Architektur kleiner Datenbank-Anwendungen zur Materialfluss-Steuerung

Dr. Gerhard A. Hergesell
Siemens AG
Nürnberg

Schlüsselworte:

Datenbank, ORACLE, Software-Architektur, Materialfluss-Steuerung, SCADA

Einleitung

Seit zahlreichen Jahren wird ORACLE RDBMS auch als Datenhaltung zur Steuerung von Materialflüssen verwendet. Es ergeben sich dabei relativ kleine Datenbanken, die jedoch hochperformant und ausfallsicher angelegt werden müssen.

Bei einer derartigen Verwendung eines relationalen DBMS sind wichtige relativ atypische Überlegungen zur Architektur der Anwendung und zum Design der Anwendungs-Datenbank nötig. Besonders eingegangen wird dabei auf die Anbindung eines SCADA-Systems, das Design der Datenbank-Objekte, das Verhältnis zu objektorientierten Konzepten und die Architektur der Dialog-Anwendung. Die vorgestellten Lösungen basieren auf scheinbar unkonventionellen Ansätzen und stellen einige derzeit herrschende Entwicklungs-Modeströmungen in Frage.

Es wird die Architektur echter Produkte und Projekte gezeigt, bei denen sich die dargestellten Konzep-

te in verschiedenartigen Materialfluss-Systemen bewährt haben, unter Anderem bei der Steuerung von Lagerhaltungen, der Automobilfertigung, der Kopplung asynchroner Fertigungsprozesse und der Gepäcksortierung bei Flughäfen.

Begriffe und Umfeld

Gewöhnlich assoziiert man die IT-Produkte und –Projekte der Siemens AG mit dem Betrieb großer Rechenzentren und der Verwaltung riesiger Datenmengen. Im kürzlich neu geschaffenen Unternehmensbereich „Infrastructure and Cities“ finden sich jedoch auch Geschäftsfelder und Arbeitsgruppen zusammen, die IT-Produkte und –Projekte realisieren, welche gleichsam als datentechnischer Überbau von Anlagen im weitesten Sinn dienen. Wir befinden uns hier also im Bereich des klassischen Anlagenbaus; die entsprechenden Anwendungen (**Leitsysteme**) steuern in diesen Anlagen die (strategischen) Abläufe und geben für die Ebene der speicherprogrammierbaren Steuerungen (SPS, etwa SIMATIC S7 ™) die Aktionen vor, mit denen der Durchfluss von Material bestimmt wird.

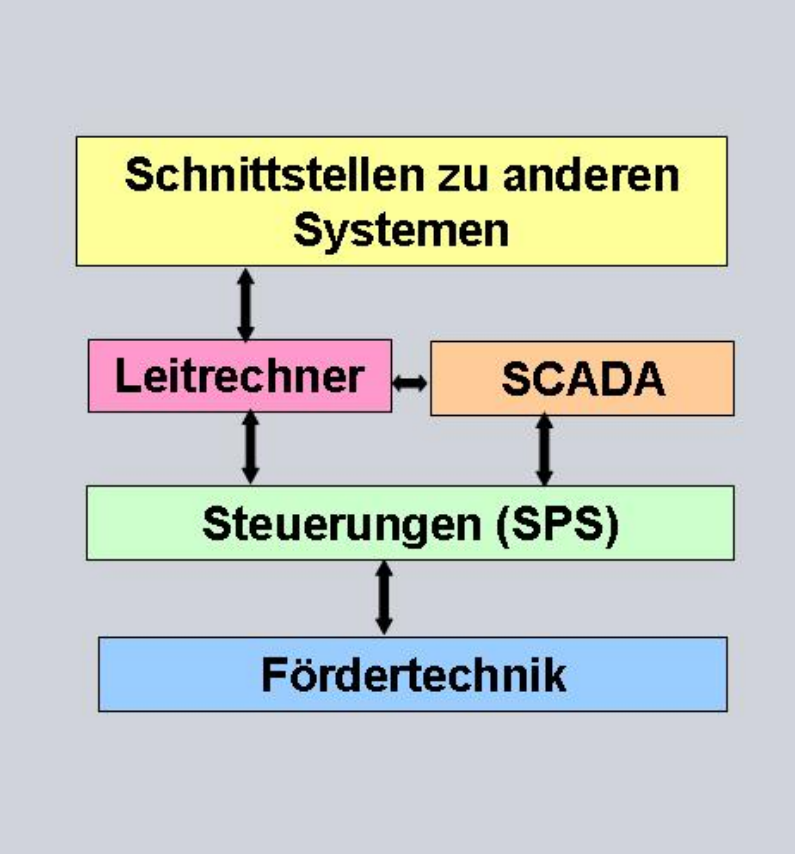

Abb. 1: Komponenten des Gesamtsystems

Damit dies nicht allzu trocken theoretisch bleibt, hier einige Beispiele (und Referenzen) für derartige Anwendungen:

- Automobilfertigung (Rohbaufertigung BMW)
- Fertigungspuffer zwischen asynchron arbeitenden Fertigungszellen (Siemens Regensburg)
- Warenlager aller Art, u.a.

Quelle Verteilzentrum Leipzig
Dänische Bettenwelt Verteillager
Edeka Nord
Energieversorgung Sachsen Betriebsmittel- und Teilelager
Logistikzentrum HARO
Chemikalienlager BAYER Metalloxide
Arzneimittel-Großhandelslager Drugofa (BAYER Consumer Care)
Süddt. Elektromotoren-Werke
Zweirad-Einkaufs-Genossenschaft
diverse Fertigungs- und Auslieferungslager bei SIEMENS

- Postsortieranlagen (Briefe, Großbriefe, Päckchen, Pakete)

- Gepäcksortierung und Frachtabwicklung in Flughäfen, u.a.
München T2
Madrid T4
London Gatwick
London Heathrow T1-T2-T4 (nicht T5!)
Beijing T3 (Olympia-Terminal)
Hongkong
Paris CDG T2
Zürich
Dubai
Seoul-Incheon
Kuala Lumpur
Delhi
Kolkata

Ich denke, dass die Begriffe „Materialfluss-Steuerung“ und „Leitsystem“ jetzt etwas plastischer und verständlicher geworden sind.

Zur Rolle der **Datenbank** in diesen Anwendungen muss man etwas ausholen. Gedanklich stellt man sich unter einer Datenbank regelmäßig eine große Menge, womöglich Terabyte oder Exabyte von Daten vor, die „auf der Platte“ stehen, im Extremfall dort „im letzten Winkel“, und von dort „mühsam“ stückweise in eine Hauptspeicher-Datenhaltung (etwa: Object Engine, Middle Tier einer Maskenanwendung) geholt werden müssen. Derartige Datenbanken treten bei einer Materialfluss-Anwendung in der Regel nicht auf. Dennoch benötigt man auch bei solchen Anwendungen eine Datenhaltung, die gewisse Mindestanforderungen erfüllt:

- Persistenz bei jedem Anlagenzustand
- Transaktionssicherheit
- einfache Abfrage-Methode (etwa SQL)
- konkurrierende verändernde Zugriffe
- sicheres und konsistentes Wiederanlauf-Verhalten

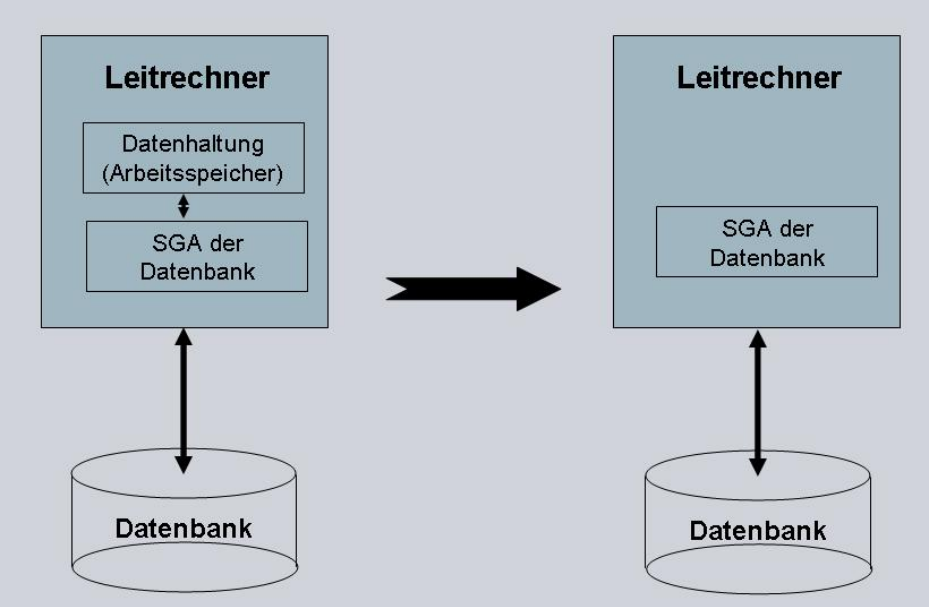

Abb. 2: Die Datenbank als einzige Datenhaltung

Wir haben dazu schon vor geraumer Zeit (ca. 1990) die Sicht auf unsere Daten umgekehrt: Anstatt uns darüber zu grämen, dass unsere Daten bei Einsatz eines RDBMS im „hintersten Winkel“ einer Externspeicher-Architektur liegen und „mühsam“ in den Hauptspeicher geholt werden müssen, begreifen wir die Datenbank als ein Stück Hauptspeicher (Shared memory, SGA), in dem unsere Daten jederzeit direkt verfügbar sind. Diesen Speicherplatz übergibt man nun der „Verwaltung“ durch ein RDBMS, damit die aufgelisteten Anforderungen erfüllt werden. Das geht natürlich nur dann gut, wenn die Datenmenge relativ „klein“ ist. In der Praxis bewegen wir uns im Bereich von ca. 300 MB bis ca. 10 GB Netto-Datenvolumen (ohne Indexe, Undo-Segmente, Redo Logs, System-Tablespace).

Bevor Sie jetzt frustriert über diese vermeintliche „Spiel-Datenbanken“ die Nase rümpfen, werde ich im Hauptteil meines Vortrages darstellen, welche Überlegungen bei einem derartigen RDBMS-Einsatz notwendig sind, welche (meist) unerwarteten Fallen in diesem Ansatz lauern und welche Konsequenzen daraus (manchmal zur Überraschung vieler Experten) folgen.

Der Datenfluss, einschließlich eines Ausflugs in die Welt des SCADA und der Steuerungen

Bei den Materialfluss-Leitsystemen haben wir es mit einer (erstaunlicherweise?) über alle Technologien der Anlagen hinweg ziemlich einheitlichen Architektur der Daten(flüsse) zu tun:

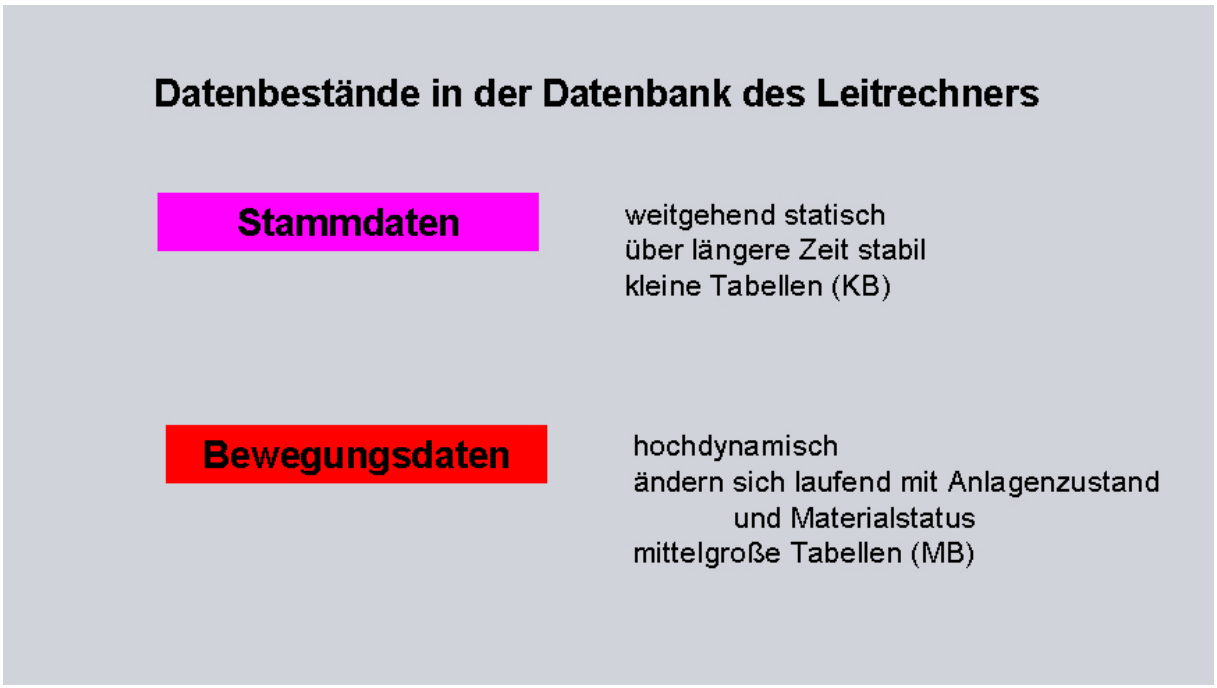

Abb. 3: Komponenten des Gesamtsystems

Da gibt es zuerst eine Vielzahl von kleinen Tabellen, die weitgehend statisch der Anwendung zur Verfügung stehen, die sogenannten **Stammdaten**. Stellen Sie sich hier, je nach Anwendungsfall, etwa die Liste aller Zielflughäfen, den Artikelkatalog, die angesteuerten speicherprogrammierbaren Steuerungen (SPS), alle Transportelemente (Fördertechnik, Sorter, Aufgabestellen und Abwürfe), Ladeträger **(**Paletten, Trays, Kästen, …) und weitere Kenndaten im weitesten Sinn der Anlage vor.

Daneben gibt es typische **Bewegungsdaten**, in denen der Zustand der Anlage sowie des gegenwärtig (einschließlich einer gewissen Zeit in Vergangenheit und Zukunft) vorhandenen Materials abgebildet sind. Hier gilt, dass die Daten sich deutlich schneller „bewegen“ müssen als das physische Material auf der tatsächlichen Anlage. Dies kann Antwortzeiten (zwischen Anfrage der SPS und Antwort darauf) im Bereich von 10 msec erfordern, bei einem Durchsatz von bis zu 1000 Transaktionen pro Sekunde.

Ein wesentlicher Anteil der Bewegungsdaten dient ausschließlich zum Treiben des **SCADA-Systems**. (SCADA = System/Supervisory Control and Data Acquisition). In diesem System wird, je nach Anlagengröße und Wunschvorstellung des Kunden, der Anlagenzustand grafisch angezeigt – über eine Video-Wand, Arbeitsplatz-Bildschirme, oder Anzeigen (Leuchten, Sirenen, …).

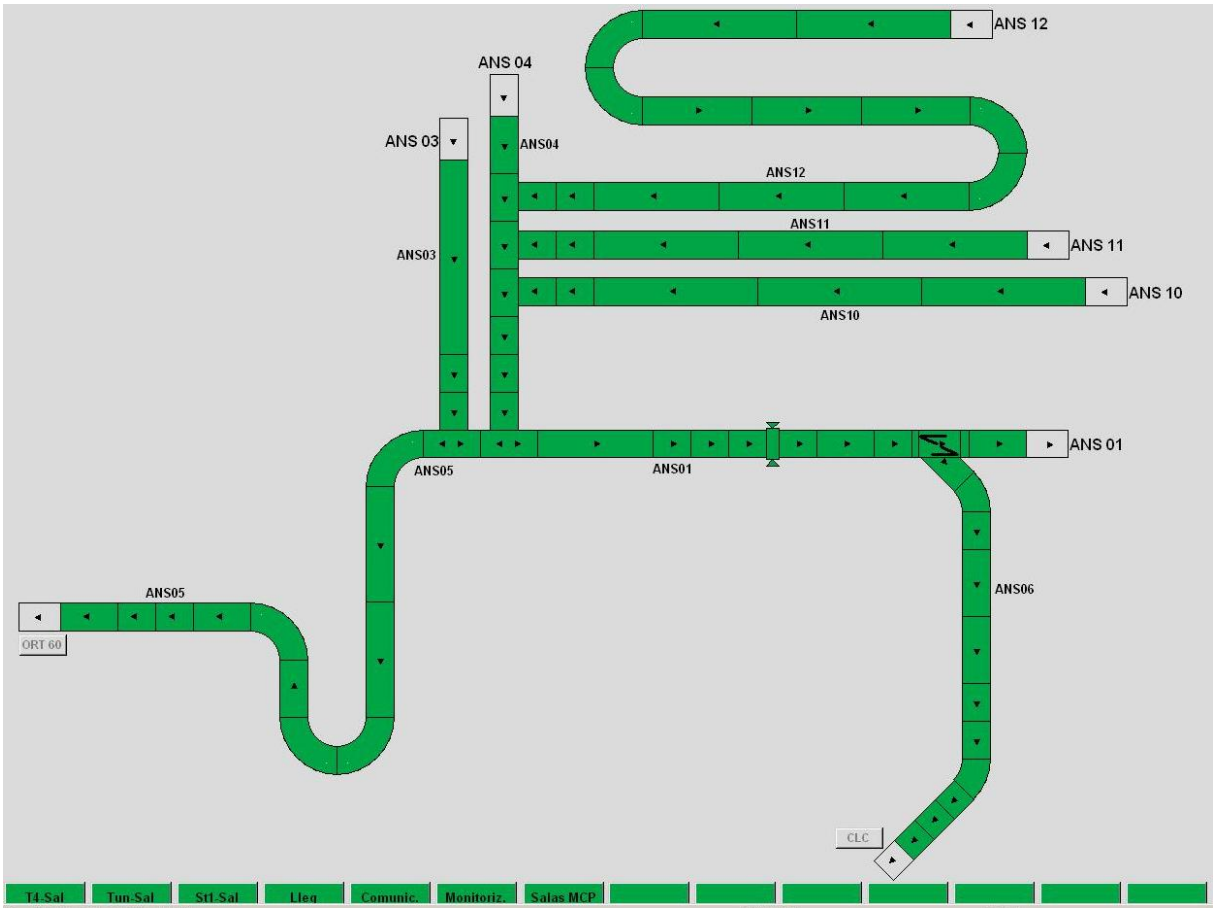

Abb. 1: Beispiel eines SCADA-Screens (Fördertechnik)

Daneben können über dieses System auch einfache Bedienungen der Anlage erfolgen (Ein-/Ausschalten von Anlagenteilen, Rücksetzen von Fehleranzeigen, Start von Steuerungen, etc.). Der insgesamt für den SCADA-Betrieb nötige Datenfluss beträgt oft mehrere Tausend Meldungen pro Sekunde – der für die Materialfluss-Steuerung relevante Anteil beträgt nur einige Prozent davon und wird in der Regel aus dem gesamten Datenfluss für das SCADA ausgefiltert.

Dies kann auf zwei prinzipiell verschiedenen Arten durchgeführt werden.

- Entweder nimmt das Leitsystem alle Daten der SPS-Ebene entgegen, speichert selbst nur die für das System selbst relevanten Daten in der Datenbank und reicht die übrigen Daten an das SCADA weiter.

- Oder das SCADA-System nimmt seinerseits alle Daten der SPS-Ebene entgegen und reicht die wenigen für das Leitsystem relevanten Daten an dieses weiter.

Für beide Lösungen gibt es Anwendungsfälle, die unter Anderem auch davon abhängig sind, welches SCADA-System (etwa: SIMATIC WinCC ™ oder LCMBASE™) verwendet wird und wie eng dieses an das Leitsystem angebunden ist (built-in oder detached).

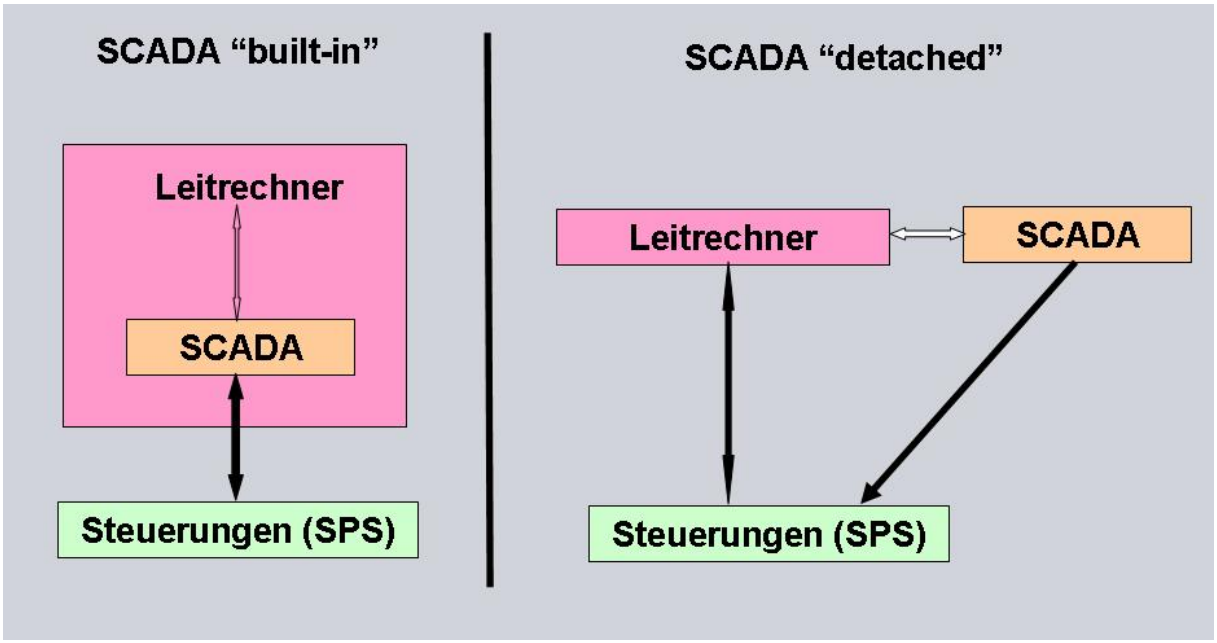

Abb. 4: Alternativen für die SCADA-Anbindung

Aufgabenverteilung zwischen SPS und Leitsystem

In vielen Systemen (Post-Sortieranlagen, viele Lager) ist eine „einfache“ Geometrie der Förderwege vorhanden, die völlig autark von den Steuerungen (SPS) verwaltet werden. Dies ist für das Leitsystem am bequemsten und wird vor allem dann angestrebt, wenn auch das SCADA-System losgelöst vom Leitsystem betrieben wird (beispielsweise mit SIMATIC WinCC™ oder bei autonomen Sorter-Steuerungen und Flurförderzeugen).

Bei komplexer Anlagengeometrie fällt dem Leitsystem auch die Aufgabe der **Wegefindung** zu. Hier ist das Leitsystem vom momentan vorliegenden Anlagenzustand abhängiger, benötigt mehr Zustandsdaten und hat eine größere Last zu bewältigen als bei „einfacher“ Anlagengeometrie. Solche Anforderungen treten häufig bei der Abwicklung von Fluggepäck oder bei einer komplexen Lagervorzone auf. In diesem Fall ist ein mit dem Leitsystem integriertes SCADA-System (etwa LCMBASE™) vorzuziehen. Die SPS-Systeme können nun „einfacher“ projektiert werden; in vielen Fällen lassen sich dann wesentliche Bausteine der SPS aus den Anlagen-Stammdaten oder sogar aus der Anlagenprojektierung (Mechanik) vorab automatisch generieren.

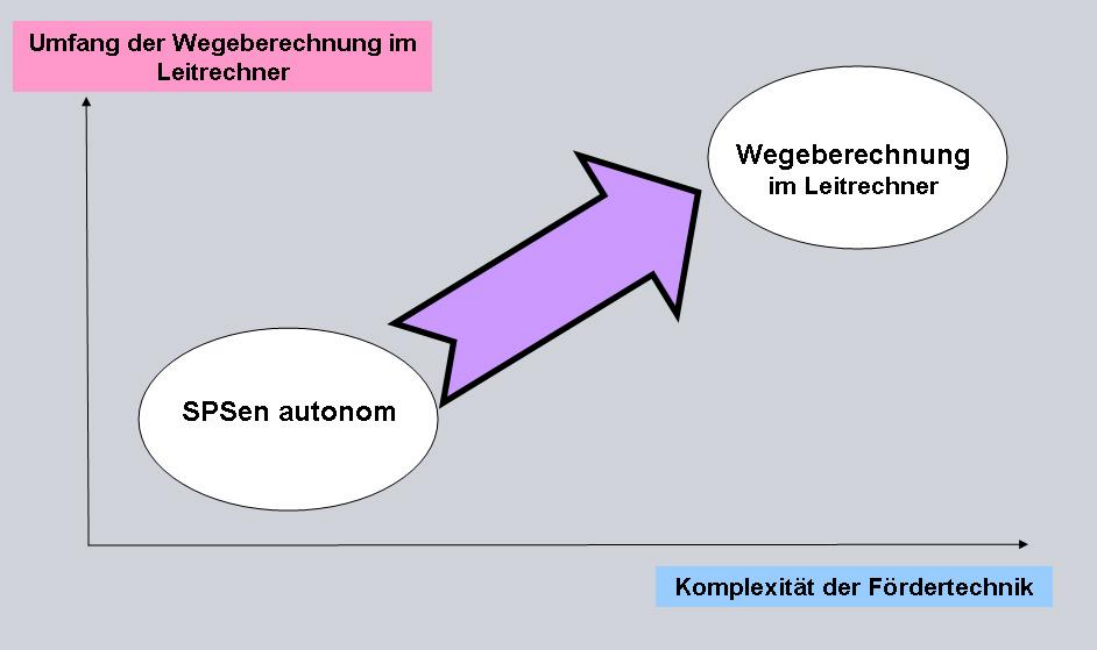

Abb. 5: Aufgabenzuordnung zwischen Leitsystem und SPSen

Das folgende Bild zeigt den Entwurf einer Kopfstation für eine Hochgeschwindigkeitsstrecke in einer Sortieranlage für Fluggepäck, einen kleinen Ausschnitt aus etwa 60 Kilometer Förderstrecken.

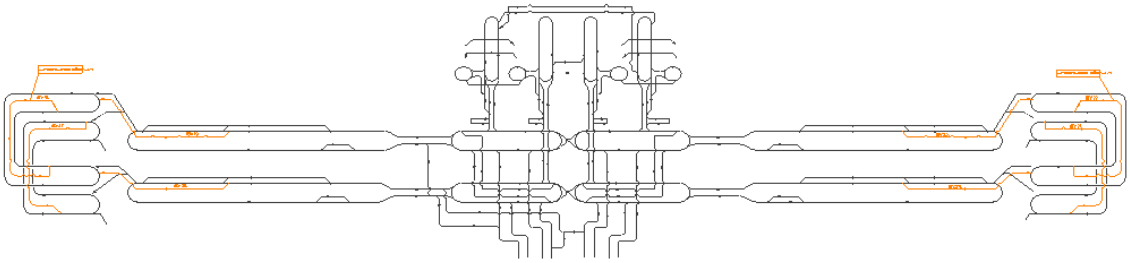

Abb. 6: Komplexe Fördertechnik: Kopfstation Hochgeschwindigkeitsstrecke einer Sortieranlage für Fluggepäck

Zurück zum **Leitsystem**:
Letztendlich gibt es noch die Schnittstellen nach „oben“, in ein Waren-Dispositionssystem, die Flugplanung, Archive aller Art und übergreifende Datenbestände. Über diese werden sowohl die Resultate der Materialfluss-Steuerung (Beispiel: Zielmeldung eines Reisekoffers, Verpackung einer Lieferung) gemeldet als auch die Auftragsdaten der Materialströme (Checkin des Reisekoffers, Kundenauftrag) abgewickelt.

Design-Überlegungen zur Datenbank

Für die Datenbank-Tabellen ergeben sich für die vorliegenden Anwendungen folgende Design-Überlegungen:

- Stammdaten können nach Bedarf (auch) denormalisiert vorliegen

Da häufig viele Stammdaten-Informationen häufig in der gleichen Zusammenstellung an die Bewegungsdaten gejoint werden, ist es ratsam, die Stammdaten (auch) so in der Datenbank zu hinterlegen, wie sie gebraucht werden, und nicht (nur) „schön“ normalisiert.

- Stammdaten brauchen (fast) keine Constraints

Die Stammdaten sind, wie schon dargestellt, weitgehend fest. Constraints aller Art sind nutzlos und verbrauchen nur unnötig Ressourcen.

- Stammdaten benötigen (fast) keine Archivierung oder Datensicherung

Zur Installation der Stammdaten hat sich das Erstellen geeigneter SQL-Skripten als einfachste Möglichkeit erwiesen – diese Skripten stellen gleichzeitig ein Archiv des Stammdaten-Bestandes dar. Mit derartigen SQL-Skripten werden auch normalisierte und denormalisierte Stammdaten aufeinander abgebildet.

- Bewegungsdaten sollten immer normalisiert vorliegen

Die Pflege nicht-normalisierter, redundanter Bewegungsdaten, insbesondere von Bestandszählern aller Art, ist auf Anwendungsebene extrem fehleranfällig. Werden derartige redundante Daten aus Performance-Gründen benötigt, so werden diese ausschließlich über datenbank-interne Trigger gepflegt. Ansonsten sollen Business Cases nicht in Datenbank-Triggern „versteckt“ werden.

- Auch für Bewegungsdaten sind referentielle oder Check-Constraints eher überflüssig (bis schädlich).

Wenn die Stammdaten sauber definiert und die Schnittstellen sauber getestet sind, dann können nirgendwo fehlerhafte Daten entstehen (Beispiele: Falsche Anlagen-Kennungen, Arbeitsplatz-Bezeichnungen, Flags). In den meisten Fällen wäre ohnehin völlig undefiniert, was bei Constraint-Verletzungen passieren soll. Wo keine Semantik (Bedeutung), da keine Funktion. Das Meta-Wissen über die Konsistenz der Daten muss in den Stammdaten und den Funktionen bereits vorhanden sein, also ist die entsprechende Überprüfung sinnlos. Andererseits gewinnt man ohne referentielle Constraints die Freiheit, die DML-Statements innerhalb einer Transaktion geeignet zu sortieren. Das ist insbesondere dann wichtig, wenn die Pflege von redundanten Zählern in allen Transaktionen die nötigen Satzsperren in einer vorgegebenen Reihenfolge erstellen muss, weil andernfalls (lästige und) unnötige Deadlocks entstehen. Wären hier referentielle Constraints vorhanden, so würden die Sätze unter Umständen in einer unerwünschten Reihenfolge gesperrt.

- Tablespaces und womöglich auch Tabellen sollen eine definierte Maximalgröße haben und können nicht „unbegrenzt“ wachsen.

Je kleiner die „überlaufende“ oder besser „an ihre Grenzen stoßende“ Einheit ist, desto leichter ist dieser Fehlerfall zu bearbeiten. Wenn es erst kracht, sobald alle Platten voll sind, dann kann man sich unter Umständen nicht mehr als normaler Nutzer zur Fehlerbehebung einloggen – was dann?

Design-Überlegungen zur Prozess-Architektur

- Es soll keine datenbehaftete Kommunikation zwischen den Anwendungsprozessen (und Masken) geben. Jeder Prozess kann zu jedem Zeitpunkt auf dem Zustand aufsetzen, wie er in den Datenbank-Tabellen vorliegt. Dies reduziert die möglichen Telegramme auf zwei Spezialfälle:

a) WAKEUP: Aufwachen und aufgrund des Zustandes der Datenbank-Tabellen tätig werden. (Schon diese Definition der Prozess-Tätigkeit legt fest, dass die Prozesse im Wesentlichen als **Funktionsbausteine** definiert sind und nicht als **Objekt-Methoden**).

b) PARAM: Lies die für die Funktion nötigen Parameter (Stammdaten) neu ein. Die entsprechenden Funktionen muss es ohnehin geben, da sie beim Start der Anwendung durchlaufen werden müssen. Dadurch werden alle Prozesse selbststartend auf jedem beliebigen (konsistenten) Datenbestand.

- In keinem Prozess sollen DDL-Aktionen vorkommen. Ausnahme: Index-Reorganisation (SQL-Job) und TRUNCATE (wenn deutlich besser als alles Andere).

- Jeder Prozess hat einen Überwachungsprozess (Keep-Alive-Kontrolle). Jeder Prozess kann zu jedem Zeitpunkt nachgestartet werden. Prozesse können bei Bedarf „partitioniert“ werden, in dem Sinn, dass ein Prozess mehrfach auf verschiedenen Datenbereichen gestartet werden kann.

Design-Überlegungen zu den Bearbeitungsmasken

- Alle einfachen und übersichtlichen Lösungen sind auch die effizientesten und wartungsfreundlichsten. In diesem Sinne sollen technologisch bedeutungslose Tools und Hilfsprozesse vermieden werden. Dies ist konkret eine Absage an „Kommunikations-Agents“, Middle-Tiers, abstrakte Datenbank-Layer und unnötige Message-Handler. In allen unseren Projekten hat sich eine herkömmliche **Client-Server-Architektur** zwischen den Masken (und Prozessen) einerseits und der Datenbank andererseits als einfachste, wartungsfreundlichste, performanteste und robusteste Lösung erwiesen.

- (Fast) keine der Bearbeitungsmasken genügt der **FORMS**-Systematik „Pflegemaske einer/weniger Tabelle(n)“. Vielmehr ist in (fast) jeder Bearbeitungsmaske „Business Logic“ enthalten, die nicht (einfach) direkt auf die Datenbank-Tabellen abbildbar ist.

- Die Realisierung der **Masken** in einer betriebssystem-unabhängigen Sprache wie JAVA und einem robusten SQL-Treiber (JDBC oder JAVA Programmer) hat sich nach und nach als kostengünstigste und stabilste Entwicklungsplattform für die Bedienmasken erwiesen.

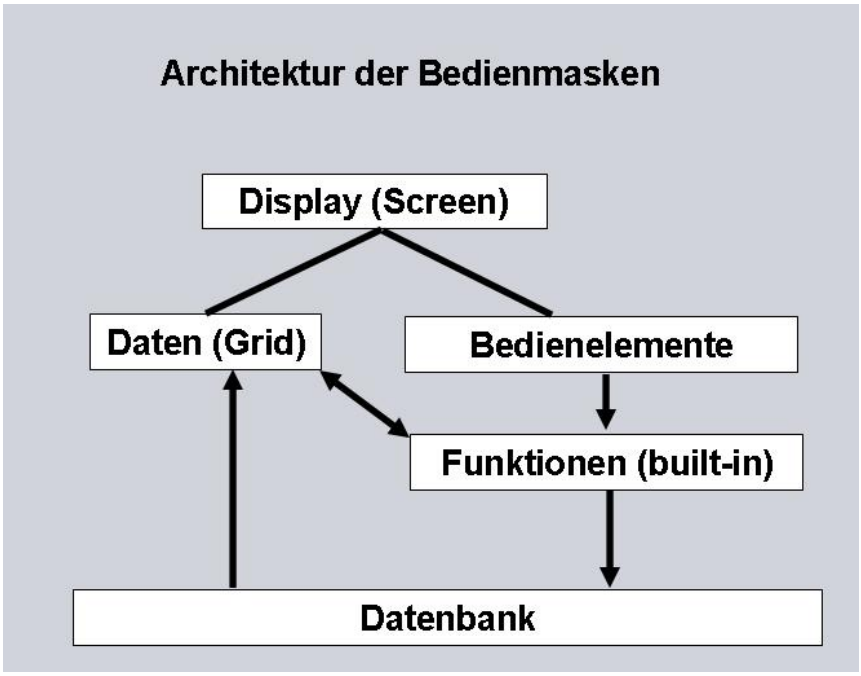

Abb. 7: Architektur der Bedienmasken

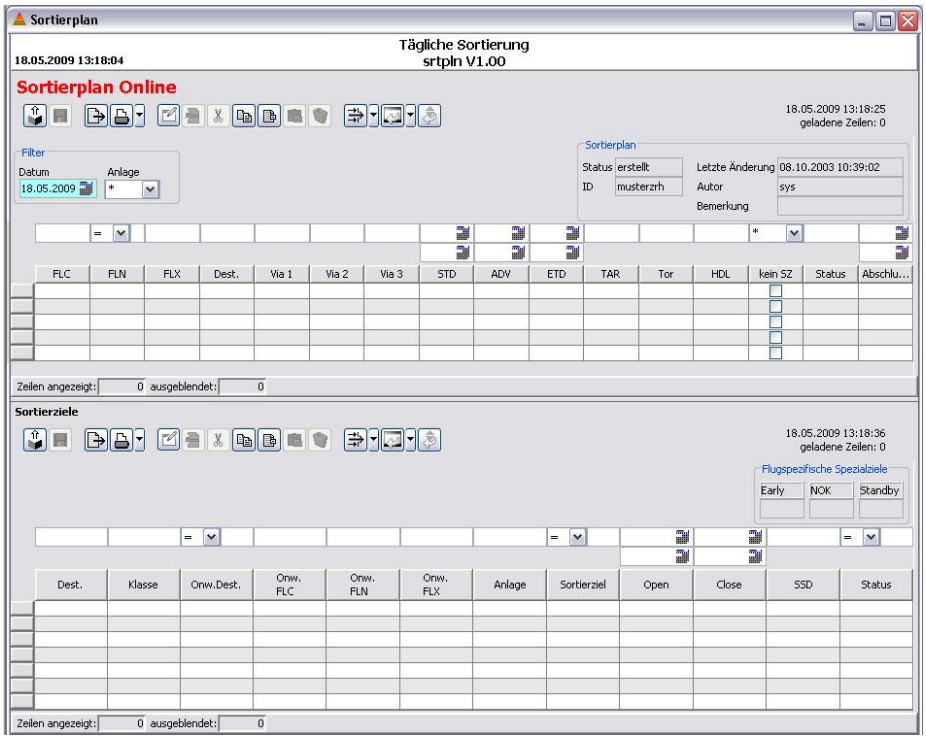

Abb. 8: Beispiel-Bedienmaske

Erfahrungs-getestete Regeln (nicht vollständig)

Löse jede Aufgabe auf dem primitivsten Level, das sie performant und wiederanlaufsicher erledigt.

Scheue (womöglich dynamisch auf der Anlage und interaktiv) generierte Software. Was nicht zu einem definierbaren Zeitpunkt als Textdatei vorliegt, die mit einem handelsüblichen Allerwelts-Editor editiert werden kann, ist kein „Source-Code“ im herkömmlichen Sinn und wird als Ergebnis der Programmierung nicht akzeptiert.

Software überprüft nicht Software: Keine systeminterne Prüfung auf Konsistenz, keine Verifikation von Übergabeparametern, kein Wildwuchs bei der Aufruftiefe. Keine Metalevel innerhalb des Systems, wenig bis kein generierter ausführbarer Code (aber: generierte Includes, Defines aller Art, Puffer, Bindevariable, … in der Entwicklungsumgebung – und nur dort).

Du sollst keine Datenhaltung außerhalb der Datenbank haben. Alle Daten sind in der Datenbank und sind nach jeder Transaktion konsistent.

Möglichst viel, was nicht der Steuerung des Materialflusses dient, sollte man aus der Anwendung und der Produktiv-Datenbank auslagern, insbesondere Langzeitarchive und Management-Reports.

Der Systemarchitekt bestimmt die Prozesskonfiguration und das gesamte Datenbank-Design, ist also auch der DBA. Keinesfalls darf jeder die DB-Objekte definieren, die er zu benötigen glaubt.

Der Systemarchitekt bestimmt auch die Nutzung spezieller Optionen. Es soll versucht werden, mit dem Funktionsumfang der Standard Edition auszukommen. Keinesfalls darf jeder ungefragt alle Konstrukte einsetzen, die er zu benötigen glaubt. (Beispiele: Materialized Views, Partitioned Tables, Functional Indexes …) .

Dokumentation

Dies ist eines der heikelsten Themen der Software-Erstellung. Es gibt keinen Code, der sich „selbst“ dokumentiert. Und am Ende jedes Projektes und jeder Produkt-Version muss noch Zeit und Geld für die Dokumentation vorhanden sein.

Voraussetzung ist, dass man komplizierte Tricks scheut wie der sprichwörtliche Teufel das Weihwasser. Code, den der Ersteller der Software nicht unmittelbar (auch nach einem oder mehreren Jahren) einsieht und ein Kollege schon zweimal nicht, verbietet sich von selbst.

Unabdingbar ist eine „Entwickler-Dokumentation“: Was habe ich wofür gemacht? Welche Layer gibt es in meiner Software? Welcher Prozess/Ablauf benötigt welche Funktion? Es sind „nebenbei“ schon viele Bugs bei der Anfertigung einer solchen Dokumentation entdeckt worden!

Anwender-Dokumentation: Nicht nur beschreiben, was man wie parametrieren kann, sondern auch, welche internen Abläufe im System damit losgetreten werden.

Dokumentation und übersichtlicher Code sind im weitesten Sinn auch schriftstellerische Tätigkeit. Übersichtlicher Code und lesbare Dokumentation gehen Hand in Hand.

Persönliche Nachbemerkung

Dieser Vortrag ist (auch) mein „Abschiedsgeschenk“ an die DOAG Community - bei der nächsten DOAG-Konferenz werde ich voraussichtlich nicht mehr beruflich tätig sein.

Kontaktadresse:

Dr. Gerhard A. Hergesell
Siemens AG
Colmberger Str. 2
D-90451 Nürnberg

Telefon: +49 (0) 911-145 7266
Fax: +49 (0) 911-145 7926
E-Mail gerhard.hergesell@siemens.com
Internet: www.siemens.de