

„RICH THIN CLIENTS“ FÜR WEB-ANWENDUNGEN

Wegen ihrer einfachen Verteilbarkeit werden Applikationen häufig mit Web-Benutzungsoberflächen ausgestattet. Das Konzept der Rich Thin Clients reichert diese recht unkomfortablen BenutzungsOberflächen mit einer größeren Interaktivität und Benutzerfreundlichkeit an. Es stellt somit einen interessanten architektonischen Ansatz für Applikationen dar, die reichhaltig vernetzte Informationsstrukturen verwalten.

Anwendungen für Call-Center stellen erhöhte Anforderungen an die Fähigkeiten und die Ergonomie der BenutzungsOberfläche. Innerhalb kurzer Zeit müssen die Kundeninformationen in übersichtlicher Form auf dem Bildschirm des Call-Center-Agenten erscheinen, damit die Kundenanfrage ohne lange Wartezeiten bearbeitet werden kann. Im Falle einer Versicherung muss sich der Call-Center-Agent beispielsweise innerhalb kurzer Zeit ein Bild über die unterschiedlichen Verträge des Kunden machen, sehen, welche Kontakte es gegeben hat und wie der Status der Kontakte ist. All diese Informationen stammen aus verschiedenen operativen Systemen und stellen ein reich verflochtenes Gebilde dar. Entsprechend den jeweiligen Bedürfnissen des Call-Center-Agenten müssen ihm diese Informationen in unterschiedlichen Sichten präsentiert werden. Der Wechsel zwischen diesen Sichten muss einfach und schnell sein.

Dieses Szenario einer Call-Center Anwendung ist typisch für viele aktuelle Problemstellungen. Ihnen allen ist gemeinsam, dass die Informationen aus verschiedenen operativen Systemen extrahiert und zu einer stark vernetzten Gesamtsicht zusammengefügt werden – sei es im Bereich *Client Relationship Management (CRM)*, wo das System unterschiedliche Kundeninformationen zu einer einheitlichen Sicht konsolidiert, oder aber im Bereich *Supply Chain Management (SCM)*, wo eine integrierte Sicht über die gesamte Logistikkette zur Verfügung gestellt werden soll. Ausschlaggebend ist, dass der Anwender effizient navigieren und immer wieder unterschiedliche Sichten arrangieren kann. Zu diesem Zweck müssen Informationen übersichtlich in Hierarchien oder Netzen dargestellt werden. Teilbereiche müssen bei Bedarf aus- und wieder eingeblendet werden. Unterschiedliche Aspekte müssen nebeneinander dargestellt oder die angezeigten Aspekte je nach Bedarf auf dem Bildschirm neu angeordnet werden.

Bei der Programmierung der BenutzungsOberflächen haben sich Web-Technologien als De-facto-Standard etabliert. Wenn es jedoch darum geht, komplexe Informationsgebilde darzustellen, hat dieser De-facto-Standard seine Defizite, da er eigentlich zur Erfassung von Informationen und zur Darstellung von Dokumenten gedacht war:

- Zur Darstellung von Informationen in einer baumartigen Struktur ist man auf den Einsatz von Java-Script mit seinen Browser-Abhängigkeiten angewiesen.
- Das Sortieren von Tabelleninhalten nach verschiedenen Kriterien bedingt immer eine zusätzliche Kommunikation mit dem Server. Dieses erneute Laden der Seite vom Server reduziert die Interaktivität und das Antwortzeitverhalten der BenutzungsOberfläche.
- Das Selektieren von einzelnen Zellen in einer Tabelle ist nur mit zusätzlichen Steuerungselementen möglich. Dieses Hilfskonstrukt mindert die Intuitivität der BenutzungsOberfläche und den Arbeitsfluss des Anwenders.
- Kontextmenüs und ähnlich mächtige Steuerungselemente fehlen.
- Navigieren durch große Informationsmengen ist – bedingt durch die blockweise Darstellung und die damit verbundenen Server-Roundtrips – recht zähflüssig und unübersichtlich.
- Tastenkombinationen (*Hot-Keys*) oder Tastaturkürzel (*Keyboard-Shortcuts*) stehen erst in neueren HTML-Versionen zur Verfügung.

Neben diesen grundlegenden Defiziten birgt die Programmierung der BenutzungsOberfläche mit Web-Technologien zusätzliche Komplexität, da hier verschiedene Technologien (HTML, JavaScript und „Cascading Style Sheets“) gleichzeitig zum Einsatz kommen – und jede dieser Technologien mit ihren eigenen Konzepten und Strukturen. Für den Entwickler der BenutzungsOberfläche bedeutet dies zusätz-

der autor



Thomas Neumann
(E-Mail: mail@thomasneumann.org)
arbeitet als Softwareentwickler und
-architekt in J2EE-Projekten.

lichen Aufwand, da er alle Technologien zu einem homogenen Ganzen kombinieren muss.

„Rich Thin Clients“: ein neuer Architekturanatz

Rich-Thin-Client-Technologien gehen dieses Problem an, indem sie auf der Client-Seite eine leichtgewichtige *UI-Engine* (Benutzungs-schnittstellen-Engine) einsetzen. Diese *UI-Engine* ist – im Unterschied zum Web-Browser – auf die Bedürfnisse von BenutzungsOberflächen dialogorientierter Anwendungen zugeschnitten. Sie ist verantwortlich für den Aufbau und die Steuerung der BenutzungsOberfläche. Zu diesem Zweck stellt die *UI-Engine* mächtige Steuerungselemente zur Verfügung, wie z.B. Tabellen, die große Informationsmengen visualisieren und sortieren können. Daneben verfügt die *UI-Engine* über eine konfigurierbare Ereignisbehandlung, die entscheidet, welche Ereignisse (*Events*) innerhalb der *UI-Engine* abgehandelt werden können und welche Ereignisse zur Steuerung der Anwendungslogik an den Server übertragen werden müssen. Darüber hinaus bietet die *UI-Engine* neben dem von den Web-Technologien bekannten *Pull*-Kommunikationsmodell zusätzliche Möglichkeiten, um z.B. eine *Push*-Kommunikation zwischen Server und Client zu realisieren.

Ähnlich wie bei einem Web-Browser erhält die *UI-Engine* die Beschreibung der Dialoge vom Server. Die *UI-Engine* bildet diese Beschreibungen auf die Technologie der Client-Plattform ab. Damit ermöglicht sie BenutzungsOberflächen mit einer größeren Interaktivität, die die technischen Möglichkeiten der Client-Plattform besser ausnutzen und so den erhöhten Anforderungen komplexer BenutzungsOberflächen besser gerecht werden.



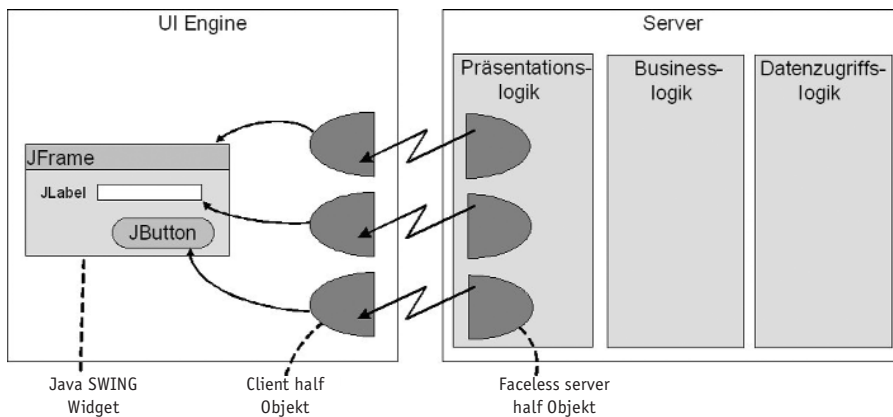


Abb. 1: Darstellung einer beispielhaften ULC-Anwendung: Die gesamte Anwendung einschließlich der Logik der Präsentationsschicht läuft auf dem Server. Die UI-Engine übernimmt den Aufbau der Benutzungsoberfläche und kommuniziert mit dem Server über das Half-Object-Muster.

Da sich die UI-Engine nur auf die Darstellung und Steuerung der Benutzungsoberfläche beschränkt, ist sie sehr leichtgewichtig. Basiert die UI-Engine auf Technologien, die im Web-Browser z. B. als Plug-In zur Verfügung stehen, lässt sie sich problemlos auch in eine HTML-Seite integrieren. Dadurch hat dieser Ansatz auch den Vorteil der einfachen und schnellen Softwareverteilung.

Somit gelingt *Rich-Thin-Client*-Technologien im Bezug auf dialogorientierte Benutzungsoberflächen die Quadratur des Kreises. Sie erweitern die auf Web-Technologien basierenden *Thin Clients* um die Elemente

- leistungsfähige Steuerungselemente,
 - konfigurierbare Ereignisbehandlung und
 - zusätzliche Kommunikationsmodelle.
- Damit gleichen sie die Nachteile herkömmlicher Web-Technologien im Bezug auf Interaktivität der Benutzungsoberflächen aus, behalten aber den Vorteil der einfachen Softwareverteilung bei.

Java als Plattform für einen „Rich Thin Client“

Was liegt bei der Realisierung dieses Ansatzes näher als die Verwendung von Java? Aufgrund der beiden grundlegenden Eigenschaften Plattform-Unabhängigkeit und Mobilität des Codes ist Java sehr gut dazu geeignet, Beschreibungen von Benutzungsoberflächen auf unterschiedliche Plattformen abzubilden. Das Produkt „UltraLightClient“ (ULC) von der Firma Canoo ist ein Vertreter dieser *Rich-Thin-Client*-Technologien, der genau diesen Ansatz verfolgt.

ULC ist ein in *Pure Java* geschriebenes Framework und basiert auf dem „Java SWING Widget-Set“. Das Framework ist in ein Client/Server-Modell aufgeteilt: Auf dem Server läuft die gesamte Anwendungslogik, inklusive der Präsentationslogik zur Steuerung der Abläufe und dem Aufbau der Benutzungsoberfläche (siehe Abb. 1). Die Client-Seite implementiert die leichtgewichtige UI-Engine, die entweder in einem *Applet* oder als eine Applikation laufen kann. Sie stellt die erweiterten Steuerungselemente, das konfigurierbare Ereignismodell sowie die zusätzlichen Kommunikationsmodelle zur Verfügung.

Die erweiterten Steuerungselemente bieten solche Features wie Kontextmenü, Tastaturkürzel und *Drag&Drop*-Funktionalität. Das konfigurierbare Ereignismodell arbeitet die Ereignisse so weit wie möglich ohne Interaktion mit dem Server selbstständig ab. Nur Ereignisse, die zur Steuerung der Anwendungslogik benötigt werden, leitet die UI-Engine an den Server weiter. Neben dem *Pull*-Kommunikationsmodell bietet die UI-Engine die Möglichkeit der *Push*-Kommunikation sowie das bedarfsgesteuerte Laden der Steuerungselement-Inhalte.

Kontrolliert wird diese UI-Engine vom Server-Teil des Frameworks. Die gesamte Anwendungslogik – inklusive aller Objekte zur Steuerung der Benutzungsoberfläche – läuft auf dem Server. Diese Objekte bieten eine zu Java SWING ähnliche Programmierschnittstelle, d. h. die Applikationslogik kreiert mit einem ähnlichen Programmiermodell wie in Java SWING auf dem Server beispielsweise ein Fensterobjekt. Das bewirkt, dass in der UI-Engine auf der Client-Seite das korrespondie-

rende Java-SWING-Fensterobjekt angelegt wird und auf der Benutzungsoberfläche erscheint. Alle auf dem Server erzeugten Objekte der Benutzungsoberfläche stellen *Proxies* für das zugehörige angezeigte SWING-Widget auf dem Client dar. Diese *Proxies* sind wiederum nach dem *Half-Object*-Entwurfsmuster (vgl. [Mes95]) aufgebaut.

Das *Half-Object*-Muster ist für die transparente Verteilung eines Objekts über mehrere Adressräume hinweg gedacht. Hierzu wird ein Objekt in mehrere Subobjekte aufgeteilt. Jedes dieser Subobjekte repräsentiert das Gesamtobjekt in seinem Adressraum und enthält die für den Adressraum relevanten Informationen und Methoden. Die einzelnen Subobjekte wiederum synchronisieren sich mittels Mechanismen der Interprozesskommunikation.

ULC wendet dieses Muster auf die *Proxies* der Java-SWING-Widgets an (siehe Abb. 1). Diese sind aufgeteilt in ein Objekt *faceless server half* und ein Objekt *client half*. Wie der Name schon sagt, residiert das *faceless server half* Objekt auf dem Server und stellt die nach außen sichtbare Programmierschnittstelle zur Verfügung, über die der Entwickler den Aufbau der Benutzungsoberfläche steuert. Der zweite Teil des *Proxies*, das *client half* Objekt, übernimmt die Darstellung der einzelnen grafischen Elemente auf der Client-Seite und verwendet hierzu das jeweils korrespondierende Java-SWING-Widget. Umgekehrt schickt es Informationen über Aktionen auf der Benutzungsoberfläche zurück an das *faceless server half* Objekt, die dann wiederum in Aufrufe der *Callback*-Routinen der Applikationslogik resultieren.

Konfigurierbares Ereignismodell

Damit die Interprozesskommunikation nicht zum Flaschenhals beim Antwortzeitverhalten wird, optimiert ULC das Protokoll zwischen UI-Engine und Server. So gibt die UI-Engine nur Ereignisse an den Server weiter, für die dort *Callbacks* registriert sind. Alle anderen Ereignisse behandelt die UI-Engine eigenständig. Validierungen von Inhalten erfolgen ebenfalls anhand von eigenen Datentypen innerhalb der UI-Engine. Eine weitere Optimierung der *Roundtrips* zum Server ergibt sich durch die Möglichkeit, Abhängigkeiten zwischen Steuerungselementen zu definieren. Die UI-Engine kann so in Abhängigkeit davon, ob z. B. ein Textfeld gefüllt ist oder nicht, eine Schaltfläche freigeben.

Zusätzliche Kommunikationsmodelle

Um die Wartezeit beim Laden größerer Datenmengen zu verkürzen, unterstützt die UI-Engine *Lazy Loading* für alle datenorientierten UI-Elemente, wie z. B. „Liste“ oder „Tabelle“. Die UI-Engine lädt nur die Objekte, die gerade zur Ansicht benötigt werden, und hält diese dann weiter in ihrem Cache vor. Blättert der Benutzer weiter, lädt die UI-Engine die nächsten anzuzeigenden Objekte vom Server.

Neben dem klassischen *Pull*-Kommunikationsmodell einer Web-Anwendung bietet ULC das so genannte *Push*-Kommunikationsmodell, bei dem der Server die Möglichkeit hat Aktivitäten auf der Benutzungsoberfläche anzustoßen. So kann der Server bei Veränderungen von Informationen die Benutzungsoberfläche über diese Änderung benachrichtigen, sodass deren Inhalt angepasst werden kann.

Zur Kommunikation zwischen der UI-Engine und dem Server verwendet ULC die Standardprotokolle „HTTP(S)“ oder „RMI over IIOP“. Insbesondere der Einsatz von HTTP(S) erlaubt auch den sicheren und einfachen Einsatz von ULC über eine Firewall hinweg in so genannten *untrusted Environments* (unkontrollierten Umgebungen).

Erweiterbare leistungsfähige Steuerungselemente

Sollten die standardmäßig von der UI-Engine zur Verfügung gestellten Steuerungselemente nicht ausreichen, so ist es möglich, die UI-Engine um eigene *Widgets* zu erweitern. Hierzu erstellt der Entwickler zuerst das Java-SWING-*Widget* oder verwendet ein bereits vorhandenes *Widget*. Im nächsten Schritt entwickelt er die beiden *half* Objekte: zuerst das *client half* Objekt, das von `com.ulcjava.base.client.UIComponent` abgeleitet wird und mit dem neuen SWING-*Widget* zusammenarbeitet. Im letzten Schritt erstellt der Entwickler das *faceless server half* Objekt, das die Programmierschnittstelle auf dem Server zur Verfügung stellt. Bei der Programmierung des neuen ULC-*Widgets* kann der Entwickler die Kommunikation zwischen den beiden *half* Objekten selber ausgestalten. Hierbei steht es ihm frei zu entscheiden, wann und in welchem Umfang die beiden *half* Objekte miteinander synchronisiert werden. Zur Realisierung der Kommunikation setzt er auf die entsprechenden Mechanismen des ULC Frameworks auf, welche die Besonderheiten der einzelnen Protokolle von ihm fernhalten.

Gute Integration in die Java-Plattform

Wie schon eingangs erwähnt, handelt es sich bei ULC um ein in Java geschriebenes Produkt, das sich nahtlos in die Java-Plattform einfügt. Es kann einfach in beliebigen *Servlet*- oder *EJB*-Containern verwendet werden. Getestet wurde es mit „BEA Weblogic“, „IBM Websphere Application Server“, „JBoss“ und „Tomcat“.

Durch die gute Integration in die Java-Plattform ergibt sich für ULC-basierte Applikationen eine große Durchgängigkeit in der Architektur. Oftmals kommen im Bereich der Benutzungsoberfläche im Web entweder Skriptsprachen („Flash“ oder „Java-Script“) oder hybride Ansätze („Java Server Pages“) zum Einsatz. Diese Mischtechniken haben Brüche in der Programmierung zur Folge. Die Zuständigkeiten der einzelnen Technologien müssen im Voraus geklärt werden. Diese Zuständigkeitsgrenzen sind dann fix und nur schwer oder mit großem Aufwand zu verschieben. Anders bei einer durchgängigen Java-Architektur: Hier lassen sich die Zuständigkeiten kontinuierlich verteilen und später auch verschieben. Neben der architektonischen Schönheit hat dies auch ganz pragmatische Vorteile. Die beteiligten Entwickler benötigen nur Erfahrungen in einer Technik und nicht in mehreren, was zum einen die Stellenbesetzung und Planung eines Projekts signifikant erleichtert, auf der anderen Seite aber auch bei der Wartung und Pflege sehr hilfreich ist, da es eine Reduktion der Komplexität bedeutet.

Vereinfachte Entwicklung von Benutzungsoberflächen

Neben der Komplexität durch die Verwendung unterschiedlicher Technologien gestaltet sich die Entwicklung einer Web-Oberfläche schwieriger, da auf jedem Entwicklungsrechner eine entsprechende Laufzeitumgebung eingerichtet werden muss. Im Rahmen der Entwicklungsarbeit muss dann jede Änderung erneut in dieser Laufzeitumgebung installiert werden. Auf diese Weise verlangsamten sich die Entwicklungszyklen und die Fehlersuche wird schwieriger.

Als Lösung für dieses Problem bietet ULC als Bestandteil des Produkts den so genannten *DevelopmentRunner*, der es dem Entwickler ermöglicht, sowohl den Server als auch die UI-Engine in einer virtuellen Java-Maschine laufen zu lassen. ULC-Applikationen können deshalb mit dem *DevelopmentRunner* ohne aufwändige J2EE-Installation in jeder Entwicklungsumgebung entwickelt werden. Auf

diese Weise benötigt der Entwickler nicht permanent eine komplette J2EE-Umgebung mit den entsprechenden aufwändigeren Deployment-Zyklen. Außerdem vereinfacht es den Einsatz eines Debuggers, da nicht mehr unbedingt ein *remote* Debugger benötigt wird.

Darüber kann der Entwickler mit dem *DevelopmentRunner* die Laufzeitumgebung seiner ULC-Anwendung entsprechend seinen realen Gegebenheiten simulieren. So ist es möglich, beim Starten des *DevelopmentRunner* Übertragungsraten für die Interprozesskommunikation anzugeben, Initialisierungsparameter zu definieren usw. Dies kann entweder programmatisch, über Kommandozeilen-Parameter oder über eine grafische Benutzungsoberfläche erfolgen.

Als Ergänzung zum ULC-Framework werden von der Firma Canoo noch zwei weitere Tools angeboten: „ULC Visual Editor“ und „ULC Load“. Bei ersterem handelt es sich um einen grafischen Editor, mit dem der Entwickler seine Benutzungsoberfläche mittels *Drag&Drop* zusammenstellen kann. Als Ergebnis seiner gestalterischen Arbeit erhält er den fertig generierten Java-Code. Der ULC Visual Editor basiert auf der Eclipse-Plattform. ULC Load ist ein Tool für Last- und Stresstests. Mit diesem Werkzeug kann der Entwickler beliebige Szenarien aufzeichnen und später in beliebiger Reihenfolge abspielen. Die Ergebnisse können dann z. B. nach MS Excel übernommen und ausgewertet werden.

Fazit

In der Vergangenheit hat sich gezeigt, dass die Möglichkeiten der Web-Technologie im Bereich Benutzungsoberflächen doch eher beschränkt sind. Gerade bei Anwendungen, die auf einem komplexen Informationsmodell operieren, ist es schwierig, mit den Mitteln der Web-Technologie eine Benutzungsoberfläche zu realisieren, die den Anwender gut bei seiner täglichen Arbeit unterstützt.

Rich-Thin-Client-Technologien bieten hier einen guten Ansatz zur Entwicklung von Anwendungen mit erhöhten Ansprüchen an die Benutzungsoberfläche. Diese neue Art von Architektur für dialogbasierte Anwendungen schlägt eine Brücke zwischen schnellem, einfachen Deployment und einer Benutzungsoberfläche, die die reichhaltigen Möglichkeiten der heute auf der Client-Seite zur Verfügung stehenden Technologien ausnutzt.

Als Erweiterung zu den bekannten Web-Technologien bieten *Rich-Thin-Client*-Technologien leistungsfähige Steuerungselemente, ein konfigurierbares Ereignismodell sowie zusätzliche Kom-



munikationsmodelle. Diese Erweiterungen ermöglichen die Entwicklung von Benutzungsoberflächen, die sich durch ein größeres Maß an Interaktivität und Intuitivität auszeichnen. Beides zusammen genommen ermöglicht es, Anwendungen zu entwickeln, die dem Benutzer „besser in der Hand liegen“.

ULC stellt dem Entwickler eine Implementierung dieses Ansatzes in Form eines leichtgewichtigen, gut zu integrierenden Frameworks zur Verfügung. Der Entwickler verwendet ein Java-SWING-ähnliches Programmiermodell auf dem Server zur Gestaltung der Benutzungsoberfläche auf dem Client. Da das Framework in *Pure Java* geschrieben ist und sich auf die vorhandenen J2EE-Standards abstützt, integriert es sich nahtlos in die J2EE-Plattform.

Mit diesen Eigenschaften ist ULC als ein Vertreter der *Rich-Thin-Client*-Technologien sehr gut geeignet zur Realisierung von Anwendungen, bei denen es besonders auf ein schnelles Antwortzeitverhalten der Benutzungsoberfläche ankommt, wie z.B. Call-Center-Applikationen, oder auch für Anwendungen, die mit stark vernetzten Informationen umgehen müssen, wie z.B. *Supply-Chain-Management*. Auch dort, wo große Informationsmengen auf der Benutzungsoberfläche handhabbar gemacht werden müssen, wie bei Applikationen aus dem Bereich Produktdaten-Management, hilft dieser Ansatz den Anwender bei seiner täglichen Arbeit besser zu unterstützen. Aufgrund der größeren Interaktivität der Benutzungsoberfläche ist dieser Ansatz auch geeignet für Applikationen zur Steuerung und

Überwachung von Abläufen, sei es im rein technischen Bereich, wie z. B. bei Produktionsprozessen, oder im kommerziellen Umfeld.

Mit ULC steht also eine Implementierung der *Rich-Thin-Client*-Technologie zur Verfügung, die den Ansatz einer Web-Architektur um Funktionalitäten ergänzt, die die Realisierung von benutzerfreundlicheren Oberflächen ermöglicht. ■

Literatur

[Mes95] G. Meszaros, Pattern: Half-Object + Protocol, in: J.O. Coplien, D.C. Schmidt, Pattern Languages of Program Design, Addison Wesley, 1995