THINKING SCOPELAND

Während Andere noch damit beschäftigt sind, aufzuschreiben, was sie denn planen zu entwickeln, sind Sie mit SCOPELAND schon fertig.

Inhalt

Einleitung: Was ist SCOPELAND und warum überhaupt? 4
Um welche Art Software geht es? 4
Wie grenzen wir uns ab? 6
Das SCOPELAND-Paradigma 8
Eine scheinbar verrückte Idee… 8
Der Lösungsansatz 8
Und so arbeitet es im Inneren 10
Von oben nach unten? Oder doch lieber von unten nach oben? 12
Das Für und Wider der Persistenz 13
Wie Datenbanken ticken 16
Das Daten-Babylonische Begriffschaos 16
Datenbanken, Tabellen und Felder/Spalten 18
Relationen 19
Die n:1-Relation (Verweis auf eine andere Tabelle) 20
Jede Datenbankabfrage hat immer genau eine Haupttabelle 22
Die Krux mit dem SQL 22
Die 1:n-Relation (Master-Detail) 24
Master-Detail-Relationen in SCOPELAND 24
Die sogenannte m:n-Relation 25
Die Datenbank – ein Netz von Relationen 26
Konsistenz 27
Vorgehensweise zum Aufbau der Datenstrukturen 27
Schlüsselfelder und Dimensionen 29
Die Würfel-Sicht auf die Daten 30
Wie sollen Tabellen und Felder heißen? 30

Indexierung 31
Das Meta-Prinzip 32
Metadaten beschreiben Daten 32
„Tabelle öffnen“ – was ist das? 33
Warum gibt es keine vorgefertigten „Abfragen“ in SCOPELAND? 34
Wie macht man das – „Tabellen einziehen“? 35
Mehrere, miteinander verbundene Tabellen öffnen 36
Auch die Datensichten sind zueinander hierarchisch 37
Bezüge (temporäre Relationen, temporäres Master-Detail) 38
Ad-hoc-Zugriff, auch für ‚Power-User‘ 39
Direkter Zugang zu den Daten? Darf man das denn dürfen? 40
Ordnung in der Datenwelt: Ordner, Freigaben und Rechte 41
End-user Computing (EUC) 42
Was bedeutet eigentlich der Begriff „Scopeland“? 42
Und wie kommt Anwendungslogik in die Metadaten? 43
Anwendungen entwickeln 46
Das Grundprinzip – kompakt im Überblick 46
Seiten und Applets 47
Wie entstehen „Seiten“? 49
Direct Views 49
Mehrere Direct Views in einer Seite 52
Oberfläche bauen = Einzelteile zusammenkleben + Form Design 54
Speicherformat einer Seite – inkl. aller Businesslogik und Oberfläche 54
Zusätzliche Logik ergänzen 57
Objekte in SCOPELAND 58
Pfadausdrücke in Direct Views und Applets 59
Regeln zu Objekten 61
Vordefinierte Zustände und Aktionen 62
UML-Prozesslogik 62
Und was sind „Aktivitäten“? Aktionen oder Zustände? 64
Wie aus Aktionen und Zuständen ein Workflow wird 65
Abgrenzung: Was man so alles unter „Workflow“ verstehen kann… 66
Warum der Workflow immer an einer Tabelle hängt 67
Wenn noch Logik fehlt: Lücken schließen mit Makros und Scripts 68
SCOPELAND für Fortgeschrittene 70
Die „Rahmenanwendung“ 70

Temporäre 1:n-Relationen (Bezüge) 71
Temporäre n:1-Relationen (Verweise) 72
Generierungen von Druckausgaben (Dokumente, Reports oder Datenbereitstellungen) 72
Schnittstellen 74
Datenflüsse: Aggregation, Transformation und vieles mehr 75
Integrierte Kartendarstellung 77
Und immer mehr Features..... 78

Einleitung: Was ist SCOPELAND und warum überhaupt?

Um welche Art Software geht es?

SCOPELAND ist eine konfigurierbare, weitestgehend programmierfreie Entwicklungs- und Laufzeitumgebung für maßgeschneiderte Datenbankanwendungen. SCOPELAND dient dazu, Kosten und Zeit zu sparen und vor allem, Verwaltungssoftware flexibler zu machen, was letztlich zu einer besseren IT-Unterstützung der alltäglichen Arbeit führt.

Aber warum?

Gibt es nicht schon genug Programmiersprachen und Entwicklungsumgebungen? Wozu dann noch etwas speziell für Datenbanken?

Zahlreiche große Unternehmen, Behörden und sonstige Institutionen benötigen immer wieder, gestern wie heute, und vielleicht auch in ferner Zukunft, in erheblichem Umfang maßgeschneiderte Softwarelösungen. Und für solche Individuallösungen sind die üblichen, eigentlich für die Entwicklung von Standardsoftware etablierten Vorgehensweisen viel zu aufwändig und die dafür ausgelegten Entwicklungssysteme einfach nicht optimal.

Individuallösungen unterscheiden sich ganz grundsätzlich von Softwareprodukten. Zunächst einmal haben bei kundenspezifischen Entwicklungen die ‚eigentlichen' Entwicklungskosten einen viel höheren Stellenwert als bei einer Produktentwicklung, wo Vertrieb und Marketing die weitaus größeren Budgets beanspruchen. Außerdem werden Individuallösungen oft von Mitarbeitern entwickelt, die nicht nur IT- sondern auch vielfältige Fachaufgaben haben, und die deshalb nicht in der Lage oder nicht gewillt sind, sich in abstrakte, allzu tief strukturierte Objektmodelle, Quellcodestrukturen und Plattformbesonderheiten hineinzudenken. Sie brauchen viel einfachere Methoden, bei denen die Inhalte und nicht die Technik im Vordergrund stehen. Und letztlich unterliegen kundenspezifische Lösungen einer viel höheren Änderungsfrequenz als Produktentwicklungen, nicht nur weil die Anforderungsbedarfe an sich häufiger sind, sondern auch weil sich die genauen Anforderungen oft erst im Zuge der Entwicklung oder während der späteren Nutzung genau genug herauskristallisieren. ‚Agile Softwareentwicklung' als Projektmanagementmethode allein genügt da nicht. Auch die Software selbst sollte in seinem innersten Kern und Wesen agil sein – nicht starr das umsetzen, was da einmal hineinprogrammiert wurde, sondern für die Veränderung ausgelegt sein.

Die meisten kundenspezifischen Anwendungslösungen sind von ihrer Natur her Datenbank-anwendungen, also Programme, deren Daten in relationalen Datenbanken abgelegt werden und deren Abläufe und Inhalte sich eng an die Natur eben dieser Daten anlehnen. Egal ob unter Windows, im Web oder auf mobilen Plattformen. Dabei geht es nicht nur darum, ob die Programme irgendwie auf Datenbanken zugreifen, denn das tun heute fast alle Programme, sondern ob die Datenbankfunktionalität, also das Ablegen, Wiederauffinden und Verarbeiten strukturierter Daten den Charakter der Programme bestimmt.

Ob CRM-Lösungen, Umweltinformationssysteme, Projektmanagement- oder Controlling-Lösungen, ERP-Systeme oder E-Government, Buchungssysteme oder was auch immer, alles das sind Datenbank-anwendungen; und ein nicht unerheblicher Prozentsatz solcher Anwendungssoftware wird kundenspezifisch aufgesetzt – um genau diese geht es hier.

Datenbankanwendungen sind nicht nur die häufigste Art maßgeschneiderter Software, sie sind auch besonders gut dafür geeignet, mit alternativen Verfahren entwickelt zu werden.

Und eines dieser Verfahren ist die SCOPELAND-Technologie.

Weil SCOPELAND keine Programmiersprache, sondern eher ein Baukastensystem von Fertigkomponenten ist, können Sie damit typische Datenbanklösungen, so wie sie auf dem Markt immer wieder nachgefragt werden, um ein Vielfaches schneller, einfacher, eleganter und auch flexibler entwickeln. Kein Vergleich zum aufwändigen Codieren hunderttausender einzelner Programmbefehle.

Möglich ist das aber nur dann, wenn es sich wirklich um Datenbankanwendungen handelt, denn diese bestehen erfahrungsgemäß aus immer denselben Grundfunktionen. Verglichen mit anderen Programmen wie beispielsweise Spielen, Routenplanern oder Mediaplayern, Simulations- oder Analysesoftware, wirken Datenbankanwendungen eher nüchtern, unspektakulär und „trocken“. Angesichts der unverzichtbaren Multiuser-Fähigkeit und Transaktionslogik, der hohen Datenschutzanforderungen, ganz spezifischer Bedienabläufe, Berechtigungskonzepte und Ergonomieregeln ist die Entwicklung solcher Programme aber dennoch keinesfalls trivial.

Typisch ist auch, dass bei Datenbankanwendungen nicht die besonders anspruchsvollen Programmfeatures die Kosten bestimmen, sondern eher die große Menge der einfacheren Programmteile. Einigen wenigen kniffligen Algorithmen stehen oft hunderte eher einfach aufgebaute Bildschirmseiten gegenüber. Will man Kosten sparen und Flexibilität gewinnen, dann geht es also weniger darum, die Entwicklung der schwierigen Programmteile zu erleichtern, sondern mit hoher Effizienz das Projekt in seiner vollen Breite abzuwickeln.

Effizienz im Elementaren statt Kunstfertigkeit im Besonderen.

Deshalb ist SCOPELAND gerade darauf ausgelegt, das Einfache, das Normale, so effizient wie möglich abzuarbeiten. Es geht darum, große Anwendungen in voller Breite vollständig oder weitestgehend ohne Programmierung umzusetzen. SCOPELAND zeichnet sich dabei durch Einfachheit und Schnelligkeit der Entwicklung sowie durch besondere Flexibilität aus.

Typische Bildschirmmasken mit einer Vielzahl an Eingabefeldern, diversen Funktionselementen, eingebetteten Tabellen, Registerordern und Schaltflächen aller Art, also einfache Dialoge, wie sie aufgrund ihrer großen Zahl die Projektkosten dominieren, lassen sich mit wenigen Mausklicks konfigurieren. Sie sind äußerlich einfach aufgebaut, aber intern dennoch technisch raffiniert umgesetzt.

Trotz der scheinbaren Einfachheit in der Breite, geht es meist um anspruchsvolle und komplexe Datenbankanwendungen, die im Multiuser-Betrieb mit Dutzenden, Hunderten oder gar Tausenden von gleichzeitigen Anwendern auf ein und denselben Daten sicher, stabil und performant laufen müssen.

Der Hauptfokus liegt nicht auf der Oberfläche,
sondern in der effizienten und sicheren Datenhaltung und -verarbeitung.

Und die Programmoberfläche folgt ebendieser Aufgabenstellung.
Immer getreu dem Motto:

Form follows function.

Wie grenzen wir uns ab?

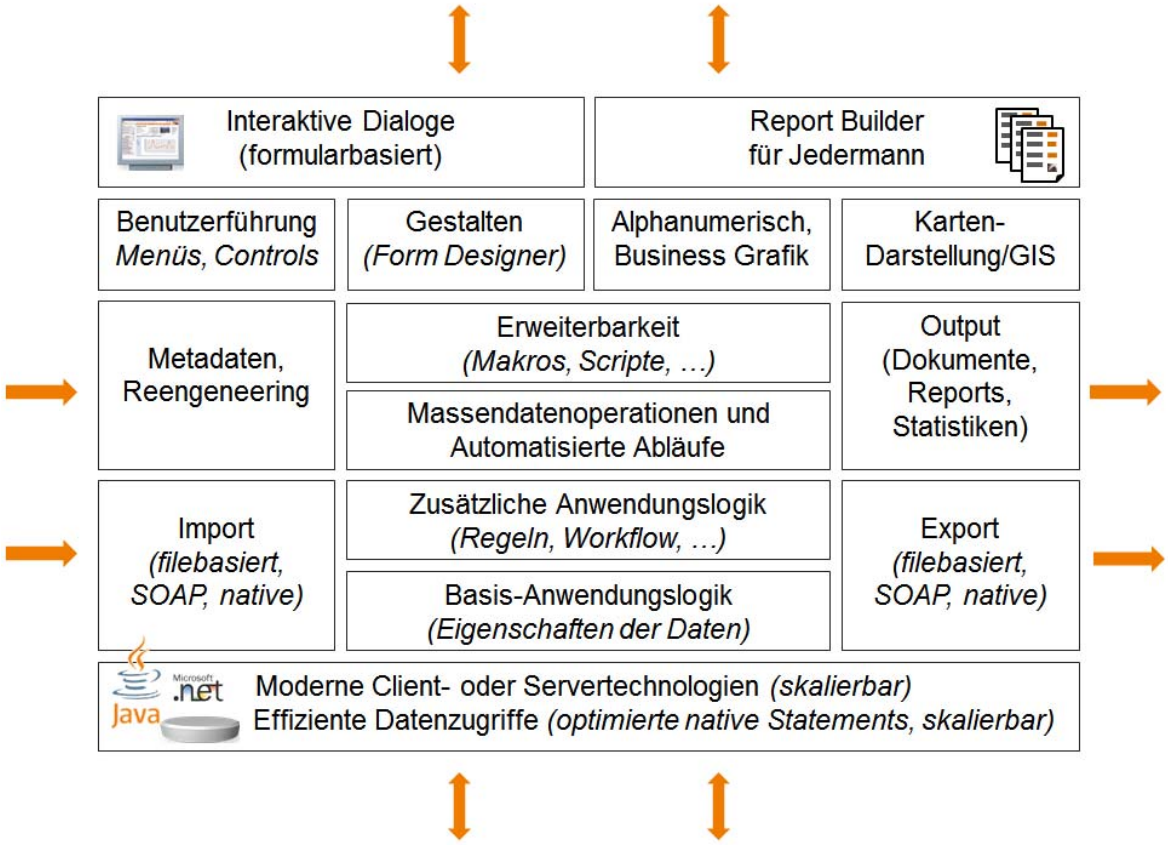
Für die Entwicklung von Fachanwendungen für die Verwaltung und anderer Datenbanklösungen konkurriert die SCOPELAND-Technologie ganz bewusst mit den etablierten Mainstream-Verfahren zur manuellen Softwareentwicklung. Es geht dabei nicht darum, die Programmierung als solche infrage zu stellen – schließlich ist das Produkt SCOPELAND selbst ja auch zu großen Teilen durch Programmierung entstanden, so wie es für Standardsoftwareprodukte selbstverständlich ist.

Nein, hierbei geht es ganz konkret und ausschließlich darum, das Kosten-Nutzen-Verhältnis echter Individuallösungen zu verbessern. Überall da, wo man keine wirklich passenden oder nur sehr teure Standardlösungen am Markt vorfindet, wird individuell Software entwickelt. Anders als bei echten Produktentwicklungen mit hohen Marketing- und Vertriebskosten, fallen hierbei die eigentlichen Entwicklungskosten und -zeiten viel mehr ins Gewicht. Ein Ziel von SCOPELAND ist deshalb, die Implementierungskosten und Projektlaufzeiten drastisch zu reduzieren und auf diese Weise das Kosten-Nutzen-Verhältnis der zu entwickelnden Fachanwendungen deutlich zu verbessern. Dabei geht es auch darum, Entwicklungsprozesse unabhängiger zu machen von konkreten Programmierer-Persönlichkeiten und die Softwarepflege von den IT-Dienstleistern, die die Programme einst entwickelt haben. Auch deshalb sind programmierfreie Lösungsansätze in diesem Umfeld die bessere Wahl.

Es geht aber nicht nur um Kosteneinsparungen. Mindestens ebenso wichtig ist das gute Ergebnis. Ein gutes Ergebnis erreicht man aber nur dann, wenn die entwickelte Software tatsächlich genau das abdeckt, was benötigt wird und nicht nur das, was mal irgendwann vorher jemand aufgeschrieben hatte. Es ist sehr schwer, fast unmöglich, im Vorfeld alle Anforderungen vollständig und korrekt zu erfassen und alle noch kommenden Änderungsbedarfe vorherzusehen. Deshalb ist agile Softwareentwicklung unvermeidlich. Wirklich agil aber kann man nur sein, wenn die Softwaretechnologie dies auch hergibt. Es reicht nicht, das Projektmanagement agil zu gestalten; die Software selbst muss agil sein, flexibel anpassbar in nahezu jeder Hinsicht – im Idealfall per Mausklick. So und nur so gelingt es, die entstehende Software evolutionär auf den tatsächlichen Bedarf hin zu optimieren und kontinuierlich an alle sich ändernden Anforderungen anzupassen. SCOPELAND-Lösungen sind an sich flexibel. Die Veränderung ist eine Kernfunktionalität von SCOPELAND.

Um die beiden Hauptziele (Kosten senken und mehr Flexibilität in der Anpassbarkeit an sich präzisierende oder ändernde Anforderungen) zu erreichen, setzt SCOPELAND darauf, Individualprogrammierung nicht nur zu unterstützen, sondern mehr oder weniger überflüssig zu machen.

Es geht darum, maßgeschneiderte Lösungen per interaktiver Konfiguration mit ingenieursmäßigen Methoden zusammenzubauen. Industrialisierung statt Handwerk.



Individuallösungen ja - Individualprogrammierung nein!

Abbildung 1: Logische Architektur
Übersicht über die Kernfunktionen und -module. SCOPELAND deckt den funktionalen Gesamtumfang typischer Verwaltungslösungen mit vorgefertigten Funktionalitäten ab, mit dem Anspruch, durch „Konfigurieren statt Programmieren" komplette Anwendungssoftware weitestgehend ohne Programmierung abzudecken.

SCOPELAND dient zudem auch noch dazu, ausgewählten Entwicklern, Power Usern und Endanwendern einen viel einfacheren Zugang zu „ihren" Daten zu ermöglichen, z. B. um auf einfachst denkbare Art und Weise Datenbanken auszuwerten, Reports zu erstellen, in den Datenbeständen frei zu navigieren und – falls man selbst der Datenherr ist – ggf. sogar konsistent auf direktem Wege Daten zu prüfen und zu korrigieren. In diesem „Ad hoc-Modus" bewegt man sich als Anwender stets exakt in seinem „Scope". Man sieht genau das, was man mit seinem Rechteprofil sehen darf und nichts anderes. Und man darf genau das tun, was man mit seinem Rechteprofil mit den Daten tun darf. Jeder Benutzer, dem man diese Freiheit einräumen möchte, soll sich in seinem Scope so frei wie möglich bewegen und nicht nur im engen Rahmen vorgefertigter Suchfunktionen und Bildschirmmasken auf seine Daten schauen dürfen. Davon leitet sich auch der Name SCOPELAND ab – eine effizientere Arbeitsweise für mündige IT-Anwender. Ein Freiraum, den man ausgewählten Fachanwendern gestatten kann, aber nicht muss. Die SCOPELAND-Anwendung kann auch, wenn man dies möchte, genauso konservativ und restriktiv daherkommen, wie Sie es von anderen Programmen gewohnt sind.

Das SCOPELAND-Paradigma

Eine scheinbar verrückte Idee…

SCOPELAND ist ein Bausteinsystem von Fertigfunktionalitäten, aber kein Set aus fertigen Programmmodulen, denn die Welt ist zu vielfältig für eine überschaubare Anzahl an Modulen.

Um Programmierfreiheit in der Softwareentwicklung überhaupt ermöglichen zu können, benötigt man vorgefertigte Softwarebausteine und -funktionen. Angesichts der Vielfalt der abzubildenden Inhalte kann man dies nicht mit groß zugeschnittenen Fertigmodulen erreichen.

Ein Bausteinsystem für Datenbankanwendungen aller Art kann nur dann funktionieren, wenn man nicht inhaltlich definierte Bausteine (wie z.B. ein Buchungsmodul), sondern technische Funktionen (z.B. Programme zur Prüfung gegen Plausibilitätsregeln, Auswahlfunktionen oder Ereignisauslöser) als vorgefertigte Elemente verwendet. Diese sind dann so vorprogrammiert, dass sie dasselbe tun wie in jedem anderen Programm, aber nicht für die jeweils ganz speziellen Daten, sondern für beliebige.

Aber auch dann ist die Vielfalt immer noch zu groß, um sie als Entwickler hinreichend gut überschauen zu können – es geht immer noch um etwa eintausend unterschiedliche Programmfeatures. Will man maßgeschneiderte Datenbankanwendungen ohne Programmierung interaktiv zusammenklicken, dann kann dies nur effizient funktionieren, wenn sich die Bausteine gegenseitig kennen und wenn sie von selbst wissen, wann sie wo und wie benötigt werden.

SCOPELAND verwendet also Bausteine, die nicht nur von selbst wissen, wie sie etwas tun, sondern auch, wann und wo sie gebraucht werden.

Es sind also Bausteine, die sich auch noch selbst zusammenfinden!

Kann so etwas funktionieren?

Der Lösungsansatz

SCOPELAND setzt auf kleinteilige, elementare Funktionalitäten, wie sie in jeder typischen Datenbankanwendung und Verwaltungslösung vorkommen; und die Entwicklungsumgebung führt den Benutzer nun so, dass in Abhängigkeit von den konkreten Datenstrukturen und -eigenschaften stets die jeweils geeigneten oder notwendigen Funktionalitäten automatisch aktiviert bzw. zur Auswahl angeboten werden. Dass das so möglich ist, beruht einzig und allein auf der Tatsache, dass typische Datenbankanwendungen trotz grundlegend anderer Inhalte doch stets dieselben üblichen Funktionselemente verwenden. Beispiele für diese allgemein üblichen Funktionselemente sind Masken- und tabellarische Darstellungen von Datensichten mit Eingabe- oder Auswahlfeldern, Master-Detail-Darstellungen, Prüfung von Plausibilitätsregeln, Ausführung von Berechnungen, statusabhängige Aktionen (Workflow) und vieles mehr.

Die verwendeten Daten, ihre Eigenschaften und Verknüpfungen zueinander, ihre spezifischen Besonderheiten in der Anwendungslogik und die Art und Weise, wie sich die Daten auf der Benutzeroberfläche präsentieren sollen, all dies wird deklarativ als sog. „Metadaten“ in einer Metadatenbank abgelegt. Die Metadatenbank ist eine zusätzliche Datenbank, meist auf demselben Datenbankserver abgelegt, in der auch die wichtigsten Anwendungsdaten liegen. Besondere Serverkomponenten sind nicht erforderlich.

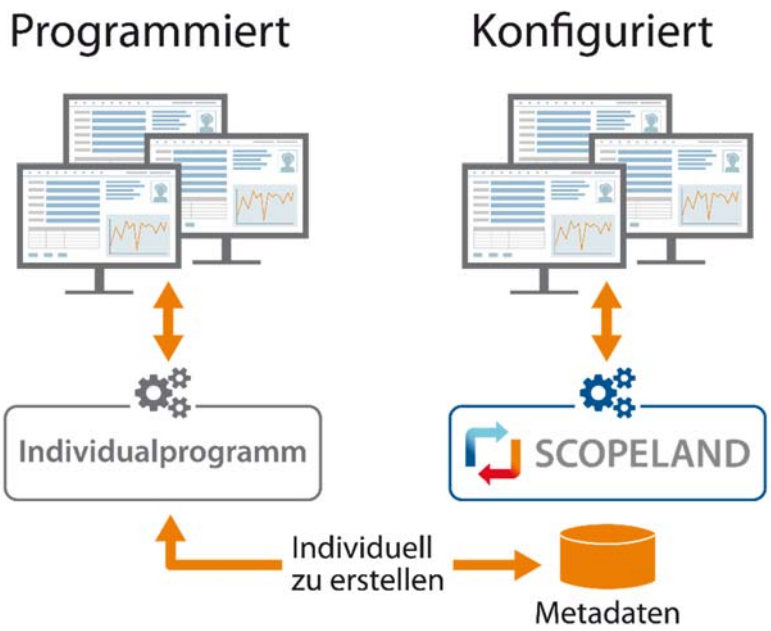


Abbildung 1: Technologie

Alternative zum klassischen Programmieransatz (links), bei dem die gesamte eigentliche Software individuell erstellt werden muss (wie eine teure, für den Einzelfall angefertigte Spezialmaschine). Ähnlich einem Roboter, der sein Verhalten aus Steuerdaten ableitet, so passt sich die „Universal Application" SCOPELAND 6 (rechts) intelligent und selbsttätig an all das an, was in den Metadaten hinterlegt wurde. Aus Sicht des Benutzers macht SCOPELAND (bzw. der daraus generierte Code) dann dasselbe wie das Spezial-programm, es hat aber nur ein Bruchteil dessen gekostet.

Die letztlich auszuführende Software wird nicht erst entwickelt, sie ist bereits vorhanden. Alle die genannten typischen Funktionalitäten sind bereits als Teil des Produkts SCOPELAND ausprogrammiert worden, nur eben in universalisierter Form. Sie passen sich an die konkreten Datenstrukturen und Eigenschaften der jeweiligen Daten, wie sie in den Metadaten beschrieben sind, eigenständig an. Die Algorithmen, aus denen jedes Programm besteht, sind bereits vorhanden, in Form der Standardsoftware SCOPELAND. Sie benötigen nur noch (deklarative) Steuerdaten, die ihnen sagen, worauf sie mit welchen Ausprägungen anzuwenden sind. Die Abläufe selbst und wann diese wo und wie angestoßen werden müssen, das weiß SCOPELAND von selbst.

Die Entwicklung von Anwendungen erfolgt mittels der Windows-basierten Client-Software SCOPELAND Direct Desk. Im Entwicklungsprozess können die gerade interaktiv zusammengebauten Anwendungen in einem Interpretermodus auch direkt ausgeführt werden. Sie reagieren immer sofort auf jede Art der Änderung an den Metadaten, was eine enorm flinke Arbeitsweise ermöglicht. Unter Umständen können bestimmte Anwendungen auch direkt im Direct Desk produktiv eingesetzt werden, was vor allem dann sinnvoll ist, wenn man die einzigartigen Möglichkeiten des Ad-hoc-Zugriffs, kombiniert mit der durchaus vorteilhaften Windows-Fenstertechnik und andere Vorteilen von Windows-basierter Software ausnutzen möchte.

Größere Anwendungen hingegen werden meist für eine der Zielplattformen WinForms.NET, ASP.NET oder Java Enterprise Edition (JEE) entwickelt. Hierzu ist in den Direct Desk ein Codegenerator eingebunden, der beim Speichern einer Seite, einer Druckausgabe oder Schnittstelle, also beim Speichern einer Anwendungskomponente, automatisch aus den jeweiligen Metadaten heraus äquivalenten Programmcode für die jeweilige Zielumgebung generiert. In diesem Fall dient der Interpretermodus lediglich der Unterstützung der Entwickler, z. B. für erste schnelle Funktionstests.

Und so arbeitet es im Inneren

Überall da, wo sonst Programmierer immer wieder erneut Code schreiben mit einer konkreten Bedeutung wie ‚hole die Daten aus Spalte x in Tabelle y', haben die Entwickler des Produkts SCOPELAND ***einmal für alle*** geschrieben ‚hole die Daten, die hier benötigt werden'. Welche Daten das sind, das ergibt sich aus dem jeweiligen Kontext von selbst.

> *Beispiel: statt immer wieder erneut den ganzen Ablauf zu programmieren, um mittels einer Listbox auf einer Bildschirmmaske für ein Datenfeld einen konkreten Wert aus einer Katalogtabelle auszuwählen, ca. 100 Zeilen Programmcode, haben die SCOPELAND- Entwickler das nur einmal getan, und zwar so, dass dies auf jedes beliebiges Datenfeld anwendbar ist. Wo die möglichen Auswahlwerte herkommen, das weiß das Programm von selbst, denn sie können ja nur aus einer Tabelle kommen, die mit dem betreffenden Feld über eine n-zu-1-Relation verbunden ist. Um diese Funktionalität auf ein Datenfeld anzuwenden, benötigt man als Anwendungsentwickler keine hundert Zeilen Programmcode, sondern nur zwei simple Konfigurationseinstellungen: eine für die Relation zwischen den beiden Tabellen und eine zweite für die Auswahl der gewünschten Methode (hier „Listbox").*

Die konkreten Algorithmen dafür sind alle schon fertig – im Produkt SCOPELAND in universalisierter Form enthalten.

Prinzip ‚Roboter statt Spezialmaschine'.
*Man muss dem Programm nur noch sagen, **was** es tun soll.*
***Wie** das zu tun ist, das weiß es von selbst.*

In diesem Sinne haben die Entwickler des Produkts SCOPELAND nun alle nur denkbaren Standardfunktionalitäten wie z.B. diverse Formen von Dateneingaben, Plausibilitätsprüfungen, Master-Detail-Darstellungen, Maskenformatierungen, Konsistenzprüfungen und vieles mehr einmal implementiert, einmal für alle. Mehr als tausend Features. Alle sind sie so gebaut, dass sie von selbst wissen, was sie tun müssen – und sogar, an welchen Programmstellen sie möglich und sinnvoll sind.

Die SCOPELAND-Idee geht aber noch weiter als nur einen Bausteinkasten anzubieten. Sie setzt auf einer Erkenntnis auf, die man mit dem etwas vereinfachenden Spruch…

‚Ein gutes Datenmodell ist die halbe Anwendung'

…gut umschreiben kann. Tatsächlich ist es häufig so, dass das Datenmodell, also die Strukturierung der in der Datenbank gespeicherten Daten, eine bemerkenswerte Analogie aufweist zu den Grundstrukturen des zu entwickelnden Programms. Ein professionell entwickeltes und gut dokumentiertes Datenmodell ist für einen guten, erfahrenen Programmierer eine ausreichende Grundlage, um ohne weitere Informationsquellen eine erste, halbwegs brauchbare Version des darauf aufbauenden Programms zu schreiben. Wenn dem so ist, warum kann das dann nicht der Computer allein tun? Alles, was ein Mensch ohne weitere Informationen kann, sollte ein Computer auch können.

SCOPELAND ist nun ein Programm, welches, aufbauend auf einem gut dokumentierten Datenmodell, genau dies tut: es baut aus vorgefertigten Bausteinen, die es auch noch selbst verwaltet, völlig

eigenständig eine erste Version der jeweils benötigten Programme zusammen. Im einfachsten Fall, nämlich dem einer simplen, aber konsistenten Datenerfassung gemäß der relationalen Strukturen in der Datenbank, ist tatsächlich nichts weiter erforderlich. Sie geben dem Computer lediglich eine gut beschriebene Datenbank, also ein dokumentiertes Datenmodell, und sie bekommen ein fast fertiges Programm zurück.

Das ist kein Zauberwerk. SCOPELAND baut einfach alles in das Programm ein, was laut Datenmodell im jeweiligen Kontext sinnvoll und möglich sein könnte: Verknüpfungen zwischen den relational verbundenen Tabellen, Formular- und Tabellenansichten, Einfügen, Löschen, Editieren, Auswahlfunktionen, Speichern, Master-Detailansichten, Prüfen von Wertebereichen und anderen Plausibilitätsregeln, sowie sonstige Konsistenzprüfungen, Datenexport- und -importschnittstellen und vieles mehr – alles, was man aus einem dokumentierten Datenmodell so herauslesen könnte. Und das ist beachtlich viel – und damit kann man tatsächlich schon produktiv arbeiten.

Leider aber ist die Wirklichkeit doch noch etwas anspruchsvoller als es ein Computer jemals vermuten könnte. Geprägt von ihren Erfahrungen mit anderen Softwareprodukten erwarten Endanwender häufig ein viel höheres Maß an Ästhetik und Bedienkomfort. Sie erwarten eine selbsterklärende Benutzerführung und ein viel höheres Maß an Sicherheit gegen inhaltliche Fehleingaben. Sie wollen besonders komplexe, bedingte Plausibilitätsregeln und an so mancher Stelle auch eine automatische Datenverarbeitung. Und sie erwarten vordefinierte Workflows, die besagen, in welchem Zustand der Daten was erlaubt ist und wie sich im Anschluss daran der Status bestimmter Daten ändert.

All diese Dinge haben zur Folge, dass doch noch Etliches zu tun ist, ehe man das zunächst computergenerierte Programm an typische Endanwender ausliefern bzw. ausrollen kann. Zu diesem Zweck sind diverse weitere „optionale“ Features ebenso vorprogrammiert vorhanden, aber man muss sie natürlich noch gezielt auf die jeweiligen Daten anwenden. Das geht im Prinzip ganz einfach, indem man aus den jeweiligen vorprogrammierten Möglichkeiten eine geeignete auswählt; und indem man Regeln formelartig dazuschreibt, die das Programm fortan automatisch berücksichtigt.

Die Arbeit des SCOPELAND-Entwicklers besteht also im Wesentlichen darin, die Generierung der automatischen Entwürfe so zu steuern, weiter auszubauen und umzugestalten, dass sich das Programm Schritt für Schritt immer weiter dem gewünschten Ergebnis annähert.

Damit ist aber nicht gemeint, dass er das zuvor generierte Programm manuell umschreiben muss, und er braucht dazu auch keine Programmiersprache. Es ist eher so, dass er mit den Augen des späteren Anwenders auf das Programm schaut und dieses interaktiv so lange verfeinert und verschönert, bis es schließlich den gestellten Anforderungen genügt.

Deshalb besteht das Erlernen von SCOPELAND auch nicht so sehr im Kennenlernen bestimmter Funktionen, sondern mehr darin, verstehen zu lernen, wie man diesen „Roboter“ dazu bringt, das zu tun, was man will.

Die Arbeit des SCOPELAND-Entwicklers bewegt sich nicht nur auf der Programmoberfläche: Es geht auch darum, die Applikationslogik interaktiv kontinuierlich auszubauen und zu verfeinern. Und häufig besteht sie auch darin, dabei auch das Datenmodell zu präzisieren und immer weiterzuentwickeln. Selbst dann, wenn man schon am Ausfeilen der Programmoberfläche ist, kann man immer noch bedenkenlos die Programmlogik und die Datenstrukturen verändern.

Von oben nach unten? Oder doch lieber von unten nach oben?

Bei SCOPELAND steht die Datenbank immer im Mittelpunkt der Betrachtung. Man kann SCOPELAND im weitesten Sinne als „datenzentrisch“ bezeichnen.

Warum? Fast alle anderen Produkte, die man entfernt mit SCOPELAND vergleichen mag, machen es doch genau andersrum: sie bieten nette Werkzeuge an, um als erstes eine hübsche Oberfläche zu gestalten und diese dann später mit Funktionalität und mit einer Bindung an Tabellen und Spalten aus der Datenbank zu untersetzen oder sogar aus dem so erkennbaren Bedarf heraus passende Datenstrukturen zu erzeugen. Warum macht SCOPELAND das nicht auch so? Ganz einfach deshalb, weil der Entwicklungsaufwand so um ein Vielfaches höher wäre. Wie bereits oben erläutert, lässt sich ein Großteil der Funktionalität aus dem Datenmodell ableiten und ein Großteil der Programmoberfläche entsprechend auch. Auf diesen Effekt wollen wir nicht verzichten.

Und es entspricht so auch besser der Natur der Sache, denn die Daten sind immer das Wesentliche. Ein Programm ist doch letztlich nur ein Fenster, um die Daten zu sehen, bzw. ein Kanal, um mit den Daten zu kommunizieren.

Am Anfang stehen immer die Daten und nicht deren Visualisierung.

Ein und dieselbe Datenbanktabelle wird in komplexeren Umgebungen vielleicht 20, vielleicht 100 Mal in den unterschiedlichsten Zusammenhängen verwendet. Die Daten sind das Primäre. Die Struktur der Daten ist der Mittelpunkt des Ganzen.

Es widerspricht der Natur der Sache, solche zentralen Strukturen aus der Oberfläche heraus zu entwickeln. Und es wäre eine Fehlerquelle, denn es ist kaum zu vermeiden, dass die so entstehenden Tabellen nicht optimal strukturiert sind, wenn man aus dem jeweiligen Anwender-Kontext heraus die Dinge immer nur von einer Seite aus sieht.

Die Struktur bestimmt die Funktionalität.
Und die Funktionalität bestimmt die Benutzeroberfläche.

Aus der Struktur heraus die Programmlogik und aus dieser dann die Benutzeroberfläche zu entwickeln, das ist ein weiterer Grundgedanken von SCOPELAND.

Datenmodelle sollen nicht nur den momentanen Erstbedarf, sondern die objektive Realität abbilden. Genau darum geht es:

Die Struktur einer Datenbank soll die Wirklichkeit abbilden
und nicht die Sichtweise eines einzelnen, zufälligen Anwenders.

Und schon gar nicht die Sichtweise eines einzelnen Programmierers.

Zugegeben, es fällt nicht immer leicht, diesem Gedankengang zu folgen und beim Entwurf einer Anwendung zunächst völlig von der geplanten und gewünschten Oberfläche zu abstrahieren. Aber genau dies ist sinnvoll und wichtig, denn dann ist man in der Lage, anspruchsvolle, komplexe Datenbanken aufzubauen, die offen sind für jegliche künftige Weiterentwicklung.

Das Für und Wider der Persistenz

An dieser Stelle soll kurz der innere Programmaufbau von SCOPELAND-Anwendungen erläutert und begründet werden. Dafür aber müssen wir uns zunächst einer der großen Grundsatzfragen heutiger IT stellen, der ‚Persistenz', bzw. verallgemeinert der ‚Ebenen' einer Software.

Es ist unstrittig, dass Software mehrschichtig aufgebaut werden sollte.
Aber wir meinen damit nicht alle das Gleiche.

Mehrschichtigkeit in dem Sinne, dass ein Teil der Programmlogik auf lokalen Endgeräten läuft, ein weiterer auf zentralen Servern und schließlich ein Teil in der Datenbank selbst, das ergibt sich quasi von selbst aus dem geplanten Einsatzzweck der Software. Auch dass einzelne Ebenen bei Bedarf über bestimmte standardisierte Schnittstellen, z.B. über WebServices, miteinander kommunizieren sollten, ist weitgehend unumstritten.

Fraglich ist aber, ob es immer richtig ist, jede dieser Schichten unabhängig voneinander zu entwickeln. Es hängt von den konkreten Umständen ab, ob es überhaupt erforderlich ist, später mal die einzelnen Schichten unabhängig voneinander austauschen und weiterentwickeln zu können oder ob es eher von Vorteil wäre, diese zusammen, aus einem Guss, aus einer Quelle heraus zu entwickeln und automatisch synchron zueinander zu halten.

Es gibt viele Einsatzfälle, insbesondere bei Standardsoftwareprodukten, wo die komplexe Applikationslogik vor etwaigen technischen Einflüssen (z.B. infolge einer Portierung auf ein anderes Datenbanksystem oder auf ein anderes Front-end-System) geschützt werden sollte. Deshalb werden sog. ‚Persistenzschichten' vorgesehen. Die Software wird so konzipiert, dass die ‚eigentlichen' Programme nicht mit der Datenbank kommunizieren, sondern nur abstrakte Objektmodelle verwenden – und der Abgleich der Dateninhalte mit der jeweiligen Datenbank wird dann einem gänzlich eigenständigen Programm überlassen. Will man von einem Datenbankhersteller zu einem Anderen wechseln, so braucht man nur diese Datenzugriffsschicht auszutauschen, ohne dass das eigentliche Programm davon tangiert wird. Dasselbe gilt, wenn man das Ausgabemedium wechseln möchte, z.B. zwischen unterschiedlichen Betriebssystemen mobiler Endgeräte.

Schön und gut. Aber wollen wir das überhaupt?

Wenn es um die Entwicklung maßgeschneiderter Anwendungssoftware geht, dann kann es zwar durchaus einmal vorkommen, dass ein Wechsel der Datenbank erfolgen muss, und dies sollte mit angemessenem Aufwand auch irgendwie möglich sein, aber real kommt das doch eher selten vor. Viel häufiger hingegen passiert es, dass die Programminhalte selbst infolge neuer Anforderungen oder Erkenntnisse verändert werden müssen. Dann muss sowohl die Anwendungslogik angepasst werden als auch, synchron dazu, das Datenmodell nebst aller Datenzugriffe, und natürlich auch die Programmoberfläche.

Hat man nun Persistenzschichten eingezogen, dann sind sie hierbei eher hinderlich, da nicht nur ein, sondern immer zwei oder mehr Programme synchron geändert werden müssen. Der

Änderungsaufwand steigt hierdurch i.d.R. auf ein Vielfaches an, verbunden mit einem immensen Anstieg des Fehlerrisikos. Ein Mittel, das eigentlich dafür erfunden wurde, mehr Flexibilität zu erreichen, und das auch von Vielen in diesem Sinne verstanden und genutzt wird, bewirkt im hier betrachteten Umfeld das genaue Gegenteil: es macht die Software starr und unflexibel.

Je nachdem, an welcher Stelle man Änderungen erwartet (in der technischen Basis oder in den Inhalten) machen solche Persistenzansätze Software entweder flexibler oder – ganz im Gegenteil – so unflexibel, dass spätere Änderungen kaum noch möglich sind.

Persistenzschichten in individuellen, kundenspezifischen Datenbankanwendungen führen zu vermeidbarer Redundanz und drastischem Mehraufwand und sie stellen zudem eine schwerwiegende Fehlerquelle dar.

Gestatten Sie uns hier bitte, auf eine Analogie aus der Welt des Bauens zu verweisen:

Wenn man als Architekt ein Haus plant, dann muss man sich auch entscheiden, wo man die tragenden Wände des Gebäudes vorsieht: innen oder außen, längs oder quer. Die tragenden Wände sind fix, alles andere ist flexibel änderbar. Genau vor dieser Frage steht man hier auch: wo sollen die tragenden Wände sein – innen oder außen, längs oder quer?

Alles zugleich kann nicht flexibel sein.
Ein in jeder Hinsicht flexibles Haus, das wäre bestenfalls ein Zelt.

Stellt man die tragenden Wände quer auf, als Persistenzschichten und getrennte Programmebenen, dann wird das System flexibel gegenüber ‚horizontalen' Änderungen wie z.B. einem späteren Austausch einer Middleware oder einem Wechsel von einer relationalen zu einer nichtrelationalen Datenbank.

‚Vertikale' (inhaltliche) Änderungen hingegen sind dann aber nur sehr aufwändig möglich. Sieht man die tragenden Wände hingegen in Längsrichtung vor, inhaltlich, so dass alle Programmteile voneinander unabhängig sind, und indem man alle technischen Ebenen in einem gemeinsamen System modelliert, dann ist es später sehr viel einfacher, die fertigen Programme an sich ändernde inhaltliche Anforderungen anzupassen.

Der Preis dafür ist allerdings, dass man an gewisse Grundprinzipien gebunden ist, z.B. an das Prinzip relationaler Datenbanken oder an einen irgendwie ‚direkten' Zugriff von einer Programmebene auf die andere. Aber das ist bei kundenspezifischer Anwendungssoftware i.d.R. kein Problem.

Deshalb verbietet bzw. erübrigt sich bei Individualentwicklung der Gedanke, zunächst ein übergreifendes Objektmodell zu entwickeln, welches sich dann in die einzelnen Seiten und Programmfunktionen hinein vererbt. Dies würde ja bedeuten, dass noch so kleine inhaltlich-strukturelle Modifikationen an einem zentralen Objekt großflächige Änderungen in der ganzen Anwendung nach sich ziehen würden – eine verheerende Fehlentscheidung mit dramatischen Konsequenzen.

Diese Objektdenkweise ist nur dann optimal, wenn man genau weiß, was ein Programm tun soll und wenn sich anschließend daran nichts mehr ändern wird. Sie ist flexibel im Hinblick auf technische Implementierungsanpassungen dieser Objekte, aber nicht im Hinblick auf variierende Inhalte. In Individualsoftwarelösungen geht es aber primär um Inhalte, die sich jederzeit ändern können. Deshalb geht SCOPELAND ganz bewusst den genau entgegengesetzten Weg: Jede Seite soll für sich alleinstehend komplett und vollständig sein, über alle technischen Ebenen hinweg und sie soll so unabhängig von jeder anderen Seite sein wie möglich.

Es liegt auf der Hand, dass bestimmte Anwendungsfälle den einen und andere einen anderen Ansatz verlangen. Wenn es darum geht, ein festes, für einen Massenmarkt gefertigtes Softwareprodukt besonders robust zu gestalten, sich als Hersteller unabhängig von bestimmter Middleware zu machen und das Programm für Dutzende sehr unterschiedlicher Browser, Endgeräte oder Zielgruppen zu optimieren, dann ist zweifellos der ‚horizontale' Ansatz der bessere. Wenn man ohnehin nicht vorhat, die Funktionalität wesentlich zu verändern und wenn die Kosten für die Programmierung gering sind im Verhältnis zu den Marketing- und Vertriebskosten des Produkts, dann schlägt Robustheit die Agilität und Portierungsflexibilität schlägt die inhaltliche Flexibilität.

Ganz anders aber im Umfeld individueller, maßgeschneiderter Softwarelösungen. Da hier eine gänzlich andere Art an Flexibilität gefordert ist, bewirkt das gutgemeinte Ebenen-Denken das genaue Gegenteil von dem, was hier nötig ist: alles wird viel teurer, dauert viel länger und nachträgliche Änderungen sind kaum noch möglich. Die Vorteile der Persistenz kommen nicht zum Tragen, aber die Nachteile wirken sich in Form zu hoher Kosten und zu langer Projektlaufzeiten sehr nachteilig aus.

Man muss sich immer vor Augen halten, dass Individualsoftware etwas grundsätzlich anderes ist als Standardsoftware.
Sie unterliegt anderen Anforderungen und verlangt demzufolge auch andere technische Mittel und Prinzipien.

Für kundenspezifische Software brauchen wir ein System, mit dem Software ebenen-übergreifend entwickelt wird, wo eine Änderung auf der einen Ebene die nötigen Anpassungen auf allen anderen Ebenen automatisch nach sich zieht oder noch besser, wo alle Ebenen aus einer einzigen, redundanzfreien Modellierung heraus automatisch erzeugt werden.

Dann und nur dann ist es möglich, mit wenigen Mausklicks in eine laufende Anwendung ein neues Feld einzufügen: in die Benutzeroberfläche, in die Anwendungslogik und sogar in die Datenbank. Und dies ohne allzu großes Risiko, dass dadurch andere Programmteile davon tangiert werden. So wird Software wirklich flexibel – hinsichtlich inhaltlicher Änderungen. SCOPELAND ist ein Softwaresystem, das für diese Art von Flexibilität konzipiert ist.

Wie Datenbanken ticken

Gestatten Sie uns bitte, uns noch einmal den elementaren Prinzipien aus der Welt der Datenbanken zu widmen. Denn auch hier gibt es so unglaublich unterschiedliche Sichtweisen auf ein und dieselbe Sache, dass wir nicht drum herum kommen, klar und deutlich zu sagen, was wir eigentlich meinen. Sie werden hier nichts lesen von imaginären 3., 5. oder gar 7. Normalformen und auch nicht von anderen akademischen Feinheiten, die in der Praxis fast ohne jegliche Relevanz sind. Sie werden aber lernen, Datenbanken so zu verstehen und so zu modellieren, wie es für die besonders ‚pragmatische' SCOPELAND-Methodik optimal ist.

Diese Thematik sei so erläutert, dass sich das auch ohne spezielles Datenbank-Know-how erschließen sollte. Lediglich allgemeine PC-Kenntnisse werden vorausgesetzt. Für den erfahrenen Datenbankprofi ist der Inhalt dieses Kapitels also nicht neu, aufgrund seiner konsequent praxisnahen Herangehensweise aber vielleicht doch interessant. Zumindest hat schon manch ein IT-Fachmann resümiert, dass er das Thema von dieser Seite zuvor noch nie betrachtet hatte. Unabhängig davon, ob ein Datenmodell konventionell oder agil entwickelt wird und mit welchen Werkzeugen man arbeitet, mag die hier dargestellte Betrachtungsweise in jedem Fall hilfreich sein. Und sie ist wichtig, da sie die Grundlage für die darauf aufbauende Anwendungsentwicklung mit SCOPELAND ist.

Ferner werden hier auch Methoden, Begriffe und Grundprinzipien eingeführt bzw. gewählt, die wir dann im Weiteren verwenden werden, also empfehlen wir in jedem Fall, dieses Kapitel zu lesen.

Das Daten-Babylonische Begriffschaos

Zur Einstimmung sei erst einmal kurz auf das Begriffschaos in der Datenbankwelt hingewiesen:

> *Wir im SCOPELAND-Umfeld haben uns entschlossen, mit den gängigsten Begriffen zu arbeiten. Das sind im Wesentlichen Tabellen, Zeilen und Spalten sowie Felder, die Identifikatoren oder Attribute einer Entität enthalten. Und wir sprechen von Relationen und Relationsarten. Ganz einfach. Aber das ist nicht selbstverständlich.*
>
> *Beispielsweise nennt ein Informatik-Theoretiker der einen oder anderen Schule das, was in der Praxis meist als „Attribut" bezeichnet wird, eine „Relation", während das, was wir und die meisten anderen eine „Relation" nennen, bei manchen „Ableitung", „Kante" o.ä. genannt wird. Die Relationsarten sind dann „Kardinalitäten" und so weiter…*
>
> *Manche Anwendergruppen wiederum haben fachspezifische Begriffswelten. So ist z.B. das, was wir ein Attribut nennen, für einen Statistiker eine „Variable", ein Identifikator eine „Dimension" und das was bei Programmierern eine Variable ist, hat eigentlich keinen Namen.*
>
> *Unsere Katalog-Tabellen gehören zu dem, was dort so alles den „Metadaten" zugerechnet wird und hat nichts zu tun mit dem, was wir als Metadaten bezeichnen und schon gar nichts mit dem, was GIS-Spezialisten ‚Metadaten' nennen.*
>
> *Tabellen mit konkreten Einzelinformationen, also normale Tabellen, sind für Statistiker „Mikrodaten" und Summen- bzw. Aggregattabellen heißen dort „Makrodaten". Einige dieser Begriffe haben sich mit dem Data Warehouse-Konzept branchenübergreifend verbreitet, so dass man ihnen jetzt häufiger begegnet. Folglich müsste man bei dem Begriff ‚Metadaten' eigentlich immer dazu sagen, was man eigentlich gerade meint.*
>
> *Bei einigen proprietären Datenbankprodukten heißen gar die Tabellen nicht Tabellen, sondern „Listen" oder „Dateien" und der Begriff „Spalte" taucht teilweise gar nicht auf. Wenn von*

‚Mehrfachwerten' oder Ähnlichem die Rede ist, ist meist das gemeint, was wir im Folgenden Detail-Daten nennen.

Und als ‚Details' bezeichnen die Einen eine 1:n-Datensicht, also mehrere Datensätze als Detailinformation zu einem Datensatz einer übergeordneten Datensicht. Andere hingegen meinen, ‚Details' einer Entität seien die einzelnen Datenfelder eines Datensatzes, also die Attribute.

Um das Sprachchaos zu komplettieren, hat die Welt der objektorientierten und der sog. ‚postrelationalen' Datenbanken nicht nur interessante technischen Neuerungen, sondern für wieder dieselben oder sehr ähnliche Dinge dann auch nochmals andere Begriffe eingeführt, auf die wir hier aber keinen weiteren Bezug nehmen wollen.

Man muss einfach damit leben, missverstanden zu werden.

Besonders gern spricht man in der IT-Welt auch von „Bäumen", womit hierarchische Strukturen aller Art gemeint sind. Das ist bei SCOPELAND-Anwendungen besonders häufig der Fall, da wir ja die logischen Strukturen hinter einem Programm stets formalisieren, um sie mit universalisierter Software abbilden zu können – und diese Strukturen sind eigentlich immer hierarchisch aufgebaut.

Ist Ihnen dabei schon mal aufgefallen, dass Bäume in der IT-Sprache logisch gesehen meist auf dem Kopf stehen? Auch dieser Begriff ist nicht wirklich gut gewählt, gleichwohl aber sehr verbreitet.

Der Hauptgegenstand einer Anwendung ist inhaltlich als „oben" zu bezeichnen und die Informationen untersetzen sich naturgemäß in Form von Detailinformationen nach unten bzw. nach rechts unten. So werden 99,9% aller Bildschirmseiten aufgebaut: Oben wird angezeigt, worum es geht und da drunter kommen dann die Einzelheiten. Also befindet sich das Hauptobjekt einer Bildschirmseite fast immer links oben, aber kaum jemals unten. Je weiter wir uns im Baum verzweigen, also von Zweig zu Zweig bzw. von Detail zu Detail weiter gehen, kommen wir auch in der Bildaufteilung der Webseite bzw. Bildschirmmaske immer weiter nach unten bzw. nach unten rechts.

Ähnlich sieht es in jeder Baumdarstellung hierarchischer Daten aus: die Untersetzungen kommen immer unter dem jeweiligen Haupteintrag, unterhalb und leicht nach rechts versetzt.

Das Hauptobjekt ist im Sprachgebrauch der IT-Menschen die Wurzel des Baums (engl.: root), und die zugehörigen Detaildaten sind die Verästelungen, die man ja bei den meisten Baumsorten eher weiter oben erwarten würde. Die Wurzel ist oben und die Äste sind unten. Die meisten Programmoberflächen stellen logisch gesehen einen Baum aus Datenobjekten dar und seine Wurzel befindet sich nicht etwa unten, sondern oben. Das ist sehr komisch, aber wir können es nicht ändern. Begriffe wie ‚Tree' (englisch für Baum), ‚Root' usw. sind nun einmal massiv etabliert und wir müssen sie benutzen.

Sprachlich merkwürdig ist auch, dass die Zweige dieser auf dem Kopf stehenden Bäume grundsätzlich immer einen Vater (engl. parent) und eventuell auch Kinder (engl. child) haben. Hier hat sich ein anderes Bildnis eingemischt, das Anwachsen einer Familie vom Großvater über den Vater zu seinen Kinder und Kindeskindern. Aber wenigstens das ist von der Sache her mal richtig, denn…

Die Wurzel eines Baums ist der Urgroßvater seiner Zweige.
Verwirrend ist nur, dass er auf dem Kopf steht…

Wir bewegen uns hier im Rahmen der in der praxisnahen IT-Industrie gebräuchlichen Begriffswelt. Sie ist aber leider niemals offiziell standardisiert worden und variiert leider von Produkt zu Produkt, von Unternehmen zu Unternehmen, sogar von Berater zu Berater. Bitte tolerieren Sie es also, falls wir hier vielleicht nicht genau Ihre Bezeichnungen treffen. Dies lässt sich nicht vermeiden.

Datenbanken, Tabellen und Felder/Spalten

Datenbanken speichern alle Arten von Informationen in Tabellen. Alles, sogar Bilder, Musik oder Videos, überwiegend aber klassische Daten wie Zahlen, Zeichenketten, Datumsangaben.

Grundelement aller relationalen Datenbanken ist eine ‚Tabelle', womit aber nicht dasselbe gemeint ist wie bei einer Tabellenkalkulation, in der Spalten und Zeilen gleichberechtigt sind. Das ist hier grundsätzlich nicht der Fall.

*Tabellen bestehen immer aus einer **festen** Spaltenanzahl und*
*einer **variablen** Anzahl von Zeilen (Datensätzen).*

Man weiß i.d.R. nicht, wie viele Zeilen eine Tabelle überhaupt enthält
und auch nicht, an welcher Stelle sich eine Zeile befindet.

Den Speicherplatz für eine Spalte in einer konkreten Zeile nennt man ein ‚Feld'.

Aufgrund der Besonderheit von Datenbanken, dass die Zeilenanzahl fast immer variabel ist, betrachtet man selten das einzelne, konkrete Feld, sondern hat immer den Blick auf alle Felder der gesamten Spalte. Und wenn man von einem konkreten Feld spricht, dann meint man das, was in irgendeiner beliebigen Zeile in dieser Spalte steht. Man behandelt, von seltenen und unerwünschten Ausnahmen abgesehen, immer alle Datensätze gleich.

Gänzlich anders als in einem Tabellenkalkulationsprogramm, wo zwischen Spalten und Zeilen kein nennenswerter Unterschied besteht und wo jedes einzelne Feld sein ganz spezifisches Eigenleben haben kann, sind bei Datenbanken alle Zeilen immer als gleichartig zu betrachten. Daraus ergibt sich das Kuriosum, dass man den Begriff ‚Spalte' häufig synonym verwendet für ‚Feld' und umgekehrt. Dies ist gewöhnungsbedürftig, aber unproblematisch, da alle Felder einer Spalte grundsätzlich immer die gleichen Eigenschaften haben.

Unterschiedliche Dinge in derselben Spalte zu speichern – sowas macht man nicht.

Es ist ein typischer Anfängerfehler, sich besonders clevere Tricks auszudenken,
die gegen diesen allgemeinen Grundsatz verstoßen.

Man weiß übrigens auch nie, die wievielte Zeile man gerade bearbeitet. Zeilen/Datensätze werden nämlich nicht über eine Zeilennummer adressiert, sondern immer nur über ihren Inhalt. Man sagt nicht etwa zur Datenbank:

> *„Zeige mir Zeile Nr. 7 an“*

sondern sinngemäß zum Beispiel:

> *„Zeige mir alle Zeilen an, in der die Spalte ‚Name’ die Zeichenkette ‚Müller’ enthält“.*

Bzw. weil es u.U. viele ‚Müllers‘ geben kann, fragt man sicherheitshalber lieber nach einer Personennummer, die man ihm (dem Datenschutzbeauftragen zum Trotz) zuvor klammheimlich vergeben hat.

> *Dieses einfache Beispiel deutet übrigens schon auf eine typische Fehlerquelle hin: Der Entwickler muss wissen, ob er auf eine solche Datenbankabfrage hin einen oder mehrere Datensätze erwarten darf. Macht er hier einen Fehler, kann das schnell mal zum Programmabsturz führen. Datenbanken sind voller Fallstricke und das an Stellen, wo man es nie erwarten würde.*

Neben den eigentlichen und wichtigsten „echten“ Tabellen gibt es nun noch eine ganze Reihe von Derivaten, z.B. „Views“ und „temporäre Tabellen“. Im Wesentlichen verhalten sie sich aber alle ähnlich, so dass sie logisch gesehen fast wie normale Tabellen behandelt werden können.

Eine Datenbank ist nun eine Ansammlung von Tabellen. Viele Datenbank-Management-Systeme („DBMS“ oder relational „RDBMS“, d.h. die Programme, die für die interne Speicherung und Verwaltung der Daten zuständig sind) kennen nun diesen Begriff der ‚Datenbank’: sie können mehrere Datenbanken parallel verwalten und auch gleichzeitig mehrere, miteinander zusammenhängende Datenbanken einer ‚Instanz‘ bedienen. Bei manchen Systemen nennt man dies, abweichend hiervon „Schemen“, manchmal auch schlicht und einfach „Dateien“. SCOPELAND benutzt hier zwecks Einheitlichkeit immer den verbreitetsten Begriff ‚Datenbanken’, auch wenn einige Produkte dies intern anders verwalten oder anders bezeichnen.

Die Gesamtheit aller zusammengehörigen Anwendungsdaten bildet also eine Datenbank. Eine Anwendung kann ggf. Daten mehrerer Datenbanken, manchmal auch aus mehreren Instanzen bzw. von unterschiedlichen Servern an unterschiedlichen Standorten verarbeiten.

Der Begriff „Datenbank“ wird aber auch (sorry: auch das ist leicht verwirrend) oftmals für die Gesamtheit aller verwendeten Datenbanken samt seiner Dateninhalte und der zugehörigen Programme benutzt, so in etwa im Sinne einer „Daten-Bank“. Im Englischen kann man dies besser unterscheiden: „**databank**“ anstelle von „**database**“. Im Deutschen muss man aus dem Kontext ersehen, was hier konkret gemeint ist und oftmals gilt das Gesagte auch für beides.

Relationen

Das Wichtigste an einer Datenbank sind die Relationen. Das sind Verknüpfungen zwischen zwei Spalten zweier Tabellen, die mehr oder weniger dieselben Werte enthalten.

Achtung: dies ist jetzt schon einmal eine erste Vereinfachung: es gibt auch sog. Mehrfachrelationen, bei denen Spalten paarweise oder gar tripelweise verknüpft werden. Das ist aber nur ein technischer Unterschied. Inhaltlich ist das dasselbe. In SCOPELAND besteht die Möglichkeit, ‚Mehrfachschlüsselfelder‘ als Pseudo-Felder zu definieren, zwischen denen dann wieder ganz normale

einfache Relationen bestehen. In anderen Systemen ist der Umgang mit solchen Mehrfachrelationen teils sehr viel komplizierter.

Und hier beginnt nun die Wissenschaftlichkeit mit einer eigenen Welt theoretisch denkbarer Relationsarten und -unterarten, die sehr abstrakt und nur schwer zu verstehen ist und leider auch so unzureichend in seinen Begriffen vereinheitlicht ist, dass dieselben Begriffe oftmals für unterschiedliche Dinge benutzt werden. Auch zur Bezeichnung der einzelnen Relationsarten werden ebenfalls sehr unterschiedliche Begriffs- und Symbolschemata verwendet.

Wir wollen jetzt den Versuch unternehmen, das Prinzip von Datenbanken so einfach wie möglich darzustellen und so, wie es auch der SCOPELAND-Entwickler im Zuge seiner Arbeit benötigt. Dementsprechend benutzen wir auch das verbreitetste Bezeichnungsschema.

Ungeachtet der theoretischen Vielfalt an Relationsarten gibt es in der realen Welt typischerweise aber nur wenige Arten, genau genommen sogar nur eine einzige. Die meisten Datenmodelle enthalten durchgängig nichts anderes als diese einzige Art. Alles andere sind andere Sichtweisen auf dasselbe bzw. extrem seltene Ausnahmefälle, die man beim Entwickeln eines Datenmodells ohnehin möglichst vermeiden sollte.

Die n:1-Relation (Verweis auf eine andere Tabelle)

Diese zentrale Relationsart, der Verweis auf eine andere Tabelle, besteht eigentlich aus nichts weiter als aus einem **Zeiger**.

Eigentlich gibt es nur eine einzige Relationsart: Ein Wert in einer Tabellenspalte repräsentiert einen Datensatz in einer anderen Tabelle.

Beispiel: eine Tabelle enthält ein Firmenverzeichnis mit Spalten/Feldern wie ‚Name', ‚Ort', ‚Tätigkeit', ‚Anzahl Mitarbeiter' usw. Eine besondere Spalte enthält nun einen ‚Branchenschlüssel', irgendeine vielleicht 5-stellige Zahl, die uns normalen Menschen eigentlich gar nichts sagt.

In einer zweiten Tabelle, dem ‚Branchenkatalog' steht nun für jede Branche eine Zeile: neben der Spalte mit dem Branchenschlüssel steht dort der Branchenname im lesbaren Klartext.

Würde stattdessen der Klartext der Branchenbezeichnung in der Firmentabelle selbst stehen, dann wäre das eine unsägliche Speicherplatzverschwendung und zudem sehr viel schlechter, da man, falls man bei den Branchennamen einmal eine Schreibweise ändern muss, dies in allen Datensätzen der Firmentabelle tun müsste. Aus diesen und vielen weiteren Gründen ist es fast immer besser, solche „Kataloge" in eigene Tabellen auszulagern, so wie in diesem Beispiel geschehen. Diesen Vorgang nennt man auch „Normalisieren". Steht alles da, wo es sein sollte, spricht man von einer „Normalform".

Zwischen der Firmentabelle und dem Branchenkatalog besteht nun also eine Beziehung, eine ‚Relation'. Die dumme Schlüsselnummer im Branchennummernfeld der Firmentabelle ***zeigt*** gewissermaßen auf die Tabelle, die den Branchenkatalog enthält. Solche Felder nennt man auch „Fremdschlüssel", weil sie auf einen Schlüssel in einer anderen (fremden) Tabelle verweisen. Der Schlüssel in der anderen Tabelle, auf den sie verweisen, wird i.d.R. als „Primärschlüssel" bezeichnet, weil das der primäre Weg ist, darin einen Datensatz zu finden – gewissermaßen als Ersatz für die in Datenbanken fehlenden Zeilennummern.

Aus Sicht der Firmentabelle sieht das wie folgt aus: Jedes Feld in der Spalte mit der Branchenschlüsselnummer in der Firmentabelle ***verweist*** auf jeweils einen Datensatz der Branchentabelle, genauer gesagt: auf die Schlüsselnummernspalte im jeweiligen Datensatz der Branchentabelle.

Es handelt sich also um eine

Verweis-Relation
(auch n:1 oder „many to one“ genannt)

Betrachtet man nun die Firmentabelle allein, dann enthält sie für einen Menschen nicht sinnvoll lesbare Daten. Erst in Verbindung mit der anderen Tabelle wird sie verständlich für uns. Hierzu werden beim Auslesen der Daten aus der Datenbank in fast allen Fällen diese beiden Tabellen miteinander verknüpft und die beiden Schlüsselfelder ausgeblendet, so dass sich nachher auf dem Bildschirm diese Kombination wie eine einzige, lesbare Tabelle darstellt.

Diese Verknüpfung beim Auslesen der Daten nennen die Programmierer auch „Join“, weil sie auf diese Weise zwei Tabellen miteinander verbinden. Von sehr seltenen Ausnahmefällen abgesehen, werden Joins aber immer nur genau dort vorgenommen, wo auch eine Relation besteht, so dass es schon fraglich ist, warum hierfür überhaupt ein anderer Begriff benötigt wird. Dinge zu verknüpfen, die nichts miteinander zu tun haben, wie beispielsweise das Alter mit der Schuhgröße einer Person, das ist schlicht und einfach Unsinn. Und Felder, die relational miteinander verbunden sind, nicht zu verknüpfen, das ist ebenso fragwürdig, weil die verwendeten internen Schlüsselnummern für Menschen ja nicht lesbar sind. Datenbanktabellen sind nur in verknüpfter Form für Menschen lesbar. Also gilt der Grundsatz:

Wo eine Relation ist, ist auch ein Join.
Und umgekehrt.

In diesem Beispiel werden nun, so kann man sich das ganz einfach vorstellen, die fehlenden Spalten, die ja zuvor in den Branchenkatalog ausgelagert wurden, in die Datensicht auf das Firmenverzeichnis mit „einbezogen“.

Will man einen Datensatz der Firmentabelle anzeigen, so blendet man einfach die dumme Schlüsselnummer aus und schreibt anstelle dessen zu dieser Schlüsselnummer die passende Klartextinformation auf den Bildschirm, die man aber aus der anderen Tabelle ausgelesen hat.

Man hat also auch nach der Verknüpfung je Datensatz wiederum nur einen Datensatz. Die notwendigen Verweise werden einfach mit in die Sicht „einbezogen“.

Es ist typisch für Datenbanken, dass alle Tabellen so massiv miteinander relational verknüpft sind, dass praktisch keine einzige Tabelle für sich allein lesbar ist und alleinstehend sinnvolle Information enthält. Erst in der Verknüpfung macht das Ganze Sinn. Viele Tabellen enthalten mehr Spalten mit anonymen, unverständlichen Schlüsselnummern, als solche mit lesbarer Information. Das ist aber letztlich unproblematisch, denn man muss nur immer alle benötigten Verweis-Relationen mit einbeziehen und schon wird alles lesbar und verständlich.

Es ist völlig normal, 5, 10 oder gar 20 Tabellen miteinander zu verknüpfen, um schließlich auf dem Bildschirm einen einzigen Datensatz mit einigen wenigen Feldern anzuzeigen. Dies lässt die Welt der Datenbanken auch komplizierter erscheinen, als sie eigentlich ist.

Jede Datenbankabfrage hat immer genau eine Haupttabelle

Interessant am vorherigen Beispiel ist auch, dass es nach der Verknüpfung der beiden Tabellen immer noch eine „Haupttabelle“, nämlich die Firmentabelle gibt. Der Branchenkatalog wurde ja nur hilfsweise mit einbezogen, um das dumme Schlüsselfeld durch Klartextinformationen zu ersetzen.

Dasselbe gilt für alle von der Haupttabelle abgehenden n:1-Verweis-Relationen und ebenso auch kaskadierend für Verweise, die aus bereits einbezogenen Tabellen heraus auf wiederum andere Tabellen verweisen, beispielsweise aus dem Branchenkatalog heraus auf einen Katalog „Industriezweige“ und so immer weiter über beliebig viele Ebenen.

Egal wie viele solcher Relationen Sie auflösen, indem Sie jeweils dafür eine weitere Tabelle mit einbeziehen, es bleibt immer so, dass es eine Haupttabelle gibt. Alles andere ist nur ergänzend „einbezogene“ Information.

Sie werden aber im Gespräch mit IT-Kollegen feststellen, dass Viele das ganz anders sehen: Nur wenige Entwicklungswerkzeuge folgen dem Prinzip, immer von einer Haupttabelle auszugehen. Stattdessen erwarten viele Tools von den Entwicklern, dass diese händisch und scheinbar willkürlich Tabellen verjoinen, um so eine bestimmte Datenmenge zu erzeugen, losgelöst von jeglichem Bezug zur einzelnen Tabelle.

Man kann getrost darauf wetten, dass die so geschulten Anwender intuitiv letztlich auch keine anderen Arten von Datensichten zusammenbauen als wir. Warum sich ihnen dabei das elegant-einfache Prinzip einer Haupttabelle mit darin (ggf. über mehrere Ebenen) einbezogenen Verweisen nicht offenbart, das ist eines der großen Rätsel der IT-Industrie. Vielleicht ist es aber auch nur eine Art Selbstschutz, damit es immer schön kompliziert aussieht.

Jede oder fast jede Datensicht hat genau eine Haupttabelle und alles Weitere wird in die Sicht mit einbezogen. Wenn es anders wäre, dann könnte man in Anwendungsprogrammen niemals Daten ändern – weil es nicht definiert wäre, worauf sich die Eingaben beziehen.

Die Krux mit dem SQL

Verblüffenderweise aber wird diese Logik nicht von SQL unterstützt. SQL ist ja die Standard-Abfragesprache für relationale Datenbanken und auch SCOPELAND verwendet SQL-Befehle, um Daten abzufragen oder in die Datenbank zurückzuschreiben. Es ist nun etwas irritierend, dass ausgerechnet die dafür entwickelte Sprache SQL den typischsten aller Abfragefälle gar nicht direkt unterstützt. Wie kann das sein?

Vielleicht hat dies historische Gründe. Die Prinzipien, nach denen SQL aufgebaut ist, machen manchmal den Eindruck als folgten sie noch immer der Logik früherer elektromechanischer Lochkarten-Büromaschinen mit Stanz-, Kopier-, Sortier- und Mischmaschinen. Und es gab ein Gerät, mit dem man zwei Lochkartenstapel anhand eines gemeinsamen Merkmals zueinander in Beziehung setzen konnte, so dass die Inhalte der jeweils zueinander passenden Karten als neuer Datensatz (verjoint) auf eine neue Karte gestanzt wurde. Das könnte möglicherweise die historische Vorlage des heutigen ‚Join‘-Prinzips darstellen. Deren Programmlogik war von den damaligen technischen Möglichkeiten geprägt und nicht optimiert für die heutige IT-Welt. Und einmal geprägte Denkmuster halten sich bekanntlich lange.

Jedenfalls ist die Verknüpfungslogik in SQL derart: man schreibt (leider) nicht, dass ein Feld durch einen Datensatz aus einer anderen Tabelle zu ersetzen ist (was perfekt dem realen heutigen Bedarf entspräche), sondern man sagt „gib mir alle Paare von Datensätzen beider Tabellen, in denen das Feld X der einen Tabelle identisch ist zum Feld Y in der anderen Tabelle“. Das und nichts anderes verstehen heutige relationale Datenbanken.

Wenn wir nun, der Natur der Sache entsprechend, eine Spalte einer über eine n:1-Relation verbundenen Tabelle in eine Sicht auf die Haupttabelle einbeziehen wollen, dann müssen wir beim händischen SQL-Schreiben diese vergleichsweise einfache Anweisung zunächst in die viel komplexere SQL-Denkweise übersetzen.

Hat man nun eine Vielzahl von Tabellen mit einbezogen und dies auch noch kaskadiert über viele Ebenen, dann wird das SQL-Statement ganz schnell unlesbar, denn in SQL werden hintereinander die Namen aller Tabellen aufgelistet, völlig unabhängig von der logischen (hierarchischen) Anordnung in der Datenbankabfrage. Und dann heißt es: „wo ist dieses Feld gleich jenem und das da gleich dem dort?“ und so weiter. Es entsteht ein bunter Kauderwelsch solcher Verknüpfungs-Joins.

Der wahre SQL-Profi zeichnet sich unter anderem darin aus, dieses Join-Chaos überblicken zu können.

Aber es kommt noch schlimmer. In der Realität will man nämlich fast immer auch diejenigen Datensätze der Haupttabelle sehen, für die kein Verweis auf den eingezogenen Katalog eingetragen wurde; in unserem Beispiel auch die Firmen, für die noch keine Branche oder eine nicht mehr gültige Branche eingetragen wurde. Eine solche Abfrage ist mit dem einfachen Join-Mechanismus gar nicht möglich!

Als Notbehelf haben die Datenbankhersteller deshalb den „Outer Join“ erfunden, der leider nur wenig standardisiert ist und auch nicht immer den Bedarf korrekt abdeckt. Dazu markiert man den Join noch mit einem Zusatz, nach welcher Seite der Relation die Verknüpfung offen sein soll für fehlende Partner-Datensätze (man nennt es „Linkes oder Rechtes Outer“), was die SQL-Statements nun vollends unlesbar und unverständlich macht. Die echte Schnittmenge (also der reine Join ohne „Outer“) zwischen zwei Tabellen, für die SQL standardisiert wurde – die braucht man in der Praxis nur sehr selten. Die unverzichtbare Outer Join-Logik von SQL hingegen ist ein hervorragendes Beispiel dafür, wie man etwas ganz Einfaches ungeheuer kompliziert umsetzen kann.

Fazit:

SQL ist für relationale Datenbanken denkbar schlecht geeignet,
aber wir haben nichts Besseres.

Um Ihnen dies zu ersparen, quält sich SCOPELAND vollautomatisch durch die Beschränkungen der SQL-Syntax hindurch, inklusive der herstellerabhängigen Besonderheiten und der Unterschiede zwischen den verschiedenen Datenbanksystemen. Sie brauchen einfach nur anklicken, was Sie gern haben wollen (welche Tabellen Sie einbeziehen möchten); alles andere können Sie der Maschine überlassen. Und fast alles, was Sie so zusammenklicken, funktioniert dann auch. Manchmal aber geht es nicht, weil bestimmte Konstellationen in SQL einfach nicht möglich sind. Sorry. Das ist halt so.

Wie fast immer und überall prägen Werkzeuge und Sprachen das Denken der Menschen. So prägt auch die Eigentümlichkeit und Beschränktheit von SQL das Denken von Softwareentwicklern. Wenn Sie als

Anwender mit gesundem Menschenverstand oder als SCOPELAND-Entwickler einem erfahrenen sonstigen Datenbankentwickler erklären wollen, welche Daten zu selektieren sind, müssen Sie deshalb das Einfache zunächst ins Komplizierte übersetzen, damit dieser Sie überhaupt verstehen kann.

Die 1:n-Relation (Master-Detail)

Das Geniale am Konzept einer relationalen Datenbank ist, dass jede Relation immer von 2 Seiten aus betrachtet werden kann und so einen gänzlich anderen Blick auf ein und denselben Sachverhalt und ein und dieselben Daten ermöglicht. Betrachtet man z.B. den o.g. Verweis auf den Branchenkatalog von der anderen Seite aus, also aus der Sicht einer konkreten Branche, dann stellt sich das alles plötzlich ganz anders dar:

Stellt man einen Datensatz dieses Branchenkatalogs in den Mittelpunkt der Betrachtung, dann „gehören" dazu jeweils viele Datensätze der Firmentabelle, nämlich alle, die dieser Branche angehören. Typischerweise stellt man dies auf dem Bildschirm dar, indem rechts neben oder unter den einen Datensatz des Katalogs (meist als Bildschirmmaske dargestellt) eine tabellarische Auflistung mit den vielen Sätzen der anderen Tabelle geschrieben wird.

Zu einem solchen „Master"-Datensatz (hier der Branche) gehören immer keiner, einer oder mehrere „Detail"-Sätze (hier Firmen). Der Begriff „Detail" meint an dieser Stelle, dass wir hiermit nähere Einzelheiten über die Branche erfahren (in diesem Fall die Detail-Information, welche Firmen in ihr tätig sind).

Folglich kann man die Relation als eine

„Master-Detail-Relation"
(auch 1:n oder „one to many" genannt)

---

bezeichnen.

Das Verblüffende daran ist, dass es in Wirklichkeit aber genau dasselbe ist wie unsere oben beschriebene „Verweis-Relation". Nur eben von der anderen Seite aus gesehen.

Die Firmentabelle in unserem Beispiel ist eine „Detailtabelle" zur Branchenkatalogtabelle, während andersrum die Branchentabelle eine „Verweistabelle" (ein Katalog) zur Firmentabelle ist.

Allerdings sei hier noch darauf hingewiesen, dass wir uns mit dem Begriff „Detail" möglicherweise schon wieder auf Glatteis begeben, denn in anderem Zusammenhang wird dieser Begriff auch anderweitig verwendet. Hat man zum Beispiel eine Auflistung von Firmen auf dem Bildschirm, zunächst nur mit Name und wenigen zusätzlichen Attributen, dann kann eine Schaltfläche, auf der „Details" steht, durchaus bedeuten, dass man beim Klick eine Maske erhält, in der alle Felder des Datensatzes angezeigt werden.

Die Begriffskombination „Master-Detail" bedeutet in der IT-Sprache je nach Kontext entweder die Relationsart bzw. eine dementsprechende Darstellung auf dem Bildschirm (Master-Datensatz oben in Formularansicht und da drunter tabellarisch die Daten aus der Detailtabelle) oder ganz verallgemeinert: oben wenig und da drunter mehr, also detailliertere Informationen.

Master-Detail-Relationen in SCOPELAND

Die Bedienerführung in SCOPELAND ist optimal auf die Logik dieser Relationsart zugeschnitten.

Daten aus der Detailtabelle in die Datensicht der Haupttabelle einzubeziehen, ähnlich wie bei Verweis-Relationen, ist von der Sache her unmöglich bzw. unsinnig, weil es i.d.R. ja mehrere Datensätze zu einem gibt – das kann man so gar nicht darstellen.

SQL verbietet es zwar nicht, auch dahin einen plumpen „Join“ zu legen, aber das Ergebnis ist dann fehlerhaft, denn es führt zu einer unsinnigen Vervielfachung der Master-Datensätze in der Anzeige – ein typischer Anfängerfehler unter Datenbankentwicklern. Um diesen Unsinn von vornherein auszuschließen, wird das von SCOPELAND im Normalfall unterbunden (für Ausnahmefälle kann man diese Sperre aber aufheben).

Selektionsbedingungen auf Detaildaten sind aber stets möglich, jedenfalls soweit die nicht ganz optimale SQL-Sprache dies erlaubt.

Um Detaildaten darzustellen, benötigt man eine zweite Datensicht, abhängig von der ersten. Einzelheiten dazu finden Sie dann im entsprechenden Kapitel über Direct Views.

Eine Ausnahme aber gibt es, in der es doch möglich und sinnvoll ist, Detaildaten wie Verweise in die Haupttabelle mit einzubeziehen, nämlich, wenn auf einer Spalte eine Aggregatfunktion liegt, so dass sich die Detaildatensicht quasi in eine Verweisdatensicht mit jeweils einem einzigen Wert verwandelt. Im obigen Beispiel etwa die Anzahl Firmen pro Branche zum Datensatz im Branchenkatalog.

Auch dies funktioniert aber nicht immer korrekt, wiederum aufgrund der Beschränktheit der Sprache SQL. Auch hier muss man als Anwendungsentwickler manchmal mit der Beschränktheit der Basistechnologien leben.

Die sogenannte m:n-Relation

Gelegentlich stößt man auf folgende Situation: Sie haben zwei Tabellen die irgendwie miteinander zu tun haben, ohne dass die eine auf die andere verweist. Beispiel: ‚Mitarbeiter’ und ‚Sprachen’. Zwischen beiden besteht eine sog. „many to many“-Relation: ein Mitarbeiter spricht ggf. mehrere Sprachen, ebenso werden aber auch die Sprachen jeweils von mehreren Mitarbeitern gesprochen.

Was ist das? Handelt sich also ausnahmsweise doch um etwas ganz anderes? Eine völlig andersartige Relation als oben besprochen?

Nein. Genau genommen handelt es sich nämlich um 2 Verweise, die beide von einer Dritten, neu zu erstellenden Tabelle namens „Mitarbeiter spricht Sprache“ ausgehen und die beide auf je eine der beiden Tabellen zeigen. Solch eine „m:n-Relationstabelle“ ist also gar nichts Ungewöhnliches und, nachdem man diese Tabelle angelegt hat, hat man wieder das einfache Bild von Verweisen und Details, genauso wie wir es oben beschrieben haben.

Die Notwendigkeit dieser zusätzlichen Tabelle ist für manch einen schwer zu verstehen, weil diese gar keine „echten“, d.h. lesbaren Daten enthält. Sie besteht ausschließlich aus zwei Zeigern (und ggf. noch aus einem technisch bedingten, meist nicht mit angezeigten internen Schlüsselfeld, welches uns hier nicht weiter interessiert), also aus zwei Schlüsselfeldern, die paarweise den Datensatz adressieren bzw. ‚**identifizieren**’, also aus zwei Identifikatorfeldern.

Es fällt nur etwas schwer, sich eine Entität vorzustellen, die aus NICHTS besteht. Aber es gibt sie doch. Die Aussage, dass ein Mitarbeiter eine Sprache beherrscht, ist ja eigentlich doch eine richtige, echte Entität. Sie hat nur keine echten Attribute. Das ist der einzige Unterschied zu einer normalen Tabelle und auch dieser Unterschied ist nicht endgültig. Man könnte sich durchaus vorstellen, irgendwann einmal noch die Zusatzinformation zu ergänzen, ob der Mitarbeiter die Sprache perfekt, gut oder nur

leidlich spricht. Dieses neue Attribut-Datenfeld gehört genau hier hin, in diese Tabelle. Und schon ist daraus eine ganz normale, ganz gewöhnliche Tabelle geworden.

In der klassischen Datenmodellierung hat man solche Verbindungstabellen häufig gar nicht eingezeichnet und lediglich einen Doppelpfeil verwendet, der besagt, dass es sich um eine „m:n"-Relation handelt. Erst beim späteren Umsetzen in eine echte Datenbank wurde dann diese Hilfstabelle schnell noch ergänzt. Wir empfehlen aber, diese von Anfang an als ganz normale Tabelle zu begreifen und mit zu modellieren. Die Tabelle nebst der beiden Relationen kann in SCOPELAND mit einer ganz einfachen Menüfunktion in einem Schritt erzeugt werden.

> *Denkbare wäre theoretisch noch eine andere Art von m:n-Relation, die aber von untergeordneter Bedeutung ist und daher in der Datenbanktheorie auch kaum Erwähnung findet. Es könnte nämlich sein, dass zwei Tabellen inhaltlich derart miteinander verbunden sind, dass ein Inhalt in einer Spalte der einen Tabelle mehrfach auftritt und in einer anderen auch. So könnte etwa in einer Mitarbeitertabelle die Augenfarbe stehen und in einer Tabelle mit Ansprechpartnern von Kunden ebenfalls. Über diese Gemeinsamkeit hinweg könnte man sich theoretisch eine Verknüpfung vorstellen, um jeweils zum Kunden farbliche passende Vertriebsmitarbeiter zu finden. Das klingt etwas schräg, aber manch ein Datenbank-Neuling mag dies vielleicht mit einer richtigen m:n-Relation verwechseln. Tatsächlich ließe sich aber auch dies ganz elementar mit zwei ganz normalen Verweisrelationen abbilden, die beide auf eine neue Tabelle „Augenfarben" verweist. Aus Sicht der einen Haupttabelle kommt man über diesen Verweis zur Augenfarbentabelle und von dort aus weiter per Detail-Relation zur anderen. Ganz einfach.*

Die Datenbank – ein Netz von Relationen

Eine Datenbank besteht nun aus vielen Tabellen mit unzähligen Relationen dazwischen. Aber eigentlich ist es doch ganz einfach, es ist immer und überall dasselbe: es handelt sich durchgängig und überall schlicht und einfach um Verweise. Die scheinbar enorme Vielfalt entsteht erst durch die unterschiedlichsten Sichten. Je nachdem, bei welcher Tabelle man anfängt, ergibt sich immer ein ganz anderes Bild von denselben Daten.

Dies ist auch einer der Gründe, warum ein und dieselbe Datenbanktabelle so häufig in den Anwendungsprogrammen verwendet wird. An irgendeiner Stelle des gesamten Verknüpfungsnetzwerks einer Datensicht taucht sie immer wieder auf, manchmal ganz unscheinbar als Verweis vom Verweis vom Verweis eines Details eines Verweises vom Detail. Irgendwo und irgendwie landet man immer wieder bei denselben Kerntabellen einer Anwendung, egal was man macht. Sie kommen quasi überall in der Anwendung vor.

Deshalb müssen sie sorgfältig und gut durchdacht angelegt werden. Das weiträumige Zusammenspiel von Tabellen und Relationen (Verweisen und Details) soll die in der realen Welt vorhandene Informationsvielfalt möglichst gut abbilden. Wenn dies gut gelingt, dann ist es später leicht, eine Anwendung drüber zu bauen.

Sorgfältig modellieren, denn: ein gutes Datenmodell ist die halbe Anwendung.

Am leichtesten fällt es uns natürlich, Datenstrukturen sauber zu modellieren, wenn wir dabei nur an die Daten selbst und nicht an die späteren Anwendungen denken. Die spätere Oberfläche vor Augen,

neigt man hingegen eher dazu, all das in eine Tabelle hineinzupacken, was später zusammen auf dem Bildschirm stehen soll. In unserem Firmenbeispiel wäre das z.B. die ‚Tätigkeit' und die ‚Branche'. Genau das wäre aber schon der erste Modellierungsfehler.

Genau andersrum geht man vor: alles was es in dieser Welt tatsächlich gibt (jede „Entität"), wird jeweils eine Tabelle. Ganz einfach. Es gibt fest definierte Branchen, also braucht man eine Branchentabelle. Es gibt Firmen, also braucht man eine Firmentabelle usw.

Wenn uns dies klar ist, dann verstehen wir auch sehr gut, warum die Weisheit immer in den Datenstrukturen liegt und nicht in den Oberflächen und warum wir immer bei der Datenbank zu denken beginnen müssen.

Konsistenz

Lebenswichtig für eine jede Datenbank ist die „Konsistenz" der Daten.

Das hat insbesondere mit der relationalen Struktur zu tun. Wie oben beschrieben enthalten die Tabellen ja zu einem großen Teil anonyme Schlüsselfelder, die auf bestimmte Datensätze anderer Tabellen verweisen. Probleme treten nun sofort auf, sobald einmal aus irgendeinem Grund, egal weshalb, in einer Spalte ein Schlüsselwert steht, für den in der Verweistabelle gar kein Datensatz vorhanden ist. So etwas führt häufig zu Fehlfunktionen, im Extremfall sogar zu Programmabstürzen.

Diese und andere Fälle von Inkonsistenzen sind ein großes Problem und sie müssen deshalb unter allen Umständen vermieden werden, was wiederum zu einer Verkomplizierung der Anwendungsprogramme führt. Das Konsistenzproblem ist auch einer der Hauptgründe, warum man nur sehr ungern erlaubt, am Anwendungsprogramm vorbei direkt in Datenbanktabellen hineinzuschreiben. Und selbst, wenn man nicht schreibend, sondern nur lesend zugreift, steht immer das Risiko missverständlicher Interpretation der Daten im Raum. Sicherheitshalber erklären deshalb die meisten Softwarehersteller „ihre" Datenbanken zu ihrem geistigen Eigentum und verbieten jeglichen direkten Zugang an der Applikation vorbei.

---

*Wir jedoch sind da anderer Meinung. Die Daten gehören dem Kunden und nicht dem Softwarehersteller. Es ist **seine** Datenbank.*

Selbstverständlich soll es ihm möglich sein, auch ohne die Applikation auf seine Daten zuzugreifen, lesend – und warum nicht auch schreibend? Natürlich darf nicht jeder User alles dürfen, aber es muss in der Hand des Kunden liegen, wem er was erlauben will.

Das Problem mit dem direkten Zugriff ist daher nicht ein Berechtigungsproblem – das wäre einfach lösbar. Das Problem ist ein technisches. Es fehlt den Datenbanken an einem entsprechenden Verfahren, Daten aus Datenbanken sinnvoll im richtigen Kontext zu lesen und konsistenzsichernd zu ändern. Diesem Problem haben wir uns gestellt. Es ist sehr wohl möglich!

Wie SCOPELAND dies löst, dazu weiter unten.

Vorgehensweise zum Aufbau der Datenstrukturen

Die Kunst ist nun, die Wirklichkeit in Form solcher Tabellen abzubilden.

Früher ging man dabei wie folgt vor: man malte für jede Entität ein Kästchen auf ein riesiges Blatt Papier (oder auf den Bildschirm eines Zeichenprogramms) und überlegte sich anschließend, wie diese Kästchen nun miteinander zu verbinden sind, also von wo nach wo eine Relation besteht und von welcher Art diese ist.

Diese streng akademische Vorgehensweise hat Vorteile und ist auch heute noch verbreitet. Sie hat aber auch einen entscheidenden Nachteil: es bewegt sich auf einer derart abstrakten Ebene, dass dies nur von wenigen, geschulten und langjährig erfahrenden sog. ‚Systemanalytikern' fehlerfrei beherrscht wird.

Oftmals hat man aber nicht diesen enormen Spezialisierungsgrad. Wer sich nicht täglich mit solchen abstrakten Modellen beschäftigt, hat immer Schwierigkeiten, den Überblick zu behalten. Ein einziger falsch eingezeichneter Pfeil bringt später die gesamte fertigprogrammierte Anwendung zum Stehen. Um dies zu kompensieren, werden dann Hilfs- und Zwischentabellen und andere Umgehungen hineingebastelt und so erzeugt man…

…Knoten im Datenmodell.

Und diese Knoten müssen die Programmierer dann mit aberwitzigen Tricks irgendwie glattbügeln, was letztlich zum sog. „Spaghetti-Code" führt. SCOPELAND priorisiert deshalb hier den anderen, pragmatischen Ansatz, ohne jedoch den „klassischen" Weg auszuschließen.

Was liegt näher, als sich zur Vermeidung von Modellierungsfehlern von Anfang an in den oben dargestellten einfach zu verstehenden Kategorien von „Verweisen" und „Details" zu bewegen?

Also nicht Kästchen malen und nicht dann darüber grübeln, ob es nun eine „zero or one to many", oder vielleicht doch eine „one or many to zero or one"-Relation werden soll. Bei SCOPELAND geht's viel einfacher:

Man fängt einfach mit einer ersten Tabelle an, nimmt sich dann diese Tabelle und überlegt, was jetzt noch fehlt: und dabei gibt es eigentlich nur zwei klar und deutlich voneinander unterschiedene Fälle:

Entweder fehlt der aktuellen Tabelle noch eine „zugehörige Detailtabelle"
oder man muss noch einen Katalog „hinterlegen".
Etwas anderes gibt es eigentlich nicht …

… bzw. nur in sehr seltenen Ausnahmefällen, die wir an dieser Stelle zunächst ausklammern können.

Nur die allererste Tabelle ist eine „alleinstehende Tabelle", alles andere wächst dann Schritt für Schritt drum herum. Und Sie müssen nicht einmal diesen Anfang machen, denn irgendetwas gibt es ja meist schon: irgendeine Datensammlung, die man importieren kann oder Sie starten gar mit einer kleinen Alt-Datenbank, die Sie automatisch analysieren lassen und die Sie dann in der beschriebenen Art weiterentwickeln können, durch Ergänzen von Tabellen und Feldern. Es werden dabei automatisch immer korrekt die physischen Strukturen und Metadatenbeschreibungen angelegt, mit der richtigen Art von Schlüsselfeldern und mit der korrekten Relationsart, sogar mit den richtigen Indexen und

sonstigen Kennzeichnern in der Metadatenbank. All dies jeweils auf Mausklick, ohne dass Sie sich dabei um technische Details kümmern müssen.

Um einen Katalog zu hinterlegen, gibt es mehrere Varianten: entweder man erzeugt einen neuen oder man verweist lediglich auf einen bereits bestehenden Katalog. Falls man einen neuen erzeugt, so kann man ein bereits bestehendes Feld gleich als Schlüsselfeld nutzen oder man lagert den Inhalt des Feldes/der Spalte durch „Normalisieren“ in die neue Katalogtabelle aus. All diese Varianten werden unterstützt. Entsprechendes gilt für das Anlegen von Detailtabellen und m:n-Relationstabellen.

Auch wenn Ihnen bei dieser Vorgehensweise manchmal der vollständige Überblick übers Ganze fehlt, die Qualität eines so erstellten Datenmodells ist in der Regel erheblich besser. Und viel einfacher ist es sowieso.

Das Beste an dieser Vorgehensweise ist, dass Sie nicht unbedingt gezwungen sind, zu Beginn das gesamte Datenmodell komplett zu entwerfen und einzutippen. Sie fangen einfach klein an, mit den Teilen, die Sie zunächst für den ersten kleinen Anwendungsteil benötigen. Und immer dann, wenn Sie feststellen, dass es in der realen Welt irgendwelche Daten gibt, mit denen Sie eventuell noch konfrontiert werden könnten, dann ergänzen (oder modifizieren) Sie das Datenmodell evolutionär um genau diese Tabellen und Felder/Spalten und bauen dann den nächsten Anwendungsteil dazu.

Wichtig dabei: lernen Sie, „in Tabellen zu denken“.

Schlüsselfelder und Dimensionen

Das eine Feld (oder eine Kombination aus Feldern/Spalten), das genau einen Datensatz exakt identifiziert. Dies wird von Datenbankexperten lt. Theorie auch als „**Primärschlüssel**“ bezeichnet.

Und all die Spalten (oder Spaltenkombinationen), die per Verweisrelation auf eine andere Tabelle zeigen, sind die „**Fremdschlüssel**“ (weil sie ja auf den Schlüssel einer „fremden“ Tabelle zeigen).

Man kann aber auch vereinfacht von Relations- bzw. Verweisfeldern sprechen. Das klingt dann zwar nicht so wissenschaftlich, ist aber letztlich verständlicher.

Im Interesse möglichst einfach zu handhabender Datenstrukturen wird routinemäßig oftmals ein automatischer Satzzähler (ein sog. „Serialfeld“) bei fast jeder Tabelle ergänzt. Verweise zeigen dann bevorzugt auf dieses Satznummernfeld und nicht auf die eigentlichen Daten. Häufig ist es zugleich der Primärschlüssel. Manchmal aber hat man am Ende nicht nur einen, sondern gleich zwei Primärschlüssel (was lt. Theorie eigentlich verboten ist), nämlich zum einen den ***inhaltlichen*** und zum anderen den ***technischen*** Primärschlüssel. Der Erstere adressiert und identifiziert einen Datensatz inhaltlich; es handelt sich folglich um den „Identifikator“. Der Andere, der mehr oder weniger unsichtbare Hilfsschlüssel, wird auch als „Unique Key“ (engl. für „eindeutiger Schlüssel“) bezeichnet.

Im Normalfall können Sie diese Thematik ignorieren und sich von dem leiten lassen, was die Bedienerführung in SCOPELAND Ihnen vorschlägt. Nur im Ausnahmefall sollte man hier manuell eingreifen und Abweichendes einrichten.

Oft gibt es eine Spalte, manchmal aber auch eine Kombination von Spalten die zusammen einen konkreten Datensatz identifizieren. In unseren obigen Beispielen sind das z.B. ein Firmenname oder einer Firmenregistriernummer, eine offizielle Branchennummer oder -bezeichnung, oder die Kombination der beiden Verweisfelder auf die Mitarbeiter- und Sprachtabellen im m:n-Beispiel. Solche Spalten werden auch (wenngleich nicht brancheneinheitlich) als Identifikatorfelder bezeichnet.

Die Würfel-Sicht auf die Daten

In Statistiken spricht man auch von den „Dimensionen“ eines virtuellen OLAP-Würfels. Die Anzahl solcher Spalten sagt aus, wie viel Dimensionen eine Tabelle hat. Der Branchenkatalog aus unserem Beispiel ist eindimensional, denn er hat ja nur einen (inhaltlichen) Schlüssel, die Branche. Ebenso unsere Firmentabelle. Unsere obige „Mitarbeiter spricht Sprachen“-Tabelle dagegen hat zwei Dimensionen: Mitarbeiter und Sprachen.

Nach diesem Prinzip kann man sich durchaus 5-, 10- oder auch 30-dimensionale Tabellen leicht vorstellen. Beispielsweise kann ein Verkaufsvorgang Dimensionen haben wie: Filiale, Produkt, Datum, Uhrzeit, Wetter, Verkäufer, Standort im Regal, Preisklasse, Altersgruppe des Kunden, Schuhgröße, Haarfarbe, Anzahl Kinder usw. In solchen n-dimensionalen Räumen kann man dann hervorragend OLAP-Analysen und Data Mining betreiben. Nicht immer sind alle Dimensionen physisch in der Tabelle enthalten. Oftmals bekommt eine Datenbankabfrage neben den „echten“ Dimensionen aus der Tabelle durch die Verweise auf einbezogene Kataloge weitere Dimensionen hinzu, z.B. führt der Verweis auf einen Städtekatalog noch weiter zum Bundesland, von dort aus zum Land und weiter zum Kontinent und zur regierenden Partei und immer so weiter. Zieht man einen Katalog auf einer Dimensionsspalte ein, so werden alle Attribute der einbezogenen Tabellen ebenfalls de-facto zu Dimensionen (Identifikatorfeldern), was zu einer enormen Dimensionsbreite in Auswertungen führen kann. Dieses laufende, kaskadierende Hinzunehmen weiterer Dimensionen nennt man auch „Snowflake-Prinzip“ oder „Snowflake-Schema“, in Abgrenzung zum einfachen „Star-Schema“, bei dem alle Dimensionen bereits von Anfang an in der Tabelle enthalten sind.

Wie sollen Tabellen und Felder heißen?

Wenn man sich die bis hier beschriebene Datenbanklogik verinnerlicht, dann ergeben sich geradezu zwingend folgende Namenskonventionen, die zwar nicht überall so üblich, aber sehr sinnvoll und für SCOPELAND-Anwendungen fast unerlässlich sind.

Dies betrifft natürlich immer die „logischen“ Namen, mit denen man als Anwender und Entwickler im Normalfall arbeitet und nicht die oftmals kryptisch kodierten technischen Bezeichnungen. Logische Namen können uneingeschränkt Leer- und Sonderzeichen enthalten, sie können etwas länger sein und sie können jederzeit geändert werden.

- Tabellenamen sollen immer die Gesamtheit ihrer Inhalte bezeichnen, also entweder die Bezeichnung der Entität in der Mehrzahl (z.B. „Firmen“) oder als Bezeichnung der Gesamtheit (z.B. „Branchenkatalog“)
- Feldnamen sollen immer bezeichnen, was das jeweilige Feld in einem Datensatz enthält, immer im Singular! Beispiele: „Alter in Jahren“, „Geburtsort“, „Wohnort des Betreibers der Anlage“.
- Fremdschlüssel, also solche Felder, die lediglich einen Verweis auf eine andere Tabelle enthalten, sollten i.d.R. so heißen wie die Tabelle, auf die sie zeigen – allerdings unbedingt im Singular, beispielsweise „Branche“ wenn das Feld auf den Branchenkatalog verweist. Er sollte explizit nicht die Eigenschaft, dass es sich um einen Schlüssel handelt, im Namen tragen (á la „Branchenschlüssel“), weil er ja nicht den Schlüssel als solches, sondern den gesamten Datensatz repräsentiert und im Programm per Klick auf diesen weiterverweist. Unter Umständen sollte dies noch ergänzt werden um eine Kontextinformation. Wenn z.B. von zwei Feldern auf dieselbe Adresstabelle verwiesen wird, dann hat man z.B. anstelle von „Adresse“ eine „Wohnadresse“ und eine „Postadresse“.

- Primärschlüssel hingegen repräsentieren sich wirklich als Schlüssel und sie sollten auch so heißen, z.B. „Branchennummer“ bzw. „Satznummer“ oder dergleichen.
- m:n-Relationstabellen sollten immer die Verbindung zwischen den beiden Tabellen repräsentieren, also beispielsweise „Sprachen der Mitarbeiter“, um von beiden Seiten aus gesehen korrekt verstanden zu werden.

Indexierung

Ein sog. Index hilft der Datenbank, schnell Ihre Daten zu suchen und bereitzustellen.

Ein oder mehrere Felder in Kombination werden als rein technische, datenbankinterne Suchhilfe, mit einem Index versehen und schon geht es schneller: immer dann, wenn nach diesen gesucht wird oder über diesen relational verknüpft wird. Ein Datenbankentwickler muss normalerweise entscheiden, wo welche Indexe in welcher Art zu setzen sind und auch dies kann man je nach Notwendigkeit, Fachwissen und der Zeit mehr oder weniger weit treiben.

Im Normalfall können Sie mit folgender simplen Faustregel auskommen: Einen Index sollte einfach jede Spalte (oder Spaltenkombination) haben, die…

1. *… sich in irgendeiner Form „Schlüssel“ nennt.*
   Das sind insbesondere die Primär- und Fremdschlüssel, also die beiden Enden einer jeden Relation, sowie eventuelle technische Hilfsschlüssel, die SCOPELAND Ihnen automatisch angelegt hat. Wenn Sie ausschließlich die o.g. SCOPELAND-Methodik benutzten, wird das auch automatisch so generiert, ohne dass Sie sich drum kümmern müssen, oder

2. *… Zeichenketten in sehr großen Tabellen enthält, nach denen Sie vermutlich häufig suchen werden (z.B. Namensfelder).*

In den jeweiligen Programmen werden Ihnen dann bei Bedarf noch weitere Optionen dazu angeboten (z.B. ist für eindeutige Schlüssel ein ‚Unique Index’ bzw. eine direkte Deklaration als ‚Primary Key’, schneller als ein normaler Index).

Die Benutzerführung von SCOPELAND ist darauf ausgelegt, dass Sie im Normalfall die Indexierung dem System überlassen können. Eine gelegentliche Kontrolle und Präzisierung ist aber dennoch anzuraten.

Das Meta-Prinzip

Metadaten beschreiben Daten

Zunächst einmal basiert das gesamte System SCOPELAND auf einer zentralen Metadatenbank. Die darin gespeicherten Metadaten beschreiben zumindest

- das Datenmodell (Datenbanken, Tabellen, Felder, Relationen) – gut dokumentiert, umgangssprachlich bezeichnet und erweitert um weitere vom Computer auswertbare Informationen, wie z.B. Wertebereiche, logische Feldarten, berechnete Spalten und vieles mehr.
- Optional zusätzliche Logik – regelbasiert abgelegt und global wirksam: Plausibilitätsregeln, Berechnungsregeln, Überlagerung von Systemereignissen u.dgl.
- Und vielleicht auch übergreifend wirksame Workflow- und Prozessmodelle
- Sowie zahlreiche weitere gemeinsam nutzbare Ressourcen wie z.B. ein gemeinsames Bilderverzeichnis und Menüs

Im weiteren Sinne zu den Metadaten gehörig (aber technisch in einer eigenständigen Datenbank gespeichert) gibt es dann noch eine optionale zentrale Benutzerverwaltung, eine Verwaltung aller hier verwendeten Datenbanken und weitere derartige Systemeinstellungen. Je nach Installations- und Betriebsvariante wird diese ggf. nur während der Entwicklung benötigt, möglicherweise aber auch im Betrieb.

Aus diesem globalen Pool an strukturierter beschreibender Informationen schöpft nun das Programm SCOPELAND, um automatisch halbwegs brauchbare Vorschläge von Anwendungsprogrammen zu generieren.

Dank all dieser beschreibenden Informationen kann SCOPELAND, oder besser gesagt die Windows-basierte Programmkomponente SCOPELAND Direct Desk, nun mit der Datenbank oder den Datenbanken kommunizieren, und sie ermöglicht es dem Benutzer, sich interaktiv, ***direkt*** und unmittelbar Zugang zu den gespeicherten Daten zu verschaffen.

---

*… daher auch der Name **Direct** Desk.*

Man kann ad hoc Tabellen „öffnen“ und spontan in jeder erdenklichen Art und Form mit den Daten frei und ungehindert umgehen. Die Möglichkeit solcher „Ad-hoc“-Zugriffe auf die Datenbank ist etwas ganz Besonderes. Eine ausführliche Beschreibung dazu finden Sie weiter unten und sie ist auch die Grundlage des besonderen Features für Endkunden, auch diesen freien Umgang mit „ihren“ Daten, bzw. mit den Daten in ihrem „Scope“ zu ermöglichen.

Für Entwickler ist der Ad-hoc-Zugriff auf die Daten zwar nicht zwingend nötig, aber eine sehr, sehr willkommene Arbeitserleichterung in unzähligen Situationen.

Ihre Daten wissen selbst, wie sie sich präsentieren sollen.

„Metadaten“ sind (lat.: *meta* = ‚über‘) Daten über Daten, also Daten, die Daten beschreiben. Wie ganz zu Beginn ausführlich erläutert, folgt SCOPELAND einem datenzentrischen Prinzip. Alles leitet sich von

den Daten ab, denn die Daten sind der Mittelpunkt und das Wesentliche einer Anwendung. Alles weitere: Anwendungslogik, Darstellungsarten von Daten usw. rankt sich rund um die Daten herum.

Eines der wichtigsten Anliegen des SCOPELAND-Ansatzes ist es, die starre Vorgehensweise aufzubrechen, nach der man zunächst alle Datenstrukturen und andere Modelle entwirft, dann ein Programm darüber schreibt und erst ganz am Ende, wenn eigentlich alles fertig ist, zum ersten Mal die Anwendung selbst sehen und mit dieser die ersten richtigen Daten eingeben kann.

Bei SCOPELAND hingegen kann man evolutionär vorgehen, auf allen Ebenen gleichzeitig: Datenmodell erweitern oder anpassen, zusätzliche Logik ergänzen, Oberflächen zusammenklicken. Alles im munteren Wechsel miteinander. Dass dies so elegant möglich ist, liegt unter anderen darin begründet, dass die reichhaltigen SCOPELAND-Metadaten dazu führen, dass sich die Daten quasi selbst kennen und dass sie scheinbar selbst wissen, wie sie sich darzustellen haben. Ein Klick, und sie haben die Daten vor sich, in lesbarer Form, intelligent miteinander verknüpft und typgerecht korrekt dargestellt.

Und es ist bekanntlich schwierig, Anwendungen über Datenbestände zu bauen, die es noch gar nicht gibt. Deshalb ist es sehr nützlich, sich ad hoc einfach immer gleich die nötigen Testdatensätze erzeugen zu können. Ideal wäre es sogar, von Anfang an unter vollem oder wenigstens halbwegs realistisch mit Testdaten gefülltem Datenbestand zu arbeiten, auch zur frühzeitigen Erkennung von Performance-Engpässen.

„Tabelle öffnen“ – was ist das?

Jede Tabelle wird, sobald sie beschrieben wurde, sofort und automatisch in der Datenbank physisch angelegt und bei Bedarf später auch verändert. Sie kann deshalb auch sofort „geöffnet“ werden, ähnlich wie man eine Datei öffnen und ansehen und ggf. auch bearbeiten kann.

Ein solches Ad-hoc- „Öffnen“ einer Tabelle bedeutet, dass der relevante Teil des Tabelleninhalts in das Programm SCOPELAND geladen und (per Default in tabellarischer Darstellung, wahlweise aber auch als Bildschirmmaske o.a.) angezeigt wird. Dies ist nicht nur ein simples Anzeigen, sondern es passiert bereits mit der vollen SCOPELAND-eigenen Intelligenz. Und natürlich mit einer ausgefeilten Optimierung darüber, wann welche Datensätze aus der Datenbank ausgelesen werden müssen, so dass dies auch für Tabellen mit Millionen von Datensätzen noch gut funktioniert.

Beispielsweise „weiß“ SCOPELAND ja von selbst, was für Daten eine Tabelle enthält. Hat man z.B. eine Spalte, die für einen Menschen nicht lesbare Schlüsselnummern enthält, die ja schließlich keinen anderen Zweck haben, als auf einen Datensatz einer verbundenen Katalogtabelle zu verweisen, dann wird man nicht mit diesen „dummen“ Nummern konfrontiert. Stattdessen werden gleich automatisch die Hauptfelder des jeweiligen Verweis-Datensatzes mit einbezogen (es wird automatisch ein „Outer Join“ gebildet). Folglich werden alle Daten immer automatisch in lesbarer Form präsentiert. Und auch gleich in einer sinnvollen Sortierung. Dies ist möglich dank der zugrundeliegenden ‚Metadaten’.

Welche Spalten der geöffneten „Haupt-“Tabelle und der eingezogenen Verweistabellen nun angezeigt werden sollen, dies kann man nun durch einfaches Ein- oder Ausklicken in einer Selektionsmaske (übers Menü bzw. Toolbar) selbst festlegen und so von dem intelligenten Erstvorschlag abweichen. Auch dies geht intuitiv einfach, in einer Baumdarstellung entlang der Verweise.

Mehr noch: die Tabelle kann (entsprechende Benutzerrechte vorausgesetzt) auch sofort mit Daten gefüllt bzw. geändert werden. Auch dies funktioniert auf intelligente Art und Weise. Will man z.B. die Branchenzuordnung in unserer Firmentabelle ändern, soll auch dies funktionieren, ohne dass man sich mit den anonymen Schlüsselnummern beschäftigen muss. Folglich wird automatisch eine

Auswahlfunktion angeboten, die man dann in wählbarer Präsentationsform (Auswahlfenster, Listbox, Radiobuttons,…) sofort benutzen kann.

Auch alle andere Anwendungslogik (siehe dazu Erläuterungen im Weiteren), die in Form von Metadaten inzwischen eingerichtet wurde, wird entsprechend sofort und vollautomatisch dabei berücksichtigt.

Und das ist schon recht erstaunlich: dieses simple „Öffnen“ einer Tabelle stellt eigentlich bereits ein voll funktionsfähiges, lauffähiges Programm dar, ein erstes Datenerfassungsprogramm für diese Tabelle.

Man kann nun tatsächlich sofort damit beginnen, Daten erfassen zu lassen oder aus externen Datenquellen dort hinein zu importieren. Je eher man viele, möglichst sogar vollständige Daten, in der Datenbank hat, umso leichter fällt dann das weitere Entwickeln.

Mit den Daten kommunizieren

Es ist sehr angenehm und elegant, während des Entwickelns ständig direkt und unmittelbar mit den Daten (bei kritischen Anwendungen natürlich mit Testdaten) kommunizieren zu können. Das gibt dem Entwickler die Sicherheit, genau das Richtige zu tun und vermeidet Fehler von Anfang an. Man sieht immer und sofort das Ergebnis seiner Arbeit.

Warum gibt es keine vorgefertigten „Abfragen“ in SCOPELAND?

Datenbanktabellen enthalten ja nur in ihrem relationalen Zusammenspiel sinnvolle Informationen. Beispielsweise ist die Branchennummer in unserem obigen Beispiel einer Firmentabelle kaum aussagekräftig; erst durch das Zusammenspiel mit der Branchenkatalogtabelle (genauer gesagt durch das Einbeziehen des Katalogs in die Firmentabellendaten) entsteht sinnvolle Information. Aus diesem Grund greift man meist nicht nur frontal auf Tabellen, sondern auf komplexe Abfragen aus mehreren Tabellen zu, die durch entsprechende Outer Joins miteinander verknüpft sind.

Bei vielen Datenbankwerkzeugen muss der Entwickler zunächst „Abfragen“ (auch Queries genannt) schreiben oder interaktiv konfigurieren; und die anschließend entwickelten Programme greifen dann auf diese fertigen Abfragen zurück. Würde man z.B. eine tabellarische Anzeige von Daten direkt mit einer Datenbanktabelle verbinden, dann würden ja (wegen der relationalen Logik) nur für den Menschen unlesbare Informationen angezeigt, zum großen Teil bestehend aus unverständlichen Schlüsselnummern. Und weil das Aufbauen eines SQL-Statements bei fast allen anderen Werkzeugen außer bei SCOPELAND eine sehr komplizierte Angelegenheit ist, separiert man die Abfragen von den eigentlichen Programmen, in der Hoffnung, diese noch mehrfach wiederverwenden zu können. Deshalb gibt es in vielen Datenbankentwicklungssystemen vordefinierte Abfragen.

Auf diesen Umweg aber kann man getrost verzichten; und auch hier stellt sich die Frage, warum es viele andere so umständlich machen, obwohl es doch viel einfacher geht.

Da man fast immer genau eine Entität der realen Welt und also genau eine Haupttabelle im Blick hat (hier entweder die Firmen oder die Branchen oder was auch immer), genügt es doch völlig, sich zunächst mit ebendieser Tabelle direkt zu verbinden und dann im Nachgang bei Bedarf die eine oder andere Tabelle in diese Datensicht interaktiv mit einzubeziehen. Auch wenn letztlich im Hintergrund ebenso komplexe Datenbankabfragen entstehen, der Anwendungsentwickler wird damit nicht konfrontiert, denn aus seiner Sicht ist es – dem gesunden Menschenverstand entsprechend – immer noch dieselbe eine Tabelle, die er zu Beginn geöffnet hatte. Alles andere, alle Einbeziehungen anderer Tabellen und alle sonstigen Einstellungen (z.B. Filter, Aggregat- und andere Datenbankfunktionen) sind lediglich optionale Verfeinerungen im Datenbankzugriff, die man auch später noch nach Belieben ergänzen oder wieder entfernen kann. Deshalb ist es auch viel intuitiver, direkt mit Tabellen anstatt mit vordefinierten Abfragen oder Views zu arbeiten.

Die späteren Anwendungen bestehen dann also aus zahlreichen komplexen Datenbanksichten und dennoch basieren diese alle stets auf genau einer Haupttabelle, auf jeweils einer Entität. So ist e, und so muss es auch sein.

Letztlich nähern sich auch handprogrammierte Anwendungen im Verlauf der Entwicklung an genau diese Architektur an, aber mehr oder weniger unbewusst und auf vergleichsweise umständlichem Wege. SCOPELAND-Anwendungen hingegen werden von vornherein so aufgebaut und sind damit von Natur aus in ihren Grundzügen logisch richtig und ergonomisch strukturiert.

Wie macht man das – „Tabellen einziehen“?

Zum Konfigurieren der Datensichten, die beim Öffnen einer Tabelle entstehen, steht eine sogenannte Selektionsmaske zur Verfügung, in der Sie alles einstellen können, was diese Datenbankabfrage angeht und noch viel mehr.

Darin werden Ihnen zunächst alle Felder/Spalten der geöffneten Tabelle angezeigt und Sie können den automatischen Vorschlag, welche Felder/Spalten sie sehen möchten, per Mausklick anpassen.

Einige der dort aufgelisteten Felder/Spalten, nämlich sämtliche Fremdschlüssel sind entsprechend als solche gekennzeichnet (bildlich dargestellt: „es liegt noch was dahinter“) und wenn Sie auf das entsprechende Feld klicken, dann wird (anhand der in der Metadatenbank beschriebenen Relation) die mit diesem Feld verbundene Tabelle in die Datensicht mit einbezogen.

Das SCOPELAND-Verfahren zum Einbeziehen von Tabellen in Datensichten erscheint als so selbstverständlich und intuitiv, dass man sich ernstlich fragt, warum das nicht Alle so machen.

Es ist kaum nachvollziehbar, warum es noch immer Datenbankprodukte gibt, bei denen man Verknüpfungen mühevoll dadurch herstellen muss, dass man die verknüpfbaren Felder zweier Tabellen manuell herausfinden und dann mit irgendwelchen Häkchen, Strichen oder sonstigen Kuriositäten miteinander verbinden muss. Wer das mal so gelernt hat, der könnte dieses archaische Verjoinen-Feature zweier Tabellen in SCOPELAND möglicherweise vermissen. Aber bitte glauben Sie uns: man braucht es nicht. Das ist schlicht und einfach branchenüblicher Unsinn.

Nun können Sie Felder der eingezogenen Tabelle in gleicher Weise herein- und herausklicken wie Sie es schon bei der Haupttabelle selbst getan haben.

Will man nun aber noch weitere Tabellen in die Datensicht mit einbeziehen und zwar solche, auf die von bereits einbezogenen Tabellen verwiesen wird, dann geht auch dies, und zwar über beliebig viele Ebenen hinweg. So kann man sich quasi durch die gesamte Datenbank hindurchklicken und alles anhakeln, was man sehen möchte. Dabei entsteht eine baumartige hierarchische Struktur von Tabelleneinbeziehungen.

Für manch einen ist es etwas verblüffend, wie es sein kann, dass sich ein wild verzweigtes Netzwerk relationaler Datenbanktabellen im Selektionskonfigurator als Baumstruktur repräsentiert. Das erscheint zunächst als ein Widerspruch, ist es aber nicht. Es ist so völlig richtig!

Aus der Sicht einer Tabelle ist ein relationales Netz immer ein Baum.

Aber warum? Nun, Verknüpfungen entstehen naturgemäß immer in Form von Einbeziehungen anderer Tabellen anstelle des jeweils abgehenden Verweisfeldes (Fremdschlüssels). Wie sonst?

Geht man nun so immer weiter vor, von einer Tabelle zur nächsten und immer so weiter, dann entsteht ganz von selbst eine Baumstruktur. Sie erreichen so alle in der Datenbank gespeicherten Informationen, die in irgendeiner Weise inhaltlich etwas mit dem jeweils aktuellen Datensatz zu tun haben. In dieser hierarchischen Datensicht kann nun jede Datenbanktabelle auch mehrfach vorkommen, aber in jeweils unterschiedlichem Kontext.

Beispielsweise kann man von einer Firmentabelle aus mehrfach auf eine Adresstabelle verweisen, einmal im Kontext „Postadresse“ und einmal im Kontext „Besuchsadresse“. Oder, mal etwas weiter hergeholt: über die Stadt, in der die Firma angesiedelt ist, zu deren Bürgermeister und dann dessen Wohnadresse oder vom Kreis oder Bundesland der Firma zur jeweiligen Hauptstadt und von da aus zur Adresse des zuständigen Gewerbeaufsichtsamtes.

Ein und dieselbe Tabelle (hier die Adresstabelle) kann also mehrfach in der Datenbankabfrage vorkommen, aber immer in anderen Zusammenhängen und folglich auch mit anderen Datensätzen! Aus Sicht der Haupttabelle sind diese Einbeziehungen völlig unabhängig voneinander und es spielt überhaupt keine Rolle, ob die vielen Einbeziehungen aus derselben oder aus unterschiedlichen Tabellen kommen, denn schließlich geht es ja um immer andere Dateninhalte.

Und so schließt sich der Kreis. So klärt sich der scheinbare Widerspruch auf: man folgt durchaus dem wild verzweigten Relationen-Netzwerk, manchmal sogar scheinbar im Kreis herum, aber es geht dabei immer um andere Dateninhalte. Nicht die Tabellen selbst, sondern die Dateninhalte sind – von einem Punkt aus betrachtet – immer hierarchisch strukturiert.

Solche Abfragen, in denen gleich mehrere Tabellen mehrfach vorkommen, sind keinesfalls eine Seltenheit. Allerdings sind die zugehörigen SQL-Statements, die man ja bei SCOPELAND glücklicherweise nicht von Hand schreiben muss, ziemlich unübersichtlich.

Mehrere, miteinander verbundene Tabellen öffnen

Man kann sich dabei nun nicht etwa nur innerhalb einer Datensicht, also einer Tabelle nebst ihrer n:1-Verknüpfungen, hierarchisch bewegen, sondern auch von Datensicht zu Datensicht. Hierbei gibt es ebenfalls ein hierarchisches Prinzip:

Eine jede Tabelle hat ja oftmals mehrere zugehörige Detailtabellen und häufig will man zu einem Datensatz einer Haupttabelle alle jeweils zugehörigen Datensätze aus solchen Detailtabellen sehen.

Beispielsweise zu einer Branchenbezeichnung alle Firmen dieser Branche und zu einer Firma alle Ansprechpartner. Solch eine „Master-Detail“-Beziehung (1:n-Relation) ist ein elementarer Grundbaustein aller Datenbankanwendungen. Dies wird in fast jedem Dialog mehrfach benötigt.

Dies ist mit SCOPELAND sehr einfach aufzubauen: man steht auf einem Datensatz der geöffneten Haupttabelle und lässt sich per Menü alle damit verbundenen Detailtabellen zur Auswahl anbieten. Per Klick wird dann diese Detailtabelle geöffnet (natürlich wieder mit derselben Eigenintelligenz, wie wir sie bereits bei der ersten Tabelle hatten) und zwar gleich so, dass sie nur diejenigen Datensätze anzeigt, die zu dem aktuellen Datensatz der Haupttabelle der ersten Datensicht gehören.

Der Mechanismus, der das Zusammenspiel beider Datensichten steuert, ist verblüffend einfach:

In der geöffneten Detailtabelle wird auf das Feld, welche in Gegenrichtung zur ersten Tabelle verweisen würde (z.B. in einer Ansprechpartnertabelle das Fremdschlüsselfeld „Firma“, welches die jeweilige Firmen-ID enthält) eine dynamische Selektionsbedingung gelegt, die auf das entsprechende Feld der ersten Datensicht verweist (hier auf die Firmen-ID im aktuell angezeigten Datensatz in der ersten Datensicht). Außerdem ist diese Detaildatensicht der ersten Datensicht logisch untergeordnet, denn sie wurde ja aus dieser heraus geöffnet. Sie ist sozusagen eine „Child-“Datensicht zur Ersten. Und wenn man in der ersten Datensicht auf einen anderen Datensatz wechselt, dann müssen logischerweise in der nachgeordneten Datensicht die Daten neu eingelesen werden. Das weiß und macht SCOPELAND selbsttätig – eben weil es sich um eine der ersten Sicht nachgeordnete Child-Datensicht handelt. Auf diese Weise wird sichergestellt, dass man immer korrekt die zusammengehörigen Daten auf dem Bildschirm sieht. Ein genial einfaches Prinzip.

Auch die Datensichten sind zueinander hierarchisch

Alles ist hierarchisch aufgebaut: nicht nur der logische Aufbau der einzelnen Datensichten, sondern auch das Zusammenspiel derselben.

---

*Die einzelnen Datensichten werden im SCOPELAND-Sprachgebrauch als „Direct View“ bezeichnet, weil sie **direkt** (und nicht über vorgefertigte Abfragen oder physische Views) entstanden sind und weil man darin bei entsprechenden Berechtigungen auch gleich **direkt** editieren kann.*

Solche Direct Views können nun, wie wir oben gesehen haben, in einem Master-Detail-Verhältnis zueinander stehen. Sie stehen immer in einem geordneten Verhältnis zueinander: Ausgehend von einem Ersten, der sozusagen global durch direktes Öffnen einer Tabelle entstanden ist, ordnen sich alle aus diesem heraus geöffneten Detail-Datensichten diesem unter.

Beispielsweise öffnet man aus einer Firmentabelle heraus eine untergeordnete Datensicht „Ansprechpartner“ oder „Mitarbeiter“, indem man die Tabelle aus dem Kontext der Firmentabelle heraus als „Detail“ öffnet. Aus dieser heraus kann man nun wiederum weitere Details öffnen, z.B. die Tabelle „Sprachen der Mitarbeiter“. Diese ist dann also der Detailtabelle nachgeordnet, liegt also schon zwei Ebenen unter der Firmentabelle.

Auf diese Weise können Datensichten beliebig tief geschachtelt werden, und dies bedeutet letztlich, dass dabei ein Baum von Datensichten (Direct Views) entsteht; in diesem Fall ausgehend von der Firmentabelle. Wechselt man ganz oben zu einem anderen Datensatz, dann wird automatisch alles Nachgeordnete aktualisiert (neu aus der Datenbank ausgelesen), damit man wieder die jeweils

zugehörigen Datensätze sieht. Wechselt man hingegen auf einer tieferen Ebene zu einem anderen Datensatz, dann muss nur der betreffende Zweig des Baumes aktualisiert werden.

Dieses Verfahren, von einer Tabelle zur nächsten zu gehen und als „Details“ zu öffnen und auf diese Weise zugleich immer sofort und vollautomatisch die jeweils zugehörigen Datensätze zu sehen, ist nicht nur zum Bau komplexer Anwendungen geeignet, sondern gerade auch im Ad-hoc-Modus, um sich mal schnell ad hoc zu informieren. Es ist einer der Schlüssel dafür, Datenbanken interaktiv lesen zu können.

Es ist übrigens sehr zweckmäßig, beim Öffnen abhängiger Direct Views immer streng der generellen Anordnungs-Ergonomieregel zu folgen, die da heißt:

Immer von links oben nach rechts unten.

Platzieren Sie also global geöffnete Tabellen nach links oben und schieben Sie sich die da heraus geöffneten Detail-Datensichten stets darunter oder rechts daneben.

Es entspricht der gewohnten Bedienfolge (zumindest von den heute verbreiteten IT-Systemen), dass sich alle Folgen von Bedienhandlungen stets nach rechts oder unten auswirken, so auch ein Satzwechsel in einem Direct View, welcher zur Folge hat, dass dann in nachgeordneten Datensichten andere Inhalte angezeigt werden.

Bei entsprechender Selbstdisziplin in der Anordnung der Datensichten wird das scheinbar komplizierte Zusammenspiel zwischen den DVs ganz leicht verständlich und plausibel.

Bezüge (temporäre Relationen, temporäres Master-Detail)

Das Master-Detail-Verhalten, wie z.B. das automatische Aktualisieren der Anzeige der Detaildaten infolge eines Satzwechsels im Master- Direct View, betrifft nicht nur die Beziehungen zwischen Direct Views, die auf „echten“ 1:n-Relationen beruhen, sondern auch auf sonstige, manuell herstellte Beziehungen zwischen Datensichten.

Will man beispielsweise zur Firmentabelle in der ersten Datensicht eine nachgeordnete haben, in der alle Sprachen angezeigt werden, die die Mitarbeiter dieser Firmen sprechen, dann wäre dies zwar durchaus entlang des Relationenmodells erreichbar, aber nicht in einem Schritt.

Für solche Ausnahmen kann man sich nun den Master-Detail-Charakter selbst aufbauen, indem man die „Sprachen der Mitarbeiter“-Tabelle direkt, also global öffnet und dann einen sogenannten „Bezug“ herstellt. Dieser temporäre Bezug wird aufgebaut zwischen der Firmenschlüsselnummer im ersten Direct View und der Firmenschlüsselnummer, die man im zweiten Direct View indirekt findet, indem man sich darin durch die einbezogenen Tabellen hangelt – in diesem Fall über die Verweisrelation zu den Mitarbeitern die dort enthaltene Firmennummer.

Solche Bezüge zwischen Datensichten, auch temporäre Relationen genannt, können Sie völlig frei aufbauen und damit auch Ausnahmefälle aller Art abbilden. So könnten Sie z.B. auch eine Datensicht anhängen, in der alle Sprachen angezeigt werden, die von den Mitarbeitern der Firma nicht oder nur zu einem gewissen Grad gesprochen werden. Oder alle Länder, in denen Sprachen gesprochen werden, die einer der Mitarbeiter der Firma verhandlungssicher spricht, und die außerdem in einem der Vertriebsgebiete liegen, in der die Firma lt. einer weiteren Tabelle „Vertriebsgebiete von Firmen“ tätig

ist. Im letzten Beispiel wirken sogar mehrere solcher temporären Bezüge parallel, sogar von unterschiedlichen Datensichten ausgehend.

Mit solch trickreichem Setzen von Beziehungen zwischen Datensichten kann man auf elegante und einfache Weise, rein deklarativ, nahezu alle denkbaren Datenbankabfragekonstellationen aufbauen, alles visuell und interaktiv und immer so, dass man sofort sieht, was man tut.

Ad-hoc-Zugriff, auch für ‚Power-User'

Eines der herausragenden Features von SCOPELAND ist die Möglichkeit, immer und überall auf einfachste Weise und in verständlicher, lesbarer Form an alle verfügbaren Informationen heranzukommen und zwar sowohl lesend als auch (konsistent) schreibend.

Dies gilt nicht nur für Entwickler oder Datenbankadministratoren. Es gilt auch für ausgewählte Endanwender, sog. „Power-User".

Des Ad-hoc-Modus von SCOPELAND ist gedacht als Befreiung aus dem strengen und engen Korsett der Applikation. Warum sollte jemand, der sowieso Datenherr für bestimmte Teile der Anwendung ist oder sonstigen uneingeschränkten Zugang hat, nicht einfach mal „direkt" in die Datenbank schauen dürfen? Warum soll er immer nur entlang der fest programmierten Bedienabläufe und über starre Bildschirmmasken gehen müssen? Warum soll er die Daten immer nur so in der Filterung und Reihenfolge und in dem Kontext sehen dürfen, wie ein Softwareentwickler sich das so gedacht hat?

Die Datenbank ist nicht Eigentum der Applikation bzw. soll es nicht sein. Sie ist Eigentum der Datenherren und sie sollen das Recht haben, mit ihren Daten zu tun, was sie möchten und wann sie es möchten – solange die Konsistenz und die Sicherheit der Daten dadurch nicht beeinträchtigt wird.

Bislang verhielt sich eine Datenbank immer irgendwie wie eine Blackbox. Aufgrund der komplexen Relationen-Struktur mit seinen vielen unlesbaren Schlüsselfeldern galt es als äußerst schwierig, nachzuschauen, was eigentlich in einer Datenbank für Informationen drin stecken. Daten waren immer nur innerhalb des Programms zu sehen, niemals direkt als solche. Und das ist nicht nur für die Anwender eine sehr drastische Einschränkung, sondern auch für die Entwickler selbst.

Der Ad-hoc-Zugriff ist eine Möglichkeit,
in Datenbanken wie in einem offenen Buch zu lesen.

… und nicht nur zu lesen, sondern darin auch…

… zu schreiben, ohne die für Datenbanken
unverzichtbare Konsistenz der Daten zu verletzen.

Und all dies gänzlich ohne ein spezifisches Anwendungsprogramm.

Wie ist das möglich? Man geht dazu genauso vor, wie oben beschrieben. Zuerst sucht man sich seinen Eintrittspunkt (eine Tabelle), öffnet diese Tabelle ad hoc mit all der oben beschriebenen Eigenintelligenz, kann darin frei selektieren, suchen und sortieren, die Datensicht mittels der Selektionsmaske anpassen und mit den so erzielten (zunächst meist tabellarischen) Daten macht man

was man will: anschauen, ausdrucken, exportieren und vielleicht auch verändern – zumindest falls das Berechtigungsprofil des Users dies zulässt.

Mit simplem Doppelklick kann man sich jeden Datensatz in einem gut gestalteten, automatisch generierten Formular ansehen und vielleicht auch (unter gesicherter Wahrung der Konsistenz der Daten) bearbeiten.

Und wenn Sie dann auf „Details“ klicken, erhalten Sie eine Auswahl aller anderen Tabellen, die im jeweiligen Kontext zur gerade angezeigten Tabelle und zum gerade aktiven Datensatz zugehörige Detail-Daten enthalten (Master-Detail-Prinzip).

So können Sie sich elegant, sowohl innerhalb der einzelnen Datenbankabfragen als auch durch immer weiterführende Master-Detail-Aufrufe, kreuz und quer durch die gesamte Datenbank hangeln. Sie sehen immer alle Daten im Klartext; alle dummen Schlüssel werden automatisch aufgelöst und sie werden somit zu lesbarer Klartext-Information.

All dies, ohne, dass Ihnen jemand dafür ein Programm hätte entwickeln müssen.

All dies geht einfach so, ganz von selbst.
Sie benötigen dafür nur korrekte Metadaten, das ist alles.
Kein spezielles Programm!

Hat man die Möglichkeiten des Ad-hoc-Zugriffs von SCOPELAND einmal verinnerlicht, dann fällt es schwer, sich vorzustellen, wie es jemals ohne dies gehen konnte. Das ist kein ausgefallen kompliziert-exotisches Verfahren, sondern ganz im Gegenteil: es erscheint dem routinierten SCOPELAND-Entwickler oder Power-User als das Natürlichste und Selbstverständlichste überhaupt.

Direkter Zugang zu den Daten? Darf man das denn dürfen?

Ja und Nein.

- ➔ Contra: natürlich nicht, zumindest nicht jeder! Und die, die es dürfen, sollen es nicht für alle Daten dürfen. Alles muss höchst restriktiv sein, Datenschutz ist wichtig.

- ➔ Pro: natürlich! Jeder für bestimmte fachliche Inhalte zuständige Datenherr hat sowieso alle seine Daten im Zugriff. Warum also sollte man ihm verwehren, direkt auf die Daten zu schauen? Warum sollte man ihn zwingen, um an seine Daten heranzukommen immer umständlich durch das schmale Fenster der Applikation zugreifen zu müssen – mit einem Vielfachen des eigentlich nötigen Aufwands?

Kann man diese beiden Sichten verbinden und es dem Einen wie auch dem Anderen recht machen? Ja, man kann.

Es ist notwendig und sinnvoll, genau zu überlegen, wer eigentlich mit welchen Daten was tun darf. Das gilt für Endanwender sowieso allemal, für Power-User oft und bei großen Projekten manchmal sogar für Entwickler und Administratoren.

Nun könnte man theoretisch auf Datenbankebene zu jeder Tabelle oder gar zu jedem Feld festlegen, welcher User in welcher Rolle dieses lesen oder verändern darf. Das geht, aber es ist eigentlich nicht die richtige Stelle und nicht die richtige Methode. Auf der einen Seite viel zu aufwändig (für Hunderte

von Benutzern bei Tausenden von Tabellen – wer soll das einrichten?) und auf der anderen Seite nicht präzise genug, denn es hängt oft vom Kontext ab, wann man welche Datensätze sehen und was man mit den Daten machen darf. Auch dafür noch zusätzliche Regeln in der Datenbank zu hinterlegen, das würde jedes vernünftige Maß an Aufwand sprengen. Außerdem hilft es wenig, wenn man auf der Oberfläche vorgegaukelt bekommt, man dürfe alles, um dann ständig von Datenbankfehlermeldungen traktiert zu werden, die einem sagen, dass man das, was man gerade tun will, nicht darf.

SCOPELAND wählt deshalb einen viel eleganteren Weg, den der „Freigabe“ ganzer Ordnersysteme voller Daten und Programme für bestimmte Benutzer oder Benutzergruppen und was da in Ordnern zusammengefasst wird, das lässt sich einfach und übersichtlich zuordnen. Mit wenigen Handgriffen und hoher Sicherheit gegen Fehleinrichtung lässt sich so präzise steuern, wer welche Teile der Datenbank sehen, schreiben oder gar verändern darf.

Ordnung in der Datenwelt: Ordner, Freigaben und Rechte

Diejenigen Ordner, die Daten oder Programme enthalten, die man nicht sehen darf, die werden gar nicht erst angezeigt. Jeder einzelne Nutzer sieht immer nur das, was er auch darf.

Das Verzeichnissystem dient zum einen der besseren Übersicht und der Navigation in den Datenbeständen, zum anderen der gezielten Freigabe für bestimmte Benutzer oder Benutzergruppen.

Es ist in jedem Fall zu empfehlen, dieses Ordnersystem inhaltlich zu strukturieren und nicht nach Art der Objekte. Darin gehören diejenigen Daten in einen Ordner, die inhaltlich zusammengehören und für die derselbe „Datenherr“ zuständig ist. Ein Datenherr ist derjenige, der ohnehin, egal ob mit oder ohne Anwendungsprogramm dazu, zuständig für die Korrektheit der eingegebenen Daten ist und die Pflicht und folglich auch das Recht hat, diese aktuell zu halten.

Jedes Verzeichnis gehört zunächst demjenigen, der es angelegt hat und ist darüber hinaus auch nur für den Hauptverantwortlichen, den „Eigentümer“ der Anwendung einsehbar. Ein Verzeichnis kann nun von diesen beiden gezielt für andere freigegeben werden. Eine Freigabe erfolgt, abgestuft nach Arten der Freigabe (z.B. nur lesend), im einfachsten Fall direkt für die betreffenden Personen (oder eine „Rolle“, in der eine Person tätig ist).

Und der eigentliche Clou am Ganzen: die Freigaben müssen nicht von der zentralen IT-Abteilung nach einem großen Masterplan vorgenommen werden, sondern man kann es dem Datenherrn selbst überlassen, ob und wem er direkt Einsicht in seine Daten gewähren möchte. Dies führt zu einer äußerst effizienten Selbstorganisation: sicher und einfach zugleich.

Alternativ zur Freigabe an Personen oder Gruppen können Freigaben auch für bestimmte „Zuständigkeiten“ erteilt werden. Der große Vorteil ist, dass dadurch die Abhängigkeit von den konkreten Personen aufgelöst wird. Inhaltliche Verantwortlichkeiten bleiben meist relativ unverändert, auch wenn Mitarbeiter der Unternehmen kommen und gehen oder ihre Funktionen wechseln. Haben Sie z.B. einen oder mehrere Controller im Unternehmen, dann ist dies eine Zuständigkeit „Controller“. Sie können sie im Rechtesystem eintragen und fortan alle Verzeichnisse, die geschäftsrelevante Daten enthalten, für die Zuständigkeit „Controller“ lesend freigeben. So einfach ist es, diesen den entsprechenden direkten Zugang zu den für sie relevanten Daten einzuräumen. Andere Beispiele für Zuständigkeiten sind z.B. „Vertriebsmitarbeiter“, „Vertriebsleiter“, „Produktionsleiter“, „Marketingmitarbeiter“ usw.

Die bei SCOPELAND gezielte Freigabe bestimmter Datenbestände für bestimmte Benutzergruppen oder für Personen mit bestimmten Verantwortlichkeiten löst dieses Problem auf elegante Weise.

End-user Computing (EUC)

Darüber hinaus gibt es auch noch einen „privaten“ Ordner für jeden Einzelnen dafür autorisierten Benutzer. Das gilt zunächst für alle Entwickler, kann aber auch auf ausgewählte Power-User ausgeweitet werden, die sich eigene Auswertungen oder individuelle kleine Programme selbst entwickeln dürfen.

Gedanklich bildet das die Struktur der Dateisysteme in einem weiträumigen Unternehmensnetzwerk nach. Fast jeder Benutzer hat, selbstverständlich, ein eigenes Dateisystem, in dem er sich frei bewegen kann und das i.d.R. für andere Endanwender auch unzugänglich ist.

Dies ist der Tummelplatz für das eigentliche „End-user Computing“. End-user Computing hat man ohnehin, auch wenn man es nicht will, nämlich in Form von zahlreichen unkontrollierbaren MS-Excel- und MS-Access-Dateien, und anderen tabellarischen Daten aller Art. Das Problem dabei ist ja, dass solche Daten meist weder professionell gesichert werden noch für übergreifende Auswertungen zur Verfügung stehen. Außerdem sind sie hochgradig redundant und es ist schwer, diese Mini-Anwendungen bei Bedarf weiter auszubauen.

Beim SCOPELAND-basierten End-user Computing geht es nicht etwa darum, allen Benutzern unkontrollierten Zugang zu den geschützten Unternehmensdaten einzuräumen, sondern im Gegenteil, die bislang unkontrolliert auf lokalen Daten verteilten Datenbestände zurückzuholen und der Kontrolle, Sicherung und Verwendbarkeit durch die IT-Verantwortlichen zuzuführen.

Clevere Fachanwender und Power-User können sich mit geringem Lernaufwand vergleichsweise anspruchsvolle Mini-Anwendungen selbst zusammenklicken. Das ist deshalb möglich, weil man dazu nicht programmieren muss und sehr gut intuitiv geführt wird. Wer einigermaßen fortgeschritten eine Tabellenkalkulation bedienen kann, der kann sich auch mit SCOPELAND seine eigenen Kleinanwendungen selbst entwickeln. Hierzu gibt man betreffenden Benutzern einfach das Recht, private Anwendungen zu erstellen und sich ggf. auch private Datenobjekte anzulegen. Dann können sie sich innerhalb ihres privaten Ordnersystems frei bewegen, ohne dadurch unkontrollierten Zugang zu anderen Daten und Anwendungen zu bekommen.

Eine besonders interessante Konstellation ist hierbei, dass diese Power-User sich ihre eigenen Kleinanwendungen bauen können, die in Teilen (allerdings meist nur lesend) Datenbestände der eigentlichen, zentralen Anwendungen verwenden, direkt und unmittelbar, ohne diese unnötig hin- und her kopieren zu müssen. Individuelle Ergänzungen und Erweiterungen der Zentralanwendung für bestimmte dafür autorisierte Benutzer sind so möglich.

Was bedeutet eigentlich der Begriff „Scopeland“?

Ein „Scope“ (engl.) ist die Menge aller Dinge, die man sehen oder tun kann. Alle Daten, die Sie (mit Ihren spezifischen Benutzerrechten) sehen und alles was sie mit diesen Daten tun dürfen, das ist Ihr persönlicher Scope, Ihr

---

Personal Scope

Jeder hat einen anderen Scope, in Abhängigkeit von seinem Benutzerprofil.

Und jeder kann sich innerhalb seines Scopes frei bewegen und selbst entscheiden, wie er die Dinge dargestellt haben will und wie er sie tun will. Das ist der zentrale Gedanke dahinter.

Man sieht daran, welche Bedeutung das Unternehmen Scopeland den Themen Ad-hoc-Zugriff und End-user Computing beimisst und welche zentrale Rolle es dabei folglich spielt, sicherzustellen, dass sich immer alles im Rahmen von klar definierten Berechtigungen und gesicherter Konsistenz bewegt. Und dies erlaubt, zumindest perspektivisch, eine völlig neue, viel einfachere und viel flexiblere Herangehensweise an manche IT-Themen. Auch wenn Sie diese Möglichkeiten für Ihre Anwender (vorerst) nicht freigeben wollen, vielleicht deshalb, weil Ihre Anwendung allzu kritische Daten und Prozesse enthält …

… zumindest die Anwendungsentwickler werden es zu schätzen wissen, sich in ihrem eigenen Revier, in ihrem

---

Scope – Land

---

wirklich ***frei*** bewegen zu dürfen.

Und wie kommt Anwendungslogik in die Metadaten?

Bei SCOPELAND gibt es keinerlei datenfreie Programme und auch keine Programme, die man erst später irgendwie mit den Datenbeständen in Verbindung bringt. Es geht schlichtweg immer um Datenbanktabellen (oder andere Datenobjekte). Und fast alles, was wir an Anwendungslogik benötigen, betrachten wir als Eigenschaften von Daten.

Einen Teil seiner Eigenintelligenz entnimmt SCOPELAND ja schon einer logischen Klassifikation der Datenobjekte. Je besser man die Daten klassifiziert, umso weniger Arbeit hat man später beim Bau der Anwendungen. Diese Informationen werden mit in den Metadaten abgelegt. Gute Metadaten sind die halbe Arbeit. Sowohl für Tabellen als auch für Felder/Spalten kann aus einer großen Vielfalt per Listbox ausgewählt werden, was für Daten das inhaltlich sind. Zum Teil werden diese Klassifikationen auch intelligent selbsttätig gesetzt, z.B. beim Anlegen neuer, verbundener Tabellen.

Aber das allein genügt natürlich noch nicht – zumindest nicht für eine richtig komfortable und robuste, endanwendertaugliche Softwarelösung. Dafür braucht man noch weitere Verfahrensweisen, um dynamisches Verhalten in die Anwendung hineinzubekommen – und auch diese lassen sich als (erweiterte) Eigenschaften von Daten verstehen.

Fast alle Anwendungslogik lässt sich auf eines zurückführen:
es sind Eigenschaften von Daten.

So ist es zum Beispiel eine Eigenschaft eines Datenfeldes, dass nur Eingaben in bestimmten Wertebereichen zulässig sind (z.B. dass das Alter eines Menschen nur bestimmte Werte annehmen kann, etwa „0 … 120“) oder dass das Eingangsdatum eines Antrags zeitlich nicht vor seinem Absendedatum liegen kann. Es gibt aber noch viel komplexere Eigenschaften.

So ist es ebenfalls eine Eigenschaft der Daten, beispielsweise einer Firmentabelle, in welche Kategorie ein Datensatz einzuordnen ist, z.B. ein Unternehmen nach Branche, Umsatz und Mitarbeiterzahl. Und es ist eine naturgegebene Konsequenz aus dieser Eigenschaft, dass diese Berechnung zu erfolgen hat, sobald die notwendigen Eingangsinformationen für diese Berechnung zur Verfügung stehen. Auf diese

Weise wird da, was man eigentlich als ein Programmalgorithmus ansieht, zu einer deklarativen Eigenschaft: eine Formel, vergleichbar zu einer Formel in einem Spreadsheet und mehr braucht man nicht. Was mit dieser Formel zu tun ist und wann das auszuführen ist, das weiß SCOPELAND von selbst – genauso wie eine Tabellenkalkulation, in der man ebenso einfach einem Feld eine Formel zuordnet und fertig! Das Programm weiß von selbst, was es mit dieser Formel zu tun hat.

SCOPELAND weiß von selbst, was mit dieser Formel zu tun ist.

Neben den einfachen Eigenschaften, die generell und immer heranzuziehen sind, gibt es nun auch solche, die nur unter bestimmten Umständen gelten. Diese nennt man „Regeln“ und sie sind immer so aufgebaut, dass sie aus dem eigentlichen Inhalt bestehen und aus einer Bedingung, die erfüllt sein muss, damit sie gelten.

Beispielsweise könnte eine Regel besagen, dass der Umsatz eines Unternehmens in einem Förderantrag kleiner sein muss als ein bestimmter Schwellwert – was aber nur dann gilt, wenn sich das Unternehmen als mittelständisch fördern lassen möchte. Meist gibt es dann für einen Sachverhalt mehrere Varianten: unter diesen Umständen gilt dies, unter jenen das und sonst jenes. Nach diesem Prinzip wird SCOPELAND zu einem „regelbasierten“ System. Die Regeln, im üblichen IT-Sprachgebrauch manchmal auch als „Business Rules“ bezeichnet, beschreiben einen wesentlichen Teil derjenigen Anwendungslogik, die sich nicht schon von selbst aus dem Datenmodell heraus ergibt. Und sie ist auf diese deklarative Weise viel kompakter, viel kürzer und zudem einfacher und übersichtlicher als eine algorithmische Notation desselben Verhaltens.

Sodann gibt es eine Vielzahl unterschiedlicher Regeltypen, z.B. welche Werte ein Feld annehmen darf, wann es sichtbar sein soll, wie es dargestellt werden sollte oder woher eine Tabelle ihre Daten bezieht und was nach dem Laden einer Seite passieren soll.

Nach diesem Prinzip lässt sich nun nahezu jegliche Programmlogik deklarativ beschreiben. Es ist zwar für manch einen ungewohnt, bei Allem in der Kategorie von Eigenschaften zu denken und es ist nicht immer auf den ersten Blick ersichtlich, wie man eine ganze, komplizierte Anwendungslösung auf Eigenschaften von Datenobjekten herunterbrechen soll. Aber es geht.

Und es ist viel einfacher als es zunächst scheint.

Die Metadatenbank wird solange mit Regeln gefüttert, bis sie ausgereift ist.

Es wird niemandem abverlangt, Alles auf einmal zu „modellieren“, was zweifelsohne eine große und allzu abstrakte Herausforderung wäre.

Man beginnt immer bei den Datenstrukturen und dann wird, Schritt für Schritt, Formel für Formel, ergänzt, was den Datenobjekten noch fehlt, um sich korrekt zu verhalten. So bauen sich auf die einfachen, strukturbeschreibenden Metadaten, diejenigen, die das Verhalten der Daten beschreiben: statische und dynamische, einfache und komplexere. Und dies ist im Prinzip dann auch schon der Kern der Anwendung. Ein spezielles algorithmisches bzw. prozedurales Programm im herkömmlichen Sinne ist überhaupt nicht erforderlich.

Die Anwendung besteht im Wesentlichen aus beschriebenen Daten.

Die Benutzeroberfläche, die man sich mit der Maus zusammenschieben kann, ist dann nur noch ein darüber gelegtes Layout nebst durchdachter Benutzerführung. Man wählt später nur noch aus, welche Informationen man an welcher Stelle der Anwendung haben will und wie sie aussehen sollen.

Was diese Daten aber tun sollen, das wissen sie eigentlich von selbst. So das Prinzip von SCOPELAND. Natürlich ist jede Idee immer nur endlich umsetzbar. In der realen Welt kommt man hin und wieder nicht drum herum, doch ein wenig Ablauflogik oben drüber zu bauen, aber es handelt sich dabei wirklich um Ausnahmefälle.

SCOPELAND-Anwendungen werden schrittweise verfeinert,
von unten nach oben. Oder auf allen Ebenen gleichzeitig.

Gute Metadaten sind die halbe Arbeit.

Sorgfalt bei der Metadatenbeschreibung zahlt sich später immer aus.

Anwendungen entwickeln

Das Grundprinzip – kompakt im Überblick

Die Entwicklung interaktiver Anwendungen (Windows- oder Webanwendungen) erfolgt in Form von „Seiten“, womit je nach Zielplattform entweder Windows-Masken oder Formulare in Webseiten bzw. ganze eigenständige Webseiten gemeint sind. Die Anwendung besteht aus einer großen Vielzahl einzelner Seiten, die gegenseitig verlinkt oder als Unterformulare (Registerordner o.ä.) in andere Seiten eingefügt werden. Jede Seite wird im Interesse größtmöglicher Flexibilität unabhängig von allen anderen entwickelt. Übergreifende Klassenmodelle, logische Persistenzschichten oder Ähnliches werden bewusst vermieden, um die Abhängigkeiten der Seiten voneinander soweit wie möglich zu minimieren. Alle Seiten greifen, von Ausnahmen wie z. B. WebServices oder Service-Aufrufen abgesehen, logisch gesehen direkt auf die Datenbank zu, sie enthalten ihre eigene vollständige Programmlogik und ihre eigene Präsentationsschicht.

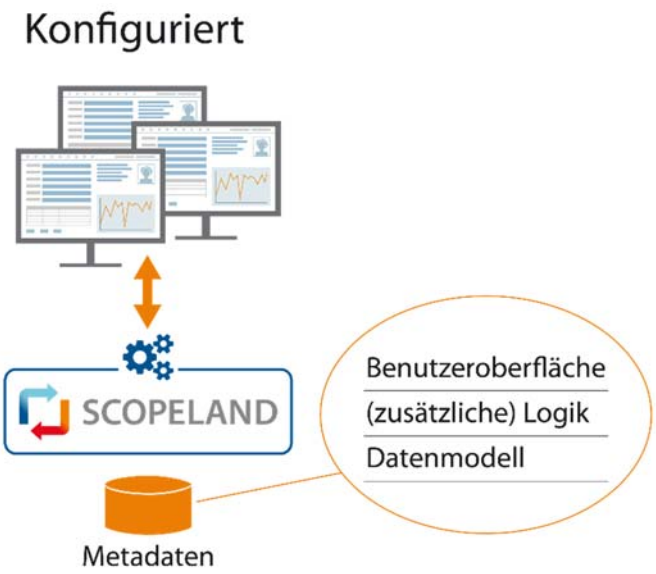


Abbildung 2: Struktur der Metadaten

Angaben, welche nicht nur die Inhalte beschreiben, sondern auch als Grundlage der Codegenerierung dienen.

Physisch kann dies, wie bei Webanwendungen üblich, je nach Zielplattform durchaus auf mehrere Ebenen aufgeteilt sein, so dass echte Multi-Tier-Anwendungen entstehen. Logisch gesehen aber bleibt alles immer eine Einheit und ermöglicht deshalb hochgradig agile Anpassungen an der Software. Da alle Informationen zu einem Datenobjekt bzw. zu seiner Verwendung in einer Seite kompakt und relational miteinander verbunden in der Metadatenbank stehen, kann jederzeit an jeder Stelle und auf jeder Ebene geändert werden, mit sehr hoher Wahrscheinlichkeit, dass keine anderen Programmteile davon betroffen sind. Dies gilt sogar für das zugrundeliegende Datenmodell: mit wenigen Mausklicks können z. B. Änderungen am Datenmodell vorgenommen werden und die betroffenen Seiten passen sich weitgehend eigenständig an diese Veränderungen an.

Die kompakte relationale Speicherung der Metainformationen ist der Schlüssel für die evolutionäre Veränderbarkeit von Softwarelösungen.

Die Vielfalt der verfügbaren Mittel zur Entwicklung anspruchsvoller interaktiver Windows- oder Webanwendungen ist sehr groß und natürlich kann hier nicht erschöpfend beschrieben werden. Beispielhaft seien hier nur einige ausgewählte Features benannt: Masken- oder tabellarische Darstellung von Daten, Eingabefelder, Listboxen, Radiobuttons, Checkboxen, Menü- und Treecontrols, Plausibilitäts- und Berechnungsregeln, Zustände, Status und Aktionen, Properties von Tabellen und Feldern/Spalten, Filterfelder, Schaltflächen, Registerordner, Hyperlinks, statische und dynamische Bilder, Rahmen und Linien, vordefinierte Formatvorlagen und vieles mehr.

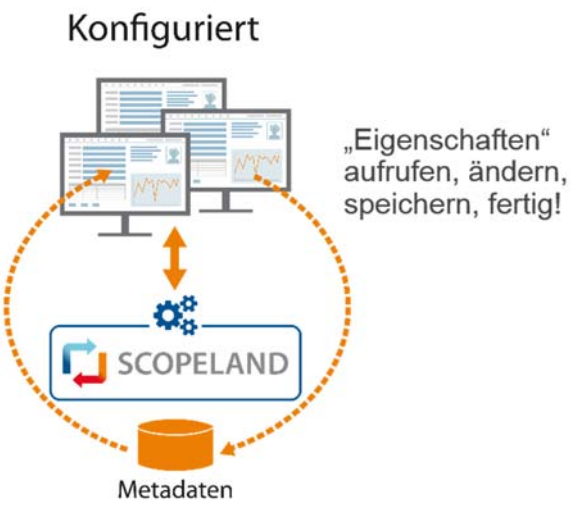


Abbildung 3: Iterative Entwicklungsmethodik

Die laufende Entwicklung und Weiterentwicklung erfolgt, indem punktuell die betreffenden Anwendungsteile laufend verfeinert und verbessert werden, ohne dass der gesamte Code einer Umprogrammierung, Versionserhöhung oder dergleichen unterliegt.

Das Schlüsselelement beim Aufbau einer interaktiven Seite sind dabei die „Datensichten“ (in der SCOPELAND-Sprache auch als „Direct View“ bezeichnet), interaktiv konfigurierte Sichten auf relational verknüpfte Datenbanktabellen, mit Filtern und Datenbankfunktionen, beschreibbar und unter Umständen angereichert um Berechnete Spalten und andere Arten von Pseudofeldern. Das Zusammenklicken solcher Datensichten erfordert keine SQL-Kenntnisse und, was besonders wichtig ist, auch keine genaue Kenntnis der zugrundeliegenden Datenstrukturen! Dieses erstaunliche Feature ist nur deshalb möglich, weil die allgegenwärtigen Metadaten ja die Daten „kennen“. Sie wissen also, welche Art von Informationen sie enthalten und wie die einzelnen Tabellen miteinander verknüpfbar sind.

Eine Seite besteht nun i.d.R. aus mehreren miteinander verknüpften Direct Views und die so aufgebaute Verknüpfungslogik zwischen den Dateninhalten einer Seite komplettiert nun die ohnehin in Form von Metabeschreibungen (Eigenschaften der Daten, Regeln, Zustände und Aktionen) bereits vorhandene Anwendungslogik. Schaltflächen wie Buttons, Icons o.ä. bekommen vordefinierte oder individuelle Aktions-Expressions zugewiesen, welche sich aus einer übersichtlichen Anzahl an Standardfunktionen wie z. B. „NewRow()“ für Satz einfügen oder „Save()“ für das Rückspeichern neuer oder geänderter Daten in die Datenbank zusammensetzen.

Damit schließt sich der Kreis und man erhält mit verblüffend einfachen Mitteln letztlich eine voll funktionsfähige Seite und in der Kombination mehrerer solcher Seiten eine komplette Anwendung. Im Idealfall entstehen so große und komplexe interaktive Anwendungen, ohne auch nur eine einzige Zeile Programmcode und ohne gar zu komplexe Formeln und Ausdrücke geschrieben zu haben.

Für Ausnahmen besteht aber immer noch die Möglichkeit, kleine Codefragmente händisch zu schreiben und so in die Metadaten einzubetten, dass sie automatisch sauber in die generierten Programme eingebettet werden – ohne dass Eingriffe am generierten Code erforderlich wären. Da diese Codeschnipsel direkt auf alle Datenobjekte und Systemfunktionen objektorientiert zugreifen können, beschränken sich diese i.d.R. auf die Definition einiger weniger Variablen und Funktionen, ohne eigene Objektmodelle, ganz einfach strukturiert und kompakt abgelegt in wenigen Zeilen. Angesichts der Einfachheit und des Ausnahmecharakters solcher eingebetteter Code-Schnipsel wird der Charakter der Programmierfreiheit von SCOPELAND dadurch in keiner Weise eingeschränkt. Nahezu alles wird lediglich konfiguriert, von der Komplexität her vergleichbar mit der Ausgestaltung einer anspruchsvollen Excel-Seite.

Seiten und Applets

Die eigentliche „Anwendung“, also das Programm, mit dem die späteren Anwender im Normalfall arbeiten werden, entsteht nun durch die Arbeit des SCOPELAND-Entwicklers, der nacheinander Seite

für Seite, Maske für Maske, Schnittstelle für Schnittstelle entwickelt, jede für sich und jeweils als eine in sich vollständige lauffähige Einheit.

In SCOPELAND gibt es keine unflexiblen Persistenzschichten, keine übergreifenden Objektmodelle oder sonst etwas, was zentral zu planen und in den einzelnen Programmteilen anzuwenden wäre. Jede Seite hat ganz bewusst und ganz absichtlich einen direkten Durchgriff zur Datenbank, weil nur so eine hinreichende Flexibilität hinsichtlich späterer inhaltlicher Änderungen gesichert werden kann, nebst einer weitestgehenden Unabhängigkeit von den einzelnen Entwicklern. Das einzig Globale an einer Seite sind die o.g. Metadaten und diese bestehen ausschließlich aus wohlstrukturierten, deklarativen Beschreibungen, im Idealfall völlig frei von jeder Art von Algorithmen. Alles nur pure Sachinformation.

Wir verwenden übergreifend für alles, was man unabhängig entwickelt, den Begriff „Seite“. Dabei kann es sich z.B. um eine Windows-Maske, eine Webseite, ein Unterformular, ein ETL- oder sonstiger Datenfluss, ein Dokumentendruck oder eine Schnittstelle handeln. Die einheitliche Nutzung des Begriffs „Seite“ für all dies hat auch den Hintergrund, dass bei SCOPELAND der innere logische Aufbau dieser Programme – wie wir noch sehen werden – in allen Fällen gleich ist. Jede Seite ist autark, technisch gesehen sozusagen ein unabhängiges Modul. Jede Seite kann problemlos entfernt, hinzugefügt oder ausgetauscht werden, als Patch nachinstalliert oder getrennt von allen anderen verändert werden. Jede Seite für sich enthält alles, was sie braucht und im Zusammenspiel mit den entsprechenden Runtime-Komponenten ist sie ein vollwertiges Programm.

Eine Applikation besteht aus einer Vielzahl von Seiten und diese können sich gegenseitig aufrufen oder einbetten oder in ein Menü eingehängt sein und in ihrer Gesamtheit stellt die Menge der Seiten das Anwendungsprogramm dar.

Alle diese Seiten bestehen selbst wiederum aus Metadaten und sie werden auch in der Metadatenbank gespeichert (als ein im Direct Desk ausführbares Programm oder zumindest als Entwurfsansicht). Die Seite ist gewissermaßen eine kleine Mini-Metadatenbank. Sie enthält den jeweiligen Ausschnitt aus der großen Metadatenbank (teilweise als Kopie) und zusätzliche, nur für diese Seite geltende Metadaten. Letztere entstehen teils durch interaktive Eingabe, beispielsweise von zusätzlichen Plausibilitätsregeln, die nur in diesem Kontext gelten und teilweise automatisch infolge der Aktivitäten des Entwicklers, z.B. wenn er Felder auf dem Bildschirm verschiebt oder abweichend gestaltet und natürlich wenn er die Auswahl der Daten, die man jeweils sieht, umkonfiguriert.

Hierbei ist es ganz entscheidend, dass auch die manuell ausgefeilten Programme (die „Seiten“) immer noch keinen manuell geschriebenen Programmcode enthalten, sondern ausschließlich wohlstrukturierte, deklarative Metadaten. Dadurch und nur dadurch ist es jederzeit möglich, alles auf allen Ebenen zu verändern.

> *Hinweis: Synonym zum Begriff „Seiten“ sprechen wir gelegentlich auch von „Applets“, was ein älterer Begriff aus den Vorläufer-Produkten von SCOPELAND ist und nichts zu tun hat mit den erst danach aufgekommenen Java-Applets. Hier meint es einfach nur eine Verkleinerungsform von „Applikation“. Meist benutzen wir hier den Begriff „Seite“ übergreifend für alles, was man mit SCOPELAND baut, und bezeichnen solche Seiten als Applets, die direkt im Direct Desk unter Windows ausführbar sind. Und Seiten, aus denen heraus letztlich extern ausführbarer Programmcode generiert wird (z.B. Webseiten oder druckbare Reports) bestehen dementsprechend aus einem Applet als der Entwurfsansicht im Direct Desk und dem daraus generierten Programmcode für das jeweilige Zielsystem.*

Wie entstehen „Seiten“?

Als Entwickler entscheidet man sich stets zuerst für eine der in den Metadaten beschriebenen Tabellen als Einstiegspunkt in die Datenbank. Meist handelt es sich um die Tabelle, welche den Hauptinhalt der jeweiligen Seite repräsentiert, beispielsweise eine Firmentabelle, wenn man Firmendaten pflegen will, eine Branchentabelle, wenn man den Branchenkatalog pflegen will usw.

Allerdings gibt es auch immer wieder Situationen, in denen es im Interesse eines optimierten inneren Aufbaus der Seite besser oder sogar notwendig ist, mit einer anderen Tabelle als Einstiegspunkt zu beginnen. Hier jeweils intuitiv sofort zu wissen, wo man jeweils optimal ansetzt, das ist eine der Dinge bei der SCOPELAND-Entwicklung, wo man als Anfänger manchmal in die falsche Richtung laufen kann und wo längere Erfahrung sehr hilfreich ist.

Den Einstiegspunkt wählt man am zweckmäßigsten mit der Menüfunktion „Neue Seite“. Hat man sich für eine Einstiegstabelle und für die gewünschte Seitenart (z.B. eine interaktive „normale Seite“) entschieden und „Fertigstellen“ betätigt, so wird vollautomatisch für genau diese Tabelle ein Entwurf einer Seite erzeugt, mit der Sie konsistent Daten eingeben, ändern und löschen können – jedenfalls sofern Ihr eigenes Rechteprofil genau dies auf diesen Daten erlaubt (was aber bei Entwicklern meist der Fall ist). Auf diese Weise kommen übrigens auch die Zugriffsrechte in die Datenbank, die die künftigen Nutzer haben sollen: sie vererben sich aus denen des Entwicklers, soweit er sie nicht einschränkt. Dadurch erhalten die späteren Anwender punktuell im Kontext dieser Seite die Möglichkeit auf Daten zuzugreifen, zu denen sie selbst keinen generellen Zugang haben.

Der Entwurf enthält noch keine Schaltflächen für derartige Funktionen, weil ja noch nicht klar ist, wo und wie die Seite später mal eingebunden wird und ob Sie solche Elemente direkt auf der Seite erwarten – hilfsweise stehen aber alle grundlegenden Funktionen im Direct Desk-Menü bereit. Die zugehörigen Programmfunktionen sind in der Seite aber quasi alle schon drin, man muss sie nur noch auf die Oberfläche bringen.

Dazu können Sie mit den entsprechenden Funktionen und Menüpunkten der Entwicklungsumgebung z.B. Schaltflächen, Hyperlinks, Registerordner für Unterformular uvm. erzeugen, wahlweise einzeln oder gleich im Block alles was Sie so benötigen.

Noch beschränkt sich eine so erstellte neue Seite auf eine einzige Datensicht, einen sog. „Direct View“, und sie sieht noch recht bescheiden aus, kann aber interaktiv immer weiter ausgebaut werden.

Direct Views

Die „Seiten“ im SCOPELAND-Sinne bestehen aus einem Baum hierarchisch angeordneter aktiver Datensichten, auch „Direct Views“ genannt. Ein Direct View ist ein aktives Objekt, welches eigenständig die jeweils aktuell gültigen Dateninhalte bereitstellt, Änderungen entgegennimmt und in die Datenbank (bzw. Datenquelle) zurückspeichert, vergleichbar zu einem sog. “Dataset“ in der .net-Umgebung. Anders als dieser enthält ein Direct View aber zusätzlich eine inhaltliche, wohlstrukturierte Beschreibung der (verknüpften) Datensicht, welche gleichfalls hierarchischer Natur ist. Ein Direct View repräsentiert jeweils eine Datensicht.

Konkrete SQL-Statements für Lese-, Einfüge- und Änderungs-Operationen sind also (zumindest auf der logischen Definitionsebene) nicht erforderlich, sondern werden bei Bedarf (während des Zugriffs oder im Zuge der Code-Generierung) dynamisch generiert. Der hierarchische Aufbau der Datensichten resultiert aus der Erkenntnis, dass sich ein netzartiges ER-Modell aus Sicht eines Knotens (einer Tabelle) immer als ein Baum repräsentiert. Dieser kann zwar durchaus über unterschiedliche

Relationspfade ein- und dieselbe Tabelle mehrfach enthalten, bezieht sich dann aber i.d.R. auf unterschiedliche Datensätze. Deshalb ist und bleibt es immer eine klare Baumstruktur.

Die Direct View-Logik geht davon aus, dass jede sinnvolle Datensicht grundsätzlich immer von einer konkreten „Haupttabelle“ ausgeht und niemals mehrere gleichrangig behandelt. Weitere Tabellen können entlang der Relationen (insbesondere der zu-1-Relationen) in die Datensicht mit „einbezogen“ werden und dies auch kaskadierend, ggf. sogar rekursiv. Schreiboperationen beziehen sich immer auf die Haupttabelle. Will man (was extrem selten erforderlich ist) ausnahmsweise eine Änderung in eine andere als in die Haupttabelle zu schreiben, so legt man sich hierfür eine weitere Datensicht an.

Hierarchisch in doppelter Hinsicht:
Jede Datensicht hat stets eine Haupttabelle (eine Entität) mit eingezogenen weiteren Tabellen und eine Bildschirmseite besteht aus verknüpften Datensichten.

Aus Sicht einer Entität ist das relationale Netz ein Baum und der Baum hat zwei Arten von Ästen: Verweise und Details. Dies kann dazu genutzt werden, mit einem einfachen Klickverfahren quasibeliebige Datensichten zusammenzustellen, ohne die dem zugrundeliegende Datensicht im Detail kennen zu müssen.

Das Einbeziehen einer weiteren Tabelle erfolgt logisch gesehen im Sinne eines „Vertiefens“ der Information, die die jeweiligen Fremdschlüsselfelder enthalten (Beispiel: ausgehend von einem Länderschlüsselfeld in einer Firmentabelle kann man in eine Ländertabelle hinein vertiefen, welche mehr Informationen über das jeweilige Land enthält). Dieses Vertiefen wird über ein Tree-Control-Verfahren durch ein Aufklappen symbolisiert. Dies ist ein sehr elegantes Verfahren zur interaktiven Definition von Datensichten, welches sehr intuitiv und leicht verständlich ist.

Auf jeden Fall enthält ein jeder Direct View stets nur Zeilen, die in einem direkten Bezug zu den selektierten Zeilen der Haupttabelle stehen. Es handelt sich also um eine flache Struktur. Master-Detail-Daten (z.B. aufgrund von zu-n-Relationen) sind grundsätzlich in weiteren Direct Views abzubilden, welche dem aktuellen Direct View i.d.R. nachgeordnet sind (Parent-Child-Beziehung).

Der Direct View enthält ferner in ebenso strukturierter Form die Auswahl der gewünschten Felder/Spalten, alle Selektionsbedingungen, Anwendung von Aggregat- und anderen Funktionen, Sortierungsangaben uvm. Der Direct View kann Pseudofelder enthalten, die keinen direkten Bezug zu konkreten Datenbankspalten haben und deren Inhalt berechnet wird (berechnete Spalten), frei eingegeben werden u.Ä. Ein Sonderfall solcher Pseudofelder sind Recherchefelder für interaktive Selektionsbedingungen. Darüber hinaus sind auch Pseudozeilen (für Gruppensummenzeilen) und andere Ausnahmen möglich.

Wir gehen davon aus und es hat sich in der Praxis immer wieder bestätigt, dass das hier skizzierte Verfahren, in eine Haupttabelle entlang der Relationen weitere Tabellen mit einzubeziehen und aus diesen die gewünschten Felder schlichtweg auszuwählen, ein geeignetes Mittel ist, um nahezu alle sinnvollen Abfragen zu definieren und somit nahezu alle sinnvollen SQL-Statements zu erzeugen. Für die verbleibenden Ausnahmefälle stehen entsprechende Sonderverfahren zur Verfügung (z.B. über „temporäre Relationen“).

Wenn darüber hinaus auf der Ebene der Basis-Metadaten für alle Objekte verständliche Synonyme und Beschreibungen eingetragen wurden, dann kann man davon ausgehen, dass man sich als Entwickler

oder Power-User (Reporting-User) mit SCOPELAND quasi beliebige Datensichten/SQL-Statements „zusammenklicken“ kann, ohne die dem zugrundeliegenden Datenstrukturen im Detail kennen zu müssen und ohne SQL zu kennen. Die so definierten Datensichten sind abstrakt definiert und RDBMS-unabhängig. Konkrete Ausprägungen im jeweiligen Native-SQL werden informativ angezeigt. Eine manuelle Änderbarkeit der Statements ist nicht erforderlich. Wer mit SCOPELAND entwickelt, benötigt keine SQL-Kenntnisse, es sei denn, zum Zweck der Fehlersuche oder des Tunings.

Der innere Aufbau eines Direct Views

Jedes Öffnen einer Tabelle, jeweils inklusive entsprechender, eingezogener Verweistabellen, stellt innerhalb von SCOPELAND einen „Direct View“ dar.

Eine davon ist immer die „Haupttabelle“, das ist die Tabelle, die man „geöffnet“ hatte.

Struktur eines Direct View (enthaltene Tabellen):

Haupttabelle → *Verweistabelle* → *Verweistabelle*

→ *Verweistabelle*

→ *Verweistabelle* → *Verweistabelle*

→ *Verweistabelle*

usw.

Jeder Direct View repräsentiert immer ein SQL-Statement, mit dem in einem Schritt die Daten aus der Datenbank ausgelesen werden können. Dieses Statement wird dynamisch erzeugt und man muss sich als SCOPELAND-Entwickler im Normalfall damit auch nicht befassen. Es ist aber vorteilhaft, die inneren Zusammenhänge von SCOPELAND zu verstehen und zu wissen, was hier in der Datenbank passiert.

Dieses SQL-Statement zum Auslesen der Daten umfasst nun in der Regel mehrere Tabellen und beschreibt eine Datensicht im Sinne der bekannten Datenbank-Welten, oftmals auch als „Query“ bezeichnet.

Es enthält auch die manuell präzisierte Feldauswahl, Selektionsbedingungen und vieles anderes, die man in der sog. Selektionsansicht eines Direct Views konfiguriert.

Dieses könnte man dann rein theoretisch auch als Datenbank-View abspeichern. Ein View ist eine in der Datenbank abgespeicherte Datenbankabfrage, welche dann anschließend so ähnlich wie eine Tabelle ausgelesen werden kann. Davon ist der Begriff „Direct View“ abgeleitet. Im Gegensatz zum Datenbank-View, der ja bekanntlich ***indirekt*** auf die Daten zugreift und auf den nur eingeschränkt schreibend zugegriffen werden kann, kann SCOPELAND aber damit direkt alles tun: uneingeschränkt lesend und schreibend auf die Datenbank zugreifen. Deshalb der Begriff „***Direct***-View. Außerdem kann sich ein Direct View auch noch zur Laufzeit dynamisch modifizieren und kann etliches enthalten, was über die Möglichkeiten einer Datenbank weit hinausgeht. Und er wird nicht in Form von schwer lesbarem, unstrukturiertem SQL-Quelltext gespeichert, sondern auf höherer Metadaten-Ebene: wohlstrukturiert und jederzeit einfach änderbar.

Schreiboperationen eines Direct Views erfolgen jedoch per Definition immer ausschließlich in der Haupttabelle. Dies ist keine Einschränkung, sondern ergibt sich aus der Natur der Sache. Die einbezogenen Verweistabellen haben ja hier nur erläuternde Funktion, um die anonymen Schlüssel in lesbare Information zu übersetzen. Würde man eingezogene Felder beschreibbar machen, dann wäre

es für den Benutzer nicht eindeutig, ob man auf diese Weise einen anderen Satz in der Verweistabelle auswählen, einen neuen Katalogdatensatz erzeugen oder gar den Kataloginhalt ändern möchte. Außerdem würde sich ja ein Ändern des Katalogs auch auf die Einbeziehungen aus anderen Datensätzen auswirken, was manch einem Benutzer vielleicht nicht klar sein könnte. Deshalb verbieten sich solche Gedanken grundsätzlich: Eingezogene Felder müssen immer schreibgeschützt sein. Will man Kataloge änderbar machen, so muss man dafür gesonderte Programmfunktionen (z.B. als Katalogpflegeseiten) bereitstellen.

Mehrere Direct Views in einer Seite

Ein SCOPELAND-Applet besteht i.d.R. aus mehreren Direct Views, im einfachsten Fall aus genau einem für die zu allererst geöffnete Tabelle und dann für jede zugehörige hier benötigte Detailtabelle einen Weiteren. Genau wie beim oben beschriebenen Ad-hoc-Öffnen von Datensichten im Master-Detail-Kontext, baut man sich auch die Anwendungsseiten aus mehreren Direct Views zusammen.

Die Montage einer Seite erfolgt durch das Verknüpfen von Datensichten, genauer gesagt teilweise durch das Öffnen von Detailtabellen zum jeweiligen Kontext (ausgehend von der aktuellen Cursorposition) und teilweise durch sonstige individuell hergestellte Bezüge zwischen separat geöffneten Direct Views.

Reale Seiten bestehen fast immer aus zahlreichen (sichtbaren oder nicht sichtbaren) Direct Views. Die Hierarchie entsteht beim Entwickeln quasi selbsttätig, z.B. beim Öffnen einer „Detaildaten“-Sicht, ausgehend von einer aktiven Master-Datensicht. Die Verknüpfung der Datensichten erfolgt teilweise automatisch, teilweise manuell, indem den jeweils nachgeordneten Datensichten Bezüge zu übergeordneten Datenfeldern als dynamische Selektionsbedingungen zugeordnet werden.

Entlang der Hierarchie, optimiert entlang dieser Bezüge, baut sich ein vollautomatischer Refresh-Mechanismus auf, der eigenständig dafür sorgt, dass alle Datensichten zum jeweils richtigen Zeitpunkt aktualisiert werden.

Das Einfügen neuer Datensätze in einen Direct View erfolgt stets unter automatischer Belegung der jeweiligen Selektionsbedingungen einschließlich der jeweils aktuellen Dateninhalte der verwendeten Bezüge. Daraus wiederum folgt, dass man bei entsprechend reichhaltiger Verwendung von berechneten Spalten und Bezügen zwischen den Datensichten durch simple Einfügebefehle neue Daten erzeugen kann, ohne dabei die rein deklarative Modellebene verlassen zu müssen.

Die Direct Views einer Seite sind nun entsprechend ihrer Relationen oder temporären Bezüge in einer Baumstruktur angeordnet.

Die zuerst geöffnete Tabelle repräsentiert immer den Direct View Nr. 1 (kurz: „DV1“). Die daran angehängten Detailtabellen bekommen eine weitere angehängte Ziffern-Stelle im Namen und heißen demzufolge nun DV11, DV12 usw. Hat man an eine Detailtabelle wiederum eine Detailtabelle angehängt, dann wird dies z.B. „DV111“.

Die Ziffernfolge „111“ des DV111 stellt nicht etwa die Zahl „Einhundertundelf“ dar, sondern ist als „Eins-eins-eins “ zu lesen. Bei mehr als 9 angehängten DVs wird dann je Stelle auf Buchstaben, beginnend mit „A“ zurückgegriffen, z.B. ist der als Elfter angehängte Direct View der DV1A. Die Stelle (hier „1“ oder „A“) ist nur ein Name, sie sind völlig gleichberechtigt, es sollten hieraus keinerlei

Vorrangregeln oder dgl. abgeleitet werden. Diese verkürzte Schreibweise ist sinnvoll, da die DV-Bezeichnung als ‚Objekt' im Weiteren sehr häufig benutzt wird.

Der jeweilige übergeordnete DV ist sozusagen Vater bzw. Mutter („Parent") seiner/ihrer Kinder, der nachgeordneten Detail-DVs („Child").

Egal wie und warum und über was für Datensichten man ein SCOPELAND-Applet zusammenbaut: es entsteht immer eine Baumstruktur der Direct Views. Diese Struktur zu verstehen ist wichtig, wir werden ihr noch häufig begegnen. Eine der Zielstellungen von Scopeland ist es ja, dass man sich auf die Inhalte konzentrieren kann und sich soweit möglich überhaupt nicht um technische Interna kümmern muss. Mit all den vielen programminternen Abläufen, wann woher welche Daten gelesen und wie sie zurückgeschrieben werden, welcher Schlüsselwert wann von wo nach wo übernommen werden soll usw. – all dies sollte SCOPELAND soweit möglich von selbst wissen.

Die baumartige Struktur von Datensichten in einem Applet hat vor allem einen technischen Hintergrund: die Refresh – Abfolge.

Einer dieser weitestgehend automatisierten Algorithmen ist die Frage, unter welchen Bedingungen die Anzeige welcher Daten aktualisiert (engl. ‚Refresh') werden muss. Und dies führt uns zu einer Baumstruktur. Wie wollen nun erläutern, warum:

Angenommen, man will den Branchenkatalog aus unserem Beispiel prüfen und ggf. bearbeiten. Dazu lässt man sich in einem Applet als ersten Direct View, also als DV1, die Daten dieses Katalogs anzeigen, vielleicht als Tabelle, oder jeden Datensatz einzeln in einer Bildschirmmaske. Darin kann man nun direkt editieren, das ist erst einmal trivial: Speichert man diese Änderungen in die Datenbank zurück, so müssen anschließend die Daten erneut aus der Datenbank gelesen und angezeigt werden, da die geänderten Daten lt. Sortiervorschrift (z.B. alphabetisch) jetzt vielleicht an anderer Stelle angezeigt werden müssen. Folglich gehört zu jedem Speichern auch immer ein Refresh. Ganz einfach.

Lässt man sich aber zusätzlich auch noch Detaildaten anzeigen (in unserem Beispiel als DV11 die Firmen, die der Branche angehören), so muss natürlich auch dies mit aktualisiert werden, denn die Änderungen könnten sich ja auf die Selektionsbedingungen der Detailtabelle in irgendeiner Art und Weise auswirken. Selbst wenn gar keine Änderungen vorgenommen wurden, allein schon beim Wechsel des Datensatzes in der Branchentabelle (DV1), müssen die Detaildaten neu gelesen werden, da wir ja jetzt in DV11 die Firmen einer anderen Branche, also völlig andere Daten sehen wollen.

Dasselbe gilt, wenn man sich andere Detaildaten anzeigen lässt, z.B. als DV12 die Branchenverbände je Branche.

Und es gilt auch in der nächsttieferen Ebene: Lässt man sich zu den Firmen in DV11 immer noch jeweils die Standorte ihrer Betriebsstätten auflisten, also als DV111, dem Detail vom Detail, dann müssen auch diese immer mit aktualisiert werden.

Es sieht jetzt fast so aus, als sollte man einfach immer alles aktualisieren, aber das wäre auch nicht richtig. Ändert man z.B. den Namen eines Branchenverbands in DV12, so kann das überhaupt keine Auswirkungen auf seinen ‚Vater' (die Branchentabelle in DV1) und auf den gesamten Zweig haben, der unter DV11 steht (Firmen und als DV111 deren Standorte). Folglich ‚vererbt' sich also die Refresh-Folge immer nur nach unten.

Jedes Speichern, jedes ‚Refresh' und jeder Satzwechsel
löst immer auch in allen darunterliegenden DVs ein ‚Refresh' aus.

Wenn man nun auch noch davon ausgeht, dass in den meisten Fällen die „oberen" DVs lediglich der Navigation und Vorauswahl dienen und die eigentlichen, zu bearbeitenden Daten immer an den Enden des Baums liegen, dann wird beim Speichern und Aktualisieren dieser Detaildaten nur ein minimaler Bruchteil der Rechenzeit benötigt, den man bräuchte, um immer alles zu aktualisieren.

Durch diese Regel werden die Zugriffe auf die Datenbank enorm reduziert und damit wird ein sehr deutlich spürbarer Gewinn an Performance erreicht – zumindest soweit die Plattform, für die wir entwickeln, diese Optimierung technisch ermöglicht.

Oberfläche bauen = Einzelteile zusammenkleben + Form Design

Hat man nun einige zusammenhängende Direct Views im Applet und hat zumindest einen davon als Maske vorliegen, so kann man die anderen einfach da draufkleben: egal was – Tabelle, Masken, Graphiken.

Einfach drüber schieben und ankleben!

Auf diese Weise setzt man den Bildschirm (die Maske bzw. Webseite) zusammen, aus zahlreichen Einzelteilen. Es müssen übrigens auch nicht immer all diese Direct Views relational zusammenhängen. Man kann einfach alles überall hineinkleben.

Möchte man noch eine Datensicht ergänzen, die sich nicht direkt als Detaildaten ableiten lässt, so kann man sie auch separat öffnen und dann über frei gestaltete Selektionsbedingungen, manuell gesetzte Bezüge (temporäre Relationen) oder auch ohne jeglichen Datenbezug anhängen.

Wenn man willkürlich Direct Views anhängt und einklebt, dann sollte man dabei aber unbedingt immer die oben beschriebene Refresh-Folge beachten! Ein solcher DV wird immer automatisch mit aktualisiert, wenn der DV aktualisiert wird, wo er angehängt wurde oder wenn in diesem ein Zeilenwechsel erfolgt.

Speicherformat einer Seite – inkl. aller Businesslogik und Oberfläche

Mit „Speichern unter…" speichern Sie das interaktiv aufgebaute Applet zurück in die Metadatenbank.

Im Zuge der Erstellung der Seiten wurden zuvor bereits neue, zusätzliche Metainformationen erzeugt und diese werden nun kompakt zur Seite in der Metadatenbank abgelegt.

Es wird bewusst vermieden, auf der Ebene der Benutzeroberfläche Programmcode vorzusehen, um das bekannte Problem des einfachen MDA-Ansatzes zu vermeiden: dass sich trotz gut strukturierter, teilweise deklarativer Modellierung keine nachträglichen Änderungen auf Modellebene einbringen lassen. Würde man über die wohlstrukturierte SCOPELAND-Logik eine Programmschicht darüber legen, dann wäre diese nach der finalen Ausprogrammierung ohne Verlust nicht mehr neu generierbar.

SCOPELAND vermeidet dieses Problem, indem auch diese logische Schicht ausschließlich aus wohldefinierten und wohlstrukturierten und rein deklarativen Metadaten besteht, welche im Zuge der interaktiven Bildschirmgestaltung entstehen. Diese zusätzlichen Metadaten beschreiben insbesondere

die Abweichungen von den Defaultwerten, Stilen u. dgl., die von dem abweichen, was sich aus den anderen Metainformationen und den eingestellten Stylesheets als Default ableiten lässt. So erhält bspw. ein Feld/eine Spalte auf der logischen Ebene der Benutzeroberfläche nur dann eine Breitenangabe als zusätzliches Metadatum, wenn mit dem Designwerkzeug eine Abweichung von der automatisch ermittelten, vorgeschlagenen Breite eingestellt wurde.

Auch diese Metainformationen werden nicht in Freitextform abgelegt, sondern eindeutig dem betreffenden Objekt zugeordnet. Durch diesen Kunstgriff können zu jedem beliebigen Zeitpunkt nach Belieben Objekte aus der Seite herausgenommen, im Format geändert, neu hinzugefügt werden usw., ohne dass die zusätzlichen Metainformationen dadurch invalid werden können. Auf diese Weise wird die besondere Flexibilität von SCOPELAND auch auf der Ebene der fertig gestalteten Anwendung beibehalten. Dies ist ein Schlüssel dafür, agile Softwareentwicklung konsequent und umfassend zu ermöglichen.

Nun stellt sich aber die Frage, wo und wie die strukturierten Metainformationen der dritten Ebene, diese „temporären Metadaten“, gespeichert werden sollen. Der naheliegende Ansatz, diese direkt in die Metadatenbank zu speichern, als Detail-Informationen zu den normalen, statischen Metadaten, scheidet aus mehreren Gründen aus. Erstens darf die Metadatenbank nicht überfrachtet werden mit halbfertigen Arbeitsständen, privaten Ad-hoc-Reports usw. Zweitens sollen Seiten, insbesondere Reports, einfach von einem Standort einer Applikation zu einem anderen übertragen werden können (Verteilungsproblem, Patches). Drittens wäre dies nicht performanceoptimal für laufzeitkritische Anwendungen bzw. für die Entwickler von generierten Lösungen und viertens ist es gar nicht immer wünschenswert, wenn Änderungen an den Datenstrukturen oder an der Businesslogik ungeprüft auf die fertigen Anwendungen durchschlagen.

Deshalb sieht SCOPELAND hier eine Redundanz vor, indem die fertigen Seiten in kompakter Form den hierfür relevanten Ausschnitt der Gesamt-Metadatenbank enthalten, mit dem, logisch getrennt, die temporären Metadaten verknüpft sind. Im Falle von Änderungen in einer der tieferen Ebenen wird die Seite „aktualisiert“, indem die gesamte untere Schicht ausgetauscht wird, aber manuell ausgelöst vom Entwickler, der sofort die Auswirkungen dieser Änderungen prüfen und ggf. behandeln kann (z.B. durch Layout-Korrekturen nach einer Datentypänderung).

Für diese Mini-Metadatenbank zur Seite wird auch weiterhin der Begriff des SCOPELAND-Applets verwendet. Das SCOPELAND-Applet ist also ein kompaktes, optimiertes Datenpaket, welches wie eine Datei gehandhabt werden kann. In interpretierenden Direct Desk-Anwendungen wird der Dateicharakter des Applets z.B. dazu genutzt, einen Cache-Mechanismus aufzubauen („Applet Cache“), der so eine aufwändige Softwareverteilung für die SCOPELAND-Anwendungen vermeidet. Außerdem sind auch Verteilungen benutzerdefinierter Reports und von Patches von Standort zu Standort möglich.

Intern werden für die Applet-Dateien zwei unterschiedliche Speicherformen vorgesehen. Für interpretierende Anwendungen wird die hochoptimierte, sehr kompakte Objektstruktur 1:1 in der Datei abgebildet, was Anwendungen ermöglicht, die denkbar klein sind und somit in großen Client-Server-Umgebungen unschlagbar schnell sind. Zusätzlich kann das Applet in Form einer wohlstrukturierten XML-Datei abgelegt werden, aus der heraus Codegenerierungen mit eigenen oder Dritt-Werkzeugen möglich sind. Diese letztere Form ist in gewisser Weise vergleichbar mit einem Modellierungsdatenpaket, während die kompakte Ablage der Objekte einem Objektcode entspricht.

Dieses XML-Format wird als XDAML bezeichnet
(XML-based Database Application Modelling Language).

Diese Bezeichnung beschreibt genau, was es ist: eine XDAML-Seite ist tatsächlich eine vollständige und im Wesentlichen deklarative Beschreibung einer Seite einer Datenbankanwendung – und dies nahezu vollständig plattformneutral. Die XDAML-Seite enthält weder eine Festlegung des RDBMS, noch eine Sprachentscheidung (Java oder C# oder C+ oder PHP) und auch keinerlei Einschränkungen hinsichtlich der Zielplattform (Web oder C/S oder Mobile usw.)

Aus diesem XDAML Format heraus lassen sich mit unterschiedlichen Codegeneratoren ausführbare Programme (i.d.R. in Form von klassischem Quelltext) generieren.

Zusätzliche Logik ergänzen

Aufbauend auf der Datenmodellebene und der DV-Logik setzt eine zweite logische Schicht auf: die der zusätzlichen Logik, die sich nicht bereits aus dem Modell ableiten lässt.

Dabei handelt es sich um konkrete Plausibilitäts-, Wertzuweisungs-, Berechnungs- und Sichtbarkeitsregeln, die, wie ja bereits der Name sagt, „Regeln“ sind.

Wir erheben den Anspruch, prozedurale Algorithmen, wenn möglich, gänzlich zu vermeiden und diese, soweit erforderlich, durch rein deklarative Ausdrücke zu ersetzen. Eine mögliche und sinnvolle Form der Notation besteht folglich darin, diese als ‚Business Rules‘ in Form von Wenn-Dann-Regeln zu notieren. Ferner die konkreten Workflows, die einem Statusfeld zuzuordnen sind und ggf. interner sonstiger Prozesse. Wie sich zeigt, lassen sich auch diese relativ einfach formalisieren und zwar nach genau demselben Schema, das wir bereits seit UML 1.0 kennen: als ein Zusammenspiel von Zuständen, Aktionen und Ereignissen.

Regeln sind eigentlich Daten und keine Programme.

Die konkreten Formeln und Ausdrücke, die da abzuarbeiten sind, befinden sich außerhalb des Programms SCOPELAND. Es sind ja nur Daten. Auch wenn aus Optimierungsgründen solche Formeln und Ausdrücke letztlich doch irgendwo in den generierten Programmcode mit eingebaut werden – logisch gesehen sind es dennoch nur Daten, genauso wie die Formeln in den Zellen einer Tabellenkalkulation.

Deshalb kann man die konkrete Programmlogik aus den universalisierten Algorithmen heraushalten; und nur deshalb ist eine Universal Application überhaupt möglich.

SCOPELAND unterscheidet bei aller Anwendungslogik zwischen global geltenden Eigenschaften und solchen, die nur „temporär“, also nur innerhalb einer Seite, gelten.

Solche Regeln, die im engeren Sinne die Eigenschaften der jeweiligen Daten beschreiben, unabhängig von ihrer Verwendung in konkreten Anwenderprogrammen, sollten auch global zu den jeweiligen Tabellen definiert werden. Sie gelten dann immer und überall, ohne dass man sie stets aufs Neue schreiben muss. Sie sind aber – und das ist der Nachteil – etwas weniger flexibel, da nicht auszuschließen ist, dass man diese später irrtümlich so verändern könnte, dass eine fertige Seite plötzlich nicht mehr funktioniert. Globale Eigenschaften müssen deshalb sorgfältig durchdacht sein und sie müssen wirklich von ihrer Natur her globale Gültigkeit haben.

Diesem Risiko kann man ganz einfach aus dem Wege gehen, indem man einfach alles lokal („temporär“) als Eigenschaften zum DV ablegt. So kann man als Entwickler störfrei arbeiten und den Rest der Welt einfach ignorieren. Leider aber erzeugt man so eine unschöne Redundanz und hat natürlich auch wesentlich mehr Arbeit damit.

Logik global oder zur Seite ablegen – das muss man immer sorgfältig abwägen. Meist ist eine gute Mischung aus beidem der beste Weg.

Objekte in SCOPELAND

Da jede Tabelle und jedes Feld/jede Spalte eine interne Nummer hat (z.B. F123 für ein Feld/eine Spalte und T234 für eine Tabelle), kann jedes Objekt innerhalb des Direct View eindeutig adressiert werden. Dies ist erforderlich, um Regeln, Bedingungen und Aktionen hinterlegen zu können, die sich auf konkrete Dateninhalte beziehen.

Und dies gilt nicht nur absolut (z.B. F123 innerhalb der Tabelle T234), sondern, und das ist hier das viel wichtigere, im Kontext aus einem bestimmten Blickwinkel, auch aus der Sicht einer ***anderen*** Tabelle bzw. einem anderen Direct View.

Zunächst ein einfaches Beispiel aus Sicht der eigenen Tabelle: Eine Verweisrelation, ausgehend von einer Haupttabelle T2, zeigt vom Feld F77 aus auf Tabelle T5. Die Relation lässt sich also (innerhalb der Tabelle T2) beschreiben mit dem Pfadausdruck:

F77.T5

Ein Feld F99 in Tabelle T5 ist nun schlicht und einfach „F99“, wenn es allein betrachtet wird. Aus Sicht der Tabelle 2 hat es dann aber die Adresse (den Pfadausdruck):

F77.T5.F99

Und ein Feld in einer dritten Tabelle, einem Verweis der Verweistabelle, heißt dann beispielsweise:

F77.T5.F99.T7.F33

Dabei wird dieser Pfadausdruck weitgehend vor dem Entwickler verborgen. An den meisten Stellen wird er so geführt, dass man immer mit Auswahlfunktionen und Klartextanzeige arbeiten kann. Es ist aber durchaus hilfreich, diese interne Kodierung zu kennen und zu verstehen.

In der normalen Benutzerführung für den Entwickler ist das Ganze dann besser lesbar. Beispielsweise ist der Klartext der Branchenbezeichnung einer Firma, der aber leider nicht in der Firmentabelle selbst, sondern in einem dahinterliegenden Katalog steht, wie folgt beschrieben:

Branchenschlüssel.Brachenkatalog.Branchenbezeichung,

… während ein Feld, das in der Firmentabelle selbst steht, direkt adressiert werden kann, also z.B. als:

Postleitzahl

Diese Pfadausdrücke sind nun Grundlage der gesamten Business-Logik in SCOPELAND. Sie sind notwendig, weil sich in realen Datenbanken niemals die gesamte zusammengehörige Information in einer einzigen Tabelle befindet – sie ist immer verstreut über das ganze Netzwerk von miteinander verbundenen Tabellen.

Hierbei genügt es nun keinesfalls, einfach nur zu sagen, welches Feld man meint und es SCOPELAND zur Laufzeit zu überlassen, den richtigen Relationspfad dahin zu finden. Das liegt daran, dass es häufig mehrere, unterschiedliche Wege von einer Tabelle zu einer anderen gibt.

Beispielsweise ist der Wohnort des Geschäftsführers einer Firma inhaltlich etwas völlig anderes als der Ort, in dem das Unternehmen ansässig ist. Zwischen den Tabellen „Firmen“ und „Städte“ bestehen also offenbar mehrere, unterschiedliche Verknüpfungen und man meint entweder das eine oder das andere.

Deshalb ist es generell unverzichtbar, bei der Adressierung eines Datenobjekts immer anzugeben, auf welchem Wege man dorthin kommt.

In diesem Beispiel in der Firmentabelle ist zwar in jedem Fall das Feld Ortsname in der „Städte“-Tabelle gemeint, aber entweder in dem Datensatz, auf den das Feld „Stadtnummer“ zeigt:

Stadtnummer.Städte.Ortsname

oder es handelt sich um einen Verweis auf eine Personentabelle, die dann wiederum ein „Wohnort“ -Feld enthält, welches dann auf die Städtetabelle zeigt.

Geschäftsführer.Personen.Wohnortnummer.Städte.Ortsname

Pfadausdrücke in Direct Views und Applets

Entsprechend dem oben Erläuterten kann man sich also, ausgehend von einer beliebigen Haupttabelle, in viele Richtungen entlang der Relationen zu anderen Tabellen bewegen und dabei verzweigt es sich immer weiter. Aus Sicht dieser Haupttabelle handelt es sich also um eine Baumstruktur!

Nach dem oben dargestellten Prinzip kann folglich jede Tabelle, die in irgendeiner beliebigen Art und Weise, direkt oder indirekt und egal über wie viele Zwischenschritte relational mit der gerade aktuellen Tabelle verknüpft ist, eindeutig adressiert werden. Beispiel:

(Akt. Tabelle) *→ Verweistabelle* *→ Verweistabelle*
→ Verweistabelle
→ Verweistabelle *→ Verweistabelle*
→ Verweistabelle
usw.

In der oben erläuterten Syntax:

T1 *→ F1.T2* *→ F1.T2.F5.T8*
→ F1.T2.F6.T9
→ F3.T4 *→ F3.T4.F7.T8*
→ F3.T4.F8.T9
usw.

Die einzelnen Datenobjekte (Felder) erreicht man dementsprechend so:

T1 → *F1*
→ *F1.T2.F5*
→ *F1.T2.F6*
→ *F2*
→ *F3*
→ *F3.T4.F7*
→ *F3.T4.F8*
→ *F4*
→ *F5*

usw.

Die Struktur einer Seite (eines SCOPELAND-Applets) ist nun ebenfalls hierarchisch aufgebaut:

DV1 ← *DV11* ← *DV111*
← *DV112*
← *DV12* ← *DV121*
← *DV122*

usw.

Ein bestimmtes Datenfeld einer jeden, auch einer eingezogenen Tabelle, hat folglich eine eindeutige Adresse innerhalb des Applets: *Direct View + Verweiskette(optional) + Feldobjekt*. Beispiele für den korrekten Objektbezug, aus Sicht der Seite:

Beispiel 1: Direct View = DV1 (das ist die Sicht auf die zuerst geöffnete Tabelle)
Darin das Feld = F66

→ *Die Adresse (der Pfadausdruck) des Feldes ist: DV1.F66*

Beispiel 2: Direct View = DV11 (die Datensicht der ersten angehängten Detailtabelle)
Verweiskette = F77.T23 (Feld Nr. 77 verweist mit einer Relation auf Tabelle 23)
Darin das Feld = F99

→ *Der Name (der Pfadausdruck) des Feldobjekts ist: DV11.F77.T23.F99*

Anmerkung: Innerhalb des DVs können die Feldobjekte auch ohne Vorsatz des DVs adressiert werden:

→ *Innerhalb des DV-Kontextes genügt als Feldobjekt: F77.T23.F99*

Mittels dieser eindeutigen Adressierung kann man nun, ähnlich wie bei einer Tabellenkalkulation, jeden beliebigen Dateninhalt mit jedem beliebigen anderen verrechnen.

Auch hierbei wird dieser Pfadausdruck weitgehend vor dem Entwickler verborgen. An den meisten Stellen wird er so geführt, dass man immer mit Auswahlfunktionen und Klartextanzeige arbeiten kann. Es ist aber durchaus hilfreich, diese interne Kodierung zu kennen und zu verstehen.

Beispiele, Anzeige:

DV1.Stadtnummer.Städte.Ortsname

DV12.Geschäftsführer.Personen.Wohnortnummer.Städte.Ortsname

An den meisten Stellen kann man bei SCOPELAND zwischen den beiden Darstellungen (Klartextanzeige oder Nummernanzeige) nach Belieben hin- und herschalten. Die Klartextanzeige ist natürlich immer besser lesbar, während die Nummerndarstellung kürzer und prägnanter ist. Manch ein Entwickler kennt nach einer Weile etliche der Nummern sowieso auswendig und kann fließend damit umgehen.

Regeln zu Objekten

Regeln werden in SCOPELAND deklarativ hinterlegt, zumindest so deklarativ wie es jeweils sinnvoll ist. Der Vorteil deklarativer Notation ist offensichtlich.

So genügt es z.B. zu schreiben, dass ein Feld größer sein muss als ein anderes (z.B. zum Feld1 die Regel: „> Feld2“), ggf. unter einer ebenso einfach notierten Bedingung (z.B. wenn „Feld3 == 7“) oder dass ein Feld gleich dem Ergebnis eines formelähnlichen Ausdrucks sein muss (z.B. „F4 *2 + 1“).

Der Entwickler muss lediglich diesen Ausdruck notieren, das ist alles. Wann diese Regel auszuführen ist, was in welcher Reihenfolge genau zu tun ist und was im Fehlerfall zu passieren hat, all dies kann dieselbe Maschine, die bereits aus dem Datenmodell heraus die Rohanwendung errechnet hat, ebenso vollautomatisch in dieselbe einfügen.

Dieser Ansatz hat eine gewisse Ähnlichkeit zu dem, wie in modernen Spreadsheet-Programmen z.B. Microsoft-Excel Formeln notiert werden. Auch da trägt man zu einem Feld, dessen Inhalt berechnet werden soll, einfach eine Formel ein und muss sich nicht weiter darum kümmern, wann und in welcher Reihenfolge solche Berechnungen zu erfolgen haben. Man beschränkt sich auf deklarative Ausdrücke; die zugehörigen Algorithmen stellt das Spreadsheet-Programm selbsttätig bereit.

Zweckmäßigerweise betrachten wir all diese zusätzlichen Regeln als ‚Properties‘ (Eigenschaften) der jeweiligen Datenobjekte, auf die sie sich beziehen. Jedes Datenfeld, jede Datenbanktabelle und jedes sonstige wie auch immer geartete Objekt kann durch Zuweisung von Regeln unterschiedlicher Typen in seinem Verhalten immer weiter verfeinert werden.

Anders als bei einem Spreadsheet genügt hier aber oftmals nicht, eine einzige Formel zu hinterlegen. Zumindest ist zwischen „harten“ und „weichen“ Wertzuweisungs- bzw. Prüfregeln zu unterscheiden. Und wenn man ohnehin zwischen mehreren Regeltypen unterscheiden muss, dann kann man auch gleich soweit gehen, eine beliebige Anzahl von Regeln, auch gleichen Typs, zuzulassen, die sich ggf. durch konkurrierende Bedingungen gegenseitig ausschließen oder auch gleichzeitig gültig sind.

Logisch gesehen erreicht man damit, dass ein jedes Objekt sich quasi selbst kennt und selbst weiß, wie es sich darstellen und wie es sich verhalten soll. Wird ein Objekt z.B. in einem Programmteil verwendet, dann bringt es seine Eigenschaften selbsttätig mit und fügt sie (gewissermaßen selbsttätig) in das entsprechende Rahmenprogramm ein. Je mehr Regeln, Workflows und Prozesse man hinzufügt, umso

feiner und präziser prägt sich das Programm aus. Und all dies komplett eigenständig, ohne dass der Anwendungsentwickler auch nur ein einziges Mal einen Algorithmus hätte schreiben müssen.

Vordefinierte Zustände und Aktionen

Wie oben bereits angedeutet, kann man in SCOPELAND vordefinierte Zustande hinterlegen und auch diese wahlweise global zur Tabelle oder auch lokal zum DV innerhalb einer Seite.

Zustände sind ein Grundelement der UML-Logik gemäß dem sog. UML-„Aktivitätsdiagramm“ und damit eine Grundlage für fast alle heute üblichen Prozess- und Workflow-Modellierungsverfahren. Zustände und deren Unterzustände sind die Basis aller Prozessabläufe und innerhalb dieser Zustände sind bestimmte, ebenso vordefinierte Aktionen möglich – wiederum global oder temporär.

Aktionen können so definiert sein, dass „ihr“ Zustand quasi nur eine Ausführungsbedingung für die Aktion ist. Aktionen können aber auch Zustandsübergänge darstellen, die Daten von einem Ausgangs- in einen Zielzustand überführen. Um letzteres technisch zu ermöglichen, werden Zustände so definiert, dass sie nicht nur eine Bedingung enthalten (wie zum Beispiel die, ob ein Feld gleich einem Wert ist), sondern zusätzlich auch noch eine kurze Aktion, die den Zustand herstellt, z.B. ein Statusfeld auf ebendiesen Wert setzt.

Aktionen enthalten neben den Zustandsübergängen („Transitions“) oftmals bestimmte auszuführende Arbeitsschritte, also Abläufe, die streng genommen gar nicht so deklarativ sind, wie die SCOPELAND-Idee dies erwarten ließe. Sie werden aber ebenso in einfachen formelähnlichen Ausdrücken („Expressions“) notiert, die bestimmte Standardfunktionen aufrufen (z.B. um Werte zu setzen). Aktions-Expressions sollen ebenso kurz und knackig sein wie die in Regeln. Also quasi-deklarativ.

UML-Prozesslogik

Zur Modellierung von Prozessen und Abläufen aller Art, sowie zur übergreifenden Definition von Ausführungsbedingungen folgt SCOPELAND den in UML1 definierten und in UML2 bestätigten Prinzipien. Im Kern geht es dabei immer um Zustände, Aktionen/Zustandsübergänge und Auslöser. Dies deckt sich im Kern mit der Beschreibungslogik fast aller modernen Modellierungssprachen.

Allerdings verfolgen wir hier eine etwas andere Absicht: in SCOPELAND geht es nicht darum, die Eigenschaften der zu entwickelnden Anwendung im Voraus vollständig zu beschreiben, sondern die entstehende Software um das Feature einer Prozess- oder Workflow-Abarbeitung zu erweitern.

Wer einmal versucht hat, eine komplette Anwendung vollständig in UML zu beschreiben, wird wissen, dass dies ein völlig unrealistisches Vorhaben ist. Es macht keinen Sinn, Details wie z.B. die Prüfung einer Plausibilität als ein Set von Zuständen (plausibel, unplausibel, gerade in Bearbeitung usw.), Aktionen (Erfassen, Ändern, Feldverlassen, Bestätigen u. dgl.) und Keyboard-Ereignissen zu beschreiben, wo doch eine simple Formel allein schon denselben Inhalt deklariert. Es wäre auch völlig falsch verstanden, wenn man UML als eine Meta-Programmiersprache benutzen würde.

UML ist keine Sprache zur Beschreibung eines Programmverhaltens,
sondern eine Entwurfssprache.

Insoweit erhebt sich natürlich die Frage, ob UML überhaupt dazu geeignet ist, ausführbare Programmlogik zu beschreiben. Das ist sie, allerdings sollte man sich hierbei auf diejenigen Abläufe

und Prozesse beschränken, die tatsächlich statusgetrieben sind und nicht etwa versuchen, triviale Details mit UML abzubilden. Letzte lassen sich weitaus besser und kompakter mit deklarativen Business Rules und anderen Eigenschaften von Daten abbilden.

Welchen Umfang hat dieser verbleibende Anteil echter Prozesslogik in typischen Datenbankanwendungen tatsächlich? Meist ist es so, dass nur einige wenige der vielen Dialoge und ihrer Daten tatsächlich prozessgetrieben sind. Auf jeden Fall handelt es sich dabei um einen Bruchteil dessen, was man mit Business Rules beschreiben kann.

Workflow- oder Prozesslogik ist das nicht immer benötigte Sahnehäubchen auf der ansonsten eher elementaren Anwendungslogik.

Dementsprechend ist auch die Entwicklungsumgebung von SCOPELAND so aufgebaut, dass Zustände, Aktionen und Auslöser die Benutzerführung für den Entwickler nicht dominieren, sondern latent im Hintergrund verfügbar bleiben. Im realen Entwicklungsprozess dominieren aus gutem Grund eher die direkt, temporär notierten Bedingungen und die ebenso direkt notierten Ausdrücke für das, was z.B. beim Betätigen einer Schaltfläche passieren soll. Hat man aber systematisch definierte Zustände und Aktionen vorliegen, so werden diese dem Entwickler immer und überall auch mit zur Auswahl angeboten.

Da UML-Prozesselemente besser systematisiert und graphisch dargestellt werden können, obliegt es der Verantwortung des Projektleiters, zu entscheiden, in welchem Umfang er seine Entwickler dazu anhält, zunächst eine UML-konforme Modellierung vorzunehmen. Es geht immer um die Frage, inwieweit es wirklich zielführend ist, alle denkbaren definierten Zustände und Aktionen den verfügbaren Ereignissen zuzuordnen und inwieweit man den Entwicklern die Freiheit belässt, auf eine systematische Modellierung zu verzichten und die Funktionalität direkt den jeweiligen Elementen zuzuordnen.

Die Entscheidung hierfür ist natürlich abhängig von der Komplexität und Kritikalität der jeweiligen Programmmodule.

SCOPELAND unterstützt in diesem Zusammenhang alle wichtigen UML-Elemente:

- Zustände nebst Unterzuständen in beliebig tiefer Schachtelung
- Aktionen mit Zustandsabhängigkeit und Nebenbedingungen
- Transitions (Zustandsübergänge), in Form von Aktionen mit Zielzustandsangabe
- Ereignisse/Auslöser einschließlich der automatischen Auslöser DO und ENTRY
- Splitting und Synchronisation
- Verzweigung und Zusammenführung

Dadurch sind auch komplexe Parallelprozesse nebst ihrer Beziehungen untereinander abbildbar. In realen Projekten werden jedoch meist nur eingleisige Prozessabläufe verwendet, so dass man die Elemente Splitting und Synchronisation beim Erlernen von SCOPELAND zunächst eher vernachlässigen kann.

Verzweigungen und Zusammenführungen hingegen sind viel einfacher zu verstehen und selbst diese werden nur selten benötigt. Sie dienen der Modellierung von Fallunterscheidungen im Prozessablauf – aber es stellt sich die Frage: gehören diese überhaupt ins Modell? Ist es nicht eher so, dass die Aktionen selbst schon die jeweils nächsten Zustände herstellen und damit die Fallunterscheidungen in

sich tragen. Verzichtet man auch auf diese Elemente, dann verbleiben neben den Zuständen und Aktionen lediglich noch die Auslöser und die setzt man ohnehin intuitiv, indem man z.B. eine Aktion auf einen Button legt. Automatische Auslöser wie „ENTRY“ (immer sofort, wenn sich der Ausgangszustand einstellt) und „DO“ (immer, solange der Zustand anhält), werden je nach Ausprägung der Programme durch einfache Checkboxen aktiviert – auch sie müssen nicht unbedingt im Vorfeld modelliert werden. Sie stellen gleichfalls meist keine Entwurfsentscheidung dar, sondern eher eine Feinausprägung der Applikation in der Endphase der Entwicklung.

> *Allerdings sollte der Entwickler dabei berücksichtigen, dass die Möglichkeiten eines Systems, festzustellen, ob ein Zustand eingetreten ist, natürlich endlich sind. Um unakzeptable Performanceeinbußen zu vermeiden, muss sich SCOPELAND darauf beschränken, dass diese automatischen Auslöser nur dann wirken, wenn das Eintreten des Zustands der SCOPELAND-Applikationslogik bekannt ist, sprich, wenn er durch SCOPELAND in Form einer Transition direkt herbeigeführt wurde. Da davon i.d.R. auch ausgegangen werden kann, ist dies aber keine sehr große Einschränkung.*

Damit vereinfacht sich die Abbildung von Prozess- und Workflowmodellen erheblich. Alles basiert ausschließlich auf Aktionen und Zuständen, sowohl logisch als auch in der technischen Umsetzung. Selbst die vier Logik-Operatoren im Prozessfluss (Splitting und Synchronisation sowie Verzweigung und Zusammenführung) werden technisch als Pseudozustände umgesetzt, die mit DO-Events belegt sind und auf diese Weise ein automatisches Weitertreiben des Prozesses sicherstellen.

Diese UML-basierten Modelle werden in SCOPELAND streng den Datenobjekten bzw. Datensichten zugeordnet. Völlig ungebundene Prozesse sind nicht zulässig. Besteht ein Programmmodul (z.B. ein Formular) aus mehreren Datensichten, dann können die einzelnen Modelle einander beeinflussen, ebenso wie auch ein Datenobjekt mehrere parallele Modelle gleichzeitig haben kann.

Und was sind „Aktivitäten“? Aktionen oder Zustände?

Lt. „UML-Aktivitätendiagramm“ und auch vieler anderer Prozess- und Workflow-Modellierungssprachen sind „Zustände“ dasselbe wie „Aktivitäten“ – aber streng von „Aktionen“ zu unterscheiden. Dies erklärt sich relativ einfach, indem man sich unter „Aktivitäten“ im einfachsten Fall User-Aktivitäten wie z.B. „Datenerfassung“ vorstellt. Das ist tatsächlich ein Zustand im Sinne von „Datenerfassung läuft gerade“.

Auch wenn es auf den ersten Blick irritieren sollte, es ist aber trotzdem so:
„Aktivitäten“ sind keine Aktionen, sondern Zustände.

Es gibt aber auch automatische Aktivitäten, die dennoch ein Zustand sind, z.B. „Datenimport“ im Sinne von „Daten werden gerade importiert“ mit dem Unterzustand „Import ist jetzt fertig“. Eine Aktion hingegen ist die technische Umsetzung des Zustandswechsels, hier z.B. das simple Setzen eines Flags, das aussagt, dass Daten vorhanden sind, was dann also technisch mit dem Übergang zum nächsten (User-)Zustand „Importierte Daten prüfen“ identisch ist.

Der Zustandswechsel, die sog. „Transition“ könnte auch aus quasi gar nichts bestehen. Falls wir nämlich den Zustand, dass Daten vorhanden sind, nicht an einem Flag, sondern daran erkennen, dass die Daten physisch tatsächlich vorliegen, löst sich der Zustandswechsel quasi von selbst aus und er macht auch gar nichts. Hieran sieht man, dass die UML-Sicht auf die Abläufe und das Programm gänzlich verschiedene Dinge sind: das Modell würde assoziieren, man bräuchte hier ein Watchdog-Programm, um bei Eintreten des Unterzustands „Import ist jetzt fertig“ vollautomatisch ein Programm

zu starten, welches einen neuen Systemzustand herbeiführt. Das stimmt so nicht: es ist programmtechnisch schlicht und einfach gar nichts zu tun.

Ein theoretisches, konzeptionelles UML-Modell beschreibt die jeweiligen Arbeitsfortschritte, also auch die Tätigkeiten des Benutzers. Es kann gar nicht funktional identisch sein zu Programmabläufen. Nur ein Teil des Modells hat eine Entsprechung im Programmcode bzw. in unseren Regeln. Hingegen ist das meiste Programmtechnische überhaupt nicht Gegenstand des Modells. Das UML-Modell hat also mehr Entwurfs- und Dokumentationscharakter und ist entgegen weit verbreiteter Vorstellungen keine gut geeignete Quelle für eine Generierung fertiger Programme. Es ist eine Illusion, dass zwischen einem UML-Modell und der daraus erstellten Programmlogik eine 1:1-Entsprechung bestehen könnte, da nicht primär die Programmabläufe, sondern in erster Linie die Benutzeraktivitäten modelliert werden.

Ein UML-Modell dient meist nur der graphischen Veranschaulichung der Bearbeitungsabläufe und taugt nur bedingt zu einer Codegenerierung. Hinter einem einfachen Pfeil kann sich ein komplexes Programm verbergen und andersrum können selbst extrem komplexe Transition-Konstrukte technisch mit einem einzigen simplen Kommando geprüft bzw. ausgeführt werden.

Wie aus Aktionen und Zuständen ein Workflow wird

Typische Verwaltungslösungen benötigen zwingend Workflow-Funktionalität. Damit ist nicht primär das Orchestrieren übergeordneter Abläufe auf sehr hoher Ebene gemeint und es ist auch nicht die aktenbezogene „Workflow“-Funktionalität von Dokumentenmanagementsystemen gemeint. Hier geht es um die anwendungsinternen Abläufe zur inhaltlichen Vorgangsbearbeitung auf Sachdatenebene. Diese Art von Workflow ist in Business-Applikationen allgegenwärtig und wird von Programmierern meist intuitiv mit programmierten Abläufen abgebildet.

Aus Performancegründen sowie wegen der manchmal unvermeidlichen Parallelverarbeitung (mehrere Stati pro Vorgang) wird hierfür in SCOPELAND-Anwendungen i.d.R. keine separate Workflow-Engine eingesetzt. Ein Status ist ein ganz normales Datenfeld in der jeweiligen Haupt-Datenbanktabelle, in dem die Vorgänge oder Sachverhalte gespeichert sind; und die Workflow-Abläufe sind ebenso selbstverständlich ein integrierter Bestandteil der ohnehin auf diese Daten wirkenden Anwendungslogik.

Der SCOPELAND-Anspruch, komplette Applikationen rein deklarativ zu modellieren, erzwingt folglich auch hierfür den Einsatz einer strukturierten Ablauflogik. Die weitgehende Ähnlichkeit zu den in UML modellierten Prozessen legt es nahe, hierfür dieselben Beschreibungselemente und Mechanismen zu verwenden.

Allerdings gibt es einige gravierende Abweichungen. Prozesse im UML-Sinn wirken generell nur temporär während der Laufzeit eines Programms oder einer Kaskade von Programmen. Der Fall, dass ein Vorgang ruht, während das Programm abgeschaltet ist, ist in UML schlichtweg nicht vorgesehen. Da die einzige sinnvolle, von den Programmen unabhängige Speichermöglichkeit für Zwischenzustände die Datenbank ist, ist es naheliegend, den jeweiligen Zustand in derselben als Datenbankfeld zu speichern. Zwar können u.U. einige denkbare Zustände auch mit freien Definitionen aus dem Zustand der Daten ermittelt werden; im Normalfall aber ist der Zustand eine konkrete Einzelinformation, die genau dort ihren optimalen Speicherplatz findet – in genau einem einzigen Datenfeld.

Daraus wiederum folgt, dass beim Workflow, anders als bei temporären Prozessen, ein Zustand/Status nicht nur zu ermitteln ist, sondern es wird auch eine Funktionalität benötigt, denselben explizit zu „setzen“: sprich, das Statusfeld mit einem definierten Wert zu beschreiben (und zwar in derselben Datenbanktransaktion wie die inhaltlichen Veränderungen an den Daten).

Das ist der Unterschied zwischen freien UML-Modellen und strukturiertem Workflow:

Ein Prozess-Zustand kennt nur eine frei definierte Prüfbedingung, ein Workflow-Status hingegen einen (meinst numerischen) Wert, der von der „Engine“ in einer Datenbank geprüft und von derselben Engine auch in dasselbe Feld hineingeschrieben wird.

Eine Aktion im UML-Prozessmodell ist technisch und inhaltlich jedoch dasselbe wie eine Workflow-Aktion und auch der Umgang mit Auslösern ist (zumindest bei manueller Auslösung) derselbe.

Was liegt also näher, als beide Prinzipien nahtlos miteinander zu verschmelzen. Fügt man zu der Definition eines Zustands (über seine Prüfbedingung) noch eine optionale Aktion hinzu, mit der man diesen Zustand explizit erzeugen kann, dann kann man mit dem UML-Verfahren ebenso auch einen jeden Workflow abarbeiten.

Hierzu ist in SCOPELAND vorgesehen, dass man einen jeden Zustand optional als Workflow-Zustand kennzeichnen kann. In diesem Fall wird er eindeutig einem Workflow-Modell (sprich: einem Statusfeld) zugeordnet und die Prüfbedingung und die Setzaktion werden per Default mit dem Beschreiben dieses Statusfelds mit der automatisch generierten Statusnummer belegt.

Somit wird der ganze Mechanismus je nach Bedarf wahlweise als Prozessmodell oder als Workflow verwendet.

Abgrenzung: Was man so alles unter „Workflow“ verstehen kann…

Der Begriff „Workflow“ oder auch „elektronische Vorgangsbearbeitung“ basiert zwar auf einem gemeinsamen Grundgedanken, es werden aber im Alltag recht unterschiedliche Dinge darunter verstanden.

Dieses Grundprinzip besteht darin, wiederholbare Bearbeitungsabläufe für bestimmte Vorgänge, z.B. zur Bearbeitung von Anträgen, auf einer hohen Abstraktionsebene zu beschreiben und die einzelnen Bearbeitungsschritte, von Arbeitsplatz zu Arbeitsplatz, kontrolliert ablaufen zu lassen.

Im Mittelpunkt steht dabei der Begriff des „Status“. Ein (Bearbeitungs-) Status ist eine frei definierte Bezeichnung und es gibt ein Regelwerk, das aussagt, was in welchem Status mit einem Vorgang getan werden darf bzw. muss.

Vielfach wird die Workflow -Idee untrennbar mit Dokumentenmanagement in Verbindung gebracht. Dies liegt sicherlich daran, dass das Workflow-Konzept sehr gut die standardisierte, behördenrechtliche Arbeitsweise in der öffentlichen Verwaltung abbildet. Und diese Arbeitsweise ist streng aktenbezogen. Ein Vorgang ist immer zugleich eine Akte, bestehend aus einer wohlgeordneten Menge von Dokumenten. Und diese Akte und ihre Dokumente werden in DMS-Systemen in elektronischer Form gespeichert und bearbeitet.

Deshalb ist es naheliegend, Workflow als ein Prinzip zu verstehen, den einen, ohnehin in der Verwaltung üblichen Status, für eine ganze elektronische Akte elektronisch zu verwalten, Weiterleitungen, Vertretungsregelungen, Wiedervorlagen usw. damit zu steuern und je nach Status den Zugriff auf die einzelnen elektronischen Dokumente lesend bzw. schreibend freizugeben. Genau dafür hat sich eine unüberschaubar große Menge an „Workflow“-Produkten und um Workflow-Funktionalitäten erweiterten Dokumentenmanagementsystemen etabliert.

Der Rationalisierungseffekt ergibt sich hier insbesondere aus der Papiervermeidung durch die elektronische Aktenführung und das Workflow-System steuert den Weg dieser elektronischen Akten von einem virtuellen Schreibtisch zum nächsten, genau entsprechend den ohnehin gültigen Verwaltungsvorschriften. Soweit, so gut.

Mit Datenbank-gestützten Anwendungslösungen, die die Bearbeitung der Vorgänge inhaltlich unterstützen, hat das eigentlich nicht viel zu tun. Aber die Grenzen sind fließend. Wenn es über simple elektronische Formulare und standardisierte Adressdaten wesentlich hinausgeht, wenn die erfassten Daten zentral und strukturiert gespeichert und teilweise automatisch weiterverarbeitet werden sollen, dann wäre es beim DMS-Ansatz nötig, ganz viel Anwendungslogik in das Workflow-System integrieren zu müssen. Und dies ist, wie jegliche Integration unterschiedlicher Produkte, immer problembehaftet. Und von der naiven Vorstellung, man könnte gleich gänzlich die gesamte Anwendungslogik aus dem Programm herausziehen und mit aberwitzig komplexen Modellierungssprachen beschreiben, möchten wir uns sowieso abgrenzen.

Eine gute technische Lösung sollte möglichst immer „aus einem Guss“ sein und nicht ein wechselseitiges Aufrufen unterschiedlicher Systeme.

Deshalb gibt es heute eine typische Art von Datenbankanwendungen, die gern als „Vorgangsbearbeitungslösungen“ oder auch als „Workflow-Anwendungen“ bezeichnet werden. Das sind im Grunde klassische kunden- oder aufgabenspezifische Anwendungslösungen, datenbankbasiert und konventionell programmiert und sie unterstützen mehr oder weniger feststehende Bearbeitungsabläufe an den Sachdaten und nicht die der Dokumente. Mit den o.g. Workflow-Produkten hat das aber fast nichts zu tun.

In Grenzfällen werden manchmal auch zwei konkurrierende und gleichermaßen unbefriedigende Ansätze für ein Projekt diskutiert: a) man nehme ein Workflow-Produkt, in das die eigentlichen Inhalte individuell irgendwie hineinprogrammiert werden oder b) man programmiere eine echte Datenbanklösung, die die Inhalte zwar sehr gut abdeckt, aber die Vorgangsbearbeitung nicht strukturiert abbilden kann. Meist aber ist zwischen beidem klar zu unterscheiden. Es gibt auf der einen Seite die „Vorgangsbearbeitung“ im Sinne einer elektronischen Akte und andererseits programmierte „Vorgangsbearbeitungslösungen“, d.h. inhaltliche Anwendungslösungen.

SCOPELAND dient nun dazu, solche Vorgangsbearbeitungslösungen programmierfrei zu konfigurieren und es bedient sich dabei des Prinzips des Workflows. Mit DMS oder Workflow-Produkten will es aber nicht konkurrieren. Es geht um einen ganz anderen Zweck.

Warum der Workflow immer an einer Tabelle hängt

Ein Vorgang hat einen Status und bestimmte Aktionen setzen einen Vorgang von einem Status auf den anderen. Ferner sind innerhalb eines Status bestimmte Aktionen möglich, die den Status nicht verändern.

Dieses Wechselspiel von Status und Aktion lässt sich sehr gut verstehen, graphisch veranschaulichen und bequem einrichten und pflegen. Das ist der Grund dafür, dieses Prinzip einzusetzen. Inhaltlich lässt sich all dies zwar alles auch mit elementareren Mitteln (Bedingungen und Wertzuweisungen) realisieren, aber das abstrakte Modell einer Vorgangsbearbeitung, die einen Vorgang von Status zu Status schiebt, gibt der ganzen Anwendungslogik eine Überschaubarkeit, die anders kaum zu erreichen wäre.

Anders als bei einer Aktenverwaltung ist der Begriff des Status hier verallgemeinert. Jedes Ding, jedes Objekt der realen Welt, kann einen „Status“ haben und bekommt damit einen Vorgangscharakter. Und ein jeder Vorgang, egal ob es sich um einen Antrag, eine Adresse, einen Messwert oder was auch

immer handelt, ist bei SCOPELAND wie jedes andere Objekt natürlich immer ein Datensatz in einer Tabelle.

Eine Tabelle wird dadurch zu einer Vorgangstabelle, dass sie ein Statusfeld enthält, eine Spalte, in der der Status eines jeden Vorgangs gespeichert wird. Für Statusfelder gelten bestimmte Konventionen, ansonsten sind es aber ganz normale Datenfelder wie alle anderen auch.

Sobald man ein Statusfeld hat, hat man auch einen Workflow und ein Modell, das man dann entsprechend ausgestalten kann. Viele Tabellen können, völlig unabhängig voneinander, jede für sich ein Workflow-Modell haben – oder auch eine Tabelle mehrere Modelle.

Das Workflow-Prinzip kann man prinzipiell für jede Art von definierten Abläufen einzelner Bearbeitungsschritte verwenden. Sinnvoll ist es, sobald andernfalls die Übersichtlichkeit über die verschiedenen Zustände und Aktionen zu dem Datenobjekt verloren ginge.

Technisch gesehen ist ein Status fast dasselbe wie ein sonstiger Zustand, eine ganz normale vordefinierte Bedingung, nur ergänzt um eine Status-Setz-Expression.

Der Grund für das Prinzip „Statusfeld“ anstelle einer separaten Workflow Engine ist schlicht und einfach die unübertroffene Verarbeitungsgeschwindigkeit.

Wenn der Status technisch ein Datenfeld ist, wie jedes andere auch, dann kann er zur gleichen Zeit und in gleicher Weise selektiert, ausgewertet und beschrieben werden wie die Sachdaten in der Tabelle. Indem diese intern gleichbehandelt werden, werden zusätzliche Datenbankzugriffe, ja sogar zusätzliche Joins vermieden, und so eine bestmögliche Performance und Transaktionssicherheit auch bei sehr großen Datenmengen erreicht.

Es hat aber noch weitere Vorteile:

Das Statusfeld kann wie jedes andere Feld in jeder Art und Weise in der Anwendung selbst verwendet werden, beispielsweise zur Statusanzeige innerhalb einer Anwendungsmaske, für die Definition, Anzeige und Auswertung von Statusgruppen oder gar für automatisierte Massendatenoperationen, in denen sachdatenabhängig ein Status zu setzen ist (z.B. eine nichtbezahlte Rechnungen bei Fristablauf auf eine höhere Mahnstufe).

Solche Massendatenoperationen können hier mit einem einzigen Update-Statement, selbst bei etlichen Millionen von Datensätzen in Sekundenbruchteilen durchgeführt werden. Bei manch einem konventionell strukturierten Workflow-System wäre hierfür wohl ein Batchprozess vom etlichen Stunden erforderlich.

Wenn noch Logik fehlt: Lücken schließen mit Makros und Scripts

Eventuelle funktionale Lücken bzw. ungewöhnlich komplexe Algorithmen werden in Makros geschlossen bzw. realisiert. Unter Makros verstehen wir hier alles, was Programmiersprachen ähnlich ist, egal ob manuell geschrieben oder automatisch generiert, egal ob tatsächlich ein Programm oder eine besondere Datenbankoperation.

Makros können wie Applets aufgerufen werden, sie können aber auch ein Anhängsel derselben sein. In solchen sog. Applet-Scripts lassen sich nun bequem Variablen, Funktionen u.v.a. definieren, die damit sofort und unmittelbar die Funktionalität des Applets erweitern.

Sobald man Makros intensiver benutzt, ist Scopeland natürlich nicht mehr so ganz programmierfrei. Aber das ist keine allzu große Einschränkung. Erstens sind sie tatsächlich nur unter besonderen Umständen erforderlich und zweitens reduziert sich hier das zu Programmierende auf einen minimalen Bruchteil dessen, was eine komplette handgeschriebene Anwendung enthielte.

Oftmals hängt es auch von Vorlieben ab, ob auch ungewöhnliche und schwierigere Funktionalitäten durch besonders clever gebaute Applets erreicht werden, oder ob man es bevorzugt, bei jeder Gelegenheit schnell mal ein Makro zu schreiben. Die meisten dieser Sonderanforderungen lassen sich wahlweise so oder so umsetzen.

Ferner können natürlich auch diverse Integrationen mit anderer Software, Spezialschnittstellen, Datenaufbereitungen u.v.a. zur Anwendung gehören. Ihre Anwendung kann auch ein eigenes Online-Hilfesystem und andere Dokumentation erfordern. Auf all dies gehen wir hier nicht ein. Einzelheiten dazu entnehmen Sie bitte der Online-Dokumentation zu diesen Themen.

SCOPELAND für Fortgeschrittene

Die „Rahmenanwendung“

Von heutigen Webanwendungen wird erwartet, dass sie in quasi jedem beliebigen Design aufwarten können, was bedeutet, dass es kein einheitliches, von der Basissoftware bereitgestelltes Menü geben kann. Um dieser Besonderheit Rechnung zu tragen, galt es, einen Lösungsweg zu finden, der auf der einen Seite in der Rahmenbenutzerführung kundenspezifische Ausprägungen aller Art ermöglicht, aber der es auf der anderen Seite dennoch möglich macht, mit geringstmöglichem Aufwand solch ein System bereitzustellen. Also Fertigsoftware, aber mit völlig freier, individueller Ausgestaltung? Wie soll das gehen?

Wir haben uns dieser Herausforderung gestellt und eine ausgefeilte Muster-Rahmenanwendung entwickelt, die nach Belieben in die jeweiligen Anwendungen hineinkopiert werden kann, wahlweise „vom Original“ oder von einer anderen Anwendung, in der sie bereits Ausprägungen erfahren hat, die den geplanten eigenen ähnlich sind. So vererben sich die spezifischen Ausgestaltungen der Rahmenanwendung in unterschiedlichen Strängen für unterschiedliche Aufgabenstellungen, bei einem relativ konstanten funktionalen Kern, der sich gut bewährt hat und über längere Zeit hinweg erstaunlich stabil geblieben ist.

Die Rahmenanwendung ist ebenfalls mit SCOPELAND entwickelt worden, hat aber zugegebenermaßen eine etwas höhere Komplexität als typische ‚eigentliche‘ Anwendungsinhalte. Durch Nachnutzung und Anpassung kommt man deshalb so sehr viel schneller zu einem funktionsfähigen Gesamtsystem, als wenn jeder Entwickler stets von Null anfangen würde. Heute werden bereits fast alle SCOPELAND-Anwendungen auf Basis der einen oder anderen Variante der Rahmenanwendung entwickelt und für die Zukunft sind noch viele andere Ausprägungen und Formen solcher Programmrahmen denkbar – auch solche, die von unseren Partnern entwickelt wurden und möglicherweise gänzlich anderen Bedienprinzipien folgen, und die durchaus auch als Open Source-Entwicklungen verstanden werden könnten. Gewissermaßen eine Evolution von Rahmenprogrammen.

Die Rahmenanwendung ist sozusagen das Skelett der Anwendung, sein Herz und seine Außenhaut zugleich.

Dieses Konzept ist auch deshalb so charmant, weil es ermöglicht, alles Komplizierte und aufwändig zu Gestaltende in diesen Rahmen zu verlagern. Angefangen vom zoombaren „Header“ der Anwendung über dynamische Menüs mit komplizierter Rechtelogik (kombinierte Funktionen- und Objektrechte in allen erdenklichen Ausprägungen), über Kommunikationsfunktionen, Basisschnittstellen, Output-Handling, Posteingangs- bzw. Arbeitskörben, bis hin zu übergreifenden Suchfunktionen – all dies kann man elegant in den Anwendungsrahmen verlagern, unabhängig von der konkreten Aufgabenstellung für ein konkretes Modul bzw. nachnutzbar von Projekt zu Projekt.

Selbst die logische Steuerung von Quersprüngen zwischen unterschiedlichen Programmteilen, das Verhalten bei Aufrufen von Choose-Applets (Auswahlseiten), speziell ausgeprägte Zurück/Vorwärtslogik, all dies kann man in den äußeren Rahmen verlagern und für viele unterschiedliche Aufgabenstellungen verwenden.

Der eigentliche Inhalt der zu entwickelten Anwendung kann dann funktional und gestalterisch relativ einfach gehalten und quasi als ‚Content‘ in den Programmrahmen eingebunden werden, vergleichbar mit dem Einbinden von Content in einem CMS-System. Es entspricht einem guten Stil, solche Content-Seiten in Kontrast zum aufwändigen Programmrahmen stets schlicht und einfach zu halten und folglich kann man sie mit SCOPELAND und einem voreingestellten Stylesheet fast automatisch generieren lassen.

Der höhere Design- und Entwicklungsaufwand steckt also im Programmrahmen und ist nur einmal für ganze Pakete von Fachanwendungen eines Kunden zu leisten, teilweise sogar nachnutzbar von Dritten, während die eigentlichen Inhalte besonders rationell in einfacher Gestaltung ‚in Serienfertigung' produziert werden können. Auch dies ist ein weiterer Schritt hin zu noch mehr Effizienz und Flexibilität bei der Entwicklung von Fachverfahren – und ein Schritt, der es nahelegt, darüber nachzudenken, SCOPELAND als einheitliche und übergreifende Plattform für alle Fachverfahren einer Einrichtung anzusehen und so die IT-Kosten insgesamt nachhaltig zu senken.

Die eigentliche Substanz einer Anwendungslösung wird nun in Form von „Content-Seiten" in die Rahmenanwendung eingebracht. Aus Gründen möglichst uneingeschränkter Flexibilität strebt man dabei an, dass jede dieser Seiten möglichst unabhängig von allen anderen ist, mit der gemeinsamen Datenbank als einzige Schnittstelle zwischen den einzelnen Contentseiten. Jede Seite für sich kann aber durchaus Unterseiten in Registerordern, Aufrufe von Choose-Applets oder ähnliches enthalten.

Temporäre 1:n-Relationen (Bezüge)

Eine Detail-Relation, egal ob permanent oder temporär, bewirkt immer, dass der jeweilige Schlüsselwert der Ausgangstabelle (des Parents) in die Detailtabelle (Child) als eindeutige Selektionsbedingung eingetragen wird.

Natürlich geschieht dies als dynamischer Zeiger und nicht als fester Wert. Ansonsten würde die Detailtabelle nach einem Scrollen in der Mastertabelle nicht die dann aktuell gültigen Werte anzeigen. Ferner bewirkt die Parent-Child-Verkettung auch gleich noch ein automatisch korrektes Refresh-Verhalten.

Daraus resultiert ein korrektes Master-Detail-Verhalten, auch dann, wenn lt. Datenmodell gar keine Master-Detail-Beziehung vorlag. Achtung: im Normalfall sollte man Wirkrichtungen, also solche Bedingungen, immer nur von „oben" nach „unten" setzen. Trickreiche Ausnahmen bestätigen jedoch diese Regel.

Je nach Zweck dieser Operation spricht man von einer dynamischen „to-many"-Relation, von einem Bezug, oder auch einfach von einer dynamischen Suchbedingung. Es ist aber immer derselbe Mechanismus.

Es muss auch nicht immer eine eindeutige Selektionsbedingung sein. Sie haben die völlige Freiheit, alles mit allem frei zu verknüpfen. Statisch oder dynamisch, hart oder weich, linksrum oder rechtsrum.

Dieser Mechanismus wird übrigens auch beim Anlegen von Suchfeldern/Recherchefeldern angewandt. Auch dies ist wieder ein komfortables, schnelles Vorgehen, das auch einzeln erreicht werden kann: man legt ein Pseudofeld an, zieht einen Bezug von diesem zu dem Datenbankfeld, nach dem selektiert werden soll und legt dann fest, ob die Selektionsbedingung eindeutig sein soll.

Hier ein Beispiel dafür, wie vielfältig man Anwendungen gestalten kann, allein mit dem Mittel solcher Bezüge:

> *In einer Tabelle soll unter bestimmten Umständen ein Datensatz erzeugt werden, der aus umfangreichen Berechnungen anderer Daten erzeugt wird. Man muss hierfür keine Abläufe programmieren, sondern man ergänzt die Berechnungen in Form simpler Formeln als „berechnete Felder" (wahlweise als SQL- oder erweiterte Expressions) in den jeweiligen DVs. Nun ergänzt man das Applet noch um einen DV, der die Zieltabelle enthält und zieht von den berechneten Pseudofeldern aus Bezüge (temporäre Relationen) auf die jeweiligen Zielfelder. Diese erzeugen ja per default eindeutige Selektionsbedingungen, so dass ein simples „DV.NewRow()" -Kommando ausreicht, um den entsprechenden Satz zu erzeugen. Es ist*

gänzlich überflüssig, irgendwelche Daten von A nach B zu schieben, das macht SCOPELAND von selbst, immer entlang der Relationen und Bezüge.

Temporäre n:1-Relationen (Verweise)

Verweisrelationen hingegen erzeugen Verknüpfungen mit anderen Tabellen innerhalb eines SQL-Statements, es werden Verknüpfungen, sog. Joins, gebildet. Auch hier stellen die permanenten, global gültigen Relationen den normalen Rahmen aller sinnvollen und zulässigen Verknüpfungen dar. Aber in Ausnahmefällen, für kompliziertere Gebilde, benötigt man durchaus hin und wieder mal eine andere Verknüpfung.

Ähnlich wie bei den temporären Bezügen/Detail-Relationen wird auch hier der Ausgangspunkt (das Schlüsselfeld) markiert und dann mit einer simplen Menüfunktion eine andere, separat geöffnete Tabelle an dieser Stelle in den Direct View eingeklinkt.

So kann man ganz ungewöhnliche Datensichten zusammenstellen oder ganz andere Daten für Auswahlfunktionen in die Anwendung einbauen.

Temporäre Verweis-Relationen werden übrigens auch dann (automatisch) gebildet, wenn sich ein neu erzeugtes Recherchefeld auf den (technischen) Primärschlüssel der Tabelle bezieht, wenn es also darum geht, genau einen konkreten Datensatz auszuwählen. Dann verweist das Pseudofeld noch einmal auf die eigene Tabelle und diese Beziehung bewirkt dann die Auswahlfunktion aus der eigenen Tabelle.

Überhaupt lässt sich mit Tricks, wie z.B. auf sich selbst zeigende Verweis-Relationen, allerlei bewerkstelligen.

Generierungen von Druckausgaben (Dokumente, Reports oder Datenbereitstellungen)

Von besonderer Bedeutung ist auch das von SCOPELAND eingesetzte Verfahren zur Generierung komplexer Druckausgaben, womit hier sowohl aus dem Workflow der Anwendung heraus generierte Dokumente (z. B. Bescheide, Rechnungen oder sonstige offizielle Papiere) gemeint sind als auch Auswertungen, Reports und Statistiken oder einfach nur die Bereitstellung von exportierten Daten zur weiteren Verarbeitung in einem anderen Format (z. B. als Excel-Datei).

Das hierfür eingesetzte Verfahren wurde originär von der Scopeland Technology GmbH entwickelt und europaweit zum Patent angemeldet. Es ermöglicht in einzigartiger Art und Weise die interaktive, vollständig programmierfreie Entwicklung von Programmen zur Generierung von Druckausgaben mit komplex ineinander verschachtelten Daten, wie sie für anspruchsvolle behördliche oder wirtschaftsrelevante Dokumente normal sind.

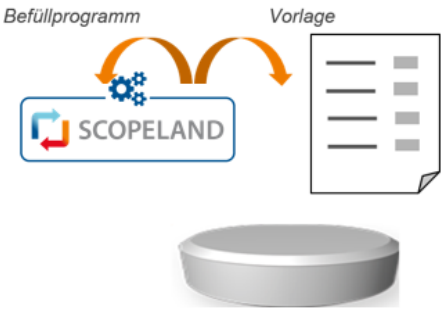


Abbildung 4: Konfiguration von Druckausgaben

Druckausgabeprogramme (Dokumente oder Reports) werden wie interaktive Seiten konfiguriert, allerdings ohne finales Layout. Daraus generiert SCOPELAND eine manuell editierbare Vorlage im Format des Zielsystems und ein zugehöriges Befüllprogramm.

Die Entwicklung erfolgt zunächst analog zu der interaktiver Bildschirmmasken: indem sich der Benutzer hierarchisch angeordnete Datensichten interaktiv zusammenstellt, vielleicht ergänzt um berechnete Spalten, Gruppensummen oder ähnliche Extras. Die druckbare Formatierung und Textgestaltung aber überlässt er den dafür besser geeigneten Standardsoftwareprodukten wie z. B. Microsoft Word oder Excel.

In ähnlicher Weise wie bei der allgemein bekannten Serienbrieffunktion gängiger Office-Produkte werden dabei Vorlagen mit Platzhaltern versehen, die dann später im Zuge der Dokumentenerzeugung durch die jeweiligen Daten ersetzt werden. Im Unterschied zur einfachen Serienbrieffunktion impliziert das SCOPELAND-Verfahren aber eine rekursive Verschachtelung mehrerer Datensichten entsprechend der sog. Master-Detail-Logik, z. B. mehrere Rechnungspositionen zu einem Adressaten, welchen wiederum mehrere Lieferpositionen zugeordnet sind oder mehrere Auflagen zu einem Fördermittelbescheid, welche wiederum jeweils durch keines, ein oder mehrere einzureichende Dokumente nachzuweisen sind.

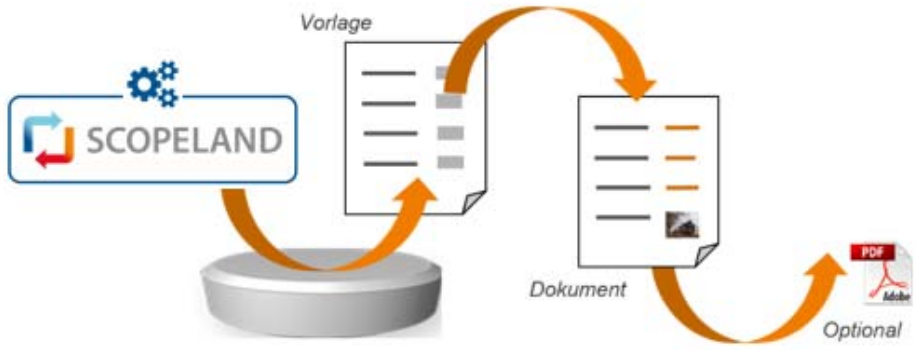


Abbildung 5: Verfahren der Dokumenten- und Reportgenerierung

Druckausgaben, die zur Laufzeit erzeugt werden sollen, arbeiten mit Serienbrief-ähnlichen Platzhaltern, allerdings mit weitaus mehr Ausgestaltungsmöglichkeiten, wie z. B. verschachtelte Master-Detail oder Master-Detail-Detail-Daten.

Die Vorlage kann angereichert werden um diverse Steuer-Tags, welche zudem auch noch eine datenabhängige Textausgestaltung ermöglichen und die Dokumentengenerierung kann sogar um Laufzeitfeatures wie z. B. eine bedingte Formatierung der dynamisch eingefügten Daten bereichert werden.

Das gesamte, oftmals recht komplizierte Paket auszugebender Daten, die Generierung eines Vorlagenentwurfs und des Programms, das diese Vorlagen dann befüllt, samt aller denkbaren Optionen und Einstellmöglichkeiten, all dies erfolgt rein interaktiv, ohne eine einzige Zeile Script oder sonstiger Codierung. Und die so entstandene Vorlage kann nun mit den Mitteln des Zielsystems wie z. B. Microsoft Word oder Excel bzw. die entsprechenden Open Office-Produkte manuell ausgestaltet werden, auch dies ohne dass vertiefte IT-Kenntnisse dafür nötig wären.

Dieses Verfahren wird sowohl von geschulten Anwendungsentwicklern verwendet, die Druckausgaben aller Art als Teil der zu entwickelnden Anwendungen konfigurieren als auch von Endanwendern, die relativ unabhängig davon eigene Auswertungen erstellen wollen.

Für letztere steht zusätzlich noch der sog. SCOPELAND Report Builder bereit, der faktisch ein abgerüsteter und leicht angepasster Direct Desk ist, funktional auf genau das beschränkt was ein Endanwender darf: in seinem Scope durch die Datenbank zu navigieren und aus den für ihn einsehbaren Daten eigene Reports erstellen und ggf. diese für Dritte freigeben.

Der Report Builder verwendet also (quasi als Abfallprodukt) die für die Anwendungsentwicklung ohnehin nötigen Metadaten und ermöglicht es Endanwendern, denkbar einfach beinahe beliebige Auswertungen selbst zusammen zu klicken, ohne SQL-Kenntnisse über mehrere Tabellen hinweg und ohne dafür die Datenstrukturen auch nur kennen zu müssen. Dies ist ein besonderes Feature, das auch die verbreitetsten Reporting-Tools und BI-Produkte i.d.R. nicht oder nur sehr eingeschränkt bieten.

Schnittstellen

Ähnlich leistungsfähig ist das von der Scopeland Technology GmbH entwickelte Verfahren zur interaktiven Konfiguration von Schnittstellen. Es basiert zunächst auf derselben zum Patent angemeldeten Basistechnologie wie die Generierung von Dokumenten und auch auf denselben Grundprinzipien. Und es unterstützt in gleicher Weise nicht nur den Umgang mit einfachen flachen tabellarischen Datensichten, sondern auch den Austausch größerer, komplexer Datenpakete samt mehrfach ineinander verschachtelter unterschiedlich gearteter Daten (Master-Detail bzw. Master-Detail-Detail-Konstrukte aus mehreren einzelnen, miteinander verknüpften Datensichten).

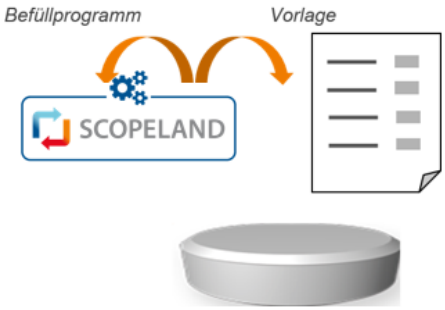


Abbildung 6: Konfiguration von Schnittstellen

Schnittstellenprogramme werden wie Druckausgaben interaktiv konfiguriert. Daraus generiert SCOPELAND eine manuell editierbare Vorlage im gewünschten Schnittstellenformat und ein zugehöriges Befüllprogramm.

Ähnlich wie bei einem Report wird auch beim interaktiven Konfigurieren von Schnittstellen zunächst das Paket der zu exportierenden bzw. zu importierenden Daten interaktiv zusammengestellt, mit denselben Bedienhandlungen wie auch bei der Entwicklung interaktiver Seiten. In gleicher Weise wird daraus automatisch eine Vorlage mit rekursiv verschachtelten Platzhaltern und ein Programm zum Befüllen der Vorlage aus dem jeweiligen Datenstand der Datenbank generiert, wiederum mit diversen Einstellmöglichkeiten und Optionen.

Dabei werden für Schnittstellen typische Zielsysteme und -formate angesprochen wie XML/XSD, CSV oder ASCII-Dateien mit festen Längen und eingeschränkt XLSX. Genau so wie bei der Dokumentengenerierung bzw. beim Reporting anschließend die erzeugten Vorlagen manuell getextet und ausgestaltet werden, erfolgt hier ein manuelles Finalisieren der Exportvorlagendateien entsprechend des jeweils vereinbarten Satzaufbaus der Schnittstellendateien. Das erfolgt direkt und unmittelbar mit den jeweils für die Formate eingestellten Editoren, ohne dass dafür vertiefte SCOPELAND-Kenntnisse erforderlich wären. Und auch für Schnittstellen gilt: nicht nur für flache Datensichten, sondern auch für weitaus komplexere Strukturen.

Dieselben, einmal erstellten Programme und Vorlagen können nun aber auch für den Import in ebendiese Datenstrukturen hinein verwendet werden. Hierzu ermittelt das generierte Programm aus der Struktur der Vorlagendatei zur Laufzeit, wie die gelieferten, zu importierenden Daten inhaltlich

zuzuordnen sind und schreibt sie an die entsprechenden Stellen in die Datenbank. Auch dies wiederum nicht nur für einfache, flache Datentabellen, sondern auch für verschachtelte Datenpakete.

Mit dieser besonders cleveren Lösung gelingt es, auch den weitaus schwierigeren Teil der Anwendungsentwicklung, die Schnittstellenarbeit, weitestgehend programmierfrei umzusetzen.

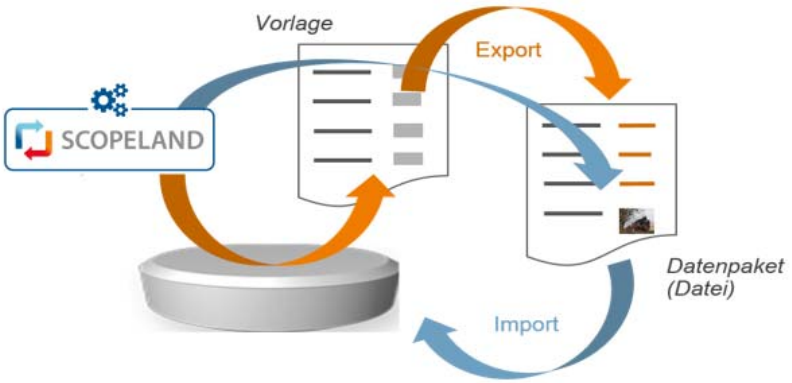


Abbildung 7: Verfahren der Bedienung von Schnittstellen

Ähnlich wie Druckausgaben arbeiten Schnittstellen mit Serienbrief-ähnlichen Platzhaltern in Vorlagendateien im Schnittstellenformat, wie z. B. XML oder CSV und dies mit weitaus mehr Ausgestaltungsmöglichkeiten, wie z. B. verschachtelte Master-Detail oder Master-Detail-Detail-Daten. Nach dem umgekehrten Verfahren erfolgen Importe, indem anhand der Vorlagendatei die Daten in der Importdatei lokalisiert und zugeordnet werden. Ein und dasselbe Schnittstellenprogramm und seine Vorlage können sowohl für den Export als auch für den Import genutzt werden.

Die oftmals bei programmierten Lösungen in die Schnittstellenprogramme eingebauten inhaltlichen Prüfungen der Daten gehören unserer Ansicht nach da nicht hin. Stattdessen werden die Daten in SCOPELAND-Anwendungen nach minimaler Formatprüfung bevorzugt so eingelesen wie sie sind und zwar zunächst in Vortabellen, wo sie dann innerhalb der Datenbank und mit den regulären Mitteln der SCOPELAND-Anwendungsentwicklung geprüft, validiert, versioniert und ggf. weiterverarbeitet werden. Durch diese Trennung von reinem physischem Importvorgang und inhaltlicher Verarbeitung importierter Daten lässt sich die sonst allzu komplexe Aufgabenstellung elegant in beherrschbare Teilaufgaben zerlegen, die programmierfrei entwickelbar sind.

Soll der Datenaustausch zudem noch mittels Webservices automatisiert werden, so werden die in beschriebener Art und Weise interaktiv entwickelten Schnittstellenprogramme anschließend in Push- oder Pull-Webeservices eingebettet, die ebenso interaktiv konfigurierend entwickelt oder aber manuell programmiert werden können. Mit extensivem Einsatz dieser Herangehensweise entwickelt man SOA-Applikationen, die sich dann in vorhandene SOA-Systemlandschaften einbetten lassen und zwar auf beiden Ebenen: sowohl Applikationen, die Webservices konsumieren als auch die Bereitstellung von Webservices und natürlich auch beides miteinander kombiniert.

Datenflüsse: Aggregation, Transformation und vieles mehr

In ähnlicher Art und Weise kann man auch Anwendungen bzw. Anwendungsteile konfigurieren, mit denen Daten im Block als Massendaten verarbeitet werden.

Auch hierbei arbeitet man mit interaktiv konfigurierten Datensichten auf relational verknüpfte Daten. Mit so erzeugten Datensichten kann man ja sehr viele unterschiedliche Dinge tun, wie z. B. exportieren, importieren, ausdrucken, auf dem Bildschirm darstellen, diese als Datenbankview definieren, so strukturierte Datenbanktabellen erzeugen uvm. Darüber hinaus kann man Datensichten aber auch dazu benutzen, um Datenflüsse zu definieren.

Abbildung 8: Datenflüsse

Datenflüsse sind Massendatenoperationen innerhalb der Datenbank und sie werden für unterschiedliche Zwecke eingesetzt, z. B. zur Voraggregation von Massendaten, als ETL-Tool oder zum Umschlüsseln und Aufbereiten importierter Drittdaten.

Das Prinzip ist ganz einfach: man definiert eine Quell- und eine Zieldatensicht, legt ein paar Optionen fest, z. B. ob vorhandene Daten überschrieben werden sollen oder nicht und überlässt alles Weitere wieder der vorgefertigten Software SCOPELAND. Aus den Konfigurationsdaten werden nun, je nach Situation und Kontext, Programme generiert, die entsprechende Datenströme von der Quell- in die Zieldatensicht bewirken. Dieser Mechanismus kann nun für die unterschiedlichsten Zwecke verwendet werden: im einfachsten Fall als ETL-Werkzeug, zur Voraggregation von Massendaten oder einfach zum Kopieren von Datensätzen. Mittels clever strukturierter Datensichten kann man z. B. über gesondert eingepflegte Relationen Querverbindungen zwischen unterschiedlichen Katalogsystemen oder Schlüsselsystematiken herstellen und auf diese Weise Daten umschlüsseln, umformatieren, prüfen, validieren oder auf sonstige Art und Weise aufbereiten.

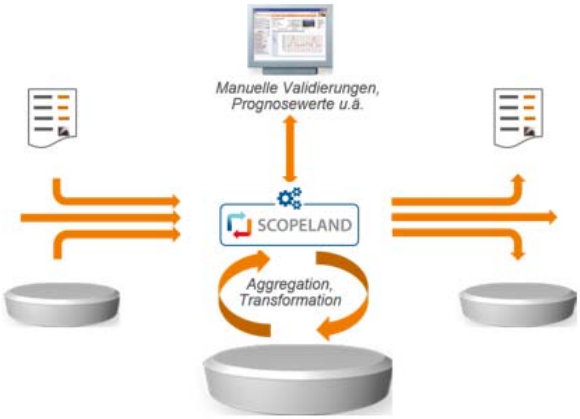


Abbildung 9:
Komplexe Datenverarbeitung

Im Zusammenspiel von Import-, Datenfluss- und Exportfunktionen, ergänzt um manuelle Datenvalidierung und -bereinigung, lassen sich komplexe und performante Datenverarbeitungslösungen aufbauen.

Die Vielfalt der Möglichkeiten, clevere Datensichten und Datenflüsse zur Datenverarbeitung einzusetzen, erschließen sich oftmals nur auf den zweiten Blick. Sie sind aber wichtig, um dem Anspruch zu genügen, auch wirklich anspruchsvolle Verwaltungslösungen komplett programmierfrei umzusetzen. Diese Mechanismen werden nicht in allen Projekten benötigt, aber sie runden den Funktionsumfang für sehr viele Arten von Massendatenverarbeitung sinnvoll ab.

Integrierte Kartendarstellung

Ein wichtiges Sonderfeature ist die standardmäßige Integration von Kartencontrols und Kartendiensten in SCOPELAND-Webanwendungen. Mit wenigen Klicks binden Sie perfekte, ausgefeilte Kartendarstellungen und Kartendienste von Google Maps, Bing Maps oder OpenStreetMap in generierte Webanwendungen ein, bidirektional synchronisiert, samt automatischer Adressverortung und vielen anderen bekannten Features und dennoch ergänzt um etliche für Fachanwendungen erforderliche Spezialfunktionen, die man sonst nur von ‚echten' GIS-Systemen her kennt. Das tollste daran: es kommt ‚out of the box'. Man muss fast nichts dafür tun. Das alles ist quasi von selbst verfügbar, in jeder Anwendung, die in irgendeiner Form Daten mit Ortsbezug enthält.

Die von den drei genannten Kartendiensten angebotenen Dienste werden, je nach Erfordernis, entweder auf dem Thin Client im Browser oder vom Web-/Application-Server aufgerufen und eingebunden. Die Kartendarstellung beschränkt sich dabei nicht etwa auf die Einbindung der Karte selbst, sondern es werden die jeweiligen dynamischen Daten der jeweils aktuellen Datensicht auf der Karte aufgeblendet. Und dies nicht nur für Punkte (geographische Koordinaten, Hoch- und Rechtswerte oder Adressen), sondern ebenso für Polygone (z.B. Flüsse, Straßenabschnitte o.dgl.) oder (halbtransparent) für Flächen (z.B. Bundesländer, Naturschutzgebiete, Vertriebsgebiete o.dgl.) – kurz gesagt: für geographische Objekte aller Art.

Alle Objekte können beschriftet und mit unterschiedlichen Symbolen oder farbig unterschiedlich dargestellt werden. Ebenso kann man per Klick oder Mouseover aufgeblendete Masken mit detaillierterer Darstellung oder aktiven Programmfunktionen anzeigen – und all dies wahlweise auf den Kartentypen des jeweiligen Diensteanbieters (z.B. Straßenkarte oder Gelände). Ähnlich wie in einem GIS-System können Objekte unterschiedlicher Datensichten in unterschiedlichen Layern dargestellt werden, unterschieden in aktive (anklickbare) und passive Layer zur rein informativen Darstellung.

Neben den viel geringeren Kosten und der weltweiten kostenfreien oder fast kostenfreien Verfügbarkeit dieser Kartendienste zeichnet sich der Lösungsweg mittels öffentlicher Kartendienste auch durch den ganz außergewöhnlich hohen Bedienkomfort dieser marktführenden Angebote aus, wie z.B. die bekannte elegante und rasant schnelle Navigation selbst bei langsamer Netzanbindung, die hohe Aktualität der Inhalte und die hohe Fehlertoleranz bei der Adressverortung.

Als Datenpool für die aufzublendenden Daten wird neben den kundeneigenen Geodaten häufig auf Open Data-Angebote zurückgegriffen. Beispielsweise sind heutzutage nahezu alle administrativen Grenzen (Staaten, Länder, Kreise, Kommunen etc.) verfügbar und dies je nach Bedarf in unterschiedlichen Genauigkeiten. Wenn man anstelle dessen oder zusätzlich die eigenen institutionsinternen geographischen Daten auf die Google-, Bing- oder OpenStreetMap-Karten aufblendet, dann stellt sich für die eine oder andere Aufgabenstellung durchaus die Frage, ob man überhaupt überall ein echtes GIS-System einbinden muss.

Auch für die Integration echter GIS-Systeme besitzt SCOPELAND eine Möglichkeit; die dafür erforderliche ‚Embedded GIS'-Schnittstelle wird schon seit geraumer Zeit unterstützt und dies wird auch weiterhin der Fall sein. Die Integration öffentlich verfügbarer Kartendienste ist in erster Linie als Ergänzung und nicht als Alternative zur echten GIS-Integration gedacht.

Besonders erwähnenswert sind noch die ergänzenden Serverfunktionen der Kartendienste, beispielsweise zur Adressverortung, zur Ermittlung von Entfernungen und Fahrzeiten bei unterschiedlichen Verkehrsmitteln, zur Umkreissuche und vieles mehr. Diese können auf denkbar einfachste Weise an allen Stellen in allen SCOPELAND-Anwendungen verwendet werden, mit denkbar geringem Entwicklungsaufwand bei sehr hohem Nutzeffekt.

Und immer mehr Features…

… kommen hinzu, von Version zu Version. Anforderungen aus strategisch wichtigen Projekten werden vom Hersteller Scopeland Technology GmbH teilweise verallgemeinert und in den Produkt-Code aufgenommen.

Nicht alles aber ist so einfach und leicht erlernbar wie SCOPELAND an sich und deshalb finden sich viele Sonderfeatures nur in der Enterprise Edition bzw. in der Extended Enterprise Edition des Produkts. Die Features können auf Nachfrage verfügbar gemacht werden und für Entwickler gibt es themenbezogene Spezial-Coachings.

Hier nur mal ein paar Stichpunkte, zu welchen Themen solche zusätzliche Produktfunktionalität verfügbar ist:

- Integrierte Terminkalender
- Webmenüs in vielerlei Ausgestaltungen
- Erweitertes Dokumentenhandling (Master-/Rahmendokumente, Textbausteine, datengetriebene Textausgestaltung uvm.)
- Webbasierte Bearbeitung von Textdokumenten und Vorlagen
- PDF-Generierung, auch auf Java-Servern
- Intelligente Druckservices
- Office-Integration (E-Mail, Termine, Kalender u.a.)
- Automatische Programme (Hintergrundprozesse, Process Server und Scheduler-basierte Lösungen)
- Listener-Funktionen in Webseiten für aktive datengetriebene Aktualisierungen
- TAPI- u.a. Integrationsverfahren für andere Kommunikationsmedien
- SharePoint-Integration
- Embedded GIS (ESRI, MapServer u.a.) usw.

Der Anspruch des Herstellers: SCOPELAND (wenn nötig mit solchen Produktergänzungen) als ***die*** Plattform für individuelle Datenbankwendungen aller Art. Es gibt keinen Grund, es anders zu tun. Mit SCOPELAND sind Sie so viel schneller und so viel besser, dass es kaum nachvollziehbar ist, warum man etwas händisch machen sollte, was auch Fertigsoftware wie SCOPELAND schon abdeckt.

Scopeland Technology GmbH
Düsterhauptstraße 39 - 40
D - 13469 Berlin
Tel. +49 (30) 209 670 - 0
Fax +49 (30) 209 670 - 111

info@scopeland.de
www.scopeland.de

Geschäftsführer: Karsten Noack
Amtsgericht Charlottenburg HRB 176787 B
USt.-Nr.: DE 183446349