Anwendungsspezifische Adaption und Erweiterung von OLAP-Anwendungen auf Basis von MDX

Thomas Hasart, Mario Gleichmann, Ilvio Bruder, Peter Forbrig, Andreas Heuer
IT Science Center Rügen, 18581 Putbus

Zusammenfassung

Die Anforderungen an OLAP-Systeme (Online Analytical Processing) [Pendse, 1998] verschieben sich von z.B. wöchentlichen zu immer kürzeren Aktualisierungsintervallen bis hin zu Auswertungen, die den aktuellen Stand des operativen Systems darstellen. Zugleich vervielfachen sich die auszuwertenden Daten. Dabei werden Funktionalitäten der Frontends wie eine zentrale einfache Schnittstelle, die neben der vom Benutzer programmierbaren Datenanalyse auch Datenmanagementaufgaben wie Import/Export, Backup/Restore, Optimierung und Rechteverwaltung beherrscht, vorausgesetzt.

Vor allem die Abarbeitungsgeschwindigkeit stellt dabei immer noch eine große Herausforderung dar. Am Markt sind zurzeit zwei Trends zu beobachten. Zum einen entwickeln Anbieter eigene In-Memory-OLAP-Datenbanken um ihre Anwendung zu beschleunigen. Zum anderen scheuen viele Anbieter den enormen Entwicklungsaufwand für eigene OLAP-Backends und setzen deshalb Datenbanken der etablierten Anbieter ein (Ansätze u.a. in [Tjoa et al., 2007]). Hier hat sich vor allem MDX als Quasi-Standard durchgesetzt.

In diesem Paper werden wir ausgehend von den Anforderungen aktueller OLAP-Anwendungen, MDX (Multi-Dimensional Expressions [Willand, 2004] [Microsoft, 1999]) und seine Splashing-Möglichkeiten analysieren und notwendige Erweiterungen herausstellen.

1 Einführung

Im Projekt Monicca zum Thema "Model-Driven Account Management in Data Warehouse Umgebungen", geht es darum, OLAP-Anwendungen von den Nutzerschnittstellen bis hin zur funktionalen Ebene adaptierbar zu gestalten. Dabei wird für Key Accounts [Sidow, 2007] ein Werkzeug entwickelt, mit dem diese ihre Kunden aufgrund verschiedener Daten aus Data Warehouses oder anderer Quellen besondere Angebote unterbreiten können. Hierdurch wird den Key Account Managern eine spezielle Schnittstelle auf die Daten der Schlüsselkunden bereitgestellt. Diese ermöglicht es, in Anwendungen umfassendere, angepasstere und geeignetere Analysefunktionalität anzubieten. Das allgemeine Problem der Personalisierung von OLAP-Anwendungen wird mittels Techniken im Bereich der Sichtdefinition und Metadaten-gestützte Verwaltung von OLAP-Data Cubes gelöst. Zur Erzeugung der Nutzersichten wird eine Modellsprache entwickelt, die nötige OLAP-Operationen für die Sichten über Verknüpfungen bis hin zur Definition der Ausgabe und der folgenden Interaktion beschreiben kann. Dieser Modell-gestützte Ansatz ist die Grundlage für das "End-User Development" mit dem Ziel, die Anwendung personalisierbar zu gestalten.

2 Architektur

Die Architekturübersicht in Abbildung 1 zeigt die verschiedenen zur Verfügung stehenden Interfaces. Der BI-Client nutzt eine proprietäre Schnittstelle, die direkt auf den OLAP-Processor zugreift. Externe Anwendungen wie Microsoft Excel oder Crystal Reports können XML/A (XML for Analyses [Simba-Technologies-Inc, 2006]) oder ODBO (OLE DB for OLAP [Microsoft, 1997]) nutzen. Beide Interfaces nutzen den MDX-Processor der die Anfragen an den OLAP-Processor weiterleitet. Als Vorteil von ODBO ist vor allem die breite Unterstützung durch die Anbieter zu nennen. Dafür ist ODBO auf die Windows Plattform beschränkt. Bei XML/A sieht es genau anderes herum aus. Dieser Web-Service Standard ist plattformunabhängig. XML/A wird

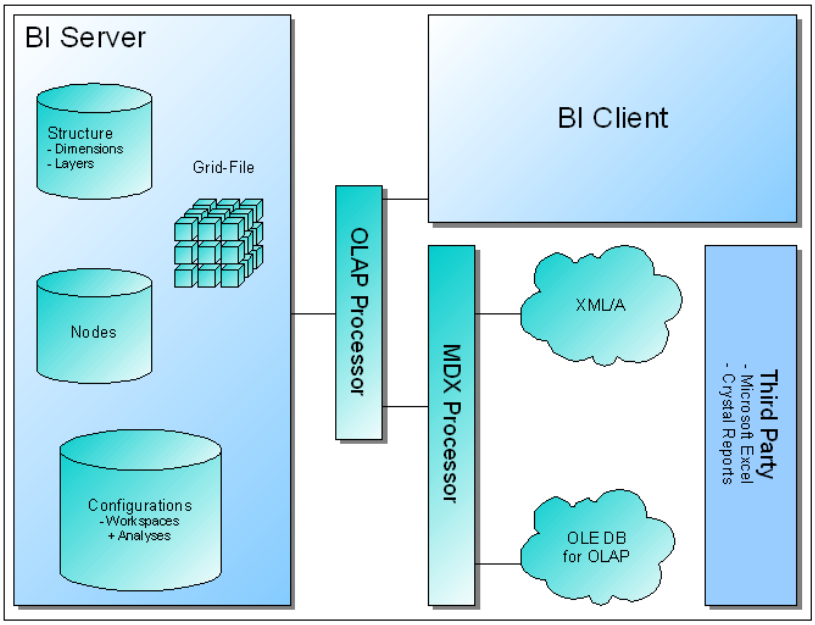

Abbildung 1: Interface Integrationsarchitektur

aber nur zögerlich von den Anbietern unterstützt weil vermutet wird, dass der XML-Overhead und der damit einhergehende Performanceverlust, Microsoft zu einer Änderung der Spezifikation veranlassen könnte. Das auf beiden Schnittstellen verwendete MDX wird nicht von allen Anbietern vollständig implementiert. So fehlen vereinzelt Prozeduren oder Prozeduren führen zu anderen Ergebnissen so dass einzelne Frontend-Anbieter nur Bindungen zu bestimmten Backends zulassen.

3 Splashing/Allocation

OLAP ermöglicht die strukturierte Analyse mehrdimensionaler Daten. Jede Dimension kann mehrere Aggregationsebenen enthalten. Beim Zusammenrechnen der Werte unterer Ebenen zur nächst höheren, existieren teilweise komplexe Abhängigkeiten. Neben der einfachen Summe und dem Durchschnitt ergibt sich nicht selten der Gesamtwert durch Gewichtung der Einzelwerte in Abhängigkeit zu einer anderen Kennzahl. Auch sind Aggregationsmuster Ebenen spezifisch so dass z.B. auf der Ebene der Monate anders aggregiert wird als auf der Ebene der Tage. Neben diesem lesenden Zugriff gewinnt aber immer mehr der schreibende Zugriff, der für die Planung notwendig ist, an Bedeutung. Bei der Planung wird oft vom Nutzer für einen bestimmten Zeitraum ein zu erreichender Gesamtwert vorgegeben. z.B.: 110% des Vorjahreswertes. Dieser Jahreswert muss nun vom System auf die darunter liegenden Ebene (z.B.: Monate oder Quartale) und von dieser weiter bis auf die unterste Ebene verteilt werden. Es müssen die Berechnungen der Aggregation also reversiert werden und deren Teilbeträge auf die Einzelwerte aufgeteilt werden. Diese Aufteilung wird Splashing oder Allocation genannt. Diese automatische Verteilung verursacht zahlreiche Probleme, die im folgenden Abschnitt näher erläutert werden sollen.

4 Probleme beim Splashing

Gelegentlich ist eine Änderung von mehreren Kennzahlen gleichzeitig notwendig, insbesondere wenn Kennzahlen voneinander abhängig sind. Dazu werden die internen Berechnungsergebnisse

der einen Kennzahl zur Änderung der abhängigen Kennzahl benötigt. Ein weiterer Aktualisierungszyklus würde nicht die gleichen Ergebnisse bringen, da die Ausgangswerte ja schon im ersten Durchlauf geändert wurden. Ein Konflikt der sich nur durch gleichzeitiges Setzen aller Kennzahlen lösen ließe.

Wird auf einen Knoten dessen Wert NULL ist verteilt kann dies leicht zum Sprengen der Speichergrenzen führen, da eine Verteilung wohl allen darunter liegenden Knoten einen Wert zu weisen wird, fortgesetzt bis in die unterste Ebene. Eine performante OLAP Lösung macht sich aber gerade zu Nutzen, dass in den seltensten Fällen alle Kombinationsmöglichkeiten Werte enthalten. An einem Beispiel soll dies verdeutlicht werden. Es sei ein mehrdimensionaler Würfel (Cube) mit den Dimensionen Zeit, Artikel, Kunde und Produktionsstandort gegeben. Es wird ein neuer Artikel A2 in die Artikelstruktur unter Oberwarengruppe OW1 und Warengruppe W1 eingefügt. Für diesen Artikel wird für das nächste Jahr 2010 eine Stückzahl von 500 angenommen und dieser Wert in der Kennzahl Menge im Schnittpunkt OW1/W1/A2, 2010 eingefügt. Ein Datenpunkt im Raum ist aber nur valid, wenn ihm ein diskreter Punkt auf der untersten Ebene jeder Dimension zu geordnet wird. Dies bedeutet, dass bei einer Ebenenstruktur Jahr, Monat, Tag in der Dimension Zeit der Wert 500 auf die Tage des Jahres aufgeteilt werden muss. Da es keine Vorgaben gibt werden nun 500/365 = 1,37 Stück auf jeden Tag des Jahres 2010 aufgeteilt. Für jeden dieser neuen Knoten müssen nun auch Knoten auf den anderen Dimensionen definiert sein. Bei vielleicht 500 Kunden erhält jeder Kunde an jedem Tag des Jahres 1,37/500 = 0,003 Stück, von denen an jedem der vielleicht 10 Produktionsstandorte 0,003/10 = 0,0003 Stück produziert werden. Rein mathematisch unproblematisch ergeben sich also 365x500x10 = 1.825.000 neue Punkte. Betrachtet man nun aber gängige Cubes, die zwischen 10 und 20 Dimensionen besitzen von denen z.B. nur die Kundendimension bei Kommunikationsunternehmen schon mehrere Millionen Kunden enthalten kann, kann man sich schnell die Ausmaße solch einer Verteilung vorstellen. Des Weiteren muss man sich verdeutlichen, dass so eine Verteilung auch nicht besonders hilfreich ist, da in der Regel pro Dimension – die Zeit Dimension einmal ausgenommen – nur ein logisch richtiger Knoten existiert. Um dieser Problematik zu begegnen gibt es vielfältige Möglichkeiten. So wäre es denkbar fehlende Verteilungsmuster aus anderen Artikeln zu nehmen – eine sogenannte Verschwägerung – oder Werte aus der Vergangenheit zu nutzen. Auch könnten Vorgaben wie nur auf den ersten, den letzten oder einen bestimmten Knoten definiert werden.

Problematisch ist auch die Berechnung von Kennzahlen auf NULL Werten. Teilweise werden globale Vorgaben wie z.B. Umrechnungsfaktoren für Produktionseinheiten benötigt. Diese können nicht als eigenständige Kennzahlen zur Verfügung gestellt werden da diese dann bei neuen Artikeln nicht existieren würden. Und demzufolge damit auch nicht gerechnet werden kann. Insbesondere bei Rechnungen mit Währungen deren Kurs sich zeitlich ändert sind weitere Schwierigkeiten zu erwarten.

5 Splashing in MDX

Nicht nur das die Serverimplementierung entsprechende Funktionalitäten zur Verfügung stellen muss um diese Verteilungen vornehmen zu können, muss auch der Nutzer die Möglichkeit haben seine Vorgaben dem System mitzuteilen. MDX kennt folgende `UPDATE CUBE` Anweisung um eine Verteilung durchzuführen:

```
UPDATE [CUBE] Cube_Name SET <update clause> [, <update clause> ...n]

  <update clause> ::= Tuple_Expression[.VALUE]=New_Value
   [     NO_ALLOCATION
       | USE_EQUAL_ALLOCATION
       | USE_EQUAL_INCREMENT
       | USE_WEIGHTED_ALLOCATION [BY Weight_Expression]
       | USE_WEIGHTED_INCREMENT [BY Weight_Expression]   ]
```

Dabei werden folgende Splashing-Möglichkeiten unterschieden:

USE_EQUAL_ALLOCATION Jeder Blattzelle, die zur aktualisierten Zelle beiträgt, wird **derselbe** Wert zugewiesen, der auf dem folgenden Ausdruck basiert:

```
<leaf cell value> = <New Value> / Count (leaf cells that are contained in <tuple>)
```

Das entspricht der einfachen Verteilung über die Anzahl der Werte, z.B. würde 1/12 des Jahreswertes auf jeden Monat verteilt werden.

USE_EQUAL_INCREMENT Jede Blattzelle, die zur aktualisierten Zelle beiträgt, wird entsprechend dem folgenden Ausdruck geändert:

```
<leaf cell value> = <leaf cell value> + (<New Value> - <existing value>) /
    Count (leaf cells contained in <tuple>)
```

Dabei wird die Differenz zwischen neuem und alten Gesamtwert durch die Anzahl geteilt und auf die Blattwerte aufgetragen.

USE_WEIGHTED_ALLOCATION Jeder Blattzelle, die zur aktualisierten Zelle beiträgt, wird **derselbe** Wert zugewiesen, der auf dem folgenden Ausdruck basiert:

```
<leaf cell value> = <New Value> * Weight_Expression
```

Der neue Wert wird mittels Weight_Expression, einem Wert zwischen 0 und 1, der im Standardfall den Anteil des alten Blattwertes vom alten Gesamtwert darstellt, gewichtet und auf die Blattzellen verteilt.

USE_WEIGHTED_INCREMENT Jede Blattzelle, die zur aktualisierten Zelle beiträgt, wird entsprechend dem folgenden Ausdruck geändert:

```
<leaf cell value> = <leaf cell value> +
    (<New Value> - <existing value>)  * Weight_Expression
```

Jeder Blattwert wird mit der durch Weight_Expression – standardmäßig ebenfalls der Anteil des alten Blattwertes vom alten Gesamtwert – gewichteten Differenz aus neuem und altem Gesamtwert erweitert.

6 Lösungsansätze

Defizite von MDX lassen sich durch Spracherweiterungen [Vaisman et al., 2002] lösen. Bei den angeführten Splashing Statements kann innerhalb einer Update Klausel nur eine Kennzahl geändert werden. Dies lässt sich durch folgende Erweiterung beheben.

```
<update clause> ::= <tuple clause> [, <tuple clause>...]

<tuple clause> ::= Tuple_Expression[.VALUE]= New_Value
    [     NO_ALLOCATION
        | USE_EQUAL_ALLOCATION
        | USE_EQUAL_INCREMENT
        | USE_WEIGHTED_ALLOCATION [BY Weight_Expression]
        | USE_WEIGHTED_INCREMENT [BY Weight_Expression]   ]
```

Durch die Angabe mehrerer Kennzahlen können nun gleichzeitig auch mehrere geändert werden.

Ein weiteres Manko in den `UPDATE CUBE` Statements ist, dass keine speziellen Vorgaben für die Behandlung von NULL Werten, also ob auf diesen verteilt werden soll oder nicht und wie, angegeben werden können. Eine denkbare Erweiterung wäre:

```
ON_NULL_VALUES <allocation clause> [, <allocation clause>...]

<allocation clause> ::=
    [     USE_ALL
        | USE_LAST
        | USE x
        | USE_PARENT
        | USE_PAST
        | USE_NONE   ]
```

In dieser Form können nun mehrere allocation clauses angegeben werden, die in der angegebenen Reihenfolge Verwendung finden. Z.B: würde

```
ON_NULL_VALUES USE_PARENT, USE_PAST, USE 0
```

bei fehlendem Verteilungsmuster zuerst beim Elternknoten nach einer Verteilung geschaut werden. Wird dort keine Verteilung gefunden wird in der Vergangenheit gesucht. Sind dort auch keine Vorgaben zu finden wird alles auf den ersten Knoten verteilt. Mit `USE_ALL` könnte man im Gegensatz dazu auf jeden möglichen Knoten, mit `USE_LAST` nur auf den letzten, und mit `USE x` auf einen bestimmten, dem x-ten Knoten (0 ist der erste) verteilen. `USE_NONE` würde gar keine Verteilung vornehmen und abbrechen.

7 Zusammenfassung und Ausblick

Bei der Umsetzung von OLAP-Anwendungen auf Basis von MDX existieren Probleme insbesondere im Bereich des Splashing. Mit relativ einfachen praktischen Erweiterungen und Anpassungen von MDX kann ein Plus an Anwendungslogik beschrieben und umgesetzt werden. In weiteren Arbeiten sollen nun die Erweiterungen implementiert und getestet werden.

Literatur

[Microsoft, 1997] Microsoft (1997). OLE DB for OLAP Overview. http://msdn.microsoft.com/de-de/library/ms714903.aspx.

[Microsoft, 1999] Microsoft (1999). Multidimensional Expression (MDX) - Referenz. http://msdn.microsoft.com/de-de/library/ms145506.aspx.

[Pendse, 1998] Pendse, N. (1998). What is OLAP? http://www.olapreport.com/fasmi.htm.

[Sidow, 2007] Sidow, H. D. (2007). *Key Account Management: Geschäftsausweitung durch kundenbezogene Strategien.* mi-Fachverlag, 8. edition.

[Simba-Technologies-Inc, 2006] Simba-Technologies-Inc (2006). XMLA, XML for Analysis, ODBO, OLE DB for OLAP, XMLA provider, bridge, server. http://www.xmlforanalysis.com.

[Tjoa et al., 2007] Tjoa, A. M., Wagner, R., Tomsich, P., and Rauber, A. (2007). *OLAP of the Future.* Österreichische Computer Gesellschaft, Wien.

[Vaisman et al., 2002] Vaisman, A. A., Mendelzon, A. O., Ruaro, W., and Cymerman, S. G. (2002). Supporting Dimension Updates in an OLAP Server.

[Willand, 2004] Willand, S. (2004). Einführung in MDX, Seminararbeit WS04/05.