Für Querdenker – Was ODI anders macht als der OWB und umgekehrt

Bianca Stolz, Negib Marhoul
ORACLE Deutschland B.V. & Co. KG
Berlin/Potsdam

Schlüsselworte
Data Warehouse, ETL / ELT, ETL Tools, DWH-Architektur, DWH Design

Einleitung
Schon vor einiger Zeit hat Oracle angekündigt, den Oracle Warehouse Builder (OWB) mit dem Oracle Data Integrator (ODI) in einem nächsten Release zu einem einzigen Werkzeug zusammenzuführen. Bezogen auf Entwicklung und Administration eines Data Warehouse treffen hier zwei Welten aufeinander, wie die dazu notwendigen Ladeprozesse entworfen und aufgesetzt werden. Viele Funktionen, die in dem einen Werkzeug zu finden sind, gibt es auch in dem anderen – nur möglicherweise unter einem anderen Namen oder in einer anderen Form der Implementation. Anhand eines Kundenbeispiels werden im Folgenden OWB und ODI einander im Kontext des DWH Designs gegenübergestellt.

DWH-Architekturen und ihre Umsetzung
Sich bei Design und Betrieb eines Enterprise Data Warehouses (EDWH, kurz: DWH) an einer DWH-Architektur zu orientieren ist hilfreich dabei, das DWH analog zu den Schwerpunkten der darin ablaufenden Operationen zu strukturieren. Obgleich eine Referenzarchitektur nicht den Anspruch einer Aufbauanleitung hat, kann besonders eine Trennung in eine Datenmanagement- und Datenzugriffsschicht die Systemübersicht verbessern und die Entwicklung eines zuverlässigen DWH-Systems fördern.

Das 3-Schichten-Referenzmodell hat sich in den verschiedensten Anwendungsfällen bewährt. Dabei lassen sich sowohl die Abstraktionsebene der Schichten als auch die Komplexität der abzubildenen DWH-Prozesse gut in die Praxis bei der Entwicklung übertragen. Das unten dargestellte Beispiel konnte alle System-, Betriebs- und Entwicklungsanforderungen mit dem von Oracle empfohlenen 3-Schichten-Modell erfüllen. Beim Prozess-neutralen Foundation Layer wurde hierbei konkret ein Operational Data Store mit hoch detailierten Daten implementiert, die in der 3. Normalform integriert und harmonisiert sind. Dedizierte Endanwender können über einen definierten Zugang auf diese Ebene die aktuellsten Informationen konsolidiert abrufen. Für die Historisierung der Daten wurde eine sog. Galaxy Schicht implementiert, um mit einem stark denormalisierten Schema eine zentrale Zugriffsschicht zur Verfügung zu stellen. Hierbei wird das zugrunde liegende Dimensionsmodell (Conformed Dimensions) als Referenz für die Datamarts und Business Layer herangezogen (Access and Performance Layer).

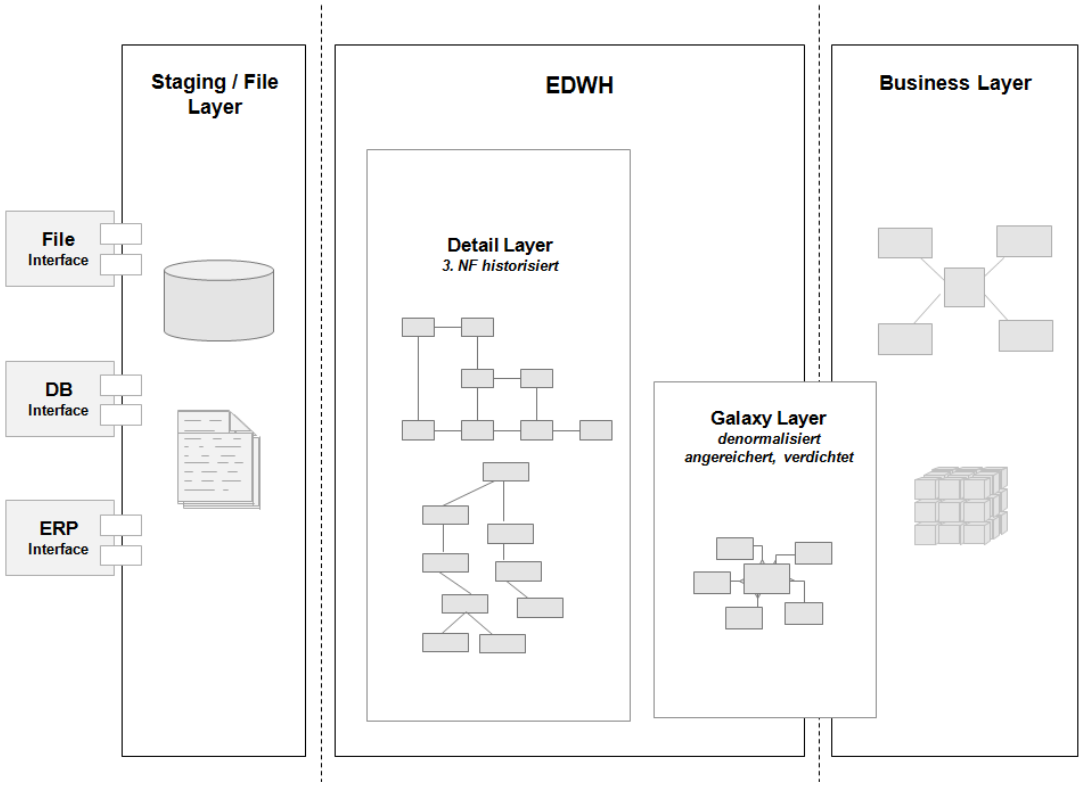

Abb. 1: DWH-Architektur eines Kundenbeispiels

Aufbau des Staging Layer

In dem angeführten Kundenbeispiel wurde in der ersten Phase des Projektes ein Staging Layer mit File Transfer Layer implementiert. Dabei bildete der ELT-Ansatz als Grundlage. Das bedeutet,, zuerst aus den Vorsystemen zu extrahieren und die Daten in das Zielsystemzu laden; alle Transformationen werden nachgelagert ausgeführt. Damit sind die operativen Systeme keiner zusätzlichen Last ausgesetzt.

Für verschiedene spezielle Funktionen im Ladeprozess stehen diverse ODI Load Knowledge Modules (LKM) zur Verfügung. Zum Beispiel kann beim Laden von großen Dateien das „File to Oracle (EXTERNAL TABLE)" Knowledge Module verwendet werden. Dieses Knowledge Module nutzt die der Oracle Datenbank bekannten Funktionen. Über ein angebundenes Filesystem werden große komprimierte Dateien automatisiert geladen. Die Konfiguration des Knowledge Module legt fest, welcher Character Set und welches Dekomprimierungswerkzeug genutzt werden soll.

Es können auch automatisierte FTP-Prozesse implementiert werden, die große Dateien vor dem Verschieben komprimieren und im Zielsystem wieder dekomprimieren. Das hat den Vorteil, dass sich die Übertragungsgeschwindigkeit deutlich erhöht . Allerdings ist hier zu beachten, dass durch die Komprimierung und Dekomprimierung im Vorfeld eine hohe CPU-Last entsteht, was in der Ressourcenbetrachtung berücksichtigt werden sollte.

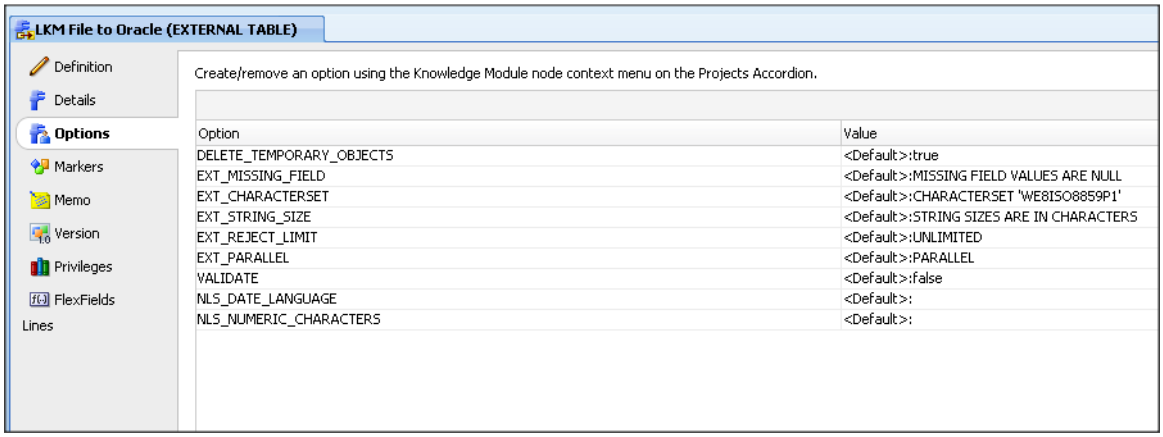

Abb. 2: Konfiguration des LKM „File to Oracle (EXTERNAL TABLE)“

Datenintegrität beim Laden

Im Falle des besagten Kundenbeispiels werden alle notwendigen Daten und Dateien revisionssicher über einen Zeitraum von ca. 4 Wochen im Staging Layer gespeichert. Damit lassen sich im Fehlerfall wiederholende Extraktions- und Ladevorgänge vermeiden, da alle Daten einfach erneut aus dem Staging Layer gezogen werden. Darüberhinaus können durch eine 1:1-Kopie der selektierten Quelldaten im Staging Layer Datenfehler bis ins Quellsystem zurückverfolgt werden (Profiling im Staging Layer), ohne die Quellsysteme selbst einer meist System- und Zeitaufwändigen Fehleranalyse zu unterziehen. Referenzielle Integrität sowie Vollständigkeit und Korrektheit der ausgewählten Daten lassen sich ebenfalls in dieser Phase des Projektes mit zentral definierten Business Rules untersuchen („Qualitygate“). Danach werden die Daten in eine Prozess-neutrale Schicht (Foundation Layer) transportiert.

Beim Einfügen der Daten ist es von Bedeutung, die Integrität aufrechtzuerhalten. Beispielsweise soll jeder Kunde eindeutig identifiziert werden können, auch wenn Kundennamen teils gleich sind. Mit Schlüsseln im Kontext eines ETL-Laufs zu arbeiten hat zwei Seiten. Einerseits ist ein Ladevorgang schneller abgeschlossen, wenn ein Schlüssel nicht validiert wird. Andererseits garantiert eine Prüfung des Schlüssels beim Hinzufügen von Daten die Integrität. Im OWB gibt es die Möglichkeit, die DML Error Logging Tables zu nutzen. Diese können im OWB 11.2 in den Eigenschaften des Tabellen- , (Materialized) View-, Dimensions- oder Cube-Operator definiert werden. Automatisch wird beim ersten Ausführen des Mappings über dieProzedur DBMS_ERRLOG.CREATE_ERROR_LOG die definierte Fehlertabelle angelegt. Dieser Mechanismus funktioniert seit OWB 10.2 sowohl bei der Row-based- als auch bei der Set-based-Verarbeitung. Wenn bei INSERT, UPDATE, MERGE, DELETE oder Multi-Table-Insert ein Fehler auftritt (wie eine Constraint-Verletzung durch doppelte Schlüssel), wird die Row ID des entsprechenden Datensatzes zusammen mit Fehlernummer, -meldung und weiteren Informationen in die Fehlertabelle geschrieben.

Sogenannte Check Knowledge Modules (CKM) unterstützen beim Einsatz mit dem ODI eine effiziente Datenqualitätsanalyse. Mit dem ODI ist es hierbei möglich, Qualitätsanalysen (Profiling) im Ladefluss (Flow Control über Check Constraints) oder auf bereits geladenen Datentabellen durchzuführen (Static Control). Durch die systemnahe Implementierung der Knowledge Modules wird das Profiling satzbasierend und parallel auf dem Staging-System ausgeführt. Bei Referenzfehlern oder allgemeinen Regelabweichungen wie bei Zeitzonen, Währungsfaktoren, Messdatentoleranzen usw. erzeugt ODI vorab automatisch Fehlertabellen und speichert sie zusammen mit den wichtigsten Kenndaten ab.

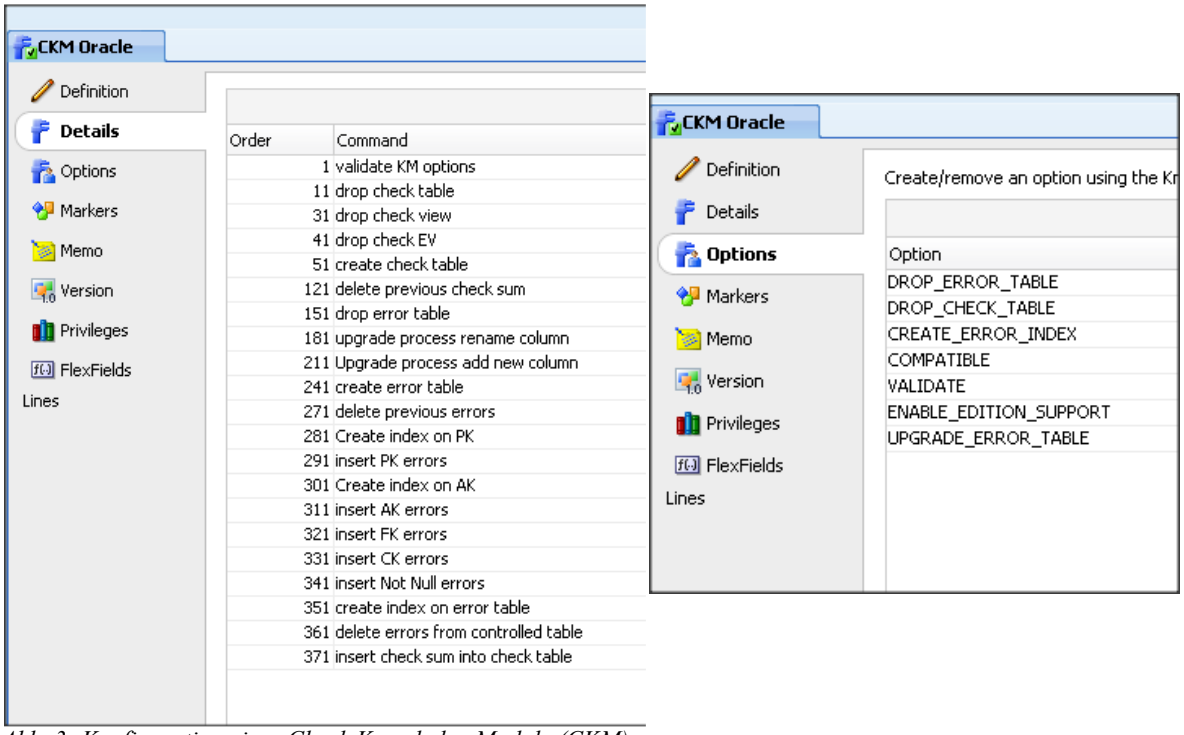

Abb. 3: Konfiguration eines Check Knowledge Module (CKM)

Wohin beim Design der Interfaces die ODI Staging Area und der Ausführungsort von Mappings, Joins und Filtern gelegt wurden, bestimmt die Art Control-Methode, die angewendet werden kann (Static oder Flow Control). Anzumerken ist, dass die „Staging Area“ im ODI einen eigenen Datenbereich bezeichnet, der entweder auf dem Quellsystem, dem Zielsystem oder in der Sunopsis Memory Engine liegen kann. Sie ist nicht unbedingt mit der bewusst gestalteten Staging Area im 3-Schichten-Referenzmodell gleichzusetzen. Werden Daten zwischen zwei Servern transferiert, ist immer ein Multi-Tech Knowledge Module erforderlich. Werden die Daten auf demselben System verarbeitet, werden dagegen Single-Tech Knowledge Modules eingesetzt. Ein Multi-Tech Knowledge Module verbindet zwei unterschiedliche Technologien miteinander, z.B. wenn ein File in eine Tabelle eingelesen wird (zwei verschiedenartige „Datenliefertechnologien“) oder eine Tabelle aus DB2 mit einer Tabelle in Oracle verknüpft werden soll. Multi-Tech Knowledge Modules lassen nur die Static Control-Methode zu. Das bedeutet, dass die Daten beim Einfügen in die Zieltabelle bereits stimmig sein müssen, da erst nach Abschluss des Interfaces die Überprüfung auf Integrität abläuft. Flow Control kann dagegen nur in Verbindung mit einem Single-Tech Knowledge Module zum Einsatz kommen, was de facto bedingt, dass sich die ODI Staging Area auf dem jeweiligen Zielssystem befindet. Dabei wird mit Hilfe einer internen Tabelle die Integrität der Daten geprüft, bevor sie dann in die eigentliche Zieltabelle kopiert werden. Dabei gilt auch die Beschränkung, dass die ODI Staging Area auf einem (relationalen) Datenbanksystem liegt, d.h. es sich dabei nicht um ein File oder einen Message Bus handelt.

Static Control kann auf einer oder mehreren Tabellen definiert werden (z.B. auf einem kompletten Data Store), Flow Control dagegen nur innerhalb eines Interfaces. Innerhalb eines Packages ist auch die Kombination beider Control-Methoden möglich, z.B. Static Control auf der Datenquelle und Flow Control auf dem Zielsystem. Geprüft wird auf Constraints, aber auch auf selbstdefinierte Bedingungen. So lässt sich die Integrität einerseits direkt auf der Datenebene und andererseits auf

einer logischen Ebene sicherstellen, wenn z.B. die im Interface definierten Bedingungen Business Rules abbilden.

Temporäre Tabellen bei der internen Verarbeitung
Zur Ausführung eines Mappings legt der OWB je nach verwendeten Operatoren interne Tabellen im Repository an. Diese Tabellen werden jedoch selten direkt vom Nutzer angesprochen. Für das Erfassen von DML-Fehlern muss explizit eine Fehlertabelle definiert werden (s.u.), in der die Datensätze erfasst werden, die nicht konform mit einem bestehendem Constraint sind. Etwaige andere Fehler während einer Ausführung werden nicht in diesen Tabellen, sondern separat in den Runtime Audit Tables (Präfix ALL_RT_) erfasst.

Im Unterschied zum OWB legt der ODI seine temporären, intern genutzte Tabellen in einer frei wählbaren Lokation an. Die internen Tabellen sind leicht zu identifizieren, da sie sich aus einem Default-Präfix und dem Namen der jeweiligen Zieltabelle zusammensetzen. Optional werden sie während der Ladephase erzeugt und automatisch wieder gelöscht. Diese temporären Integrationstabellen speichern die Zwischenergebnisse während der Transformation. Damit wird die Transformation der Daten unabhängig vom eigentlichen Zielsystem. Somit kann in der Flow Definition des ODI Interfaces die Transformationstechnologie (über die ODI Staging Area) frei gewählt werden.

Je nach Lage der ODI Staging Area und der verwendeten Knowledge Modules ist die Anzahl der temporären Tabellen unterschiedlich, da die Kombination entscheidet, ob eine interne Tabelle zur weiteren Verarbeitung nötig ist oder nicht. Beim Laden und Transformieren der Daten werden am häufigsten die Knowledge Module aus den Bereichen Laden, Integration und Prüfen angewendet. Dabei kann der Nutzer den Ort selbst bestimmen, an dem die internen Tabellen angelegt werden, und auch, ob diese Tabellen nach der Ausführung wieder gelöscht werden sollen. Die bereits erwähnten Load Knowledge Modules (LKM) transportieren die Daten auf das System, wo sich die ODI Staging Area befindet. Dabei entsteht eine 1:1-Kopie der Daten in temporären Tabellen (Default-Präfix C$_). Integration Knowledge Modules (IKM) führen die definierten Verknüpfungen, Filter und weitere Transformationen auf den geladenen Daten aus und schreiben die resultierenden Daten entweder zur weiteren Verarbeitung in eine interne Tabelle (Default-Präfix I$_) oder in die Zieltabelle. Check Knowledge Modules (CKM) legen Fehlertabellen an, sofern die Daten auf Constraints oder andere Bedingungen geprüft werden sollen. Während für jede Zieltabelle eine eigene Fehlertabelle angelegt wird (Default-Präfix E$_), gibt die SNP_CHECK_TAB Fehlertabelle das Gesamtbild über alle aufgerufenen Packages einer Session wieder. In den E$_ Fehlertabellen sind alle einzelnen Datensätze samt Row ID und Fehlermeldung zu finden, die einen Contraint oder eine selbst definierte Bedingung verletzen. In der SNP_CHECK_TAB stehen zusätzlich die dazugehörigen Schemata der Datensätze in den E$_ Fehlertabellen. Die erzeugten Fehlertabellen werden in das Work Schema des Physical Schemas der ODI Topology angelegt. Diese Tabellen können für weitere Zwecke wie Datenanalyse und Fehlerkorrektur herangezogen werden.

Im Kundenbeispiel wurden für die verschiedenen Phasen des Projektes bzw. Schichten/Layer des DWH ein eigener Data Server mit Verbindungsparametern und User zum Datenbanksystem definiert. In jedem dieser Data Server werden die einzelnen Physical Schemas (Quelldaten, Data Stores) definiert. Neben dem eigentlichen Datenschema in Form dieses Physical Schemas wird ein Work Schema definiert. Im Work Schema werden alle Laufzeitinformationen zu DML und Fehlern in den oben genannten Tabellen hinterlegt.

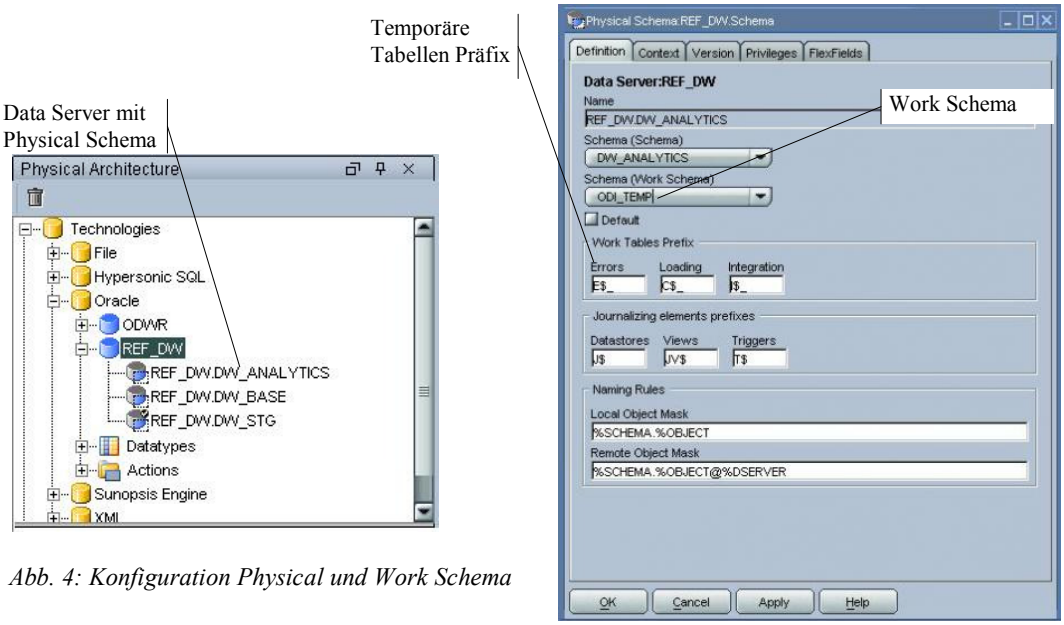

Abb. 4: Konfiguration Physical und Work Schema

Das Deployment und die Lade-Prozesse

Beim Deployment wurde insbesondere beim OWB darauf geachtet, möglichst viele Aufgaben direkt innerhalb der (Ziel-)Datenbank zu verarbeiten, um sich „unnötige“ ELT-Strecken zu ersparen. Jede ELT-Strecke bedeutet CPU-Zyklen, Zeiteinsatz sowie u.U. Systemübergänge, weshalb es oft sinnvoll ist, 1:1-Kopien zu vermeiden. Ist das Zielsystem eine Oracle-Datenbank, so legt der OWB beim Deployment bei Bedarf die entsprechenden Strukturen (Tabellen, Indizes, Mapping-Prozeduren) in dem Zielsystem an und führt die erzeugten PL/SQL-Prozeduren im Zielschema aus. „Bei Bedarf“ meint, dass die Objekte, die während des Modellierens im OWB auf logischer Ebene aufgebaut wurden, beim ersten Deployment erstmalig im Zielsystem physisch angelegt werden. Der ODI legt bei Ausführung eines Interfaces ebenfalls die jeweilige Zieltabelle an, sofern sie noch nicht existiert, jedoch wird im ODI weniger „modelliert“ im eigentlichen Sinne des Wortes als im OWB. Obwohl natürlich auch im ODI die Möglichkeit existiert, neue Tabellen zunächst logisch aufzubauen und dann physisch im Zielsystem anzulegen (z.B. über den Common Format Designer).

Bei der Gegenüberstellung von OWB und ODI sollte auch eine weitere wichtige Komponente angesprochen werden, nämlich die Services bzw. Agenten im Kontext von Deployment und regelmäßigen Ladeprozessen. Der Warehouse Builder nutzt den Control Center Service, um Aktionen von außerhalb der Datenbank anzustoßen, z.B. die Ausführung eines PL/SQL-Skripts. Außerdem sammelt dieser Service im Control Center Informationen zur Ausführung von Mappings und zu Deployments, die nach Objekt oder nach Job eingesehen werden können. Dieser Control Center Service kann auf mehreren Servern installiert sein. Es kann jedoch immer nur ein Control Center samt dazugehöriger Configuration zur Laufzeit aktiv sein.

Ab dem Release 11.2 des Warehouse Builder können auch die Knowledge Modules des ODI importiert und verwendet werden. Diese werden im OWB „Code Templates“ genannt und ermöglichen eine beliebige Kombination von Quell- und Zielsystemen. (Handelte es sich bei dem Zielsystem um eine Datenbank eines anderen Herstellers, so schrieben bis zum Release OWB 11.2 die erzeugten PL/SQL-Prozeduren die resultierenden Daten in ein Flat File, das dann in das Zielsystem geladen wurde, oder nutzten ein Transparent Gateway.) Um ein solches Code Template Mapping auszuführen oder um Web Services zu erstellen, gibt es nun für den OWB einen zusätzlichen Control Center Agent. Das ist eine Java-basierende Laufzeitumgebung, die genauso funktioniert wie im ODI: Die jeweiligen SQL-Befehle werden direkt zur Laufzeit erzeugt und vom Agenten auf den entsprechenden Systemen zur Ausführung gebracht. ODI bedient sich also eines Agenten-Konzepts,

um die gewünschten Aktionen auszuführen. Dabei läuft ein ODI Agent in einer JVM (Java Virtual Machine), der zur Laufzeit die entsprechende SQL-Befehle erzeugt für anzusteuernde Datenbank. Durch Verwendung der Knowledge Modules kann auf die jeweilige Technologie des Systems (Oracle, DB2, usw.) aufgesetzt werden statt mit reinem Standard SQL per ODBC/JDBC.

Die Betriebsumgbung des ODI bildet den Mittelpunkt für die vollständige Implementierung und den Betrieb des DWH. Dabei spielen Agenten eine zentrale Rolle für die Ausführungen aller Prozesse auf den jeweiligen Endsystemen. Im Vergleich zum zentralen Control Center und der Laufzeitumgebung des OWB können mehrere ODI Agenten gleichzeitig verschiedene Prozessabläufe aus dem gleichen Repository bedienen. Sie führen die einzelnen Prozesse Befehl für Befehl auf den dazu vorgesehenen Systemumgebungen aus. Dabei ist für die Laufzeitumgebung der „standalone Agenten“ minimal eine einfache Java Laufzeit Umgebung (JVM) auf einem Server notwendig. Als zweite Alternative können die Agenten als Dienst in einem WebLogic Server implementiert werden. Hier können die Agenten als Cluster konfiguriert und somit die Verfügbarkeit und gleichmäßige Lastverteilung durch ein vorgeschaltetes Load-Balancing sichergestellt werden.

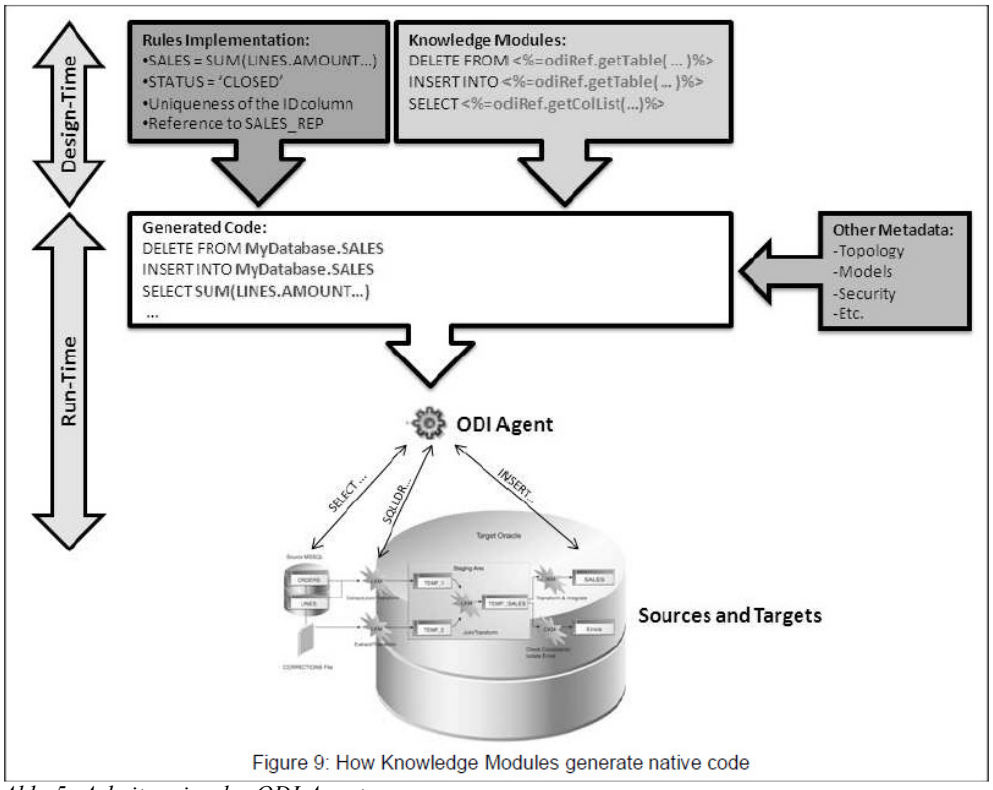

Abb. 5: Arbeitsweise des ODI Agenten

Bei der Ausführung eines Interfaces bestimmen die Knowledge Modules, welche technische Funktionen für den Datentransport und der Datenintegration für die einzelnen Quell- und Zieldatenspeicher sowie den implementierten Datenfluss herangezogen werden sollen. So werden z.B. durch das Load Knowledge Module “SQL to SQL” die Daten per JDBC über den Agenten aus dem Quellsystem gelesen und in das Zielsystem geschrieben. Der Codegenerator erzeugt für jeden Prozessschritt zur Laufzeit den übersetzen Code und füllt alle ODI-spezifischen Platzhalter wie die Verbindungsparameter u.a. aus, so dass eine Multi-Konfiguration durch den festgelegten Context möglich ist. Dieser Context legt ähnlich wie die OWB Configurations fest, wo das Deployment und

die Ausführung der Interfaces erfolgen soll Der Context läst sich auf System-Ebene festlegen und steht Projekt-übergreifend zur Verfügung. Dagegen werden im OWB die verwendeten Objekte je Projekt jeweils importiert und eine Configuration lässt sich nur im Rahmen eines Projekts verändern.

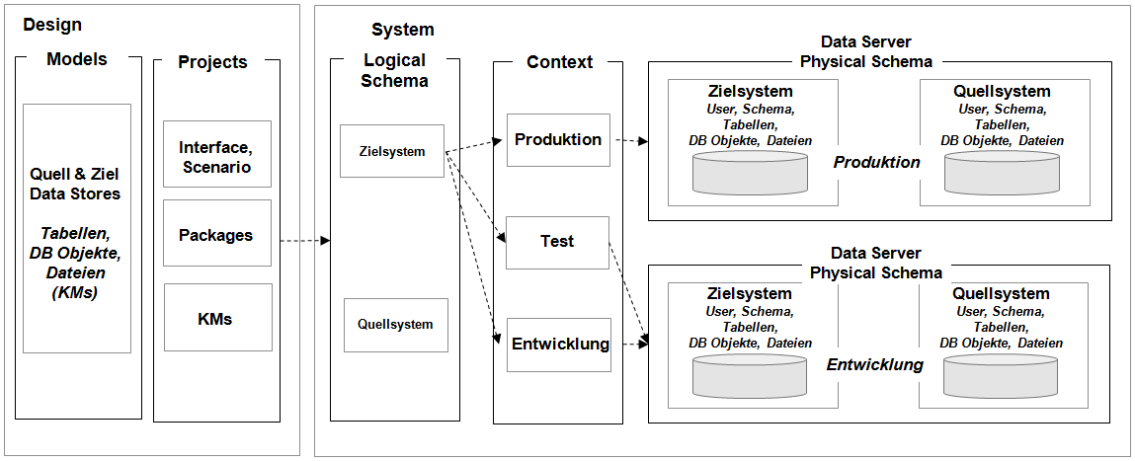

Abb. 6: Ausführungsort durch den Context festlegen

Speicherung der Metadaten

Wer den OWB bereits seit längerem einsetzt, kann sich vielleicht noch an die Zeit erinnern, in der zwei Repositories verwendet wurden: Das Warehouse und das Runtime Repository. Während im Warehouse Repository alle wichtigen Metadatenfür das Design und Deployment der Warehouses hinterlegt waren, enthielt das Runtime Repository alle Laufzeit-Statistiken zu den jeweiligen Ausführungen. Mit dem OWB 10.2 gab es schließlich nur noch ein zentrales Repository. Configurations sind dabei das Mittel des OWB, verschiedene Umgebungen (wie Test/Prod/Dev) mit unterschiedlichen Configurations sauber voneinander zu trennen anstelle von separaten Work Repositories (ODI, s.u.). In OWB 11.2 ist es noch einfacher geworden, mehrere Configurations nebeneinander zu verwalten. Configurations legen fest, welches Default-Verhalten bzw. Default-Eigenschaften die Objekte besitzen sollen, z.B. in welchen Tablespaces sie abgelegt werden.

Der ODI arbeitet mit mehreren Repositories, dem Master Repository und mindestens einem Work Repository. Das Master Repository enthält alle (zentral zu verwaltenden) Informationen, z.B. über Nutzer, Zugriffsrechte, Profile sowie topologische Angaben zu der Art der angebundenen Systeme, Datenbank-Schemata, versionierten und archivierten Objekten. In dem Work Repository befinden sich die eigentlich entwickelten Objekte, insbesondere die Interfaces (Mappings und Data Flows). Dort sind u.a. Metadaten, Constraints, Feld- und Spaltendefinitionen, Packages, Knowledge Modules, Ausführungsdaten und Logs erfasst. Wie bereits erwähnt können mehrere Work Repositories aufgesetzt werden. Damit lassen sich mehrere Entwicklungsumgebungen sauber voneinander trennen und an spezielle Anforderungen anpassen. Enthält ein Work Repository ausschließlich Ausführungsinformationen, wie es typischerweise auf einem Zielsystem der Fall ist, wird es im ODI auch als Execution Repository bezeichnet. Wie und wo die einzelnen Mappings und Transformationen eines Interfaces auszuführen sind, ist im Work Repository hinterlegt. Die auszuführenden Prozeduren werden nur im Work Repository gespeichert, jedoch nicht im (Ziel-)System selbst wie im Falle des OWB. Der ODI Agent holt sich über das Master Repository die notwendigen Verbindungsparameter vom Quell- und Zielsystem und führt die generierten Programmschritte für das entsprechende System aus. Nach einem festgeschriebenen Ablaufplan verarbeitet der Agent so den gesamten Programmcode auf dem entsprechendem System.

In dem Praxisbeispiel stellte sich letztlich folgende Konfiguration als praktikabel heraus:

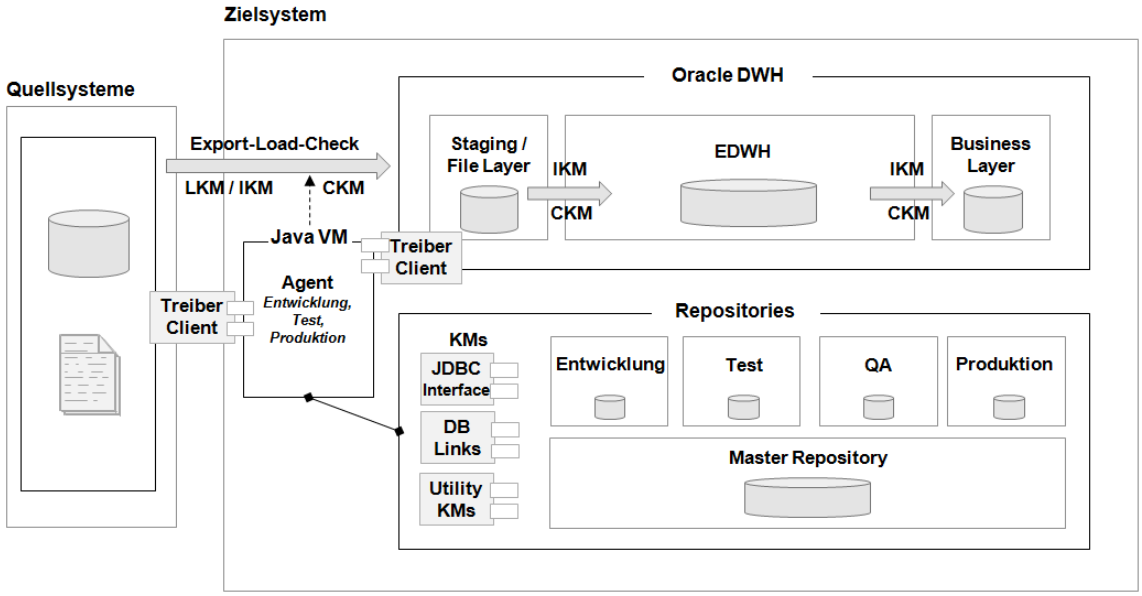

Abb. 7: Setup des Kundenbeispiels mit ODI

Hier wurde der standalone ODI Agent auf dem Zielsystem installiert. Obgleich nicht notwendig, befinden sich das Master Repository wie auch das Produktions-Repository (Work Repository) auf der gleichen Datenbank wie das DWH. Im Unterschied zum OWB sind Master und Work Repositories unabhängig von der Zieldatenbank.

Fazit

Die Unterschiede in der Architektur von OWB und ODI bieten interessante Möglichkeiten für die (physische) Implementation eines DWH. Ein Unterschied zwischen beiden Werkzeugen besteht in der Plattform(un)abhängigkeit. Durch den ersten Schritt in Richtung Zusammenführung beider Werkzeuge über die Knowledge Module (ODI) bzw. Code Templates (OWB) ist dieser Unterschied schon deutlich geringer geworden. Ein weiterer Punkt ist die Aufgabenverteilung und Ausführung. Während auf Seiten des OWB sämtliche Aufgaben zentral zusammengefasst werden, lässt der ODI eine bewusste Verteilung von Aufgaben oder Einzelschritten zu. Die Verteilung wiederum bestimmt, welche Funktionen für die weitere Verarbeitung der Daten im ODI zur Verfügung stehen, weshalb es wichtig ist, diesen Umstand beim Design zu beachten.

Kontaktadressen:

Bianca Stolz
ORACLE Deutschland B.V. & Co. KG
Schloßstraße 2
13507 Berlin

Telefon: +49 (0) 30435795163
Mobil: +49 (0) 1605318118
E-Mail bianca.stolz@oracle.com
Internet: www.oracle.de

Negib Marhoul
ORACLE Deutschland B.V. & Co. KG
Schiffbauergasse 14
14467 Potsdam

Telefon: +49 (0) 3312007217
E-Mail negib.marhoul@oracle.com
Internet: www.oracle.de