Scout Html UI

Offline Manifest

12.08.2014

André Wegmüller

Claudio Guglielmo

Referenzierte Dokumente:

* Offline Challenges.pptx

Contents

[Ausgangslage 1](#_Toc395772132)

[Fazit 2](#_Toc395772133)

[Schwierigkeiten 3](#_Toc395772134)

[Generisches read-only 3](#_Toc395772135)

[Umsetzungsvarianten 4](#_Toc395772136)

[Offline Server 4](#_Toc395772137)

[JavaScript Scout-Client 4](#_Toc395772138)

[Daten Import/Export 5](#_Toc395772139)

[Notizfunktion (à la Post-It) 5](#_Toc395772140)

[Fazit 5](#_Toc395772141)

[Abgrenzung 6](#_Toc395772142)

[Proof of Concept 7](#_Toc395772143)

[Grundlage 7](#_Toc395772144)

[Ergänzen von browser-seitigem Verhalten 8](#_Toc395772145)

[Programmatische Erzeugung von Adapter 8](#_Toc395772146)

[Fazit 10](#_Toc395772147)

Ausgangslage  
Mit *Html UI* ist im Folgenden die UI Schicht von Scout gemeint, welche als Ersatz für die bestehende RAP Lösung funktionieren wird. Das *Html UI* ist zu etwa in gleichen Teilen in Java (serverseitig) und in JavaScript (browserseitig) implementiert. Die beiden Komponenten kommunizieren über ein JSON-Protokoll miteinander.

Das *Hmtl UI* und die Businesslogik der damit gebauten Applikation "lebt" zu 95% auf einem Applikations-Server der im Offline-Fall nicht zur Verfügung steht.

Im Fall vom *Html UI* schickt der Scout-Server dem Browser nur Informationen wie eine Tabelle, ein Form, ein Baum darzustellen ist und welche Server-Funktion bei einer UI-Interaktion aufgerufen werden soll. D.h. das JSON-Protokoll zwischen Server und Client (=Browser) überträgt keine Business-Daten sondern immer nur Daten über UI-Elemente, die zufälligerweise auch Business-Daten enthalten können. Die einzige Möglichkeit, die wir haben um eine Offline-Applikation zu realisieren, müsste auf Grundlage dieser Protokoll-Daten erfolgen.

Das client-seitige *Html UI* enthält keinerlei semantische Information zu Business-Objekten. Der Offline-Client weiss z.B. nicht, dass alle Forms die *Person Xyz* anzeigen die selbe Person betreffen. Ohne weitere Massnahmen würde das Form im Offline-Fall mehrfach an den Client übertragen, wenn es von verschiedenen Orten aufgerufen wird.

Für eine vollwertige Offline-Applikation müsste die bestehende Scout-"Konfiguration" um etliche semantische Informationen und für Verhalten im On-/Offline Modus, evtl. sogar um JavaScript-Code ergänzt werden, die dann vom Browser interpretiert werden. Uns ist auch nicht klar, wie genau die Anforderung "Generisches read-only mit Ausnahmen" umgesetzt werden sollte (z.B. ein bestimmtes Feld in einem Form soll sich im Offline-Fall anders verhalten).

Mit vertretbarem Aufwand wäre einzig und allein eine *reine* read-only Ansicht denkbar (siehe Kapitel Generisches read-only). Auch eine Funktion beliebigen, unvalidierten Text in ein textfeldartiges Widget (Stringfield, Smartfield, etc.) tippen zu können, wäre denkbar aber nur begrenzt sinnvoll, weil:

Kunden werden Abstriche nicht in Kauf nehmen werden, nur weil unsere Lösung Web-basiert ist. Mein Versicherungsvertreter kann auf seinem Laptop ohne Probleme einen neuen Vertrag eröffnen und anspruchsvolle, komplexe Vorgänge über das UI seiner Applikation abwickeln (die vermutlich ein Fat-Client ist, ich habe nicht gefragt). Solchen Kunden eine schlechtere Lösung zu verkaufen ist m.M. schwierig bzw. sollte gar nicht erst von uns vorgeschlagen werden. Diese Lösung würde auch nur einfachste Hello-World Anwendungen abdecken können. Typische Anwendungen wie „wenn Checkbox X checked ist, muss Feld B mandatory werden“ könnten damit aber schon nicht mehr umgesetzt werden.

Ich glaube auch nicht dass der Kunde unsere "einfache" Lösung akzeptiert, die wir v.a. darum vorschlagen, weil wir technisch nichts besseres machen können. Der Kunde will eine *gute* Lösung, die Technik dahinter sollte ihn nur am Rande interessieren müssen.

Die hier beschriebenen Einschränkungen sind konzeptioneller Natur und haben wenig mit HTML, Browser oder dem neuen GUI zu tun. Wir versuchen jetzt konzeptionell Probleme zu lösen, die auch mit dem Fat-Client (Swing / Rayo) nicht einfach zu lösen wären z.B. Daten offline auf dem Client speichern und ohne Scout-Server zu verarbeiten. Erschwerend kommt hinzu, dass das der Scout-Client im Fall vom *Html UI* ebenfalls auf dem Server "lebt" und somit auch Dinge, die für den Swing-Client einfach sind, wie Feld-Validierungen oder beliebigen konfigurierten Java-Code ausführen, etc. im Browser nicht mehr funktionieren.

## Fazit

Mit dem *Html UI* alleine können keine anspruchsvollen Offline-Apps gebaut werden. Das in der Theorie einfache Konzept "read-only mit Ausnahmen" funktioniert nicht, bzw. lässt sich nur mit erheblichem Aufwand und komplizierten Konzepten lösen. Aus unserer Sicht ist das eine Sackgasse. Die Einschränkungen sind erheblich, Wünsche von Kunden könnten nicht flexibel umgesetzt werden.

Schwierigkeiten  
**Probleme mit Release-Wechsel** mit der auch die Zeitmaschine von BSI CRM zu kämpfen hat und die für *Html UI* ebenfalls gelöst werden müssten. Diese können auftreten wenn ein Client mit altem Release offline Daten erfasst  und dann wieder online auf einen Scout-Server mit höherer Version versucht Daten zu synchronisieren. In diesem Fall wären die lokal gespeicherten Daten nicht mehr kompatibel zum zentralen Server. Es müssten dann auch Konzepte für Daten-Migration her oder der Sync könnte nicht durchgeführt werden.  
  
Der **begrenzte Local Storage** der Web-Applikationen zur Verfügung steht (5 MB). Ein Versicherungsvertreter möchte vielleicht auch Dokumente die in der Datenbank gespeichert sind zum Kunden "mitnehmen". Diese würden den Local Storage sehr schnell füllen.

## Generisches read-only

Hier zeigen wir auf, was mit einem generischen read-only Ansatz umsetzbar ist, bzw. was die Limitierungen sind. „Generisch“ meint hier: der Scout-Programmierer muss wenig unternehmen bzw. programmieren, um eine Aktenkoffer-Funktion für eine oder mehrere Entitäten zu bekommen.

**Umsetzbar / funktionsfähig:**

* Baum, Tabelle, 360-Grad Sicht
* Browser-seitige Charts und Map (Table-Controls)
* Browser-seitige Tabellen-Sortierung / Filterung

**Limitierungen:**

* *Nur* Baum, Tabelle, 360-Grad Sicht
* Alle Felder im UI sind disabled (= read-only)
* Keine MessageBoxes oder weiterführenden Dialoge
* Keine klickbaren Buttons, Links und Menüs
* Keine Smartfields
* Keine Wizards
* Keine Dokumente
* Keine Behandlung von speziellen UI-Events innerhalb vom Form (z.B. klickbare Adress-Tabelle im Personen-Dialog)
* Keine Ausnahmen

Umsetzungsvarianten

Wie kann „Offline“ mit dem neuen Html UI trotzdem umgesetzt werden? Die folgenden Ideen zeigen, welche Möglichkeiten wir für eine vollwertige Offline-Lösung haben. Annahme: wir implementieren das Aktenkoffer-Feature als Offline-Applikation (gemäss Offline Challenges.pptx)

Offline Server  
Es wird eine eigenständige Offline-App mit Scout entwickelt. Diese kann natürlich Code mit der Online App sharen (z.B. Forms). Wichtiger Unterschied: der Client hat auch eine Scout-Server Komponente, diese kann in derselben VM-Instanz wie der Client laufen und könnte sogar eine embedded Datenbank ebenfalls in derselben VM haben (erprobt in BSI POS) - ist also nicht "schwergewichtig". Damit lassen sich Offline-Applikationen ohne irgendwelche Einschränkungen bauen.   
      
**Varianten:** a.) via http und localhost mit *Html UI*  🡪 benötigt .exe und Browser  
 b.) Rich Client (Swing/Rayo)  🡪 nur ein .exe  
  
Für den Einsatz auf Laptops wäre das eine vernünftige und erprobte Lösung. Für Smartphones   
und Tablets muss eine Alternative her. Je nach Anwendungsfall (Beratung in Filiale) wäre hier aber denkbar, dass der Server via WLAN erreicht werden kann. Für Kundenbesuche in einem Offline-Tal könnte ebenfalls ein Kleincomputer den Server mit WLAN zur Verfügung stellen und das Tablet/Smartphone darauf connecten. Wir können auch prüfen, ob der Scout-Server auf einem Smartphone gestartet werden kann.

Aufwand: L (technisch erprobt)

JavaScript Scout-Client  
Die Offline Anwendung ist eine reine JavaScript Applikation, welche die JavaScript-Library inkl. den UI-Elementen vom *Html UI* verwendet (die gleichen wie der Online Scout Html UI Server). Daten müssen der JavaScript App in einem zu definierenden Format zur Verfügung gestellt werden. Validierungen, die heute für diverse Felder in Java implementiert wurden müssten ebenfalls neu in JavaScript implementiert werden (aufwändig aber wiederverwendbar).  
  
Die Offline-App muss ohne Unterstützung von Scout / SDK geschrieben werden. Da Offline-Apps aber meistens eine sehr spezifischen Funktion haben, ist der Aufwand dafür überschaubar.

Grosser Aufwand fällt auch im Bereich Daten-Import / Schnittstellen an. Die JavaScript-Applikation muss Daten der Online-Applikation in einer bestimmten Form verarbeiten. BSI CRM arbeitet normalerweise immer auf Basis von FormData. Für die JavaScript Applikation wäre zu überlegen, ob dieses Format Sinn macht, oder ob ggf. ein Tabellen-artiges Format besser geeignet wäre.

Evtl. wäre es möglich via Scout / Html UI einen Export von einem konfigurierten Form zu machen und diesen als Basis für die JavaScript Applikation zu verwenden. Der Aufwand ein bestehendes Form-Layout in JavaScript nachzuimplementieren würde damit entfallen.

Aufwand: XXL (technisches Neuland)

## Daten Import/Export

Der *Aktenkoffer* in BSI CRM 15 ist ein reines Online-Tool. Hat aber die Möglichkeit Daten in einem definierten Format für eine Offline-Anwendung zu exportieren. Dieses könnte von der Offline-JavaScript oder Offline-Server Anwendung gelesen und geändert werden. Der Aktenkoffer hätte eine Sync-Funktion um die Daten mit dem Online-System zu synchronisieren und ggf. Konflikte zu behandeln.

Aufwand: L

Notizfunktion (à la Post-It)  
Als Ergänzung zum einfachen, generischen read-only Offline. Siehe referenziertes pptx.

Forms könnten mit Notiz-Zettelchen (ohne jegliche Semantik) "beklebt" werden. Die Notizzettel sind eine reine browser-seitige und offline-fähige App die sich mit dem *Html UI* integriert. Als Notiz könnte z.B. ein Versicherungsvertreter schreiben "Neue Adresse für Kunde XYZ erfassen". Wenn er wieder online ist kann er die Notiz manuell mit copy & paste ins Online UI übernehmen.

Aufwand: S  
  
Änderungskontrolle (à la Word)  
Als Ergänzung zum einfachen, generischen read-only Offline. Siehe referenziertes pptx.  
  
Eingabefelder auf einem Form könnten mit einer Änderung markiert werden (vorher/nachher). Diese Änderung wird rein browser-seitig gespeichert. Wenn man wieder online ist, kann man den Inhalt von der Änderungskontrolle ins richtige Feld übertragen.

Diese beiden Konzepte helfen leider nicht, wenn man komplett neue Daten strukturiert erfassen will (z.B. eine neue Versicherungs-Police).

Aufwand: S

Fazit  
Eine gute Lösung lässt sich nur mit *JavaScript Scout-Client* oder *Offline Server* erreichen. Offline Server hat für uns den Vorteil, dass wir die volle Unterstützung vom Scout SDK haben und Code sharen und wiederverwenden können. Evtl. könnte mit einer lokalen, embedded Datenbank und Hibernate sogar die Datenzugriffsschicht wiederverwendet werden. Offline JavaScript hat den Vorteil, dass wir damit auch Offline auf Smartphones und Tablets funktionieren können.

Abgrenzung  
Das Thema **Synchronisierung** sollten wir beim *Html UI* ausklammeren. Das hat nichts mit UI zu tun, sondern muss gesondert behandelt werden. Wir können uns im Rahmen vom Projekt *Html UI* allenfalls um die Optik / Ergonomie von einem solchen Feature kümmern. Die technische Umsetzung hat in der Vergangenheit schon oft hohen Aufwand verursacht und eine einfache, generische, technische Lösung zu finden, sollte nicht Teil von diesem Projekt sein.

Das Thema **Verbindungs-Unterbruch** hat nicht direkt etwas mit den hier beschriebenen Offline-Szenarien zu tun. Ziel ist ein robustes Verhalten der Applikation bei kurzzeitigen Verbindungsunterbrüchen (z.B. eine Fahrt durch einen Tunnel bei mobiler Verbindung). Die Applikation soll bedienbar bleiben. UI-Elemente werden read-only, ausser vielleicht Textfelder, damit der Benutzer weiter tippen kann. Baum, Tabelle, TableControls sollen weiterhin bedienbar sein. Benutzer sieht, dass zur Zeit keine Verbindung hergestellt ist. Die Seite versucht automatisch, die Verbindung wieder herzustellen.

# Proof of Concept

Eine Offline-App mit *Html UI* kann grundsätzlich auf zwei Arten realisiert werden (Details siehe oben):

1. Browser-seitige JavaScript Applikation
2. Lokaler Scout-Server (http)

Im Folgenden untersuchen wir v.a. die browser-seitige Lösung mit JavaScript. Als POC wurden zwei Anwendungen implementiert:

1. Ergänzen von einem bestehenden, konfigurierten Form-Feld mit rein browser-seitigen Verhalten.
2. Vollständig, programmatisches Erzeugen eines Forms, ohne Konfiguration.

## Grundlage

Eine Scout-Applikation hat immer einen Desktop und typischerweise eine Outline mit Tree, Table und Forms. Um einfach eine Startansicht im Offline-Modus zu erhalten kann die initiale JSON-Response hart-kodiert abgespeichert und im Offline-Fall geladen werden. Damit erhält man ohne viel Programmieraufwand eine gute Startgrundlage. Im Offline-Fall kann die normale (online) Session.js durch eine OfflineSession.js ersetzt werden, welche nicht den Server kontaktiert, sondern statisch oder lokal erzeugte JSON Responses zurückgibt bzw. die entsprechenden JavaScript-Objekte.

Main.js:

**var** jsonSessionId = '1:1407850659764';

**var** session = **new** scout.OfflineSession($(**this**), jsonSessionId, userAgent);

OfflineSession.js:

scout.OfflineSession.prototype.\_sendRequest = **function**(request) {

**this**.\_requestsPendingCounter++;

**var** data = **this**.\_offlineAjax(request);

**this**.\_processSuccessResponse(data);

};

scout.OfflineSession.prototype.\_offlineAjax = **function**(request) {

**if** (request.startup) {

**return** JSON.parse('{"events":[{"id":"1: … }}}');

} **else** **if** (!**this**.modelAdapterRegistry[455]) {

**return** JSON.parse('{"events":[{"id":"39","…:{}}');

} **else** {

**return** {events:{}, adapterData:{}};

}

};

### 

## Ergänzen von browser-seitigem Verhalten

Diese Anwendung ist nicht nur im Offline-Fall nützlich, sondern kann auch verwendet werden, um bestimmte Felder im Online-Fall mit einem client-seitig ausgeführten Verhalten zu erweitern. Z.B. eine spezielle Feldvalidierung, für die der Server nicht kontaktiert werden soll. Die Grundidee ist, sich bei der Adapter-Erzeugung einzuklinken.

scout.OfflineSession.prototype.getOrCreateModelAdapter = **function**(id, parent) {

**…**

adapter = **this**.objectFactory.create(adapterData);

**this**.\_setOfflineEventHandlers(adapter);

…

**return** adapter;

};

scout.OfflineSession.prototype.\_setOfflineEventHandlers = **function**(adapter) {

**var** id = adapter.id;

**if** (id == "46") {

adapter.setEventHandler(**function**(adapter, eventType, eventData) {

alert(‘hello world!’);

});

}

};

Die ID ist allerdings ungeeignet zur eindeutigen Identifikation eines Adapters. Besser wäre es die eindeutige ClassID zu verwenden oder die Namen-Hierarchie z.B. FormName.GroupBoxName.FormFieldName. Die Referenz via Name wäre auch gut geeignet, um zur Laufzeit Adapter innerhalb eines Forms zu referenzieren.

Fazit: auf diese Weise lässt sich recht einfach Verhalten *ergänzen*. Praktisch ist, dass der grundlegende Aufbau von Forms (Struktur, Layout, etc.) nicht von Hand erstellt werden muss.

## Programmatische Erzeugung von Adapter

Hier sollen ohne gespeicherte JSON-Daten on-the-fly Adapter erzeugt werden. Einstiegs-Punkt ist ebenfalls ein Custom-EventHandler wie er oben beschrieben wird. Auch wenn die Adapter programmatisch erzeugt werden, müssen die notwendigsten Properties übergeben werden. Hier fällt auf, dass z.Z. viele Properties übergeben werden müssen, die eigentlich default sein müssten (z.B. visible oder enabled = true). Hier gibt es Optimierungspotential. Diese Properties müsste der Server auch gar nicht mehr schicken. Problematisch ist auch noch die Reihenfolge der Initialisierung: wenn ein Adapter versucht beim Aufruf von init Child-Adapter anzulegen, erwartet er, dass diese bereits im adapterDataCache vorhanden sind. Darum müssen z.Z. Adapter die in der Hierarchie zuunterst stehen, zuerst initialisiert werden. Auch die parent/child Verlinkung muss noch manuell vorgenommen werden, da dies z.Z. nur in getOrCreateAdapter gemacht wird. Das sollten wir ändern, so dass jeder Adapter das selber richtig macht, beim Erzeugen der Child-Adapter. Beispiel-Code:

adapter.setEventHandler(**function**(adapter, eventType, eventData) {

**var** session = adapter.session;

**var** desktop = session.desktop;

**var** form = **new** scout.Form();

**var** okButton = **new** scout.Button();

okButton.init({

id:'1003',

objectType:'Button',

label:'Ok',

systemType:3,

enabled:**true**,

visible:**true**

}, session);

okButton.execSave = **function**() {};

okButton.setEventHandler(**function**(adapter, eventType, eventData) {

**var** vorname = adapter.parent.findAdapter('1004').

findAdapter('1005').getText();

alert('Danke und auf Wiedersehen ' + vorname + '!');

desktop.removeForm2(form);

});

**var** stringField = **new** scout.StringField();

stringField.init({

id:'1005',

objectType:'StringField',

enabled:**true**,

visible:**true**,

label:'Vorname',

labelVisible:**true**,

gridData:{x:0,y:0}

}, session);

**var** groupBox = **new** scout.GroupBox();

groupBox.init({

id:'1004',

objectType:'GroupBox',

formFields:[stringField.id],

enabled:**true**,

visible:**true**,

gridData:{x:0,y:0}

}, session);

**var** rootGroupBox = **new** scout.GroupBox();

rootGroupBox.init({

id:'1002',

objectType:'GroupBox',

formFields:[okButton.id, groupBox.id]

}, session);

form.init({

id:'1001',

displayHint:'dialog',

rootGroupBox:rootGroupBox.id,

objectType:'Form'

}, session);

rootGroupBox.parent = form;

rootGroupBox.addChild(groupBox);

rootGroupBox.addChild(okButton);

okButton.parent = rootGroupBox;

groupBox.parent = rootGroupBox;

groupBox.addChild(stringField);

stringField.parent = groupBox;

desktop.addForm2(form); });

## Fazit

Die *Html UI* JavaScript-Library ist recht gut geeignet, um programmatisch verwendet zu werden. Grundsätzlich fehlt aber jegliche Unterstützung durch ein Framework/API. Der Programmierer muss alle Events selber handeln. Wünschenswert wäre ein fest vorgegebener Ablauf wie im Java AbstractForm. Auch vor-definierte Feld-Validierungen und Handler (exec\*) fehlen. Schlussendlich müsste ein grosser Teil vom Java Scout Client in JavaScript nachprogrammiert werden. Zum jetzigen Zeitpunkt ist das aber unmöglich und würde auch den Rahmen vom *Html UI* Projekt sprengen. Es macht eher Sinn, dieses Vorhaben mit dem ersten konkreten Usecase umzusetzen, sofern wir uns überhaupt dafür entscheiden einen JavaScript Offline-Client zu bauen (anstatt einen lokalen Offline-Server zu verwenden).

Umsetzen sollten wir hingegen jetzt schon die default-Properties, da diese auch verwendet werden können, um den Datenverkehr zwischen Client und Server zu reduzieren.