Bi-Temporale Datenmodellierung
Keine Angst vor der 4. Dimension

Dr. Gernot Schreib
b.telligent GmbH & Co.KG
München

Schlüsselworte

DWH, Datenmodellierung, Normalform, Data Vault, bitemporal, berichtstreu, berichtswahr

1 Einleitung

Die bitemporale Datenmodellierung gilt als Königsdisziplin in der Data Warehouse-Datenmodellierung. Entsprechend viel Respekt wird ihr von den für die Implementierung und den Betrieb Verantwortlichen entgegengebracht. Bei genauerer Betrachtung liegen die Schwierigkeiten dabei nicht in der Modellierung oder dem ETL-Prozess. Mit einem guten 3NF- oder durchgängigen Data Vault-Modell können vielmehr durch einen konsequenten Historisierungsansatz (SCD 2) für alle Tabellen – auch den Bewegungsdaten-Tabellen – einfache, generische Ladeprozesse konstruiert werden, im referenzierten Kundenprojekt mittels eines PL/SQL-Frameworks.
Wie sich während der Entwicklung der Präsentationsschicht sehr deutlich gezeigt hat, liegt die Herausforderung vor allem in der Verwendung des bitemporalen Modells. Die Entscheidung, welcher Zeitpunkt für die Auswertung(en) gewählt werden soll, das Joinen von Historientabellen, die korrekte Kombination des fachlichen Zeitstrahls mit dem technischen (bitemporalen), etc., ist bei der Fülle der Möglichkeiten oft alles andere als trivial.

Im Projekt wurde deshalb ein generischer View-Layer eingeführt, der für jede Tabelle des bitemporalen Modells fünf Views enthält, die die verschiedenen Anwendungsfälle wie z.B. "aktuelle Sicht", "letzte gültige Zeile", "berichtstreu", etc., zur Verfügung stellt. Der Report- oder Anwendungsentwickler erhält so durch die Wahl gleichartiger Views eine zeitlich konsistente Sicht mit i.d.R. nur noch fachlichen Attributen und kann sich so voll auf die Anwendungslogik konzentrieren. Dieser View-Layer wurde dankbar angenommen.

Die vorgestellten Konzepte werden an Hand von SQL-Beispielen des Kundenprojekts erläutert.

2 Entwicklung der DWH-Datenmodellierung

Wie alle Disziplinen der Informatik war in der DWH-Datenmodellierung in den letzten 30 Jahren ein stetiger Wandel zu beobachten. Ausgehend von der Theorie der Normalformen für relational Datenbanken

> "The key, the whole key, and nothing but the key. So help me Codd!"

hat sich die DWH-Modellierung von vertikalen, hochaggregierten Sternschemen (Kimball, Ralph (1996). The Data Warehouse Toolkit. Wiley) über Normalformmodellierung ohne oder mit künstlichen Primärschlüsseln (PK), der inzwischen klassischen 3-Schichten-Architektur zu modernen Mehrschichten-Data-Vault-Modellen hin entwickelt.

Während sich die Schichtenarchitektur aus der Notwendigkeit der Abstraktion und Transformation von Daten heraus entwickelte, fand parallel eine Evolution in der Datenmodellierung (ER-Modell, Tabellenmodell, Datenbankschema) statt. Treiber waren hier Anforderungen aus dem täglichen

Geschäft, die zu Konzepten wie z.B. „slowly changing dimensions“ (SCD), berichtstreue Reports usw. führten. Diese Konzepte sollten und sollen vor allem die Komplexität, die sich durch die Einführung von weiteren v.a. technischen Zeitlinien ergibt, beherrschbar halten. Sind sowohl die fachliche als auch die informationstechnische Zeitlinie (Wann stand die Information zur Verfügung?) in der Modellierung vorhanden, so spricht man von einem bitemporalen Datenmodell. Die sicherlich aktuell modernste Variante der bitemporalen Datenmodellierung ist das unter Data-Vault bekannt Modellierungskonzept von Dan Linstedt.

2.1 Klassische bitemporale Datenmodellierung

In der klassischen bitemporalen Datenmodellierung werden jeder Datenzeile zwei Datums-/Zeitfelder (von–bis) hinzugefügt, die den Gültigkeitszeitraum der in der Zeile enthaltenen Information bestimmt. I.d.R. werden die sukzessiven von-bis Zeiträume lückenlos bestimmt, so dass das von-Datum des logisch folgenden Satzes dem bis-Datum des vorherigen entspricht. Die Zeile mit dem aktuell gültigen Wert wird durch ein spezielles bis-Datum (z.B. NULL oder der maximal mögliche Wert) gekennzeichnet. Es gibt noch weitere Feinheiten der Modellierung, allen Varianten gemeinsam ist, dass die gewünschte lückenlose Darstellung des Gültigkeitszeitraums mit Datenbankmitteln (Constraints) so nicht sichergestellt werden kann. Dies ist alleinige Aufgabe des ETL-Jobs.

2.2 Data Vault Modellierung

Der Data Vault Ansatz verfeinert die klassische Datenmodellierung und gehört auch zu den 3NF-Modellierungs-Techniken. Diese Modellierungsvariante setzt konsequent die theoretischen Erkenntnisse der letzten Jahre um. Er vermeidet alle Verletzungen des 3NF-Paradigmas, z.B. wird konsequent das "valid_to"-Feld vermieden. Die nachfolgenden Ausführungen gelten aber auch – um nicht zu sagen insbesondere – für diese Modellierungstechnik.

2.3 Komplexität bei durchgängiger SCD2-Modellierung

Obwohl ein durchgängiges SCD2-Modell viele Vorteile hat, findet man überraschend wenige DWHs, die das Konzept konsequent umgesetzt haben. Woran liegt das? Zum einen ist die SCD2-Logik nicht einfach zu implementieren und selbst gängige ETL-Tools haben sich in der Vergangenheit in diesem Bereich als so fehlerhaft oder inperformant erwiesen, dass die Funktionalität nicht benutzbar war. Manuell Programmierung erfordert ein gewisses Maß an Erfahrung im Umgang mit größeren Datenmengen und im Falle Oracle speziell mit dem Verhalten des Optimizers. Um die Aufwände hier nicht ausufern zu lassen, werden dann oft nur dedizierte Tabellen mit SCD2-Logik versehen. Diese Reduktion der Komplexität auf ETL-Seite wird aber mit einer deutlichen Erhöhung der Komplexität auf der Auswertungsseite bezahlt. Der Hauptgrund der Vermeidung von SCD2 liegt möglicherweise gerade hier. Es bedarf nicht wenig Erfahrung und Übung mit einem SCD2-Modell komplexe Business-Logiken zu etablieren. Die Zunahme der Möglichkeiten – zu jedem Zeitpunkt in der Vergangenheit liegt der genaue Informationsstand vor – wirkt auf den Business-Analysten nicht selten als Erschwernis und nicht als Erleichterung. Und die Vermischung von SCD2-Tabellen mit nicht historisierten Tabellen – die gut gemeinte Reduktion der Komplexität auf ETL-Seite – tut ihr übriges, da man sich in jedem Einzelfall überlegen muss, ob die gewünschten Daten zum gewünschten Informationszeitpunkt zusammenpassen und die gestellt Frage auch tatsächlich beantwortet wird. Die Performance von SCD2-Tabellen (sowohl beim Beladen wie bei der Benutzung) ist ein weiterer zu beachtender Punkt. SCD2-Tabellen wachsen naturgemäß deutlich schneller als das nichthistorisierte Pendant.

All diese Überlegungen zusammengenommen, ist es nicht verwunderlich, dass in der Vergangenheit konsequente SCD2-Modell nur selten entstanden sind und die Historisierungstechnik zwar sehr häufig, aber eben nur dediziert zum Einsatz kam.

3 Das Kundenprojekt

Im referenzierten Kundenprojekt bestand die Aufgabe ein bestehendes DWH durch eine Neukonzeption abzulösen (DWH 2.0). Das bestehende DWH hat die durchaus lebhafte Firmengeschichte einer B2B-Bank mit Käufen und Verkäufen von Geschäftsbereichen, geänderten Geschäftsmodellen usw. erlebt und ist im Betriebszeitraum von ca. 15 Jahren inzwischen so gealtert, dass ein Neubau nötig wurde.
Die aus den Fachbereichen heraus gestellte Forderung nach Berichtstreue (Berichtswahrheit), d.h., dass sich ein einmal erstellter Bericht nicht mehr verändern darf, implizierte, dass weite Teile des Core-Modells mit technischen Zeitstempeln ausgestattet werden mussten. Um die Entwicklungskosten gering zu halten, wurde entschieden, das Core-Modell grundsätzlich, einheitlich und ohne Ausnahmen mit einer standardisierten SCD2-Logik zu versehen.

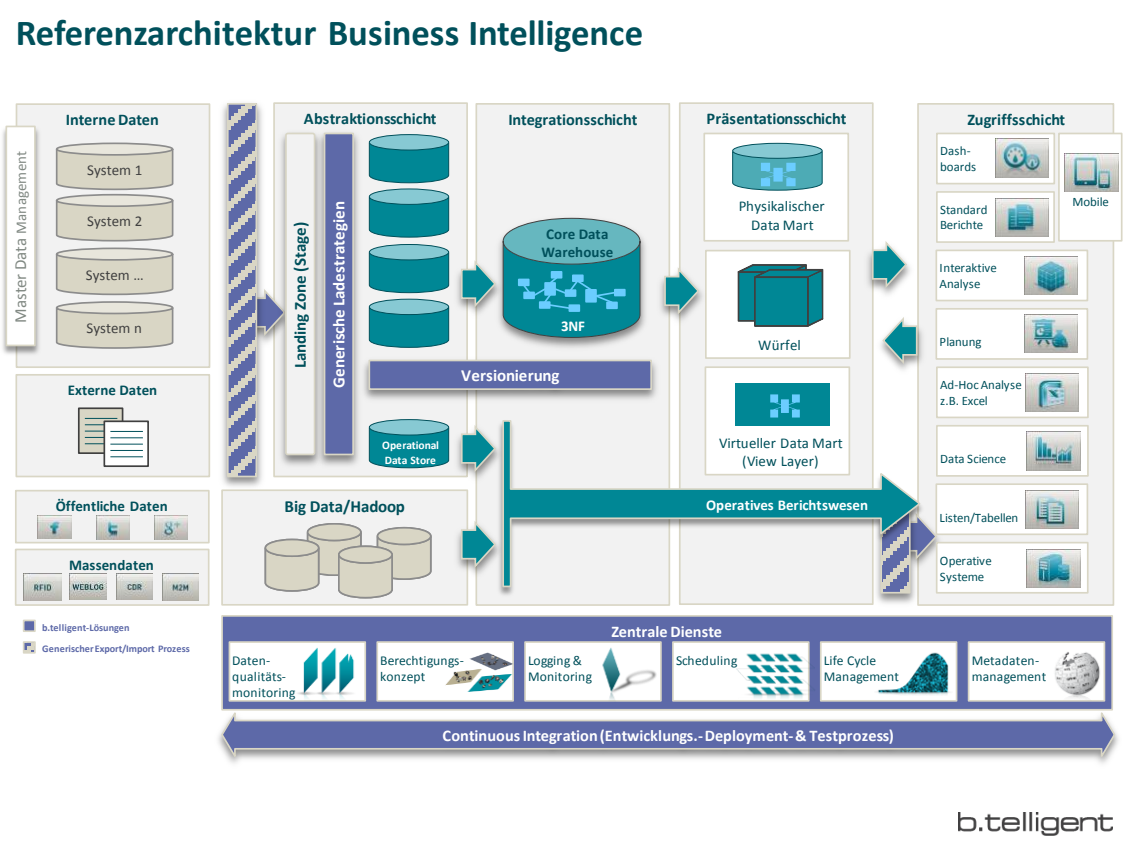

Abbildung 1: BI-Referenzarchitektur

Das DWH wurde in klassischer 3-Schicht Architektur aufgebaut. Die Abstraktionsschicht, bestehend aus einem Stage- und Source-Bereich, nimmt die Daten der Vorsysteme auf. Zunächst werden diese (als Full- oder Delta-Load) in den Stage geladen. Der Source-Bereich enthält eine vollständige Kopie der gelieferten Daten, je fachlichen PK eine Zeile mit der Information des letzten Updates dieser Zeile und ggf. der Information der Löschung dieses Satzes. Aus dem Source-Bereich werden die geänderten Daten für den Core-Bereich ermittelt (CDC, change data capture).

Der Core-Bereich ist durchgängig mit SCD2-Logik modelliert und bildet die quellsystemunabhängigen Entitäten des Geschäftsmodells ab. Die Beladung des Stage-/Source-/Core-Modells erfolgt mittels generischer bzw. automatisch generierter SQLs. Der Code-Generator ist in PL/SQL implementiert und kann über Tabellen parametrisiert werden.

4. Die Präsentationsschicht

Während die einheitliche Vorgehensweise für das ETL der Abstraktions-/Integrationsschicht gemessen an der Komplexität trotzdem einfache, performante und gut wartbare Prozesse ermöglichte, gestaltete sich der Aufbau der Präsentationsschicht (Application-Layer) deutlich schwieriger. Dies ist jedoch nicht weiter verwunderlich, da nun beim Übergang des Core-Modells zur Präsentationsschicht die i.d.R. komplexe Business-Logik zu implementieren ist. Dabei erweist sich das bitemporale Core-Modell leider zunächst als weiterer Komplexitätstreiber. Die Möglichkeiten sind durch die zusätzliche Zeitlinie deutlich erweitert worden. Um hier v.a. korrekte Abfragen und in Folge dann auch wartbare Prozess zu erhalten, bietet es sich an, die Aufgabe in Teilschritte zu zerlegen.

In den wenigsten Fällen benötigt man in der Applikation die volle Sicht auf die historisierten SCD2-Tabellen. Viel häufiger ist man an den aktuellen Daten interessiert oder benötigt keine spezielle Sicht in die Vergangenheit. Zur Unterstützung des Aufbaus einer Präsentationschicht ist es daher sinnvoll, für die verschiedenen Anwendungsfälle eine generische View-Schicht auf das Core-Modell zu erstellen. Im vorliegenden Kundenprojekte besteht diese View-Schicht aus fünf Views pro Core-Tabelle:

- View für „Current“ (nur die gültigen Einträge)
- View für „History“ (identisch zu Core-Tabelle)
- View für „Last Row“ (alle gültigen + die letzte Zeile der geschlossenen Einträge)
- View für „History by Day“
- View für „Berichtstreue“

Aufbauend auf diese View-Schicht wurde die Objekte der Präsentationsschicht, Data Marts, BO-Universen, etc. erstellt bzw. nachfolgende Applikationen (OLAP-Würfel) versorgt.

4.1. View für „Current“, „History“ und „Last Row“

Die am häufigsten genutzt View ist sicherlich die, die nur den aktuellsten Satz der historisierten Tabelle je PK ermittelt. Für die Repräsentation des aktuell gültigen Satzes wurde für die valid-to-Spalte der Wert „31.12.9999 23:59:59“ – der maximale Wert des DATE-Datentyps – verwendet. Die Darstellung der Current-View ist nun einfach (siehe Abbildung 2: Current-View).

Current-View

```
CREATE VIEW BIZ.T /* or BIZ.T_C */
AS
SELECT
  T.DWH_ID,
  …,
  T.DWH_VALID_FROM
FROM CORE.T T
WHERE T.DWH_VALID_TO=CAST('31.12.9999 23:59:59' AS DATE)
;
```

b.telligent

Abbildung 2: Current-View

Mit der „History“-View (siehe Abbildung 3: History-View) soll erreicht werden, dass die historisierte Core-Tabelle direkt aus der Präsentationsschicht angesprochen werden kann und kein Durchgriff auf dem Core-Layer erforderlich ist. Dies ermöglicht es, ein durchgängiges Rechte-/Benutzerkonzept umzusetzen. Weitere Applikations-Logik ist in dieser View nicht enthalten.

Die View „Last Row“ zeigt analog der View „Current“ alle aktuellen Datensätze einer Tabelle. Im Falle einer Löschungen (z.B. einer Ausprägung in einer Dimension) wird zusätzlich die zeitlich letzte Zeile aller bereits geschlossenen Zeilen selektiert. Falls es auf dieser Tabelle keine Löschungen geben sollte, ist diese View identisch zur View „Current“. Im Vortrag wird der SQL-Code präsentiert.

History-View

```
CREATE VIEW BIZ.T_H
AS
SELECT
  T.DWH_ID,
  …,
  T.DWH_VALID_FROM,
  T.DWH_VALID_TO
FROM CORE.T T
;

/* SELECT * FROM CORE.T */
```

b.telligent

Abbildung 3: History-View

4.2 View für „History by Day“ und „Berichtstreue“

Die Views „History by Day“ und „Berichtstreue“ sind der eigentliche Mehrwert der generische View-Schicht auf dem Core-Modell. Sie kommen in unterschiedlichen Anwendungsszenarien zum Einsatz:

Einer der Vorteile der Mehrschichtarchitektur des DWHs ist die Trennung des ETL-Ladelauf-Rhythmuses vom dem der Berichtserstellung (siehe Abbildung 4: Rhythmus ETL vs. Bericht). Mit Ausnahme der Berichte des operativen Monitorings, sind Fachbereiche meist nicht erfreut, wenn sich wesentliche Kennzahlen untertags ändern. Daher ist es sinnvoll ein Berichtsumfeld zur Verfügung zu stellen, das innerhalb eines Zeitraum eines Tages (oder halben Tages, die Granularität ist wählbar) stabil bleibt. Dies erreicht man durch Festlegung einer Uhrzeit, z.B. 23:59:59 Uhr, die repräsentativ für den gewünschten Tag steht. Sind mehrere ETL-Läufe untertags erfolgt, so wird für den Tag der Datensatz mit der festgelegt Uhrzeit ausgewählt und verwendet.
Die „History by Day“-View setzt dies um. Zu jedem Tag (eines Kalenders) wird genau eine gültige Zeile je PK ermittelt.

Der Join zwischen zwei oder mehr SCD2-Tabellen ist alles andere als trivial. Die View „History by Day“ hilft auch beim verjoinen von History-Tabellen, wie im Vortrag dann ausführlicher erläutert werden wird.

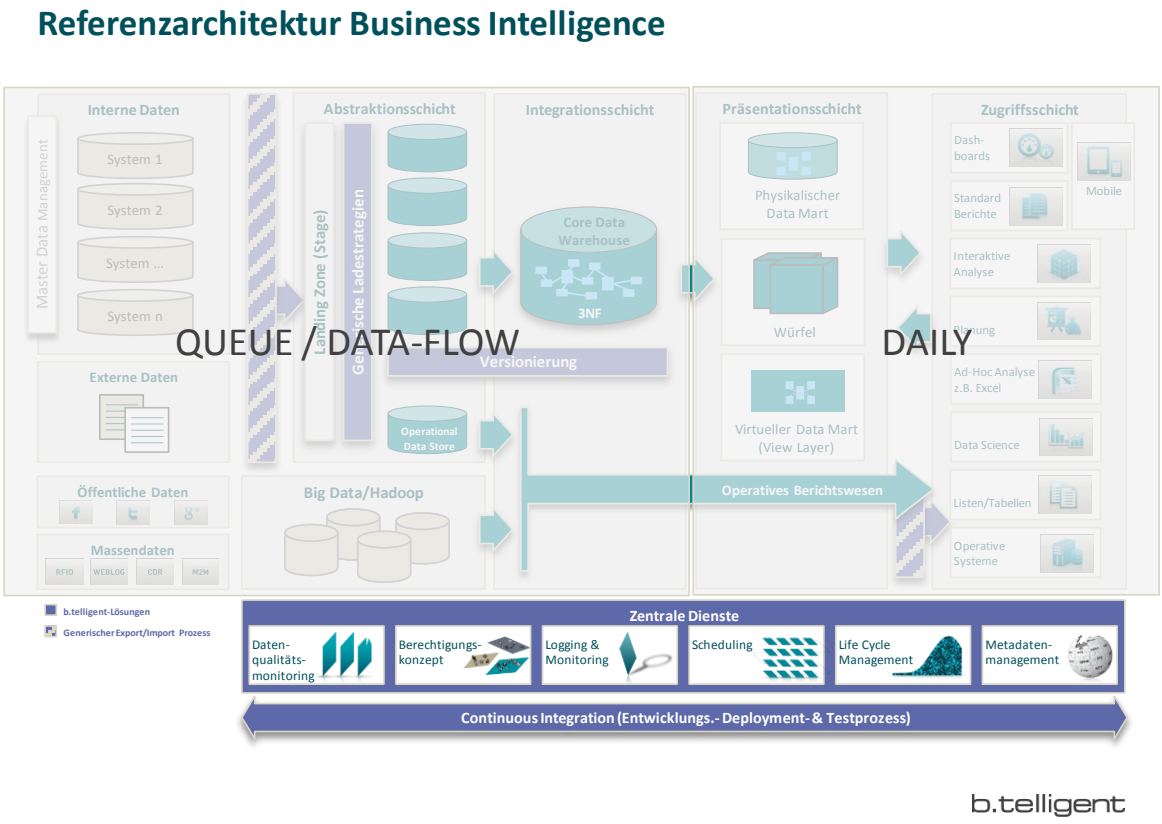

Abbildung 4: Rhythmus ETL vs. Bericht

Ist das Konzept „History by Day“ vollständig umgesetzt, ist der Weg zur Berichtstreue nicht mehr weit. Benötigt wird eine Tabelle, die zu jedem fachlichen/semantischem Datum (z.B. Buchungsdatum) einen technischen Informationsstand-Zeitstempel (Datum, Uhrzeit) enthält. Mit Hilfe dieser Tabelle lassen sich nun die Zeile aus den Historien-Tabellen selektieren, die der fachlichen und der informationstechnischen Zeitschiene entsprechen (siehe Abbildung 5: Berichtstreue-View). Die finale berichtstreue View enthält keine technische Zeitlinie mehr, sondern nur noch die fachliche Zeit. Für den Anwender präsentieren sich die Daten semantisch analog der Current-View, aber berichtstreu.
Diese wesentliche Erleichterung wurde wie der gesamte View-Layer dankbar angenommen.

Alle Views lassen sich (immer wieder) automatisiert erstellen und benötigen keinen manuellen Eingriff in der Programmierung.

Berichtstreu-View

```
CREATE VIEW BIZ.T_B
AS
SELECT
  S2T.DAY_SEMANTICAL AS DAY,
  T.DWH_ID,
  …,
  T.DWH_VALID_FROM
FROM BIZ.SEMANTICAL2TECHNICAL S2T INNER JOIN CORE.T T
       ON T.DWH_VALID_FROM <= S2T.DAY_TECHNICAL < T.DWH_VALID_TO  /* Pseudo-SQL */
;
```

b.telligent

Abbildung 5: Berichtstreue-View

5 Best practice & Empfehlungen

Im Rückblick ermöglichte die Entscheidung für das Schichtenmodell mit View-Layer, einen schlanken, ohne Ausnahmen standardisierten ETL-Prozess von den Quellen bis in das Core-Modell zu implementieren. Die vereinheitlichte Vorgehensweise erhöhte die Komplexität bei weitem nicht in dem Maße wie befürchtet. Im Gegenteil, alle Tabellen des Core-Modells können mit einem einheitlichen Verfahren sowohl beladen, als auch abgefragt werden. Die Komplexität liegt daher eher unter derjenigen eines Modells mit Mischbetrieb (dedizierter Einsatz von SCD2-Tabellen). Die Einführung eines View-Layers als Zugriffsschicht für das Core-Modell erlaubt eine konsistente Nutzung der historisierten Tabellen bei gleichzeitiger Reduktion der Komplexität. Die Verwendung von Views gleichen Typs, z.B. „Current“ oder „Berichtstreu“, garantiert zum einen die Konsistenz, ist für den Anwender jedoch nicht komplizierter als die Verwendung eines nicht historisierten Core-Modells.
Ein weiterer Vorteil ist, dass der View-Layers unterschiedliche Rhythmisierungen der Applikationen (vom ETL bis zu den Reports und Applikationen) unterstützt.

Die Empfehlung lautet daher, das Core-Modell als voll historisierendes NF-Modell anzulegen und in einer Schicht von generische Views den Anwendern und Applikationen die verschiedenen historischen Sichten in unterschiedlichen Komplexitätsstufen (von „current“ bis „berichtstreu“) einheitliche Verfügung zu stellen.

Kontaktadresse:

Dr. Gernot Schreib
b.telligent GmbH & Co.KG
Georg-Brauchle-Ring 54
D-80992 München

Telefon: +49 (0) 89-1222811 85
E-Mail gernot.schreib@btelligent.com
Internet: www.btelligent.de

b.telligent ist eine Unternehmensberatung, die auf Einführung und Weiterentwicklung von Business Intelligence, Customer Relationship Management und E-Commerce in Unternehmen in Massenmärkten spezialisiert ist.

Der Fokus liegt dabei auf der kontinuierlichen Optimierung von Geschäftsprozessen, Kunden- und Lieferantenbeziehungen durch den Erkenntnisgewinn aus der Verdichtung, Analyse und Prognose von systemübergreifenden Geschäftsdaten. So lassen sich Margen erhöhen, Kosten senken und Risiken besser kontrollieren.

Kunden von b.telligent sind Branchenführer aus den Bereichen Telekommunikation, Finanzdienstleistung, Handel und Industrie.