

Smart Clients, Teil 1: Was ist ein Smart Client? Definition und Abgrenzung

Ganz schön smart, dieser Client!

■ VON CHRISTIAN CAMPO



Fat Client, Applet, Flash, HTML, AJAX oder eine Mischung? Was war Ihr letzter Versuch, eine Anwendung möglichst leicht und verträglich zur Verfügung zu stellen? Waren Sie mit dem Ergebnis zufrieden? Vielleicht ist ein neues Konzept für Ihre Anforderungen besser geeignet. Eines, das so elegant in der Benutzerführung ist wie ein Fat Client, aber ansonsten viele Vorteile, die wir aus Webanwendungen kennen, übernommen hat – ein Smart Client eben.

Smart Clients übernehmen von Fat Clients den überaus hohen Benutzerkomfort, den Thin-Client-Ansätze bisher nicht überzeugend erreichen konnten. Das schließt scheinbar einfache Dinge mit ein, wie die direkte Validierung und Plausibilisierung aller Eingaben, die Verarbeitung im Hintergrund ohne Wartezeit, aber auch die Möglichkeit, laufend zwischen verschiedenen Teilen einer Anwendung zu wechseln.

Das Kernstück eines Smart Client ist die Möglichkeit, komplexe fachliche Logik sowohl auf dem Client als auch auf dem Server zu implementieren. Die Entscheidung über die richtige Aufteilung zwischen Client und Server orientiert sich an fachlichen Anforderungen und weniger an technischen Zwängen. Ändern

sich die Anforderungen, ist es möglich, Module nachträglich vom Server zum Client (oder umgekehrt) zu verschieben. Ein Vorteil, den andere Client/Server-Modelle so nicht bieten können. Last, but not least gelingt die Integration mit lokal installierter Software wie Office oder Mail.

Im Gegensatz zu Fat Clients sind Smart Clients Leichtgewichte. „Copy & Run“-Installation mit Drag & Drop, kein lokaler Datenspeicher und ein vergleichsweise kleiner Footprint genügen, um mit einem Smart Client zu arbeiten. Auch läuft er auch auf einem USB-Stick, da keine Installation oder Registrierung im System notwendig ist. Beim Start kann er automatisch via Software-Update auf den aktuellen Stand gebracht werden.

Seine Daten werden auf einem zentralen Server über Web Services angebunden, gut verschlüsselt mit HTTPS und bei Bedarf auch mit Client-Zertifizierung. Die Daten auf dem Server sind, wie wir es von Webanwendungen nicht mehr missen wollen, immer auf dem aktuellen Stand (keine

Smart Clients – die Reihe

- Teil 1: Was ist ein Smart Client? Definition und Abgrenzung
- Teil 2: Aspekte einer Softwarearchitektur für Smart Clients (ab S. 101)
- Teil 3: Effektive UI-Entwicklung für Smart Clients
- Teil 4: Web Services manchmal besser ohne SOAP
- Teil 5: Smarte Softwareaktualisierung

Datensynchronisierung wie beim Fat Client). Für die Kommunikation zwischen Smart Client und Backend genügt das Internet oder eine einfache mobile Anbindung. Trotzdem ist ein Smart Client nicht für jede Anwendung geeignet. Am besten wird das sichtbar, wenn man die verschiedenen Konzepte miteinander vergleicht.

Fat Client

Der Name Fat Client deutet schon darauf hin, dass ein Großteil der gesamten Anwendung auf dem Client liegt und nur ein kleiner Teil auf dem Server. Ein Fat Client enthält die gesamte Fachlogik und in der Regel einen lokalen Datenbestand. Dieser wird regelmäßig (ob alle paar Tage oder alle paar Minuten) mit einem zentralen Serverbestand abgeglichen.

Durch die lokale Datenbank steigen die Anforderungen (CPU, Speicher, Festplatte) an die Client-Hardware. Weiterhin sind diese Daten immer nur so aktuell wie der Stand der letzten Synchronisierung. Die Vorteile des Fat Client sind der gute Benutzerkomfort und die Möglichkeit, sich mit lokalen Anwendungen eng zu integrieren.

Das Fat-Client-Konzept funktioniert aber nur dann gut, wenn die Daten, die die Benutzer verändern, inhaltlich gut gegeneinander abgegrenzt werden können (jeder bearbeitet seine eigenen Kontodaten, Kundendaten oder E-Mails). Verändern mehre-

re Benutzer dieselben Informationen, führt das bei der Synchronisierung zu Konflikten, die nur manuell bereinigt werden können.

Thin Client

Ein Thin Client, also ein Browser mit einer Webanwendung in HTML, AJAX oder Flash, ist für den Endbenutzer sehr einfach aufzurufen. Die Software ist bereits vollständig vorhanden, Installation und Aktualisierung von Programm und/oder Daten sind kein Thema. Es genügt der Aufruf eines URL. Allerdings funktioniert ein Thin Client auch nur, solange er mit seinem Backend verbunden ist. Jede Verzögerung oder Unterbrechung im Netz wirkt sich direkt auf das Verhalten der Anwendung aus.

Auch wenn gerade AJAX-Anwendungen manchmal überraschende Funktionalitäten vorführen können, sind die Möglichkeiten, fachliche Funktionen verbunden mit Benutzernavigation auf den Client zu verlagern, technisch bedingt eingeschränkt. Schwierig sind Features wie parallele Bearbeitung von mehreren Vorgängen in einer Anwendung, Bearbeitung von komplexen Datenstrukturen und aufwendige Kalkulationen auf dem lokalen Client.

Also Smart Client!

Ein erhebliches Manko von Thin Clients sind die unterschiedlichen Programmiermodelle auf Client und Server. Komponenten und Strukturen können nicht gemeinsam benutzt werden. Smart Clients können bei Client und Server die gleichen Datenstrukturen verwenden und fachliche Komponenten am besten Ort platziert werden.

Auch Fat Clients erlauben es nicht, eine Komponente mit wenig Aufwand auf den Server zu schieben. Sie betrachten den Server als mehr oder weniger intelligenten Datenspeicher, aber nicht als Möglichkeit, Funktionalität zentral auszuführen. Bestimmt durch ihre Architektur kann man nicht einfach fachliche Komponenten herauslösen und zentral auf dem Server ausführen – im Gegensatz zum Smart Client.

Hier ein Beispiel, anhand dessen man die verschiedenen Client-Server-Modelle miteinander vergleichen kann. Betrachten wir das Online-Banking im Webbrowser, also Thin Client. Auf HTML-Webseiten kann man sich einen Überblick verschaffen und Aktionen wie Überweisungen,

Wertpapiere handeln etc. ausführen. Die Daten sind immer aktuell, der Benutzungskomfort ist, meiner Erfahrung nach, recht eingeschränkt.

Auch wenn noch nicht alle technischen Möglichkeiten (wie AJAX) ausgeschöpft sind, ist die Benutzerführung recht einfach gestrickt. Es ist nicht möglich, Aktionen parallel auszuführen, zum Beispiel Überweisungen ausfüllen, während man nebenher die vergangenen Umsätze durchblättert. Jedes Konto wird getrennt behandelt, jede Bank hat ihre eigene Webanwendung mit eigener Anmeldung und natürlich ein unterschiedlich gestaltetes Frontend.

Schauen wir uns als nächstes einen Fat Client für die Finanzverwaltung wie zum Beispiel „WISO Mein Geld“ oder „QUICKEN“ an. Die Anwendung hat – verglichen mit der obigen Browseranwendung – eine sehr komfortable Benutzerführung. Viele Aktionen können lokal durchgeführt, beispielweise Auswertungen, Statistiken und Übersichten, oder zumindest vorbereitet werden, zum Beispiel Überweisungen.

Alle Daten werden in einem lokalen Datencontainer vorgehalten. Die Synchronisierung mit der Bank wird manuell ausgelöst und ist ein für den Anwender spürbarer Schritt: Auslösen, Warten auf das Ergebnis, Überprüfen und möglicherweise Kategorisieren der empfangenen Daten. Mit Ausnahme des Dat

enaustausches ist der Fat Client weitestgehend autonom.

Was einem Privatmann genügt, ist für eine Firma nicht mehr ausreichend. Diese hat mehrere Konten und deutlich mehr Transaktionen. Dieses Volumen kann mit einem Web-Frontend nicht mehr übersichtlich behandelt werden. Gefordert werden hier die Möglichkeiten eines Fat Client, ohne die Daten auf einem lokalen PC speichern zu müssen. Es sollen ja mehrere Anwender gleichzeitig zugreifen können. Zudem sollen die Daten mit zentralen Anwendungen, wie einer Auftragsverfolgung oder einer Buchhaltung, verknüpft werden.

Für diese Anforderungen ist der Smart Client die beste Lösung. Mit seiner zentralen Datenhaltung, der komfortablen Benutzerführung auch bei größeren Datenmengen, der Möglichkeit zu lokalen Auswertungen und der Integration mit

Rich Client = Smart Client?

Im Web werden die Begriffe Rich Client und Smart Client meist synonym verwendet. Gelegentlich wird ein Rich Client aber auch mit einem Fat Client gleichgesetzt. (vgl. Wikipedia). Sehr prominent besetzt wird der Begriff Rich Client durch die Rich Client Platform (RCP) des Eclipse-Projekts.

RCP unterstützt eine ganze Reihe der hier vorgestellten Anforderungen an Smart Clients. Copy & Run, Softwareaktualisierung und einen modularen Aufbau durch Plug-ins (hier Module). Das größte Manko ist die (bisher) fehlende gleichwertige Plattform auf dem Server. Diese würde es erlauben, bereits existierende fachliche Komponenten auch auf dem Server zu benutzen. Darüber hinaus könnten Komponenten, ohne zusätzlichen Implementierungsaufwand, zwischen Client und Server verschoben werden. Hier ist mit der Rich Server Platform (RSP) das entsprechende Gegenstück gerade erst im Entstehen.

Fremdanwendungen stellt er eine Alternative zu Fat und Thin Client dar. Hier nun die wesentlichen Eckpunkte, die für einen Smart Client typisch sind.

Benutzungskomfort

Ein Smart Client hat eine sehr komfortable Benutzerführung, die vergleichbar mit derjenigen auf dem Fat Client ist. Dazu gehören lokale Plausibilisierung (Überprüfung eines Feldes), Validierung (Überprüfung aller Felder im Kontext und gegen die existierenden Daten), Verknüpfung von mehreren Masken, einfaches Wechseln zwischen verschiedenen Teilanwendungen, parallele Bearbeitung von mehreren Vorgängen und Integration mit lokalen Anwendungen (Office, Mail).

Wichtig ist ein „performantes GUI“, bei dem nicht für jede Benutzerinteraktion der Server befragt werden muss, sondern nur dann, wenn im Kontext der Interaktion Daten zwischen Client und Server wandern, beispielsweise bei Aktionen wie Kundenakte suchen, speichern oder Adressenprüfung. Das wird als Benutzerkomfort erlebt und fördert die Akzeptanz.

Verteilte Komponenten in einer gemeinsamen Architektur

Im Smart-Client-Modell benutzen Client und Server die gleiche Basis, auf der Komponenten, Datenobjekte (ValueObjekte) und Schnittstellen (Interfaces) entwickelt werden. Beide Teile, Client und Server, können daher in einer Entwicklungsumgebung erstellt werden. Vieles ist auf beiden Plattformen nutzbar, im Gegensatz zum Thin Client. Ein Verschieben von Funktionalität zwischen Client und Server ist aus diesem Grund einfach möglich.

Komponenten, die auf dem Server laufen, erreicht der Client über Web Services, verschlüsselt und internetfähig. Es ist sinnvoll, die Web-Service-Schnittstellen speziell auf die Anforderungen des Smart Client abzustimmen. Das heißt, die Daten, die übertragen werden, müssen sich eng an die Darstellung auf dem UI und die Anforderungen der lokalen Komponenten anlehnen. Es macht demgegenüber wenig Sinn, die Web-Service-Schnittstellen auf Wiederverwendbarkeit außerhalb des Smart-Client-Umfelds zu optimieren. Der Smart Client muss auch

in Umgebungen mit niedriger Bandbreite wie mobiler Kommunikation ohne große Verzögerung Daten mit dem Server austauschen können. So wird der Eindruck unterstützt, dass man den gleichen Benutzerkomfort wie bei einem Fat Client genießt.

Keine lokale Datenhaltung

Ein Smart Client hat keine veränderlichen Daten bei sich lokal gespeichert. Alle Daten werden direkt vom Server geladen und dort auch wieder gespeichert. Damit können viele Anwender problemlos auf den gleichen Datenbestand zugreifen. Neben den Daten enthält der Server auch fachliche Module. Diese können die Daten validieren, plausibilisieren und kompliziertere finanzielle Kalkulationen durchführen.

Statische Daten wie Länderlisten und Codeverzeichnisse können beim Smart Client lokal gespeichert werden. Das widerspricht nicht dem Smart-Client-Grundkonzept und kann die Performance erheblich steigern.

Integration fremder Software

Existierende zentral laufende Software lässt sich über Web Services in den Smart Client integrieren. Dabei verbleiben Daten und Fachlichkeiten der Fremdsoftware auf dem Server, und der Smart Client integriert die Informationen, die er über Web-Service-Aufrufe bezieht, in seine eigene Anwendung.

Im nächsten Schritt kann man auch Teile des existierenden GUI der Fremdsoftware in den Smart Client übernehmen, als eigenständigen Teil der Smart-Client-Gesamtanwendung. Voraussetzung ist die Benutzung eines gemeinsamen Basis-Frameworks, durch das die verschiedenen Komponenten einfach miteinander verbunden werden können. Die Komponenten der Fremdsoftware kommunizieren, genauso wie die eigenen Module, über Web Services mit ihrem fachlichen Gegenstück auf dem Server.

Integration lokaler Standardsoftware

Auf fast allen PCs ist bereits Standardsoftware installiert, die sich gut in den Smart Client integrieren lässt. Das kann der Export der Daten in Excel, die Erzeugung von Serienbriefen in Word oder die Gene-

rierung von Mails in Outlook sein. Auch PDF Viewer und sogar ein eingebettetes Browserfenster lassen sich im Smart Client integrieren und von dort aus steuern. Mit einer vergleichbaren Integration ist ein Thin Client überfordert.

Software-Update

Durch die enge Abstimmung zwischen Smart Client und Server ist es notwendig, dass der Smart Client immer auf dem aktuellen Softwarestand passend zum Server ist. Kompatibilität über mehrere Versionen wäre möglich (solange sich die Schnittstellen zum Backend nicht ändern), ist aber mit viel Aufwand verbunden. Daher ist es die einfachere Lösung, wenn sich der Smart Client beim Start selbst aktualisiert. Das geschieht automatisch und am besten ohne Eingriff des Benutzers.

Ein performantes Software-Update trägt viel zur Akzeptanz bei den Benutzern bei. Ein Ansatzpunkt dafür ist es, nur die gerade benötigten Komponenten eines Smart Client (inkrementell) zu aktualisieren (z.B. bei einer langsamen Internet-Anbindung).

„Copy & Run“ Install

Ein Smart Client kann kompakt gehalten werden, da er lokal kaum Daten vorhält. Die Installation besteht aus einem intelligenten „Drag & Drop“. Das beeinflusst vorher installierte Software nicht und nimmt auch keine Veränderungen an den Einstellungen im Betriebssystem vor (z.B. Registry). Alle Laufzeitkomponenten, die der Smart Client zur Ausführung braucht, bringt er selbst mit. Nur lose gekoppelte Standardsoftware wie Office, Mail und PDF Reader werden zur Laufzeit dynamisch gefunden. Ein Smart Client erfüllt damit die Voraussetzungen, um auf einem USB-Stick gepackt und von dort aus gestartet zu werden.

Zielgruppe von Smart Clients

Im Verhältnis zum einfachen Eintippen eines URL für den Start einer Webanwendung ist der Aufwand für die Installation eines Smart Client natürlich etwas größer. Selbst ein kompakter Smart Client ist, wenn er eine JRE enthält, doch mindestens 40 MB groß, die einmalig heruntergeladen werden müssen.

Der Aufwand, einen Smart Client zu benutzen, muss deshalb für den Benutzer

einen angemessenen Vorteil bieten. Die Erfahrung zeigt, dass sich Smart Clients besonders für Anwendungen lohnen, die tagtäglich berufsmäßig eingesetzt werden. Beispiele sind Außendienstmitarbeiter im Feld oder Mitarbeiter, die von verschiedenen Filialen aus auf einen zentralen Datenbestand zugreifen müssen und deren Anwendungsszenario von den oben beschriebenen Eckpunkten deutlich profitiert.

Smart Clients sind unter gewissen Umständen auch für private Heimanwender interessant, wie der Google Earth Client (auch ein Smart Client) zeigt, z.B. wenn die Datenmenge einfach zu groß ist, so dass sie nicht auf dem Client vorgehalten werden kann. Oder wenn mehr Interaktion, als im Browser möglich ist, verlangt wird, dann scheut auch ein relativ großer Download die Benutzer nicht.

Hello World!

Wie fängt man am besten an, sich seinen Smart Client zu bauen? Weder kann man sich seinen Smart Client über einen intelligenten Wizard generieren lassen noch eine schlüsselfertige Version herunterladen. Auf der grünen Wiese anfangen und alle Basiskomponenten selbst entwickeln ist auch sehr aufwendig. Am besten ist es, sich die bereits vorhandenen Frameworks für Smart Clients anzuschauen und eines davon auszuwählen. Diese bringen schon einen Laufzeitcontainer mit, für den Komponenten entwickelt werden müssen.

Neben den gängigen Aspekten bei der Erstellung einer neuen Anwendung

wie Datenmodellierung und GUI-Design kann man jetzt daran gehen, die Komponenten aufzuteilen. Da auf Client und Server sehr ähnliche Randbedingungen herrschen, kann man Komponenten zwischen Client und Server hin- und herschieben. Immer wieder gilt es, die Entscheidung zu treffen, „Wo wird was platziert?“ (Client oder Server).

- Ist die Auftragskalkulation besser auf dem Client oder auf dem Server aufgehoben?
- Wo werden Druckdokumente erzeugt?
- Wo Statistiken berechnet?
- Wo Massenbriefe erzeugt?
- Wo druckt man sie aus?
- Wer baut eine Verbindung zu Partnern auf, um dort Daten abzuholen?

Alles Fragen mit keiner universellen Antwort und ein so komplexes Thema, dass wir es detailliert in einem eigenen Artikel (siehe S. 101) besprechen. Aber das ist durchaus *das* Thema, das einen Smart Client so spannend und interessant macht. Einfach die Möglichkeit zu haben, Funktionalität (durch Deployment und Konfiguration) zwischen Client und Server zu verteilen, je nachdem, wo es besser hingehört und ohne bei den technischen Möglichkeiten Kompromisse machen zu müssen. Das ist Smart-Client-Technologie!

Fazit

Wie ich hoffe, haben Sie „Appetit“ auf den Smart Client bekommen. Der Fat Client kommt nur für Anwender in Frage, bei denen jeder Client auf einem abgesteckten Subset der Daten Veränderungen

durchführt. Konkurrierende gleichzeitige Veränderungen führen bei der Synchronisierung des lokalen Datenbestands mit dem Server zu Konflikten, die ein manuelles Eingreifen erfordern.

Lassen Sie sich auch nicht von Web 2.0 aus der Thin-Client-Welt täuschen. Die Anfangserfolge sind sehenswert, aber was, wenn die Datenobjekte auch nur ein bisschen komplex strukturiert werden und die Datenmenge wächst? Dann werden Browser wieder kriechen lernen! (Probieren Sie mal eine Google Map mit 300 oder mehr Markierungspunkten.)

Smart Clients sind eine Lösung mit einem sehr ausgereiften Programmiermodell auf Client *und* Server, dem nicht so schnell die Puste ausgeht, einer Benutzungsoberfläche, mit der das Arbeiten angenehm ist, und immer aktuellen Daten. Das gemeinsame Programmiermodell garantiert, dass Client- und Server-Komponenten sehr gut aufeinander abgestimmt sind. Falls eine Komponente dann doch zwischen Client und Server verschoben werden muss, ist das mit geringem Aufwand möglich, da der Code oft ohne inhaltliche Änderung neu platziert wird. Sie sparen Nerven und Programmieraufwand.



Christian Campo ist IT-Consultant bei com-people AG in Frankfurt. Schwerpunktmäßig beschäftigt er sich mit Client-Server-Frameworks, Web Services und verteilten Serveranwendungen auf Basis von Tomcat.

Smart Clients, Teil 2: Aspekte einer Software-Architektur für Smart Clients

Wer macht was?

■ VON OLIVER KLEBER

Bei einem Smart Client ist nicht schon durch die verwendete Technologie vorgegeben, ob eine Komponente auf dem Client oder auf dem Server realisiert wird. In Verbindung mit einer auf Services basierenden Architektur ist dies abhängig von den speziellen Anforderungen an die Anwendung. Welche Kriterien sind dafür zu beachten und was bedeuten diese für die Verteilung?

Smart Client Frameworks verwenden Techniken wie konfigurierbare Services und Dependency Injection, um die Abhängigkeiten zwischen den Komponenten zu minimieren. Damit ist es mit wenig Aufwand möglich, bei Bedarf einen Service vom Client auf den Server zu verlagern und umgekehrt. Falls sinnvoll, kann ein Service auch in zwei oder mehr Ausprägungen implementiert werden. Denkbar sind so zentrale Lösungen mit der vollständigen Funktionalität, die bei Verbindungsproblemen gegen lokale Lösungen mit reduzierten Möglichkeiten ausgetauscht werden.

Die Entscheidung, ob in einem Smart Client eine Komponente auf dem Client oder auf dem Server realisiert wird, ist in erster Linie abhängig von den Rahmenbedingungen der konkreten Anwendung. Die Architektur eines Smart Client erlaubt viel Flexibilität. Damit treten bei der Suche nach dem optimalen Design zusätzliche Fragen der Softwareverteilung auf. Oft muss eine Komponente in einen Client- und einen Server-Teil aufgespalten werden, um zu einer optimalen Lösung zu gelangen.

Ansteigende Komplexität

Das Beispiel eines Angebotsdrucks zeigt, dass es selten eine allgemein gültige Lösung für die Positionierung einer Komponente gibt.

Drucken? Wieso ist das hier überhaupt ein Thema? Gedruckt wird doch auf dem Client. Der hat den passenden Drucker-treiber, der Drucker steht in der Nähe, und

der Benutzer hat den Ausdruck sofort in der Hand. Folglich wird das Druckdokument auch auf dem Client erzeugt. Ist das wirklich immer so?

Ein Fall aus der Praxis: Ein Außendienst-Mitarbeiter will in seinem Heim-Büro ein Angebot für einen Kunden ausdrucken. Dazu benötigt er die Daten des Kunden, die einzelnen Positionen des Angebots, deren Preise und die Vertragsbedingungen. Im einfachsten Fall werden vom Server nur die einzelnen Positionen mit den aktuellen Preisen abgefragt und aufsummiert. Häufig ist das Sortiment zu umfangreich und ändert sich zu schnell, um es auf dem Client zu halten. Die Kundendaten werden nur für die Anschrift verwendet und beeinflussen die anderen Bestandteile des Angebots nicht. Die Vertragsbedingungen sind immer identisch. Hier spricht tatsächlich alles für die einfache Realisierung auf dem Client. Das Angebotsdokument wird zusammengestellt und ausgedruckt (Abb. 1a).

Die Firma kann aber auch die Politik vertreten, dass Angebote an Kunden nur über eine zentrale Instanz abgeschickt werden dürfen. Gründe dafür können zum Beispiel die Möglichkeit, zusätzliche Unterlagen wie Werbematerial beizupacken, günstigere Kosten durch Massenversand oder der Wunsch nach Kontrolle sein. Mit den Kunden ausgehandelte Sonderkonditionen werden so zentral protokolliert und überwacht. Bei diesem Szenario müssen die Kundendaten und die Positionen des An-

gebots auf den Server übertragen werden. Dieser bestimmt die Preise, stellt das komplette Angebot zusammen, veranlasst den Ausdruck und den Versand (Abb. 1b). Der Außendienst-Mitarbeiter bekommt höchstens noch eine Kopie für seine Unterlagen.

Zurück zum Ausdruck im Heim-Büro. Jetzt soll das Angebot mit einem anspruchsvollen Layout gestaltet werden. Vielleicht mit Fotos der einzelnen Positionen oder Grafiken, welche die versprochene Wertentwicklungen darstellen. Insbesondere in B2C-Beziehungen ist dieses Vorgehen weit verbreitet, um den Kunden vom Unternehmen zu überzeugen. Anstatt diese Daten einzeln auf den Client zu übertragen, bietet sich, genau wie beim zentralen Versand, eine Generierung des Druckdokuments auf dem Server an. Möglicherweise werden nötige Daten sogar erst zu diesem Zeitpunkt dynamisch über Services von den Lieferanten erfragt. Danach wird das fertige Dokument in einem Standardformat, zum Beispiel in PDF, an den Client geschickt

Smart Clients – die Reihe

- Teil 1: Was ist ein Smart Client? Definition und Abgrenzung (ab S. 97)
- Teil 2: Aspekte einer Softwarearchitektur für Smart Clients
- Teil 3: Effektive UI-Entwicklung für Smart Clients
- Teil 4: Web Services manchmal besser ohne SOAP
- Teil 5: Smarte Softwareaktualisierung

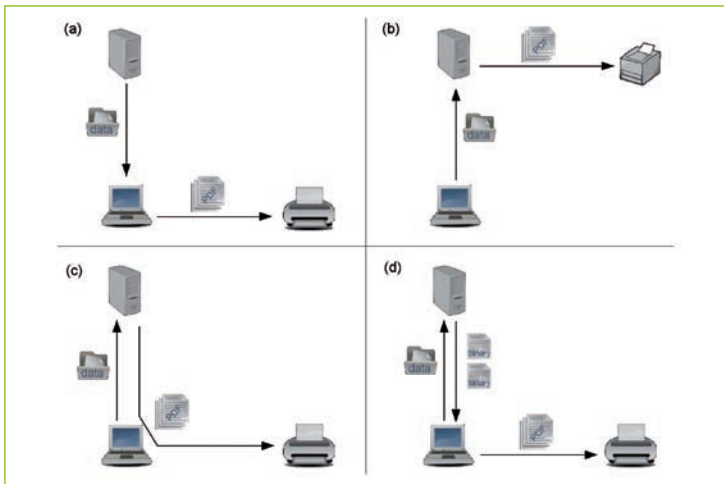


Abb. 1: Drucken – Verteilung der Aufgaben zwischen Client und Server

und erst dort ausgedruckt (Abb. 1c). Die dabei zu übertragende Datenmenge ist bei einer DSL-Verbindung unproblematisch. Lösungen auf Basis von Thin Clients verwenden häufig dieses Vorgehen.

Noch komplexer wird das letzte Szenario unter folgenden Rahmenbedingungen. Der Außendienst-Mitarbeiter soll das Angebot während des Termins beim Kunden ausdrucken. Die Anbindung an den Server erfolgt über GPRS/UMTS. Sofort wird die Größe des fertigen Druckdokuments wieder zum Problem. Damit die Übertragungszeit im akzeptablen Rahmen gehalten werden kann, muss das optimale Zusammenspiel zwischen Client und Server gefunden werden. Das Druckdokument wird in Bausteine zerteilt und auf dem Client wieder zusammengesetzt (Abb. 1d). Caching-Strategien halten die beständigeren Bausteine lokal vorrätig. Die jeweils am besten geeigneten Komprimierungsverfahren werden eingebunden. Das interne Dateiformat des Druckdokuments muss bekannt sein und manchmal sogar proprietär erweitert werden. Auch in einem Smart Client kann der hierfür nötige Aufwand ziemlich groß werden. Aber ohne Smart-Client-Technologie ist dieses Szenario kaum realisierbar.

Offline-Fähigkeit

Das Beispiel des Angebotsdrucks zeigt, wie ein Smart Client von seiner ständigen Online-Anbindung profitiert. Dennoch gibt es gute Gründe, ihm ganz oder teilweise die Fähigkeit zum Offline-Betrieb zu geben. Ein Außendienst-Mitarbeiter, der gerade beim Kunden sitzt und mit dem Laptop keine Verbindung über die GPRS/UMTS-

Karte bekommt, wird seinen Kunden kaum fragen wollen, ob er dessen DSL-Anschluss mitbenutzen darf. Wenn ihre Anwendung ein ähnliches Szenario vorsieht, sollte zumindest eingeschränktes Arbeiten auch ohne Verbindung zum Server möglich sein.

Obwohl die regionalen Lücken in der Abdeckung mit mobilen Telekommunikationsverbindungen immer kleiner werden, ist deren Verbindungsqualität nicht immer ausreichend. Deshalb kann es sinnvoll sein, auf Wunsch die Anzahl und die Dauer der notwendigen Serverzugriffe zu minimieren. Geeignete Techniken hierfür sind unter anderem intelligentes Caching und zusätzliche lokale Services, die im Bedarfsfall als Ersatz für die Web Services dienen können.

Legacy-Komponenten

Kaum ein Projekt beginnt auf der grünen Wiese. Stattdessen werden produktive Anwendungen erweitert oder abgelöst. Aus vielen Gründen kann es dann sinnvoll sein, Teile des schon vorhandenen Codes in die neue Anwendung zu integrieren.

Unabhängig von allen anderen in diesem Artikel aufgezählten Kriterien gilt für Legacy-Komponenten die Regel, dass sie am besten auf der Plattform aufgehoben sind, für die sie ursprünglich entwickelt wurden. Komponenten enthalten oft plattformspezifischen Code und laufen deshalb ohne Anpassungen auch nur in einer Umgebung. Selbst Code in einer eigentlich plattformunabhängigen Sprache wie Java ist häufig für eine Zielumgebung optimiert.

Wenn zum Beispiel eine Tarifberechnung bisher Bestandteil eines Fat Client war, dann ist dieser Code höchstwah-

scheinlich nicht Thread-sicher. Um diese Funktionalität auf den Server zu verlagern, gibt es mehrere Möglichkeiten, die jedoch alle problematisch sein können:

1. Den vorhandenen Code wegwerfen und die Funktionalität Multithread-fähig neu implementieren. Um den dafür nötigen Aufwand zu rechtfertigen sind sehr gute Argumente nötig, denn der alte Code läuft bereits und ist getestet. Falls dieser noch in anderen Anwendungen weiter verwendet wird, verdoppelt sich der Aufwand für die Wartung.
2. Den vorhandenen Code umbauen, um ihn nachträglich Multithread-fähig zu machen. Im Nachhinein ist das nicht trivial, dafür aber fehleranfällig. Das gilt insbesondere, wenn die ursprünglichen Entwickler nicht zur Verfügung stehen.
3. Den vorhandenen Code einpacken und mit einer Warteschlangenverwaltung oder Ähnlichem dafür sorgen, dass er intern nur single-thread aufgerufen wird.

Dann muss aber jeder Benutzer warten, bis alle anderen vor ihm in der Warteschlange fertig sind. In Grenzen lässt sich dieses Problem entschärfen, wenn der Server mehrere Instanzen der Komponente startet und die Aufrufe auf diese verteilt. In Frage kommt diese Lösung hauptsächlich für zustandslose Services. Der erhöhte Ressourcenbedarf wird dann in Kauf genommen.

Umgekehrt ist Code, der für den Server optimiert wurde, meistens nicht die ideale Implementierung für den Einsatz auf einem Client. Kriterien wie Speicherverbrauch und Datenbank-Zugriffe spielen dort eine andere Rolle.

Bei der Integration von Legacy-Komponenten ist zu beachten, ob und wie sie das Interaktionsverhalten beeinflussen. Beispielsweise erfolgen Eingabeprüfungen bei einem Thin Client in anderen Einheiten und zu anderen Zeitpunkten als bei einem Fat Client. Wenn die verwendeten Legacy-Komponenten andere Aufteilungen zwischen Client und Server haben als der Rest der Anwendung, kann dies zu uneinheitlichem und daher für den Benutzer unverständlichem und frustrierendem Verhalten führen. Eine Anwendung sollte immer ein einheitliches, erwartungskonformes Interaktionsverhalten haben. Falls dies

nicht möglich ist, muss das Design der Benutzungsoberfläche die Unterschiede verdeutlichen. Das kann so weit gehen, dass eine ursprünglich für den Ablauf in einem Browser geschriebene Komponente integriert wird, indem die neue Anwendung einen Browser einbindet und die Komponente auf diesem Weg steuert. Benutzer, die schon mit der vorherigen Version gearbeitet haben, wollen das bekannte Verhalten wiedererkennen und nicht umlernen.

Performance

Schon seit längerer Zeit gibt es kaum noch Anwendungsfälle, die auf einen Server zurückgreifen müssen, weil die Rechenleistung einer Client-CPU nicht ausreicht. Der Regelfall ist, dass der Client schnell genug ist und der Benutzer nur noch beim Zugriff auf den Server wartet. Auf diesem liegen die zentral verfügbaren Fachkomponenten und die großen Datenbanken.

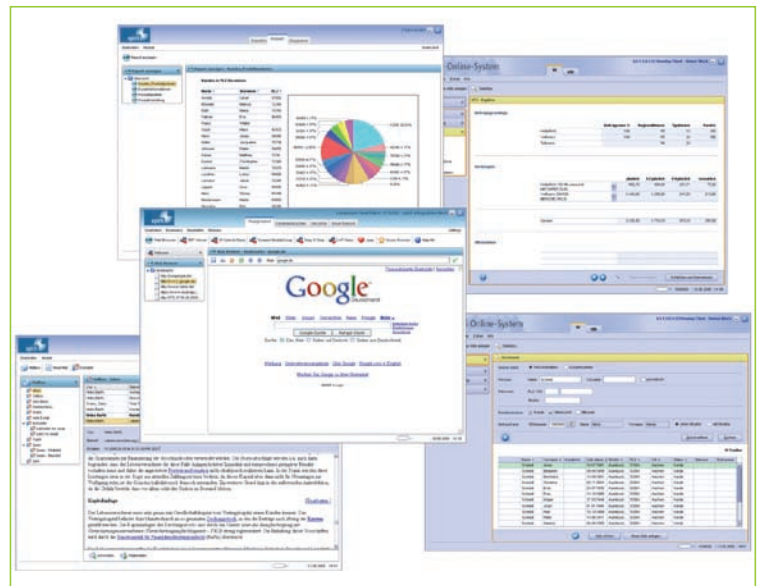
Für die gefühlte Leistung sind daher dessen Antwortzeiten entscheidend. Wenn der Benutzer weiß, dass ein größeres Dokument übertragen wird, akzeptiert er auch einige Sekunden Wartezeit. Kritisch sind dagegen die Latenzzeiten bei vielen kleinen Anfragen, wenn diese während der normalen Interaktion anfallen. Hier liegt eine Gefahr, denn das Konzept des Smart Client verleitet leicht zu einer solchen Vorgehensweise. Insbesondere wenn während der Entwicklung Client und Server auf demselben Rechner liegen und somit die Verbindungszeiten keine Rolle spielen. Daher ist es wichtig, so früh wie möglich Tests in einer produktionsnahen Umgebung durchzuführen.

Der Prozessor auf dem Client-Rechner ist selten ausgelastet und wartet hauptsächlich auf Eingaben des Benutzers. Auf der anderen Seite ist ein Serverprozessor, der Berechnungen für einen Benutzer ausführt, während dieser Zeit nicht für andere Benutzer verfügbar. Deshalb gehören rechenintensive Prozesse in den meisten Fällen auf den Client.

Sicherheit

Welche Art von Security wird bei Ihnen benötigt? Ein öffentliches Informationssystem kommt ohne spezielle Authentifizierung und Autorisierung aus. Ein System, das Außendienst-Mitarbeitern die Arbeit

Abb. 2: Screenshot eines mit spirit erstellten Smart Client, rechts ein Beispiel für einen integrierten Browser



mit internen Firmendaten ermöglicht, muss dabei die aktuellen Rechte des Benutzers beachten. Eine Online-Banking-Anwendung hat so hohe Anforderungen an die sichere Übermittlung der Daten, dass bestimmte Transaktionen explizit bestätigt werden müssen. Der Benutzer darf damit nur Zugriff auf sein Konto haben und nicht auf andere Konten oder auf interne Daten der Bank.

In den allermeisten Fällen werden die grundlegenden Bestandteile der Security, Authentifizierung und Autorisierung, auf einem Server implementiert sein. Auf dem Client sind hauptsächlich die Aufrufe der entsprechenden Services zu finden. Der Smart Client stellt in diesem Zusammenhang das funktionale Gerüst bereit, um zum einen die Übergabe von Berechtigungsnachweisen und zum anderen die serverseitige Anbindung einer anwendungsspezifischen Implementierung der Authentifizierung zu unterstützen. Darüber hinaus werden die Kommunikationsschnittstellen zum Session Handling und zum Bereich Autorisierung definiert und integriert.

Wenn Benutzer unterschiedliche Rechte haben können, sollte die Benutzungsschnittstelle diese Rechte widerspiegeln. Einfache Anpassungen, wie zusätzliche Menüpunkte, kann auch ein Fat Client umsetzen. Durch eine Versetzung oder Beförderung des Mitarbeiters kann er aber in neue Rollen kommen, aus denen andere Arbeitsszenarien und der Bedarf nach zusätzlichen Komponenten resul-

tieren. Die dynamische Anpassung an die aktuellen Rechte des Benutzers ist ein Gebiet, auf dem der Smart Client seine Stärken ausspielen kann. Veränderte Rechte werden bei der Anmeldung festgestellt und umgesetzt, neue Funktionalitäten werden automatisch vom Server nachgeladen und integriert.

Transaktionen

Eintypische Java EE-Anwendung verwendet Datenbank-Transaktionen auf dem Server. Dagegen wird ein Fat Client, der über den Server direkt auf eine Datenbank zugreift, sich auch selbst um die Transaktionen kümmern. Einem Smart Client stehen beide Möglichkeiten zur Verfügung.

Eine Smart-Client-Architektur ermöglicht zusätzlich noch eine echte, verteilte Verarbeitung von Objekten. Geschäftsobjekte können beliebig zwischen Clients und Servern transferiert und auf beiden Seiten verändert werden. Dazu werden Transaktionen auf Objekten benötigt. Eine solche Transaktionsklammer umfasst typischerweise einen geschlossenen Geschäftsvorfall und enthält alle durchgeführten Änderungen innerhalb einer definierten Menge von Geschäftsobjekten.

Dabei müssen alle Änderungen von Properties und Relationen der Geschäftsobjekte in der Objekttransaktion protokolliert werden. Diese Deltas können dann durch ein *commit* übernommen oder durch ein *rollback* wieder rückgängig gemacht werden. Neben der rein loka-

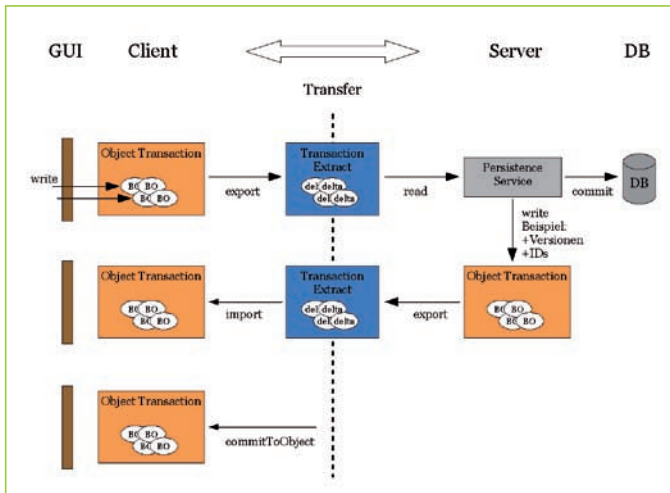


Abb. 3: Optimierter Datentransfer mit Objekttransaktionen

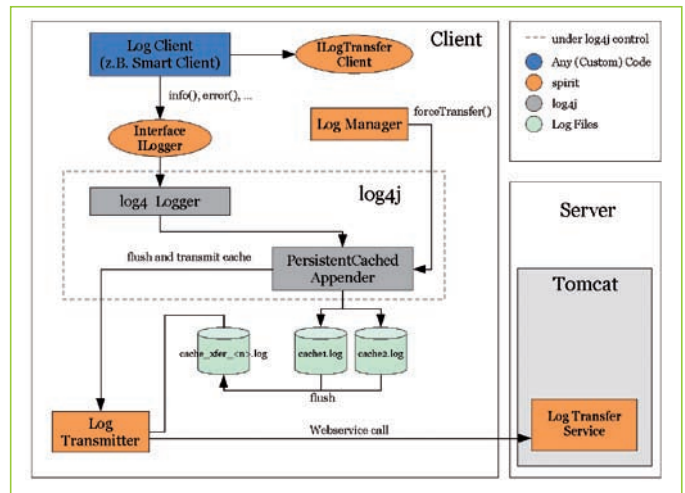


Abb. 4: Log-Transfer-Komponente des spirit Smart Client Framework

len Nutzung auf dem Client oder der Umsetzung auf Datenbank-Transaktionen kann die Objekttransaktion auch für die optimierte Übertragung von Änderungen in einer Menge von Objekten zwischen Client und Server benutzt werden.

Abbildung 3 zeigt ein Beispiel für einen solchen Transfer. Um das Datenvolumen auf ein Minimum zu reduzieren, werden die Änderungen aus einer Objekttransaktion in einen Extrakt exportiert. Nur dieser wird auf den Server übertragen und dort durch Zugriffe auf die Datenbank ergänzt. Die dadurch veränderten Objekte kommen auf dem gleichen Weg zurück auf den Client.

Preferences

Hier soll primär der Speicherort für die Preferences angesprochen werden. Welche Arten von Preferences gibt es denn? Gewöhnlich wird zwischen User-Preferences und System-Preferences unterschieden. Benutzerbezogene Einstellungen enthalten spezifische Informationen für einen einzelnen Benutzer: die Benutzerdaten selbst, die bevorzugte Gestaltung der Oberfläche, Spracheinstellungen und andere Optionen. Für jeden Benutzer eines Systems gibt es einen eigenen Satz. Wenn der Benutzer seine Einstellungen vorfinden soll, unabhängig davon, auf welchen Client-System er sich anmeldet, dann müssen diese Preferences unter seiner Kennung zentral gespeichert und nach der Anmeldung auf den Client übertragen werden. Lohnt sich dieser Aufwand wirklich, oder genügt nicht auch die lokale Speicherung?

Erfahrungsgemäß sitzen letztendlich die meisten Benutzer jeden Tag vor demselben Rechner, der nur alle paar Jahre gegen ein neueres Modell ausgetauscht wird.

Systembezogene Einstellungen enthalten Informationen über die aktuelle Umgebung. Pfade im Dateisystem, die Auflösung des Monitors und die verwendete Netzwerkverbindung machen genau auf diesem einen Rechner Sinn. Es gibt nur einen Satz pro Rechner. Und der gehört wohl kaum auf den Server. Spätestens bei dem URL des verwendeten Web Services ist die Notwendigkeit einer lokalen Speicherung offensichtlich.

Logging

Eine Anwendung soll nicht nur während der Entwicklung, sondern auch in der Produktion Logging-Informationen erfassen können. Entweder nur zur Fehlersuche oder zusätzlich zur Protokollierung, um das Nutzungsverhalten auszuwerten oder Bedienungsprobleme zu erkennen. Es ist schließlich schon von Interesse, ob Bestellvorgänge öfters bei der Eingabe der Kreditkartennummer oder nach der Anzeige der Versandkosten abgebrochen werden. Auf dem Server ist es einfach, das Logging dynamisch zu konfigurieren, zu sichern und auszuwerten. Auf dem Client ist das eigentlich auch einfach. Allerdings ist nicht der Benutzer an den Logs interessiert, sondern der zentrale Support.

Die am weitesten verbreitete Lösung bindet den Benutzer trotzdem in diesen Prozess ein. Nachdem er sich mit einem Pro-

blem an den Support gewandt hat, erhält er Anweisungen, wie er das Logging einschaltet, wo er das Resultat findet und wie er es übermittelt. Eine anschließende Synchronisierung mit dem dazugehörigen Server-Log muss manuell erfolgen. Dieses Verfahren ist unbequem und fehleranfällig.

Eine andere Lösungsmöglichkeit überlässt die Kontrolle und Steuerung dem Server. Unbedingt vorher zu klären sind hierbei die Fragen des Datenschutzes. Schließlich lässt sich so auch eine sehr weitgehende Überwachung des Benutzers realisieren.

Bei einer automatischen Lösung überträgt der Client die gesammelten Log-Einträge regelmäßig an einen Server. Jeden einzelnen Log sofort zu übertragen wäre dagegen keine gute Idee. Praktikabel und weniger belastend ist es, die Log-Einträge auf dem Client zwar zu sammeln, aber nur im Fehlerfall zu übertragen. Sonst werden sie periodisch einfach wieder gelöscht. Für das Sammeln, Synchronisieren und Auswerten der Logs empfiehlt sich ein eigener Server. Abbildung 4 stellt Ihnen eine Log-Transfer-Komponente vor.

Fehlerbehandlung

Die Verarbeitung von aufgetretenen Fehlern ist eng verwandt mit dem Logging. Clientseitige Fehler werden in der Regel lokal verarbeitet. Für die zentralen Web Services sind diese nicht von Interesse. Durch die Benutzeroberfläche muss eine geeignete Präsentation erfolgen, für den Support werden Log-Informationen erzeugt und möglicherweise auch automatisch übertragen.

Serverseitige Fehler sind für den Client dann interessant, wenn sie dessen Arbeit beeinflussen. Ob dabei besser eine auf dem Server aufgetretene native Exception oder eine verständlichere Nachricht übermittelt wird, hängt vom angestrebten Benutzerkreis ab.

Universelle Lösungen sind im Falle von Fehlern nötig, die nicht von den eigenen Services, sondern vom Webserver oder von der Transportschicht kommen. Ihre Behandlung wird dadurch erschwert, dass sie bei der Implementierung des Clients oft gar nicht bekannt sind. Für die Erkennung, Klassifizierung und Verarbeitung solcher Fehler bietet sich ein globaler Handler an.

Hilfe und Dokumentation

Es ist nicht verboten, sich von anderen, gut durchdachten Lösungen anregen zu lassen. Empfehlenswert ist zum Beispiel ein Blick auf das Hilfe- und Supportcenter von Windows. Hilfe und Dokumentation komplett auf dem Client, das klingt logisch und funktioniert auch. Andererseits wird der größte Teil davon erfahrungsgemäß nie angeschaut werden. Ein Smart Client soll doch online verteilt und aktualisiert werden. Macht es da nicht Sinn, wenn diese Informationen auf dem Server bleiben, bis sie gebraucht werden? Natürlich muss der grundlegende Teil, zum Beispiel die Anleitung zur Fehlerbehebung bei Netzwerkproblemen, immer lokal verfügbar bleiben. Ist in ihrem Anwendungsfall eine verteilte Dokumentation denkbar? Erst

wird die Information lokal gesucht und bei Misserfolg auf den Server weitergeleitet? Dann ist es kein allzu großer Schritt mehr bis zum Angebot, direkt eine Verbindung zum Supportcenter herzustellen.

Installation und Update

Ein grundlegendes Merkmal eines Smart Client ist die Möglichkeit, ihn online zu verteilen und zu aktualisieren. Eine automatische, zwangsweise Aktualisierung stellt sicher, dass jeder Benutzer immer nur mit der neuesten Version arbeitet. Wartung und Support sind erheblich einfacher, wenn ältere Versionen nicht unterstützt werden müssen. Zentrale Services müssen nur mit einer einzigen Version des Clients zusammenarbeiten.

In vielen Anwendungen muss die Datenbasis oder die Programmlogik in regelmäßigen Abständen aktualisiert werden, zum Beispiel bei jährlichen Anpassungen an geänderte Steuergesetze. Das beeinflusst die Frage, ob eine Komponente am besten auf den Client oder auf den Server gehört. Wo liegt die aktuelle Wetterkarte und wo die Kartenbasis eines Routenplaners? Zwei maßgebliche Faktoren sind also die voraussichtliche Häufigkeit einer Aktualisierung und die dabei zu übertragende Datenmenge.

Daten, die sehr schnell veralten und/oder nur sporadisch gebraucht werden, gehören nicht auf den Client. Aber auch Daten, die vielleicht nur einmal im Jahr

aktualisiert werden müssen, dann aber viele Megabyte Update bedeuten würden, sind auf dem Server besser aufgehoben. Um der Bedeutung der automatischen Verteilung und Aktualisierung als eine der Kernkomponenten eines Smart Client gerecht zu werden, wird sich ein eigener Artikel mit deren Technik befassen.

Fazit

Ein Smart Client macht die Entwicklung einer Anwendung nicht automatisch einfacher. Seine Stärke ist die Flexibilität, die mehr Möglichkeiten der Umsetzung erlaubt. Ob ein Service auf dem Client oder auf dem Server liegt, wird nicht durch die verwendete Technologie vorgegeben. Stattdessen liegt es in der Verantwortung des Entwicklungsteams, die für die gegebene Problemstellung jeweils am besten geeignete Lösung zu finden. Falls sich diese Lösung später doch nicht als optimal herausstellt oder sich die Rahmenbedingungen ändern, kann der Service leicht verlagert werden. Das verringert das Risiko und erleichtert den Einstieg in die Entwicklung mit einem Smart-Client-Framework.



Oliver Kleber arbeitet als IT-Consultant bei der compeople AG in Frankfurt. Seine Schwerpunkte sind Client-Server-Frameworks, agile Entwicklungsmethoden und Build-Management.

Links & Literatur

[1] spirit Smart Client Framework:
www.compeople.de/smartclient.html