2 Der Retail Store der Migros

Der Retail Store der Migros war der Auslöser für die Entwicklung vieler in diesem Buch vorgestellter Konzepte. Ich möchte dieses erfolgreiche Entwicklungsprojekt daher hier kurz vorstellen. Das Kapitel soll Ihren Appetit anregen und Ihnen zeigen, dass es möglich ist, komplexe Webanwendungen mit hoher Dialoglast und mit Sitzungskontexten auf BSP-Basis zu realisieren. Der Retail Store selbst und dessen Funktionen sind nicht Inhalt dieses Buches. Auch bekommen Sie mit diesem Buch keine Storelösung zum fertigen Download.

2.1 Was ist der Retail Store?

Der SAP Retail Store (SRS) ist eine von SAP ausgelieferte Webanwendung, die es dem Filialmitarbeiter ermöglicht, sein tägliches Geschäft vom Webbrowser aus zu erledigen: Er kann Filialaufträge erfassen, um zusätzlich benötigte Artikel vom Zentrallager zu bestellen, Wareneingänge und Warenverschiebungen buchen, Kundenaufträge erfassen und kann die Inventur durchführen – im Übrigen bietet der Store eine Vielzahl von Listenauswertungen, Statistiken, Übersichten über alle Bereiche des Geschäfts zur Anzeige oder zum Ausdrucken an. Hier eine Sicht auf das Grundmenü des SRS, wie er bei der Migros produktiv im Einsatz ist (siehe Abb. 2–1).

Der große Vorteil des SAP Retail Store liegt – neben seiner einfachen, intuitiven Bedienungsoberfläche – in der *Plattformunabhängigkeit*. In der Filiale wird nur ein PC mit Netzwerkanschluss und Webbrowser benötigt. Weitere Einstellungen sind nicht nötig. Im Gegensatz dazu erforden SAP-GUI-Anwendungen die Installation zusätzlicher Software auf allen PCs und erhöhten Ausbildungsaufwand, da die Bildschirmmasken (Dynpros) der SAP-Anwendungen zwar mächtige Funktionen bieten, aber ohne Schulung in der Regel nicht effizient bedient werden können.

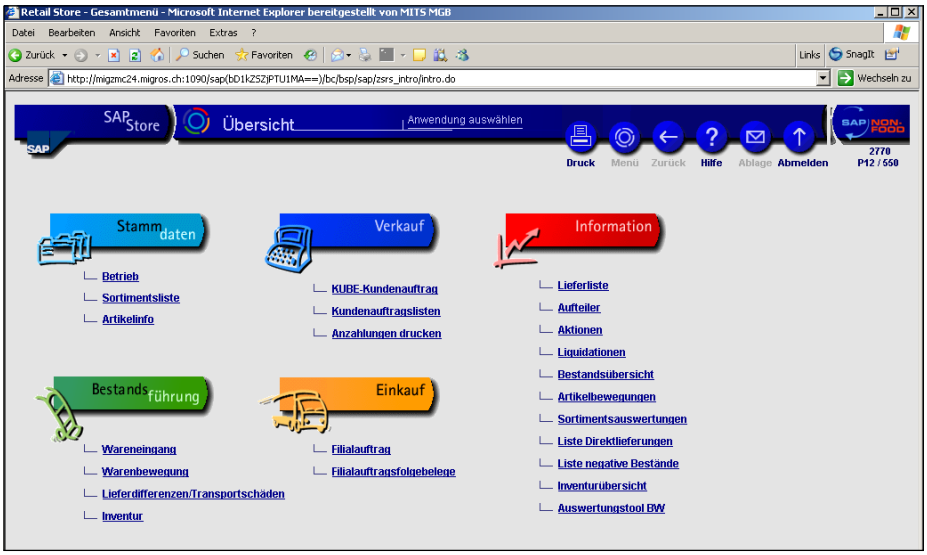


Abb. 2–1 *Das Hauptmenü des SRS*

Demgegenüber weist der SRS folgende Nachteile auf:

- Die technische Basis, auf der der SAP Retail Store realisiert ist, der *Internet Transaction Server* (ITS) ist mittlerweile veraltet. Neue Applikationen der SAP werden nicht mehr auf Basis des ITS entwickelt, sondern mit Web Dynpros oder Business Server Pages (BSP).
- Erhöhte laufende Betriebskosten durch zusätzliche Server: Aufgrund der ITS-Technologie werden zum Betrieb des SAP Retail Store neben den SAP-Application- und Database-Servern noch weitere ITS-Server benötigt, die die Aufgabe haben, die aus dem Netz eingehenden Requests SAP-Sitzungen zuzuordnen, die SAP-Applikationslogik aufzurufen, und die Rückgabedaten in HTML aufzubereiten.

  Dieser Punkt wäre zwar mit Basisrelease 6.40 entschärfbar, denn ab diesem Release ist der ITS in den SAP Web Application Server integriert. Für das NonFood-Projekt war dies leider zu spät: Für den Go-Live-Termin August 2003 war an ein Basis-Upgrade nicht mehr zu denken. Immerhin erfolgte der Go Live letztlich mit Basisrelease 6.20 und dem »Enterprise« Anwendungsrelease 4.70.
- ITS-Webapplikationen sind grundsätzlich zustandsbehaftet (*stateful*). Das bedeutet, bei jeder Anwahl eines SRS-Links wird eine neue SAP-Sitzung erzeugt und für eine gewisse Zeit aufrechterhalten. Die Sitzung bleibt auch dann erhalten, wenn der Anwender seinen Webbrowser schließt. Das bindet Ressourcen auf dem SAP-System, die sinnvoll anderweitig verwendbar wären, z.B. für die regelmäßig eingeplanten Jobs. Zwar ist die Zeitspanne bis zum Beenden der Sitzung im Prinzip einstellbar, jedoch darf sie nicht zu kurz

gewählt werden, damit die Anwender nicht während der Transaktionsbearbeitung plötzlich ihre Daten verlieren.

- Die Implementierung des SRS trennt nicht klar die zur Präsentation der Daten nötige Logik von der Anwendungslogik. Die Dynpros, auf denen die ITS-Services basieren, sind als Dynpros eigentlich reine Präsentationsobjekte; aber bereits die Module ihrer Ablauflogik vermischen die Datenpräsentation mit Applikationslogik. Das erschwert insgesamt die Wartbarkeit und Erweiterbarkeit des SRS. Das gilt insbesondere für die Darstellung der Daten auf verschiedenen Frontend-Typen. Es ist zwar auch im Rahmen des ITS möglich, je nach Frontend unterschiedliche Datenansichten an den Client zurückzugeben, jedoch mit einem erheblichen Anpassungsaufwand.

Für den Bereich »Non Food« der Migros, der mit ca. 900 Filialen und rund 400.000 Artikeln produktiv ist, ist das Thema Ressourcenschonung von hoher Priorität. Darüberhinaus bedient das Non Food System nicht nur Supermärkte, sondern auch Fachmärkte (Hobbymärkte, Gartencenter, Metall- und Möbelfachgeschäfte), die in den Verkaufsbereich der Migros integriert sind.

Daraus resultiert eine Vielzahl geschäftsspezifischer Anforderungen, die mit dem SAP-Standard nicht abgebildet werden können. Die einfache und flexible Erweiterbarkeit des Retail Store war vor diesem Hintergrund ein Muss.

2.2 Das Projekt »SRS auf BSP«

Die Leitung des Projekts »Zentrale Warenwirtschaft« der Migros fällte den Entscheid, den von SAP ausgelieferten SRS auf neuer technischer Basis zu implementieren. Schon zuvor hatte es einen Stopp aller mit dem Retail Store verknüpften Modifikations- und Erweiterungsarbeiten gegeben. In Gesprächen mit der Entwicklungsabteilung wurden die Anforderungen der Migros für eine Ausschreibung weiter konkretisiert:

- *Web-AS-basierte Lösung*
  Die Web-Requests sollen vom Web Application Server des SAP-Systems beantwortet werden. Der ITS muss aus der Systemlandschaft entfernt werden können.
- *Gleiche Oberfläche*
  Die Migros war schon vor dem »NonFood« mit dem »Food/Near Food« Sortiment in einem separaten SAP-System produktiv. Dieses Projekt, das auf einer unabhängigen Systemlandschaft läuft, arbeitet mit dem Retail Store des SAP-Standards. Da den Filialmitarbeitern nicht zuzumuten ist, für »Non Food« eine andere Oberfläche als für »Food/Near Food« präsentiert zu bekommen, müssen die Standard-Services des Retail Store für Non Food gleiches Aussehen haben wie der ITS-basierte Standard Retail Store.

- *BSPs mit dem MVC-Architekturmuster*
  Der SRS soll in Form von BSP-Applikationen portiert werden, die dem »Model-View-Controller«-Architekturmuster (MVC) entsprechen. Das heißt: Die Sicht auf die Daten (Views), die Abfolge der Bilder und Eventbehandlung (Controller) sowie die Applikationslogik (Model) sollen in klar getrennten Ebenen implementiert werden.

  Die so genannten »BSP-Seiten mit Ablauflogik« wären zwar leicht erstellbar gewesen, waren aber nach dieser Vorgabe verboten, da sie die strikte Trennung von Applikations- und Präsentationslogik wieder aufheben. Die Chance eines wartungs- und erweiterungsfreundlichen Entwurfs, die sich mit der Neuimplementierung bot, sollte auch genutzt werden.
- *Standard aufrufen*
  Die Applikationslogik soll, wenn möglich, durch den Aufruf von BAPIs oder APIs realisiert werden. So wird doppelter Code vermieden. Außerdem werden Bugfixes oder Programmverbesserungen, die durch SAP Support Packages hereinkommen, auch im neuen Retail Store sofort wirksam.
- *Zustandslos*
  Die Anwendungen sollen zustandslos (*stateless*) implementiert sein. Das heißt, nach Erzeugung des HTML, das als Antwort an den Client geschickt wird, soll die SAP-Sitzung beendet werden. Es sollen keine Sitzungen im Hauptspeicher des Systems vorgehalten werden.
- *Kein Aufwand auf Clientseite*
  Auf Clientseite darf kein zusätzlicher Installationsaufwand entstehen; ein JavaScript-fähiger Webbrowser mit aktivierten Cookies muss genügen! Das bedeutet insbesondere ein klares Nein zu allen Arten von Plugins.

Der Zeitplan war ehrgeizig gefasst. Der ITS-basierte Retail Store sollte, mit allen bis zum Projektbeginn von der Migros und ihren Beratungspartnern erfolgten Modifikationen und Erweiterungen, innerhalb von nur drei Monaten in das BSP-Umfeld portiert werden.

Drei Beratungsfirmen stellten sich dieser Herausforderung und sandten der Migros Angebote. Die Entscheidungsfindung war schwierig. Das Team von sechs Köpfen, auf das die Wahl fiel, war in der Lage, die Umsetzung des SRS-Contents tatsächlich fristgerecht fertigzustellen. So entstand der »neue SRS« der Migros – optisch ähnlich der bestehenden Lösung (wie gewünscht), jedoch in seinen internen Strukturen gründlich modernisiert.

Noch als man in der letzten Phase des Projekts war, wurde der neue SRS umfangreichen Tests unterzogen. Nach Fertigstellung des Produkts – vor dem ersten Go Live Termin – mussten diese Tests jedoch intensiviert werden, da nun noch die zuvor eingefrorenen Migros-spezifischen Erweiterungen implementiert wurden. Letztlich waren alle im Store abgebildeten Prozesse von den Erweiterun-

gen betroffen. Nun folgte eine »heiße Phase« von Korrekturen und Weiterentwicklungen.

Bis zum Go Live waren die größeren Schwierigkeiten mit dem Retail Store alle beseitigt, nun ging es um kleinere oder niedrig priorisierte Themen. Das Feedback der Filialmitarbeiter war durchgängig positiv. Insbesondere die vollständig neu entwickelte Applikation zur Kundenauftragsbearbeitung erfreute sich großer Beliebtheit.

Der neue SRS erfüllt im Wesentlichen alle Vorgaben der Projektleitung. Die Elemente seiner Benutzeroberfläche (nicht die Applikationen!) entsprechen exakt der Oberfläche des Standard Retail Stores – mit einigen, genau definierten Ausnahmen. Er ist fast durchgängig *zustandslos* realisiert: Die Benutzerkontexte werden nicht im Hauptspeicher vorgehalten, sondern für jeden Dialogschritt auf- und wieder abgebaut.

2.3 Das Framework

Es wurde schon bei den Vorbereitungen des Projekts klar, dass eine erfolgreiche und solide Lösung auf einem klar definierten, robusten und effektiven Framework aufbauen musste. Das Framework muss nicht nur sicherstellen, dass wiederkehrende Aufgaben zentral gelöst werden, sondern auch, dass die einzelnen Applikationen nach einem gleichen Schema entworfen werden. Im Grunde muss der gesamte Ablauf der Requestbearbeitung durch das Framework vorgedacht werden. Die speziellen Details der Implementierung werden dann an den vom Framework vorgesehenen Stellen implementiert:

> *Das Framework bestimmt die Architektur Ihrer Anwendung. Es definiert die Struktur im großen, ihre Unterteilung in Klassen und Objekte, die jeweiligen zentralen Zuständigkeiten, die Zusammenarbeit der Klassen und Objekte sowie den Kontrollfluss. Ein Framework legt diese Entwurfsparameter im voraus fest, so dass Sie, der Anwendungsentwickler und Programmierer, sich auf die speziellen Details Ihrer Anwendung konzentrieren können. Das Framework enthält die Entwurfsentscheidungen, die in seinem Anwendungsbereich allgemein anzufinden sind.*[1]

Durch die Vorgabe, dass die Implementierung durch MVC-basierte Business Server Pages erfolgen sollte, war das übergeordnete Framework bereits vorgegeben: Das BSP-Framework von SAP. Das Framework des SRS folgt diesem übergeordneten Framework durch Vererbung: alle involvierten Klassen erben von den SAP-Standardklassen.

---

1. Gamma/Helm/Johnson/Vlissides, *Entwurfsmuster*, Addison-Wesley 1996, S. 37

Die *Navigationsstruktur* des Retail Store ist einfach: Von einem zentralen Service, der ein Menü aller möglichen Anwendungen anbietet, kann in die einzelnen Anwendungen verzweigt und wieder zurücknavigiert werden. Die einzelnen Anwendungen enthalten ihrerseits meist ein Einstiegsbild, in dem Selektionen eingegeben werden können, und ein Übersichtsbild, in dem eine Liste angezeigt oder ein Anwendungsobjekt (Filialauftrag, Warenbewegung o.ä.) erfasst und verbucht werden kann. Eine Quer-Navigation von einer Anwendung in eine andere war im ursprünglichen Entwurf nicht vorgesehen.

Dies spiegelt sich in einer entsprechend einfachen Struktur des BSP-Frameworks: Es gibt für alle Controller des SRS eine gemeinsame Oberklasse `ZCL_CO_SRS`, die ihrerseits vom Controller des BSP-Frameworks erbt (`CL_BSP_CONTROLLER2`). Insgesamt gibt es also in der Regel eine dreistufige Klassenhierarchie.[2]

Für die Modelklassen, die die Applikationslogik enthalten, wurde im ersten Entwurf ein einfacher Weg gewählt: Alle Modelklassen erben von einer zentralen Klasse `ZCL_SRS`, die als *Applikationsklasse* ausgewiesen wird. Methoden der Applikationsklasse werden nur vom Controller aufgerufen, nicht von Views oder von anderen Orten. Dies wurde als Programmierrichtlinie festgelegt, obwohl natürlich die Ubiquität der Applikationsklasse eine missbräuchliche Verwendung ermöglichen könnte. Da diese Programmierrichtlinie auch eingehalten wurde, ist eine nachträgliche Umstellung mit verhältnismäßig wenig Aufwand möglich.

Gemeinsame Bildbereiche werden von einer zentralen BSP-Applikation `ZSRS` erstellt und können durch Aufruf von Methoden der gemeinsamen Controller-Oberklasse `ZCL_CO_SRS` in die HTML-Antwort eingefügt werden. Dazu gehört beispielsweise die Navigationsleiste, die oberhalb aller BSP-Anwendungen zu sehen ist. Sie bezeichnet den Ort, an dem man sich zur Zeit im Store befindet und bietet Buttons zur Rückwärtsnavigation, zur Rückkehr ins Menü, zum Aufruf der Dokumentation und zum Abmelden an:

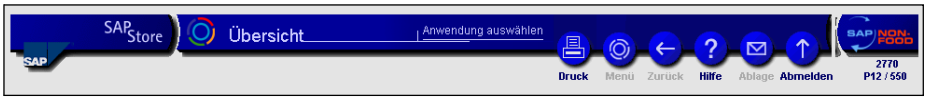


Abb. 2–2 *Die Navigationsleiste*

An der Stelle, wo dies nötig ist, kann die Instanz des Anwendungscontrollers diese Leiste durch Aufruf der Methode `navigation_bar( )` generieren.

Neben gemeinsamen Views gibt es aber auch gemeinsame Elemente des User Interface, wie zum Beispiel *Buttons*

2. Eine Ausnahme bildet hier die Kundenauftragsanwendung, die aufgrund ihrer Komplexität noch weitere Unterklassen ableiten muss und effektiv vierstufig ist.

Abb. 2–3 *Buttonleiste im SRS*

oder *Tabellen*,

Artikel zum Thema KABELZUBEHOER

Sammelartikel Variante Streckenartikel

Webpanel default

Artikel	Artikeltext	Zus.Menge	BestME	CU/TU	Bestand	Akt.	Akt. Sollbst.	Akt. Meldebs	off. Zugän
612015300000	HW-REIHENKLEMMEN 12P		CU	10	18.000		0	0	0.000
612015400000	HW-REIHENKLEMMEN 12P		CU	10	19.000		0	0	0.000
612036200000	STECKKLEMMEN		CU	10	9.000		0	0	0.000
612036300000	LEUCHTENKLEMMEN 5 ST		CU	10	9.000		0	0	0.000
613061600000	KABELBAENDER 153 X 4		CU	5	5.000		0	0	0.000
613061700000	KABELBAENDER 188 X 4		CU	5	5.000		0	0	0.000
613061900000	HW-KABELBAENDER 150X		CU	5	5.000		0	0	0.000
613062000000	KABELBAENDER 200X4,8		CU	5	5.000		0	0	0.000

Zeile 1 bis 8 von insgesamt 9 Zeilen

Abb. 2–4 *Tabelle im SRS*

die durchgängig im Retail Store ein einheitliches Aussehen haben sollen. Diese gemeinsamen Elemente des User Interface wurden durch eine eigene Tag Library (BSP-Extension) realisiert. Welche Anforderungen sich an die Tags ergeben und wie sie parametrisiert werden können, werden wir noch ausführlicher behandeln.

2.4 Zusammenfassung

Den Retail Store auf BSP-Basis neu zu implementieren, war ein ehrgeiziges, technisch anspruchsvolles Unterfangen, das zum damaligen Zeitpunkt darüberhinaus mit einigen konzeptuellen Ungewissheiten behaftet war.

Im Nachhinein können wir jedoch sagen: Das Konzept hat sich bewährt! Das System läuft stabil und mit Antwortzeiten, die sich sehen lassen können. Hier ein Auszug aus der Systemstatistik (ST03) für den Juli 2006 (siehe Abb. 2–5).

Wir sehen, dass der SRS[3] mit über 6 Mio. Requests im Juli eine durchschnittliche Antwortzeit von 0,38 Sekunden realisiert und damit sogar um 0,4 Sekunden

Systemlastübersicht: Mittlere Zeiten pro Schritt in ms

Tasktyp	# Schritte	Ø Zeit	Ø CPU-Zeit	Ø DB-Zeit	Ø Zeit	Ø Wartez.	Ø Roll-In~
ALE	45,344	286.1	58.5	58.3	0.0	8.8	0.9
AutoABAP	85,437	620.3	305.5	89.9	0.0	0.3	1.6
Background	2,073,984	11,082.0	1,353.7	6,907.0	4.3	44.6	4.3
Batch Input	471	220,936.0	33,161.2	164,719.3	0.0	0.3	3.7
Buffer Sync.	174,055	94.4	34.7	75.6	0.0	0.3	0.0
Dialog	5,126,532	779.0	204.1	370.2	0.0	0.7	1.6
HTTP	6,524,115	379.5	144.2	229.8	0.0	0.4	0.0
RFC	8,754,776	2,808.8	583.9	2,026.8	0.0	8.9	1.5
Spool	210,189	459.6	56.3	63.4	0.0	7.2	0.2
Update	1,253,042	469.0	114.3	348.2	0.0	12.2	0.0
Update2	1,873,406	237.5	30.0	160.9	0.0	47.6	0.0
andere	765,143	12.3	1.1	4.5	0.0	0.2	0.0

Abb. 2–5 *Systemlastübersicht*

besser als die GUI-Anwendungen ist (Zeile »Dialog«). Das ist natürlich auch das Ergebnis kontinuierlicher Bemühungen um Performance-Verbesserung.

Es mag manchen erstaunen zu hören, dass sich derartige Antwortzeiten mit einer *fast durchgängig zustandslosen* BSP-Anwendung realisieren lassen.[4] Man könnte befürchten, dass die Performance absackt, wenn nicht der vollständige Sitzungskontext im Speicher des Applikationsservers bereitgehalten wird. Schließlich müssen ja die benötigten Daten bei jedem Dialogschritt aus der Tabelle der serverseitigen Cookies nachgelesen werden. Vor allem für so mächtige Applikationen wie die Kundenauftragserfassung könnte man hier das Schlimmste befürchten.

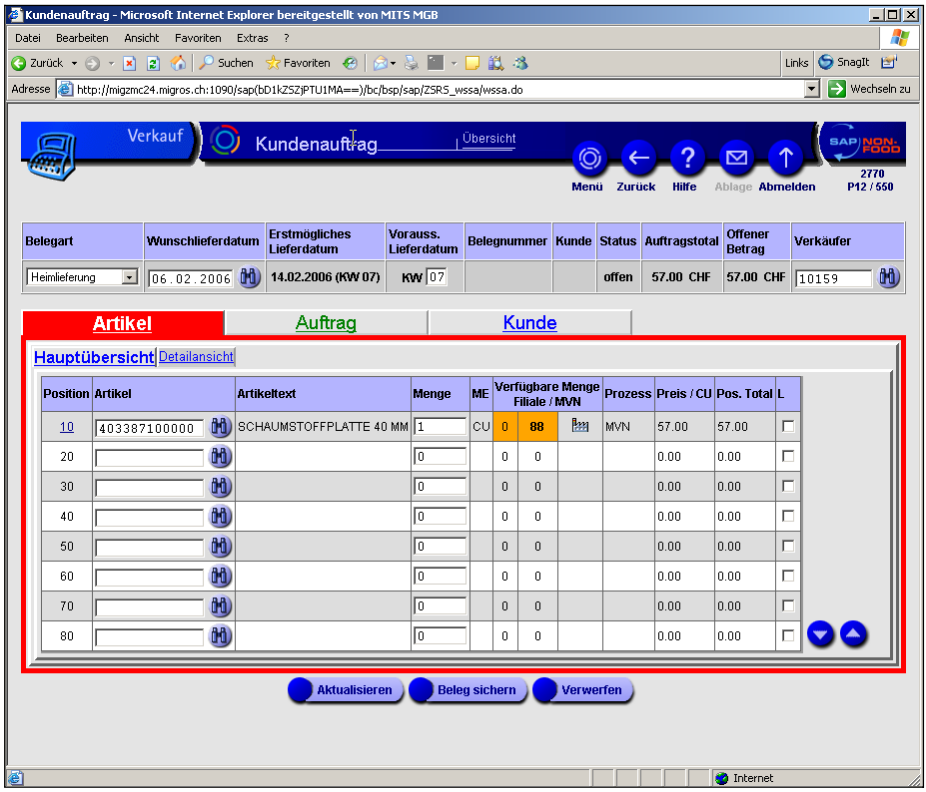
Zur Illustration zeige ich Ihnen hier einen Screenshot aus dem »KUBE-Kundenauftrag«, einer Eigenlösung für die Migros-Fachmärkte, die jedoch letztlich auf die SD-APIs zum Bearbeiten von Kundenaufträgen zurückgreift. Der KUBE-Kundenauftrag bietet

- ein Migros-eigenes Versandkonzept, mit dem die verschiedensten Lieferprozesse abgebildet werden können: Heimlieferungen, Kundenabholungen, Sofortaufträge, und dies alles sowohl für Strecken- als auch für zentral disponierte und beschaffte Artikel;
- eine Eigenlösung zur Abwicklung von Anzahlungen;
- das Ein- und Ausbuchen von Ware aus dem Filialbestand (für Kundenabholungen);

---

3. Der SRS ist praktisch die einzige Anwendung, die in diesem SAP-System den HTTP-Port verwendet. Die Antwortzeiten in der Zeile »HTTP« *sind* also die SRS-Antwortzeiten.
4. Der einzige zustandsbehaftete Dialogschritt – ein View in der Applikation »Lieferdifferenzen« – macht lediglich 0.3% des gesamten HTTP-Traffics aus und spielt daher für diese Bewertung keine Rolle.

- selbstverständlich die Standard-SD-Funktionalitäten wie die Pflege von Rabatten, die Preisfindung und Verfügbarkeitsprüfung, dazu eine Übersicht verfügbarer Bestände in anderen Migros-Filialen und
- für Streckenartikel sogar eine Berücksichtigung der Lieferantenferien!



© SAP AG

Abb. 2–6 *Der SRS-Kundenauftrag*

Doch obwohl in der Kundenauftragserfassung *bei jedem Dialogschritt* simulativ ein Kundenauftrag unter Verwendung des APIs `SD_SALES_DOCU_MAINTAIN` angelegt wird, wobei die Benutzereingaben interpretiert, die oben genannten Funktionen ausgeführt und die internen Auftragsdaten jeweils neu ermittelt werden, fällt der Kundenauftrag mit einer durchschnittlichen Antwortzeit von rund 0.6 Sekunden nicht sonderlich aus dem Rahmen. Das zeigt, dass das Konzept zustandsloser Anwendungen tragfähig ist!

Darüberhinaus hat »stateless« den Vorteil, dass mit den Ressourcen schonend umgegangen wird, weil die Sitzungsdaten nicht im Hauptspeicher gehalten werden. Das ist besonders für Internet-Anwendungen interessant, aber auch für Anwendungen in Firmennetzen, für die eine hohe Dialoglast erwartet wird (wie

den Retail Store – er produziert bei der Migros, wie Sie sehen, allein ebensoviele Dialogschritte wie sämtliche GUI-Transaktionen zusammen!).

Wer »stateless« programmiert, wird seinen Programmierstil verändern. Es ist nicht mehr, wie in der konventionellen ABAP-Entwicklung, alles überall verfügbar. Man muss sich Rechenschaft über die tatsächlich benötigten Daten ablegen. Denn die Daten, die den Dialogschritt überleben sollen, müssen explizit benannt und im serverseitigen Cookie fortgeschrieben werden.

In dem Framework, das ich in diesem Buch vorstelle, geschieht dies durch Implementierung des Interface `ZIF_SERIALIZABLE`. Jedes Objekt, das dieses Interface implementiert, kann sich serialisieren und deserialisieren. Das heißt, sein Zustand kann in einem abstrakten Datencontainer (einem `XSTRING`) abgelegt und daraus auch wiederhergestellt werden. Dadurch wird automatisch das Bewusstsein auf das Datenmodell gelenkt. An die Stelle »des Systems« als universell genutzter Datenhaufen treten ein objektorientiertes Modell mit spezifischen Zuständigkeiten und ein »*on demand*«-Programmierparadigma:

> *Daten sind jeweils dann zu ermitteln, wenn sie auch aktuell benötigt werden – das ist effizienter, als alles zu ermitteln und im Speicher vorzuhalten!*