Mobile Application Framework auf der Baustelle

Marcus Hammer
virtual7 GmbH
Karlsruhe

Schlüsselworte
Mobile Application Framework, REST, A-Team Mobile Persistence Accellerator

Einleitung

Die Firmengruppe Max Bögl möchte auf der Baustelle den Bauleitern die Möglichkeit geben, die Bauteile und deren Status möglichst einfach zurück zu melden. Die Rückmeldung für Planung, Produktion und Fertigstellung von Bauteilen sollen tagesaktuell bereitgestellt werden.

In Kombination mit Barcodes und einer kompletten Offline-Funktionalität stellte dies verschiedene Herausforderungen.

Die Projektziele:

- Vereinfachung der Rückmeldung auf der Baustelle
- Reduzierung von Mehrfacheingaben
- Nutzung der Statusdaten zu projektbezogenen jedoch anonymen Analysezwecken
- Tagesaktueller Fertigstellungsstatus im Modell
- Tagesaktueller Produktionsstatus im Modell
- Erhöhung Nutzung mobiler Endgeräte auf der Baustelle
- Abfragen von bauteilbezogenen Daten
- Zukünftige Erweiterung der bauteilbezogenen Status mit Fertigteilabruf von der Baustelle

Zentrale Statusdatenbank Bereitstellung

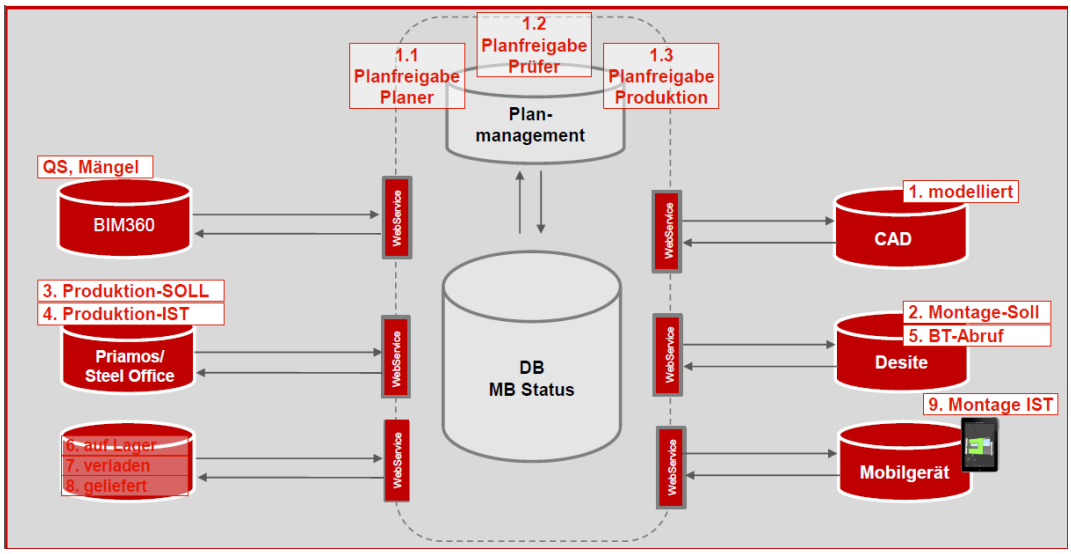

Abbildung 0.1 - Zentrale Statusdatenbank Bereitstellung

Eine zentrale Status-Datenbank bei Max Bögl stellt den Dreh- und Angelpunkt der Daten dar. Hier wird über verschiedene Planfreigabeebenen der Aufbau der Baustelle geplant. Mittels CAD wird die Planung in einem 3D Modell modelliert und als Plan- und Ist Status zugehörig zu Punkten in diesem 3D Modell gespeichert. Die Kommunikation der verschiedenen Systeme mit der Datenbank geschieht ausschließlich über Webservices.

Mobile Datenerfassung auf der Baustelle

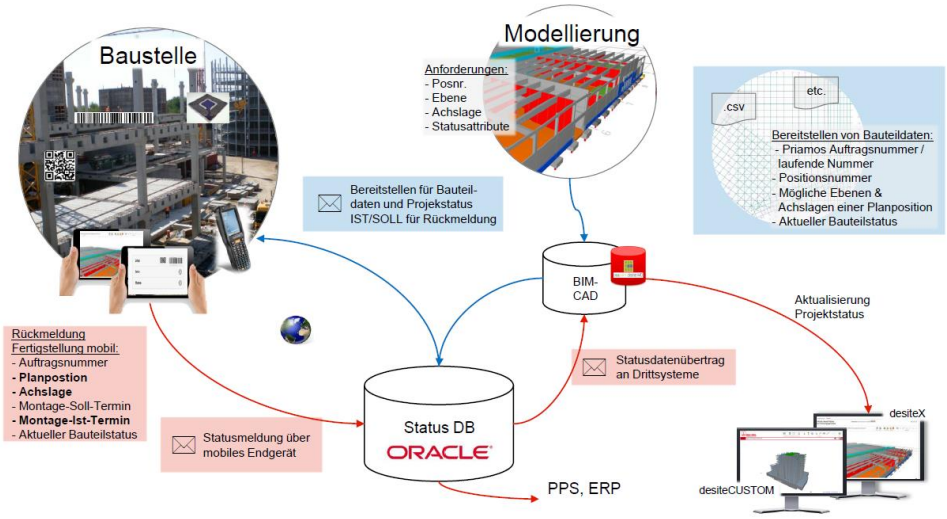

Abbildung 0.1 - Mobile Datenerfassung auf der Baustelle

Der Webservice wird als REST mit JSON als Inhalt übertragen. Am mobilen Gerät sollen für bestimmte Projekte gezielt alle Bauteile übertragen werden, um eine komplette Offline-Funktionalität zu ermöglichen. Dies soll dem Baustellenleiter auch in schwierigen Gebieten die Rückmeldung ermöglichen.

Die Herausforderung besteht darin, dass es sich hierbei um bis zu 20.000 Bauteile pro Projekt handelt und ein Benutzer der App mehrere Projekte haben kann!

Als Backend-Layer haben wir uns für das AMPA (A-Team Mobile Persistence Accellerator) Framework entschieden, welches als Add-On für MAF von Oracle zur Verfügung gestellt wird. Dies kapselt bereits die offline Modifikation von Daten und kann diese auf ein Ereignis hin bei Online-Erreichbarkeit zurück schreiben.

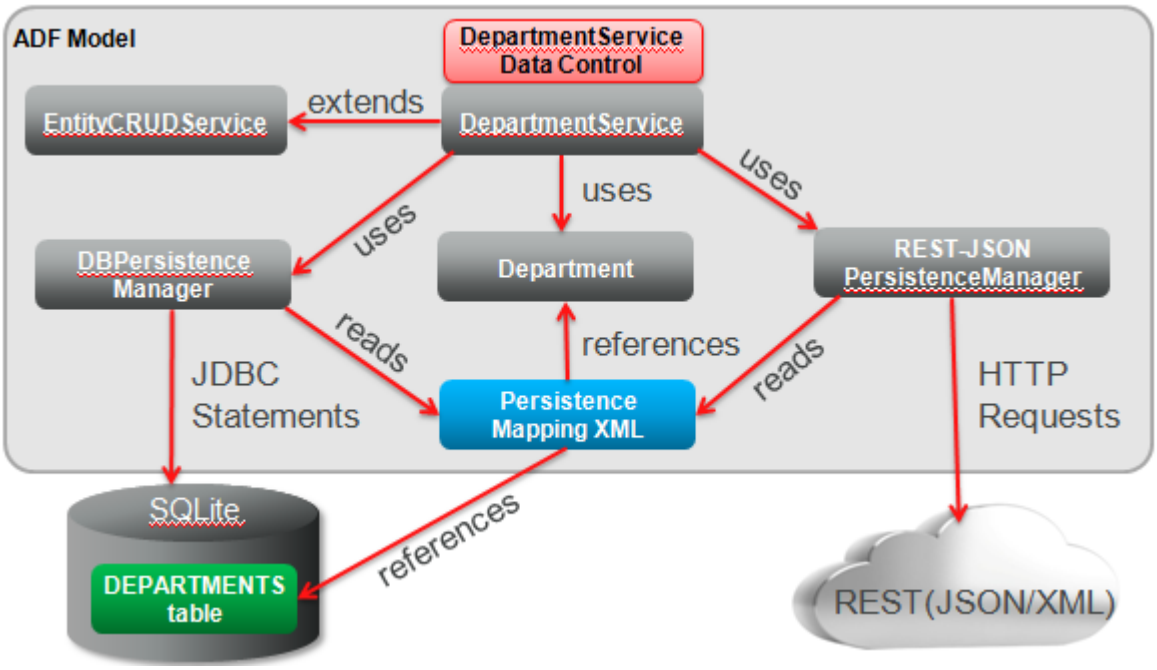

Abbildung 0.2 - A-Team Mobile Persistence Accelerator

Folgende Herausforderungen mit AMPA mussten gelöst werden:

1. Benutzerspezifische Ablage der Daten, so dass ein Benutzer die lokalen Daten des anderen Benutzers am selben Device nicht modifiziert oder löscht.

2. Der Abgleich der Daten gegenüber des Webservice darf die lokalen Informationen in einer Tabelle nicht leeren, da es zu viele Daten sind, um sich immer alles neu abzuholen → ein Delta-Load musste realisiert werden, welches abhängig von einem Zeitstempel lediglich die veränderten Daten Synchronisiert

3. Wir haben Parent-Child Beziehungen in den Daten, welche als Sub-JSON-Arrays eines JSON-Objekts transportiert werden. Damit AMPA dies korrekt interpretieren kann, mussten Anpassungen vorgenommen werden.

4. AMPA versucht die empfangenen Daten gegen Listen abzugleichen und ist sehr allgemein gehalten. Im Schnitt ist das zwar nett für die Aufbereitung der Daten aber schlecht für die Performanz. Wir mussten diverse Validierungen und Abgleiche eliminieren, um die Daten schneller über AMPA hinweg lesen zu können.

5. Das Laden der Daten vom Webservice in die SQLite DB und das Laden der Daten von der SQLite DB in den Memory sollte komplett im Hintergrund passieren, so dass

die App nicht blockiert. Da dies u.U. bedeutet, dass diverse Bindings in der Oberfläche noch keine Daten haben, musste dies überall speziell beachtet werden. Das Laden selbst wurde mit Threads realisiert.

Bauteilerfassung

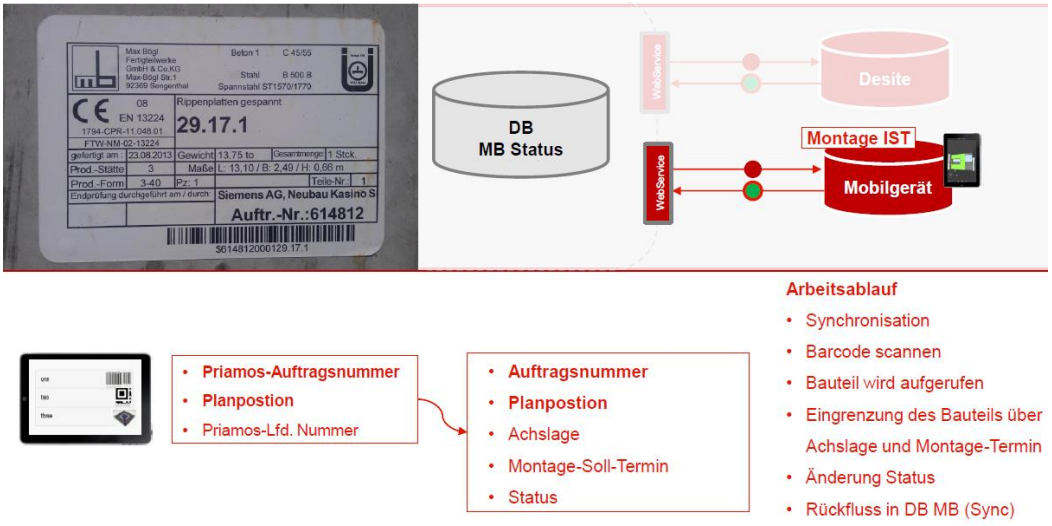

Abbildung 0.1 – Bauteilerfassung

Im Frontend war es die Anforderung, dass die Bauteile manuell gesucht, parallel aber auch mittels Barcode Scanner erkannt und daraufhin mit diesem interagiert werden kann. Die Barcodes sind je nach Hersteller aber verschiedener Natur, so dass die Bauteile entweder direkt ein Muster für das korrekte Auseinandernehmen des Barcodes mitbringen oder allgemeine Muster über den Webservice pro Projekt bereitgestellt werden. Anhand dieser wurde dann mit regulären Ausdrücken der Barcode getrennt und mit den daraus extrahierten Informationen nach dem dazugehörigen Bauteil gesucht.

Die Herausforderung an dieser Stelle bestand im Barcode Scanner. Für Cordova gibt es nicht sehr viele Barcode-Scanner. Der Platzhirsch ist von PhoneGap und kann zwar gut QR Codes lesen, scheiterte aber an den sehr langen und dünnen Barcodes von Max Bögl.

Nach einer längeren Untersuchung des nativen Quellcodes für Android und iOS haben wir zwei entscheidende Settings ausfindig gemacht:

- Qualität des aufgenommenen Bildes (Default auf Mittel)
- Größendefinition des ausgeschnittenen Scanner-Bereichs (bei hohen Auflösungen zu klein, da Angaben in Pixel)

Nach Optimierung dieser Parameter und Neukompilierung des nativen Codes, konnte final auch MAF die Barcodes scannen und sofort auswerten. Ein offenes Problem bleibt aber leider noch. Dieser Barcode Scanner unterstützt den Landscape-Mode nicht korrekt. Wird er ermöglicht und im Porträt-Mode gestartet, so bleibt das Bild des Scanners schwarz. Dies war keine Option und es war leider nicht ausreichend Zeit, um sich im nativen Code des Problems zu widmen.

Erfahrungen mit MAF

Das Mobile Application Framework selbst eignet sich gut für die Entwicklung mobiler Anwendungen, sofern man die App entsprechend der Funktionalitäten von MAF strukturiert. Wenn man Feature-Basiert denkt und die Features möglichst nicht übergreifende Informationen benötigen, ist alles gut.

Leider wird beim Wechsel von einem Feature auf ein anderes der Garbage Collector aktiv, so dass unter Umständen der Cache des verlassenen Features geleert wird.

Das Burger-Symbol des Spring-Boards ist per Default in iOS links orientiert in der oben Bar. Warum es in Android auf der rechten Seite platziert ist bleibt ein Rätsel. Man möchte doch nicht das Springboard links über einen Button rechts oben öffnen?

An verschiedenen Stellen ist man gezwungen eigene JavaScript Listener an die UI Komponenten zu hängen. Wir haben z.B. eine Autosuggest Funktionalität auf diesem Weg realisiert. Leider wird bei Aktualisieren der AMX Seite jeder Custom-JavaScript-Listener aus der Seite wieder entfernt, so dass man sich wieder explizit darum kümmern muss sie wieder hinzuzufügen.

Fazit

Neben den speziellen Herausforderungen, der enormen Datenmenge und der kompletten Offline-Funktionalität, konnte in diesem Projekt MAF seine Stärken zeigen. Die Möglichkeit über Cordova Plugins leicht bestimmte Hardware-Komponenten anzusprechen und die ADF-ähnliche Entwicklung ermöglichen es in kürzester Zeit eine hybride App bereit zu stellen. Alle Probleme die uns hier begegnet sind wären auch in einer nativen Entwicklung auf die ein oder andere Weise aufgetreten, wobei man dann alles für mindestens zwei verschiedene Welten (Android und iOS) entwickeln und lösen müsste.

Kontaktadresse:
Marcus Hammer
virtual7 GmbH
Zeppelinstraße, 2
D-76185 Karlsruhe

Telefon:	+49 (0) 721-619 017 50
Fax:	+49 (0) 721-619 017 29
E-Mail	hammer@virtual7.de
Internet:	www.virtual7.de