Webbasierte Anwendungen – Architektur und Umsetzung Einsatz modernster Technologien für Datenhaltung und Präsentation nach dem Model-View-Control Architekturmuster am Beispiel „CLAAS Telematics“

André Kluge

Agrocom GmbH & Co. Agrarsystem KG
Potsdamer Strasse 211
33719 Bielefeld
kluge@agrocom.com

Abstract: Bei „CLAAS Telematics“ fiel die Wahl für die Architektur des Systems auf die Java Technologie. Um die Entwicklung der Java Webanwendung zu beschleunigen, standardisierte Prozesse und Standards wie JavaServlets, JavaBeans und XML zu nutzen, kommt ein Open Source Web-Framework zum Einsatz. Eine weitere Anforderung an die Architektur des Systems ist die Unabhängigkeit von einem SQL-Dialekt bzw. einem Datenbanksystem. Diese Anforderung wird durch einen Persistenz-Layer erfüllt.

1 Problem der optimalen Potenzialausschöpfung von Hochleistungsmähdreschern

Betriebsleiter berichten während der Getreideernte von signifikanten Leistungsunterschieden und erheblichem Aufwand bei der optimalen Führung von 2 oder mehr Mähdreschern. Die Fahrerprofile zeigen beträchtliche Unterschiede im Schulungsgrad der einzelnen Fahrer, in Erfahrung im Umgang mit der Maschine oder Nutzung der Fahrhilfen. Des Weiteren ist der Mähdrescher die Leitressource und sein Ausfall führt zum Stillstand der gesamten aufnehmenden Logistikkette.

All diese Probleme führen dazu, dass das enorme Leistungspotenzial eines Hochleistungsmähdreschers nicht optimal ausgenutzt werden kann.

2 Anforderungen an das System

Aus den genannten Problemen ergaben sich die Anforderungen an ein System, das die Betriebsleiter während der Erntekampagne mit aktuellen Leistungsdaten seiner Maschinen versorgt um sicherzustellen, dass die Mähdrescher optimal ausgenutzt werden. Aber auch die Historiendaten seiner Maschinen sollten im System gespeichert werden für weitere Analysen.
Eine zentrale Datenhaltung und Nutzung von Standard-Webbrowsern ist eine weitere Anforderung an das System, um Installationszeit und –kosten zu sparen, Standardtechnologien zu nutzen und einen zentralisierten Zugriff auf die Daten zu erhalten.
Ferner sollte es in der Lage sein, die heutigen Anforderungen an mobile Telekommunikation durch standardisierte Lösungen abzudecken und für zukünftige Anforderungen und Technologien offen zu sein.

Dem Ziel ein offenes, erweiterbares und anpassbares System zu erhalten, wurde durch eine offene und skalierbare Systemarchitektur, sowie einem dynamischen und erweiterbaren Datenmodell Rechnung getragen.

3. Systemüberblick

Für Claas Telematics fiel die Wahl auf die J2EE Technologie[1] [2]. Gründe dafür waren Hersteller- und somit auch Plattformunabhängigkeit zu sein, um auf zukünftige Anforderungen freier reagieren zu können. Lizenzkosteneinsparungen, da man Komponenten und Tools aus der Open Source Welt nutzen konnte und vor allem der Verbreitungsgrad der Java Technologie war ausschlaggebend für diese Entscheidung.

3.1 Systemarchitektur von Claas Telematics

Um der Anforderung nach einer skalierbaren Architektur genüge zutragen, wurde sich für eine 3-Tier Architektur[3] entschieden. Man trennt dabei 3 Schichten des Systems logisch voneinander, die Präsentationsschicht (client tier), die Logikschicht (application-server tier) und die Datenschicht (data-server tier).

Die Präsentationsschicht ist für die Darstellung der Daten und die Benutzerschnittstelle verantwortlich, die Logikschicht kapselt die Geschäftslogik und Anwendungssteuerung und die Datenschicht ist verantwortlich für das Laden und Speichern der Daten.

[1] Internetquelle: http://java.sun.com/javaee/
[2] Internetquelle: http://de.wikipedia.org/wiki/J2EE
[3] Internetquelle: http://de.wikipedia.org/wiki/Three-Tier-Architektur

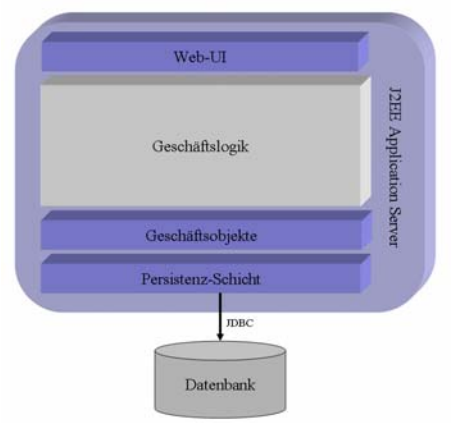

Abbildung 1 - Architektur Claas Telematics

Der Vorteil einer solchen Architektur liegt darin, dass die 3 Schichten physikalisch voneinander getrennt sein können und somit eine Lastverteilung stattfindet. Ferner bietet sie Vorteile bei der Wartung und Erweiterung des Systems, da z.B. eine Schicht ausgetauscht werden kann, ohne Änderungen an den anderen Schichten vornehmen zu müssen. Der Aufbau nach einer 3-Tier oder bei sehr großen Systemen auch n-Tier Architektur ist typisch für eine Webanwendung. Viel Entscheidender für die Architektur des Systems ist der Aufbau der Anwendung innerhalb des Applikations-Servers selbst.

3.2 Web-Oberfläche

Für die Web-Oberfläche hat sich das MVC Model 2 bewährt. Das MVC Model 2 ist eine spezialisierte Variante für Webanwendungen des Model-View-Control Architekturmusters, erstmals 1979 von Trygve Reenskaug[4] beschrieben, das bei GUI Oberflächen quasi Standard ist.
Durch dieses Model erhält man ein flexibles Programmdesign um Änderungen oder Erweiterungen einfach durchzuführen und es ermöglicht die Wiederverwendung von einzelnen Komponenten.

Das MVC Architekturmuster besteht aus 3 Komponenten.
Das Model enthält die darzustellenden Daten und den aktuellen Status des Systems dargestellt durch die Geschäftsobjekte und der Geschäftslogik.
Die Präsentation ist für die Darstellung der Daten aus dem Model zuständig. Die Darstellung in Web-Anwendungen erfolgt meistens durch HTML oder XML-Seiten, die durch Java Server Pages (JSP) erzeugt werden.
Die Steuerung verarbeitet die durch den Benutzer ausgelösten Requests des Web-Browsers und ruft die entsprechende Geschäftslogik auf. Diese Schicht enthält die Intelligenz und steuert den Ablauf der Präsentation.

[4] Internetquelle: http://de.wikipedia.org/wiki/MVC

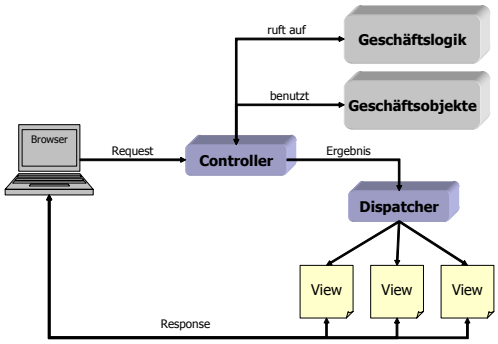

Abbildung 2 - schematische Darstellung des MVC Model 2

Für die Umsetzung des MVC Model 2 hat sich für Claas Telematics das Webframework Jakarta Struts[5] bewährt.

3.3 Geschäftslogik

Für reine Web-Anwendungen kann die Geschäftslogik in einfache Java-Klassen ausgelagert werden, die über klar definierte Schnittstellen aus der Oberfläche heraus genutzt werden. Hier muss darauf geachtet werden, dass die Schnittstellen zur Geschäftslogik klar von der Oberfläche getrennt werden, um die Schichtentrennung sicherzustellen.

3.4 Geschäftsobjekte

Die Geschäftsobjekte stellen das Datenmodell der Anwendung dar. Sie bilden den Grundstein der Anwendung und sind damit die Grundlage aller weiteren Schichten. Änderungen an den Geschäftsobjekten, die Attribute oder Methoden-Signaturen betreffen, haben fast immer direkte Auswirkungen auf alle darüberliegenden Schichten und sind daher zu vermeiden.

3.5 Persistenz-Schicht

Die Persistenz-Schicht trägt die Verantwortung die Geschäftsobjekte zu persistieren, also in nicht-flüchtige Speichermedien zu speichern. Die Geschäftsobjekte sind durch einfache Java-Objekte (oft als *Plain-Old Java Objects – POJO* bezeichnet) realisiert, die direkt in eine Datenbank durch einen entsprechenden Persistenz-Mechanismus gespeichert werden. Klassische Anbieter solcher Persistenz-Tools sind beispielsweise Oracles TopLink und Hibernate[6].

---

[5] Internetquelle: http://struts.apache.org/
[6] Internetquelle: http://www.hibernate.org/