakten auf den Punkt bringt.

- Der **ABLE-Editor** ist in Eclipse das Projekt **ablegui**
- Die Klasse **AbleDBImportBeanInfo.java** im **WLPrediction**-Projekt wurde von Hagmann aus Projekt **ablegui** importiert, nicht aus able!
- **WICHTIG:** Bei der IBM ist es üblich, dass alle paar Monate sämtliche **Passwörter geändert werden müssen**. Das kann zu ernsthaften Problemen führen. Es muss garantiert werden, dass **sämtliche Vorkommen der Benutzername/Passwort-Kombo** mit dem neuen Passwort **ersetzt werden**!
  Wenn man das Passwort mehrmals falsch eingibt, dann wird der Benutzer für den Großrechner gesperrt. Da zum Verbinden mit der Datenbank das selbe Passwort verwendet wird, kann es schnell passieren, dass man sich indirekt über die Anwendung mehrmals mit dem falschen Passwort anmeldet und in der Folge der Accout gesperrt wird!
  Am besten beim Ersetzen des Passworts die Suchfunktion von Eclipse verwenden, damit nichts vergessen wird (3x im Quellcode, 3x in WLPDefault.properties).
- Nach einer **Passwortänderung funktionieren** die **WlpAgenten** auch **nicht mehr**. Eine Änderung ist meines Wissens nicht möglich, die Agenten müssen neu erstellt werden.
- **WICHTIG:** Zum Starten des **ABLE-Editors (ablegui)**, müssen folgende Argumente als **VM arguments** in der **Run configuration** mitgegeben werden:
  -Dable.home="${workspace_loc:WLPrediction\Able_2.5.0}" -Dable.prefdir="${workspace_loc:WLPrediction}"
- **WICHTIG:** Im ABLE-Editor **muss** unter **Preferences...** das **Plugin Jar Directory** spezifiziert werden. Das ist das Verzeichnis für die selbst erstellten ABLE-Beans. Ohne diese Angabe ist es im ABLE-Editor nicht möglich einen WlpAgent zu erstellen! Im vorliegenden Fall wäre das:
  ...\Diplomarbeit\workspace\WLPrediction\Able_2.5.0\lib\plugin
- Nach jeder erfolgten Änderung im **WLPrediction**-Projekt, sollte man das Projekt neu Erstellen, um vom **ABLE-Editor** verwendete jar-Dateien (**wlpcore.jar**,...) auf den neuesten Stand zu bringen.
- Für die **graphische Anzeige** der **Vorhersage** folgende URL verwenden:
  http://localhost:8081/img/basicChart?table=prediction
  Außer dem Parameter **'table'** gibt es auch noch **'begin'**, **'end'** (beides als Timestamp), so wie **'length'** für die Anzahl anzuzeigender Datensätze.
  Note: Außer 'table' hat bei mir nichts geklappt.
- Die Daten für neu erstellte WlpAgents wurden **hart reingecoded** (MetricId, InstanceId, Timeintervall,...). Das wird in der Klasse **WlpAgentenImpl.java** erledigt.
- Um das vorhanden **WLP-Framework** das erste mal in **Eclipse einzubinden**, kann man bei der Frage nach dem Workspace einfach den **Pfad zum alten Workspace angeben**.

B. Inhalt der beigefügten DVD

Die beigefügte DVD weist folgenden Inhalt auf:

- Digitale Version der Diplomarbeit:
  Aufgeteilt in Ausarbeitung, Präsentation, Bedienungsanleitung und verwendeter Literatur
- Verwendete *Eclipse*-Software mit funktionsfähigem *WLP-Framework* im *Workspace*
- Generell verwendete Literatur und andere Diplomarbeiten im Rahmen des *Joint Research Projects*
- JDBC-Treiber für das Herstellen einer Verbindung zur *DB2 for z/OS* Datenbank
- Lauffähige Version des *SQuirreL* SQL-Clients (*Weekly Snapshot*) inklusive verwendeter SQL-Skripte
- Sonstige zur Verwendung heruntergeladene Software

Die komplette Verzeichnisstruktur findet sich zusätzlich in Abbildung B.1.

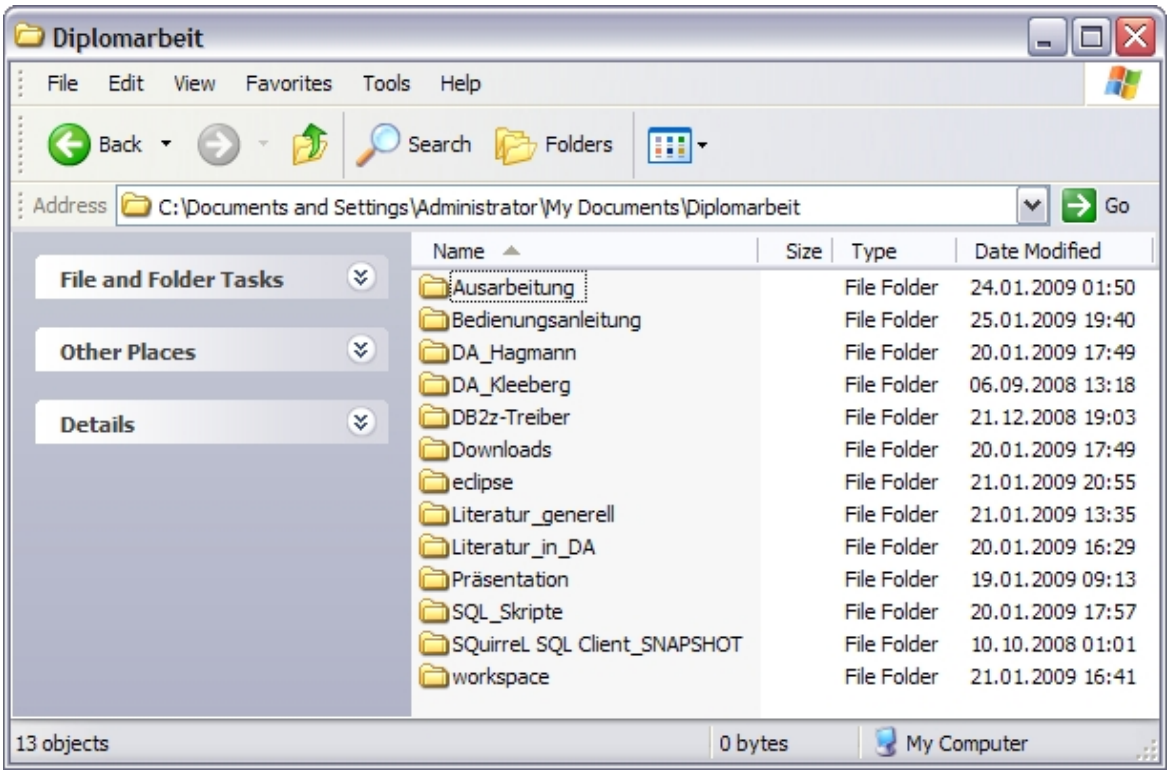


Abbildung B.1.: Inhalt der beigefügten DVD

Literaturverzeichnis

[Apa08] Apache DB Project. Apache Derby. Webseite, Dezember 2008. http://db.apache.org/derby.

[Cod08] Codehaus. What is jaxen? Webseite, Dezember 2008. http://jaxen.codehaus.org/faq.html.

[Dis08a] Distributed Management Task Force (DMTF). Common Information Model (CIM) Standards. Webseite, Dezember 2008. http://www.dmtf.org/standards/cim/.

[Dis08b] Distributed Management Task Force (DMTF). WBEM: CIM-XML protocol specification. Webseite, Dezember 2008. http://www.dmtf.org/standards/wbem/CIM-XML.

[Geb06] Clemens Gebhard. Datengewinnung und Parameterbestimmung zur Lastvorhersage in z/OS. Master's thesis, Eberhard Karls Universität Tübingen, October 2006.

[Hag07] Michael Hagmann. Architektur und Integration der Workload-Vorhersage auf Basis neuronaler Netze in z/OS. Master's thesis, Eberhard Karls Universität Tübingen, October 2007.

[Har08] Elliotte Rusty Harold. XOM FAQ: "What's the difference between XOM and JDOM?". Webseite, Dezember 2008. http://xom.nu/faq.xhtml#d0e24.

[IBM06] IBM Press Release. IBM Transforms Database Market With Introduction of DB2. Webseite, Juni 2006. http://www-03.ibm.com/press/us/en/pressrelease/19781.wss.

[IBM07] IBM. DB2 9 pureXML Guide. Redbook sg247315, Januar 2007. http://www.redbooks.ibm.com/abstracts/sg247315.html.

[IBM08a] IBM. Agent Building and Learning Environment (ABLE). Webseite, Dezember 2008. http://www.alphaworks.ibm.com/tech/able.

[IBM08b] IBM. DB2 Homepage. Webseite, Dezember 2008. http://www-01.ibm.com/software/data/db2/.

[IBM08c] IBM. IBM Data Studio Developer. Webseite, Dezember 2008. http://www-142.ibm.com/software/dre/ecatalog/detail.wss?locale=de_DE&synkey=R491071Z63067U48.

[IBM08d] IBM. z/OS Capacity Provisioning. Webseite, Dezember 2008. http://www-03.ibm.com/servers/eserver/zseries/zos/wlm/release_history/cp/index.html.

[JDO08] JDOM Project. JDOM Project Mission. Webseite, Dezember 2008. http://www.jdom.org/mission/index.html.

[Kle06] Sarah Dorothea Kleeberg. Neuronale Netze und Maschinelles Lernen zur Lastvorhersage in z/OS. Master's thesis, Eberhard Karls Universität Tübingen, August 2006.

[Kry08] Krysalis Community Project. JCharts. Webseite, Dezember 2008. http://jcharts.sourceforge.net.

[Met08] MetaStuff, Ltd. dom4j FAQ: "How does dom4j relate to JDOM?". Webseite, Dezember 2008. http://www.dom4j.org/dom4j-1.6.1/faq.html#whats-dom4j-v-jdom.

[Mor08] Mort Bay Consulting. Jetty. Webseite, Dezember 2008. http://www.mortbay.org/jetty.

[Ope08] Open Group. OpenPegasus: „C++ CIM/WBEM Manageability Services Broker". Webseite, Dezember 2008. http://www.openpegasus.org.

[Seu08] Holger Seubert. Umsetzen einer Persistenzstrategie mit Data Studio Developer & pureQuery. PDF, Dezember 2008. http://www-05.ibm.com/de/events/data-studio/pdf/Data-Studio-Developer-1_2.pdf.

[Sta08] Standards Based Linux Instrumentation for Manageability (SBLIM). CIM Client for Java. Webseite, Dezember 2008. http://sblim.wiki.sourceforge.net/CimClient.

[Ull08a] Christian Ullenboom. Galileo Computing: Java ist auch eine Insel. Webseite, Dezember 2008. http://openbook.galileocomputing.de/javainsel7.

[Ull08b] Christian Ullenboom. Galileo Computing: Java ist auch eine Insel, Kapitel 14.3: Die Java-APIs für XML. Webseite, Dezember 2008. http://openbook.galileocomputing.de/javainsel7/javainsel_14_003.htm#mjd62c1928c73c88bad15dc7a76a8c53bc.

[Wan06] Jim Wankowski. Quest Software: Breaking DB2 Platform Barriers. Webseite, Oktober 2006. http://www.quest.com/documents/landing.aspx?id=1079&technology=2&prod=&prodfamily=&loc=.

[WG08] Gerd Wagner and Glenn Griffin. SQuirreL, ein universeller SQLClient. PDF, Dezember 2008. http://www.squirrelsql.org/paper/SQuirreL_de.pdf.

[Wik08] Wikipedia. DB2 History. Webseite, Dezember 2008. http://en.wikipedia.org/wiki/IBM_DB2#History.

[Wor08a] World Wide Web Consortium (W3C). Document Object Model (DOM). Webseite, Dezember 2008. http://www.w3.org/DOM.

[Wor08b] World Wide Web Consortium (W3C). DOM-Baum für Beispiel-Tabelle. Webseite, Dezember 2008. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/introduction.html#ID-E7C30821.

[Wor08c] World Wide Web Consortium (W3C). XML Path Language (XPath). Webseite, Dezember 2008. http://www.w3.org/TR/xpath.