FirstSpirit™
Your Content Integration Platform

UX-Bridge Whitepaper

Version	**1.2**
State	**RELEASED**
Date	**2012-08-13**
Department	Professional Services
Author/ Authors	Andreas Knoor
Copyright	2012 e-Spirit AG
File name	UX-Bridge Whitepaper_DE

e-Spirit AG

Barcelonaweg 14
44269 Dortmund | Germany

T +49 231 . 477 77-0
F +49 231 . 477 77-499

info@e-Spirit.com
www.e-Spirit.com

e-Spirit AG

Inhaltsverzeichnis

1 Einführung und Vision

1.1 Websites und User Experience

Die Anforderungen an moderne Webauftritte sind in den letzten Jahren rasant gestiegen. Diese Dynamik bezieht sich neben der Optimierung der redaktionellen Prozesse zunehmend auf die Funktionalitäten des Webauftritts, welcher immer mehr als interaktives Kommunikationsmedium mit Kunden oder Interessenten wahrgenommen wird.

Die Bedürfnisse bzgl. der Steuerung dieser Kundenkommunikation werden zunehmend anspruchsvoller, weil die Erwartungshaltung auf Seiten des Konsumenten auf einem sehr hohen Niveau liegt. Der anspruchsvolle Kunde erwartet eine hervorragende „User-Experience“, wo dieser Begriff eine Vielzahl von Aspekten wie gute Usability, zielgruppenspezifische Ansprache sowie die Auslieferung von Informationen zum richtigen Zeitpunkt in der richtigen Granularität umfasst. Auch die Performance einer Website bekommt einen immer höheren Stellenwert, sowohl bei der Benutzerakzeptanz als auch bei Aspekten wie dem Suchmaschinen-Ranking.

1.2 Hybrid-Architektur

Um den steigenden Anforderungen gerecht zu werden, ist eine adäquate technische Lösung auf Seiten des Content Management Systems (CMS) notwendig. Das breite Spektrum der User-Experience Anforderungen macht eine ungewöhnlich flexible Architektur des CMS unerlässlich. Dies gilt besonders für Aspekte, die klassische gegensätzliche Ziele darstellen, wie z.B.

- Hohe Dynamik der Inhalte versus performante Auslieferung der Website
- Stabile Systemarchitektur versus Integration von vielen verschiedenen Werkzeugen

Um trotz dieser Trade-Offs eine für den jeweiligen Kunden optimale Lösung bieten zu können, stellt FirstSpirit eine besonders anpassbare Architektur zur Verfügung, die sogenannte „Hybride Architektur“.

Die hybride Architektur von FirstSpirit macht es gerade in integrationslastigen Szenarien sehr leicht, eine passgenaue Lösung zu etablieren, die den spezifischen

Eigenschaften und Anforderungen des konkreten Webauftritts Rechnung trägt. Eine „One-Size-Fits-All“ Architektur funktioniert erfahrungsgemäß nur in sehr simplen Anwendungsfällen. Mit zunehmender Anzahl und Komplexität von Webauftritten steigt jedoch die Vielfalt der zu integrierenden Drittsysteme. Gerade in größeren Unternehmen ist dies ein häufig anzutreffendes Merkmal, da neben der CMS-Komponente noch weitere Systeme wie E-Commerce-Shops, Enterprise Portale aber auch selbst entwickelte Webanwendungen vorgegeben und einzubinden sind. In solchen Fälle lässt sich durch die Hybrid-Architektur von FirstSpirit eine feine und vor allen Dingen projektindividuelle Abstimmung zwischen Performance, Stabilität, und Wartbarkeit realisieren.

FirstSpirit folgt mit seiner hybriden Architektur der Regel:

> „So viel Dynamik wie nötig, so viel Inhalte vorgenerieren wie möglich.“

Unter einem hybriden CMS verstehen wir somit ein Content Management System welches jedem Kunden die Wahlfreiheit lässt, welche Elemente einer Website voll dynamisch und welche Elemente bereits fertig vorgeneriert ausgeliefert werden.

Technisch bedeuten die beiden Varianten:

- Voll-dynamisch: Die redaktionellen Inhalte werden zum Zeitpunkt des Benutzerzugriffs auf eine Webseite (Request-Zeitpunkt) live aus einem oder mehreren Content-Repositories ausgelesen und zusammengestellt.

- Vorgeneriert: Die redaktionellen Inhalte sind bereits physikalisch in einer Webseite (z.B. einer HTML-Datei) enthalten und können direkt an den User ausgeliefert werden.

Eine Besonderheit des Hybrid-Ansatzes von FirstSpirit liegt darin, dass die Entscheidung für die eine oder andere Variante sehr feingranular für jedes Element einer einzelnen Seite festgelegt werden kann. Konkrete Beispiele für diese Art der Architektur werden in Kapitel 2.2 erläutert.

1.3 UX-Bridge: Connecting Content and User-Experience

Mit dem Trend zu immer interaktiveren Websites, kommt dem Thema „dynamischer Zugriff auf redaktionelle Inhalte“ eine stetig steigende Bedeutung zu. Immer dort, wo eine Vorgenerierung von Inhalten nicht möglich ist, muss dynamisch auf die CMS-Inhalte zugegriffen werden. Dynamisch bedeutet hier, dass sich der Inhalt potentiell für jeden Website-User und zu jedem Zeitpunkt ändern kann. Jede Information die länger als 1-2 Minuten auf der Website steht und für alle Benutzer identisch ist (z.B. ein Artikel bei einem News-Portal), gilt hier *nicht* als dynamisch, sondern als

potentiell vorgenerierbar. Nichtsdestotrotz nehmen die dynamischen Szenarien zu, da eine personalisierte Content-Auslieferung einen wichtigen Beitrag zur User-Experience liefern kann.

FirstSpirit bietet mit dem UX-Bridge-Modul eine Infrastruktur für die Anforderung einer dynamischen Content-Delivery-Plattform. Somit ergänzt das Modul den hybriden Ansatz um eine Standardinfrastruktur für dynamische Content-Auslieferung. Als Faustformel für die beiden hybriden Architekturvarianten lässt sich festhalten:

- Vorgenerierte Content-Auslieferung eines redaktionellen Inhaltes:
  → FirstSpirit Standard Deployment

- Voll dynamische Content-Auslieferung eines redaktionellen Inhaltes
  → UX-Bridge

Wichtig ist zu beachten, dass diese Entscheidung nicht global für den ganzen Webauftritt, sondern jeweils einzeln für jede Art von Content-Element getroffen werden kann (siehe Beispiele in Kapitel 2.2).

Beim Einsatz der UX-Bridge ist somit pro Projekt und pro (feingranularen) Anwendungsfall zu entscheiden, ob die Dynamisierung fachlich und technisch notwendig bzw. sinnvoll ist (siehe oben: Trade-Off Dynamik versus Performance). Auf Basis dieser Entscheidung lassen sich dann sehr leicht intelligente und Anwendungsfall-optimierte Lösungen realisieren.

1.4 Kundenbindung durch Interaktion

Um eine echte Interaktion mit den Benutzern einer Website zu ermöglichen, reicht eine reine Content-Auslieferung, selbst wenn sie personalisiert und kontextbezogen erfolgt, nicht aus. Vielmehr existiert eine Vielzahl Anwendungsfällen in denen Informationen von der Website zurück Richtung Content-Erstellung / CMS fließen müssen. Beispiele für Anwendungsfälle eines solchen Rückkanals sind User-generated Content (UGC) oder auch statistische Daten bzgl. des Benutzerverhaltens, die in einem zweiten Schritt wieder zur Optimierung der Content-Auslieferung verwendet werden können.

Mit Hilfe der UX-Bridge können zum einen UGC-Inhalte über einen Standardweg abgelegt und wieder ausgelesen werden. Gleichzeitig lässt sich die UX-Bridge nutzen um bestimmte Informationen von der Website (z.B. wie viele Kommentare hat mein Artikel schon bekommen) direkt ins Redaktionssystem zurückfließen zu lassen, um sie dort passend weiter zu verarbeiten, z.B. für Ranking-Listen.

1.5 Mobile First

Die Verwendung von mobilen Endgeräten zur Darstellung von (Web-)Content nimmt seit einigen Jahren exponentiell zu. Neben der „normalen“ Website ist es inzwischen fast schon eine Standardanforderung auch einen mobilen Ausgabekanal für Smartphones oder Table-PCs zu unterstützen. Der „Mobile First“ Ansatz geht sogar noch einen Schritt weiter und empfiehlt die mobile Variante einer Website noch vor der konventionellen Variante zu entwickeln. Die Gründe liegen darin, dass man sich im mobilen Kanal deutlich mehr auf die wesentlichen Punkte, d.h. den für den Kunden „relevanten“ Content, beschränken bzw. fokussieren muss. Weiterhin gibt es auf mobilen Geräten neben einigen technischen Einschränkungen (z.B. Bildschirmgröße) auch erweiterte Eigenschaften wie z.B. Standorterfassung per GPS, Gestensteuerung, wechselnde Seitenverhältnisse bei Hoch- und Querformat oder eine Offline-Fähigkeit. Diese müssen schon bei der initialen Konzeption einer Website mit berücksichtigt werden, da sie ohne eine komplette Neukonzeption nur schwer nachträglich hinzufügen sind.

Durch die strikte Trennung von Inhalt und Layout erlaubt FirstSpirit sehr leicht eine Erzeugung von Mobile-konformen Inhaltsfragmenten, z.B. als XML oder HTML-Fragment.

Die UX-Bridge ergänzt diesen Ansatz um eine flexible Content-Delivery-Infrastruktur, die die Anforderungen an mobil darzustellenden Content optimal unterstützt. Ein konkretes Beispielszenario ist in Kapitel 2.2.6 beschrieben.

1.6 Neue Anforderungen: Big Data, NoSQL, Realtime Analytics

Moderne Webauftritte stellen immer höhere Anforderungen an die zugrundeliegende technische Infrastruktur.

Die zu verwaltenden Datenmengen nehmen sprunghaft zu, weil neben den klassischen redaktionellen Inhalten auch zunehmend User-generated Content (UGC) sowie statistische Daten aus dem Verhalten der Website-Benutzer verwaltet und ausgewertet werden müssen. Die Speicherung und Verwaltung solch großer Datenmengen wird auch mit dem Begriff „Big Data“ assoziiert.

Mit der steigenden Menge von Daten geht gleichzeitig auch eine Diversifizierung der Datentypen einher. Stark redaktionelle getriebene Webseiten bestehen häufig aus unstrukturierten Daten, während Website-Daten aus Backendsystemen (z.B. Produktinformationen) eher stark strukturiert sind. User-generated Content wie Kommentare oder Bewertungen sind meist sehr einfach und flach strukturiert. Eine weitere Art von Benutzerdaten spiegeln Abhängigkeiten oder Beziehungen wider

(wer hat sich was angesehen, wer kennt wen, etc.). Gerade in Online-Communities ist dieser „Social-Graph“ eine wichtige Informationsquelle für Benutzerinteraktionen. Ein weiterer Anwendungsfall für Beziehungen zwischen Content-Elementen ist das sogenannte „Semantische Web“. Dort werden durch die explizite Modellierung von Verbindungen die Inhalte zueinander in Relation gesetzt, z.B. für bessere Suchmöglichkeiten.
Aus diesem Grund werden in Zukunft neue Arten von Datenablagen eine immer größere Rolle spielen, z.B. NoSQL oder Graphdatenbanken.

Neben der Speicherung von redaktionellen und benutzerbezogenen Daten kommt der Auswertung dieser Daten ein immer höherer Stellenwert zu. Bei der Datenanalyse geht der Trend zunehmend in Richtung Realtime-Reporting. Häufig werden auch Methoden aus dem Datawarehousing und der Business Intelligence verwendet, um zu relevanten Analyse zu kommen.

Beim Entwurf einer zukunftsfähigen Architektur für eine Website sind die erwähnten Randbedingungen zu berücksichtigen:

- Ständig steigenden Datenmenge
- Verwaltung sehr unterschiedlicher Datentypen
- Steigende Anforderungen bei der Auswertung von Daten (Reporting)

Bemerkenswert ist, dass die klassischen Content Management Repositories für diese umfangreichen Anforderungen überhaupt nicht entworfen wurden. Aus den Trade-Offs der Anforderungen lässt sich ableiten, dass ein „One-Size-Fits-All“-Ansatz für die Persistenz der Daten auf Dauer nicht tragfähig ist (siehe Kapitel 1.2).

Die FirstSpirit UX-Bridge folgt daher einem neuen Ansatz: Die UX-Bridge Architektur erlaubt eine flexible Auswahl der Datenablage (Persistenz) für den dynamischen Zugriff, abhängig von den fachlichen und technischen Anforderungen (siehe Kapitel 2.1.3.2).

1.7 Cloud-Ready Solution

Die UX-Bridge wurde von Anfang an so entworfen, dass ein Einsatz der Komponente in der Cloud problemlos möglich ist. Gerade Aspekte wie

- Skalierung und Performance
- Ausfallsicherheit / Failover

- Betriebskosten und Total-Cost-Of-Ownership (TCO)

sind gute Indikatoren für den Betrieb der UX-Bridge in einer Cloud-Infrastruktur wie z.B. Amazon. Gleiches gilt analog für den Betrieb der kompletten Website und natürlich auch für das FirstSpirit Redaktionssystem. Wie bei allen Architekturvarianten ist auch dies eine *optionale* Möglichkeit, die in der jeweiligen Kunden- und Projektsituation auf Eignung geprüft werden muss.

1.8 Content-Integration-Plattform

Neben der dynamischen Auslieferung von redaktionellen Inhalten in Richtung Website ist die Über- bzw. Weitergabe von Content in Richtung Drittapplikationen ein wichtiges Einsatzszenario für die UX-Bridge. Beispiele für diesen Anwendungsfall sind:

- Übergabe von Content an selbst entwickelte Webanwendung (z.B. Produktkonfigurator oder Filialsuche)
- Übergabe von Content an Cloud/SaaS Anwendungen, die in die Website mit eingebunden werden sollen und auf/mit redaktionellen Inhalten arbeiten (z.B. externe Recommendation-Engine oder eine Suchmaschine)

Während das FirstSpirit-Redaktionssystem als Content-Integration-Plattform für die Backend-Seite fungiert, nimmt die UX-Bridge diese Rolle auf der Live-Website ein.

2 Architekturbeschreibung

Die UX-Bridge stellt eine Erweiterung der klassischen vorgenerierenden FirstSpirit Architektur dar. In Abgrenzung zu einem „Volldynamischen CMS“ geht man bei FirstSpirit davon aus, dass der Content im Redaktionssystem (Backend) zusammengestellt und im Rahmen einer sogenannten „Generierung“ (häufig) auch schon mit dem passenden Ziellayout versehen wird. Das Ergebnis sind dann z.B. fertige HTML-Seiten. Anschließend erfolgt ein Übertragen (Deployment) der erzeugten Dateien Richtung Livesystem, z.B. ein Web- oder Application-Server. FirstSpirit dient somit als Content-Integration-Layer für redaktionellen Inhalte und unterschiedliche angebundene Backendsysteme (siehe Abbildung 1).

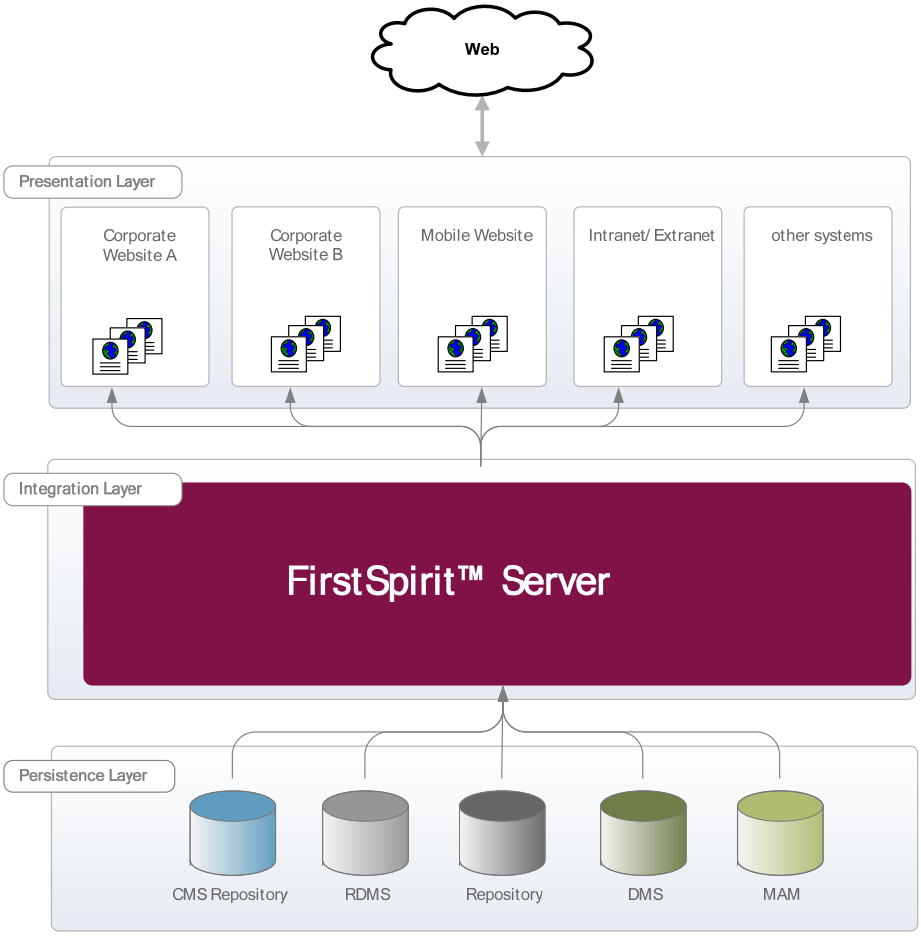

Abbildung 1: FirstSpirit Architektur

2.1 Architekturübersicht UX-Bridge

Im Rahmen der Hybrid-Architektur ergänzt die UX-Bridge FirstSpirit um eine dynamische Content-Auslieferung mit Standardmittel. Andere (UX-Bridge freie) Alternativen zur dynamischen Content-Auslieferung sind übrigens ausdrücklich möglich. Jedoch sollte zunächst geprüft werden ob ein standardnaher Ansatz wie die UX-Bridge nicht mehr Vorteile bietet.

In der UX-Bridge Architektur werden diejenigen Content-Elemente, die dynamisch über eine Webapplikation ausgelesen werden sollen, nicht wie üblich als fertige (HTML-)Dateien auf den Webserver gelegt. Stattdessen gibt es ein sogenanntes „Live-Repository“, welches die relevanten Daten von FirstSpirit aufnimmt und an die dynamischen Webapplikationen weitergibt (siehe Abbildung 2).

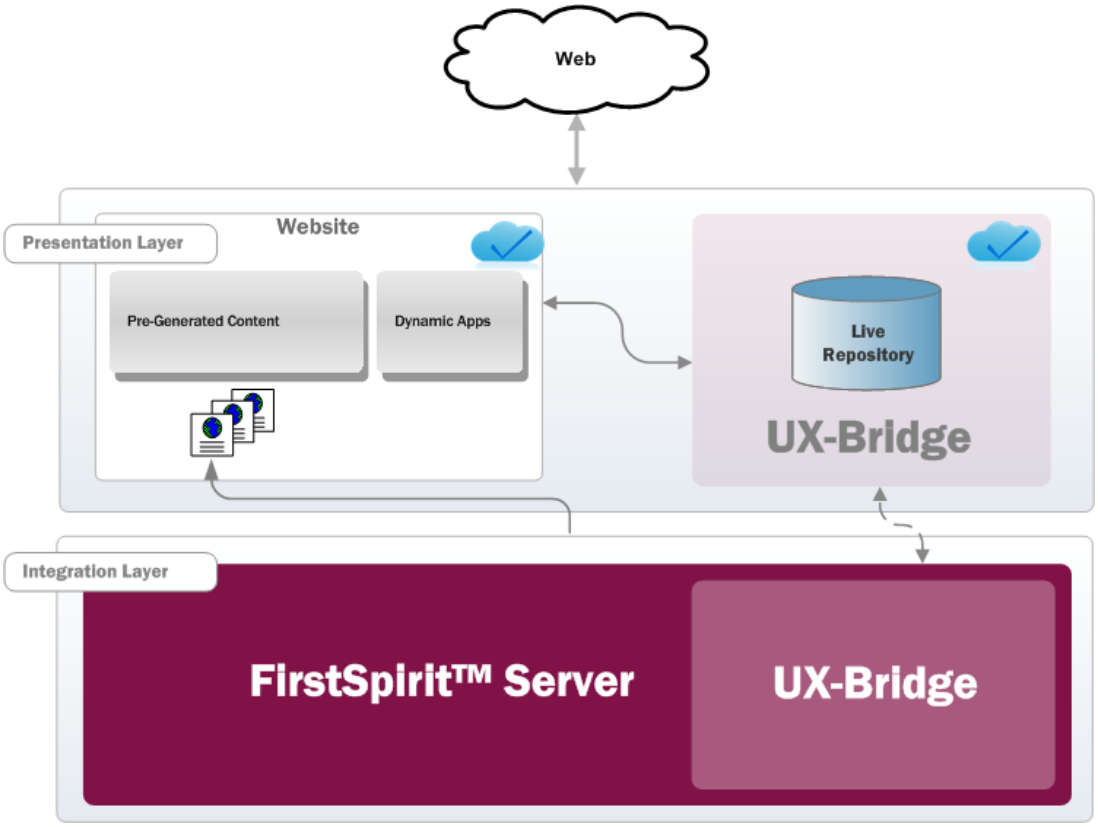

Abbildung 2: Vorgenerierung in Kombination mit der UX-Bridge

Wichtig ist hierbei, dass die beiden Ansätze „vorgenerierte Dateien“ (linke Seite der Grafik) und „Live-Repository“ (rechte Seite der Grafik) in der Regel kombiniert eingesetzt werden (siehe Abbildung 3).

So kann in einem News-Portal z.B. die normale Pressemitteilung komplett statisch vorgeneriert werden (als fertiges HTML-Fragment), während auf derselben Seite ein

„Top-Rating-Widget“ in der Marginalspalte dynamisch die Überschrift derjenigen Pressemitteilungen „live“ anzeigt, die am besten bewertet worden sind (in der Abbildung rot umrandet).

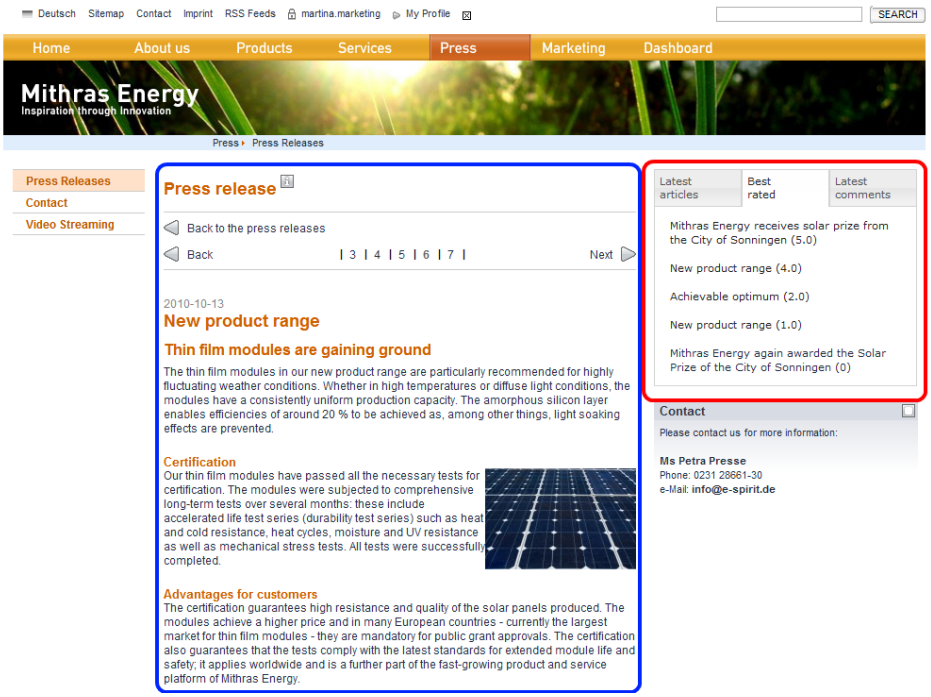

Abbildung 3: Vorgenerierung und Dynamik

Die hybride Architektur von FirstSpirit erlaubt eine beliebige Mischung solcher vorgenerierten und komplett dynamischen Inhalte.

2.1.1 Inhalte aktualisieren

Wenn wir bei dem oben beschriebenen News-Szenario bleiben, so erfolgt eine Aktualisierung der Website über folgende Schritte:

1. Ein Redakteur schreibt eine neue Pressemitteilung, die anschließend per Workflow freigegeben wird.

2. Der letzte Workflow-Schritt besteht aus der Veröffentlichung der Pressemitteilung, die sich wieder aufteilen lässt:

    a. Die Pressemitteilungsdetailseite wird komplett vorgeneriert (als statisches HTML) und anschließend auf den Webserver übertragen. Gleiches gilt für die News-Übersichtsseite, auf der die neue Mitteilung als erste Meldung erscheinen soll.

b. Die für die dynamische Anzeige relevanten Daten der Pressemitteilung (hier Überschrift der News) werden von FirstSpirit erzeugt und direkt (ohne die Erzeugung einer Datei) in das Live-Repository überführt. Das Format der Daten ist beliebig, wobei die Referenzimplementierung XML verwendet, welches HTML-Fragmente enthalten kann.

2.1.2 Inhalte ausgeben

Nach der Content-Aktualisierung ruft ein Website-Benutzer die Presseübersichtsseite auf, welche die neuste Meldung als ersten Link enthält. Ein Klick auf diesen Link öffnet die gerade aktualisierte Detailseite (siehe Abbildung 3). Rechts in der Marginalspalte der Seite ist eine dynamische WebApp, die zum Request-Zeitpunkt Kontakt zum Live-Repository aufnimmt und die Top-News ausliest und anzeigt.

Mit welchen technischen Mechanismen (z.B. über welches Webframework) die WebApp ihre Inhalte aus dem Live-Repository ausliest, ist von der Architektur nicht vorgegeben. Hier können beliebige passende Frameworks und Protokolle verwendet werden.

2.1.3 UX-Bridge Architektur im Detail

Nachdem der grobe Mechanismus zum Content-Update mit der UX-Bridge erläutert wurde, werden in diesem Kapitel die technischen Details des Moduls näher beschrieben (siehe Abbildung 4: UX-Bridge im Detail).

Auf der „Live-Seite“ (dem Presentation-Layer) besteht die UX-Bridge aus drei Komponenten:

- UX-Bus (oder auch Content-Bus):
  Der UX-Bus bildet die zentrale Infrastrukturkomponente um Inhalte von FirstSpirit in ein oder mehrere Live-Repositories zu verteilen. Technisch basiert die Referenzimplementierung des UX-Bus auf ActiveMQ (siehe http://activemq.apache.org/). Gleichzeitig kann der UX-Bus auch genutzt werden um Drittsysteme mit relevantem FirstSpirit Content zu versorgen (z.B. einen Suchindex oder eine Recommendation-Engine). Dritter Einsatzzweck des UX-Bus ist der Aufbau eines Rückkanals von der Website Richtung FirstSpirit Backend (siehe Kapitel 1.4).
  Der UX-Bus bildet somit die zentrale Integrationskomponente auf der Live-Website (siehe Kapitel 1.8), über die FirstSpirit mit den Repositories und

WebApps interagieren kann, aber ggf. auch die WebApps untereinander. Mit Hilfe des UX-Bus können somit sämtliche beteiligten Komponenten Daten und Events austauschen.
Technisch basiert der UX-Bus auf einer Message Oriented Middleware (MOM). Somit sind die Komponenten der UX-Bridge nur lose miteinander gekoppelt. Gleichzeitig ist durch eine asynchrone Kommunikation und den Einsatz von Warteschlangen die Skalierbarkeit gewährleistet. Die Referenzimplementierung des UX-Bus verwendet dafür ActiveMQ (siehe http://activemq.apache.org) als Message Broker und Apache Camel (http://camel.apache.org/) für das Routing und die Konvertierung der Nachrichten.

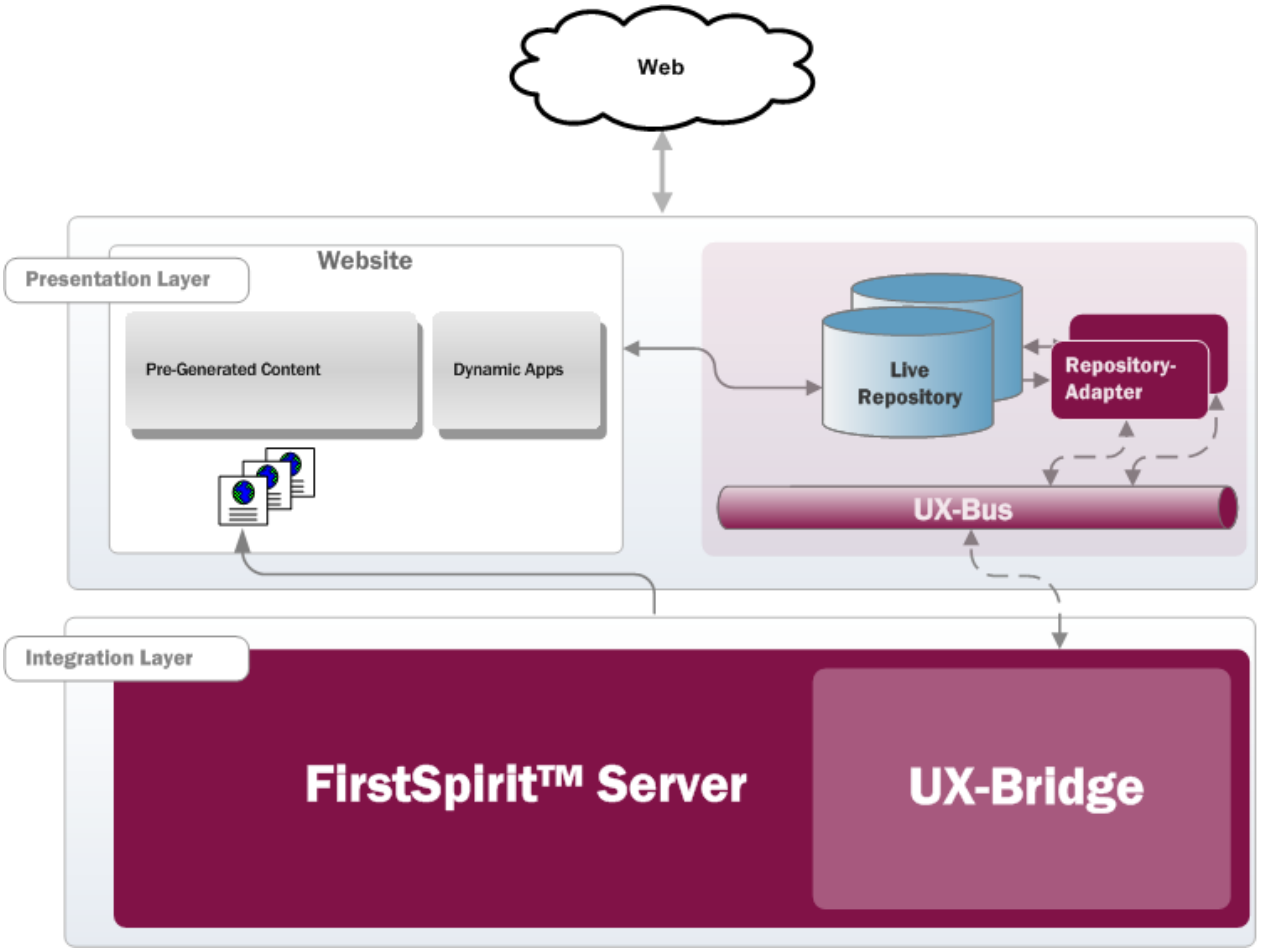

Abbildung 4: UX-Bridge im Detail

- Live-Repository:
  Das Live-Repository ist eine Datenhaltungskomponente, die von FirstSpirit gefüllt und von Webapplikationen ausgelesen wird. Wichtiges Architekturparadigma der

UX-Bridge ist hier: „Die Art und Anzahl der Repositories wird nicht vorgegeben“, weil je nach Aufgabenart unterschiedliche Repositories unterschiedlich gut geeignet sind (siehe Kapitel 2.1.3.2).
In vielen Fälle kann es auch sinnvoll sein, das Live-Repository nicht ausschließlich von FirstSpirit aus zu befüllen, sondern auch von der Website aus. Dies ist z.B. bei der Verwaltung von User-generated Content (UGC) relevant. Im Live-Repository wird dann FirstSpirit Content (z.B. Pressemitteilungen) mit UGC verknüpft (z.B. Bewertung der Pressemitteilungen).

- Repository-Adapter:
  Der Repository-Adapter übernimmt das Einfügen der FirstSpirit-Inhalte in das Content-Repository. Gleichzeitig erfolgt hier auch das „Mapping“ des projektspezifischen FirstSpirit-Datenmodells eines Content-Objektes (z.B. ein XML-Konstrukt mit der Pressemitteilung) auf das Datenmodell im Repository, welches meist von der Webapplikation bestimmt wird (z.B. eine relationale oder NoSQL Datenbank). Damit hat der Repository-Adapter immer einen projektspezifischen Anteil, d.h. für jede Art von Daten ist eine passende Mapping-Logik zu implementieren.

2.1.3.1 Wahl des Live-Repositories

Die Auswahl der „richtigen“ Live-Repositories ist eine wichtige Architekturentscheidung innerhalb des Projektes. In der Planungsphase eines Projektes sind immer zwei wichtige Fragen zu beantworten:

a) Welche redaktionelle Daten sollen/müssen in ein Live-Repository überführt werden?

b) Welche Art von Repository ist für diese Art von Daten und (Web-)Anwendung optimal?

Generell macht die UX-Bridge hier keine festen Vorgaben, da im Sinne einer aufgabenangemessenen Auslieferungsinfrastruktur verschiedene Kriterien berücksichtigt werden müssen:

- Art der Webapplikation und Daten:
  Welche Art von Webapplikation in Bezug auf die Datenpersistenz soll konkret verwendet werden? Wie sieht das Datenmodell der Anwendung aus und in welcher Art von Persistenz lässt sich dieses Datenmodell optimal abbilden? Ist ein statisches Datenmodell passend oder ist eine dynamische Erweiterbarkeit

notwendig? Ist z.B. Transaktionalität wichtig oder liegt der Schwerpunkt auf der Verfügbarkeit[1]?

- Performance- und Skalierungsanforderungen:
  Wie sind die Anforderungen Richtung Performance und Skalierbarkeit des Repositories? Ist eine horizontale Skalierbarkeit wichtig? Wie relevant ist das Abfedern von Lastspitzen? Wie elastisch muss die Performance angepasst werden können?

- Drittsysteme:
  Welche weiteren Systeme müssen auf die Daten zugreifen können? Mit welcher Art von Persistenz kommen die Drittsysteme am besten zurecht?

- Tool- und Framework-Support:
  Mit welcher Art von Repository kommt das eingesetzt Webframework besonders gut zu Recht? Wofür gibt es fertige Plugins? Gibt es weitere Tools (z.B. für Import/Export/Reporting), die auf das Repository zugreifen müssen?

- Knowhow im Team:
  Mit welcher Art von Persistenz sind die Entwickler am besten vertraut? Mit welchem System ist die beste Entwicklungsperformance zu erwarten?

Dies sind alles Fragen, die vor der Auswahl eines Repositories gestellt und beantwortet werden sollten, um auf dieser Basis dann eine Entscheidung zu treffen.

Für einen ersten Überblick, sind hier einmal die „üblichen“ Arten von Repositories aufgeführt:

- Relationale Datenbank (Oracle, MySQL, PostgreSQL etc.)

- NoSQL Datenbanken (MongoDB, Cassandra, Redis, SimpleDB, DynamoDB, etc.)

- Index-basierte Speicher (Lucene, Solr, Elasticsearch, Sphinxsearch, etc.)

2.1.3.2 Polyglot Persistence

Bei der beschriebenen Auswahl eines passenden Repositories für konkrete Anwendungsfälle kann es durchaus vorkommen, dass (sogar innerhalb eines Projektes) mehrere unterschiedliche Repositories zum Einsatz kommen, um eine aufgabenangemessene Lösung zu realisieren. Dieser Architekturansatz wird auch

[1] Der beschriebene Trade-Off bei den Repository-Eigenschaften wird auch als „CAP theorem“ bezeichnet (http://en.wikipedia.org/wiki/CAP_theorem)

„Polyglot Persistence“ Strategie genannt. Hinter diesem Begriff steht der einfache Ansatz:

> „Wähle die passende Persistenz für die den jeweiligen Anwendungsfall“

Der *gleichzeitige* Einsatz unterschiedlicher Persistenzarten für Daten ist dabei ausdrücklich erlaubt, wenn dies fachlich sinnvoll ist. Weitere Details zum Thema „Polyglot Persistence“ ist hier zu finden: http://martinfowler.com/articles/nosql-intro.pdf

Die UX-Bridge folgt genau diesem Ansatz, da in jedem Kontext die fachlichen Anforderungen die Auswahl der konkreten Persistenzarten bestimmen (siehe auch Kapitel 1.5).

2.2 Beispiele für Einsatzszenarien

Um die Frage zu beantworten, in welchen Szenarien die UX-Bridge sinnvoll eingesetzt werden kann, sind im Folgenden einige Praxisbeispiele aufgeführt. Zur Unterstützung tragfähiger Architekturentscheidungen wird in diesem Zusammenhang insbesondere betrachtet, welche Aspekte dynamisiert und welche sinnvollerweise vorgeneriert werden. Auch die Wahl eines passenden Repositories wird pro Anwendungsfall diskutiert.

2.2.1 News-Szenario inkl. User-generated Content (UGC)

Betrachten wir erneut das News-Szenario aus Kapitel 2.1. In einer Website gibt es eine Vielzahl von Nachrichten, die über die FirstSpirit Datenquellen verwaltet werden. Auf der Website gibt es News-Übersichtsseiten (mit den aktuellsten 10 News auf der ersten Seite) sowie News-Detailseiten (siehe Abbildung 5: News-Szenario). Auf den Detailseiten ist neben der Pressemitteilung (blauer Rahmen) auch ein Top-News-Widget (roter Rahmen), welches dynamisch die Top-News anhand bestimmter Kriterien (am besten bewertet, etc.) darstellt.

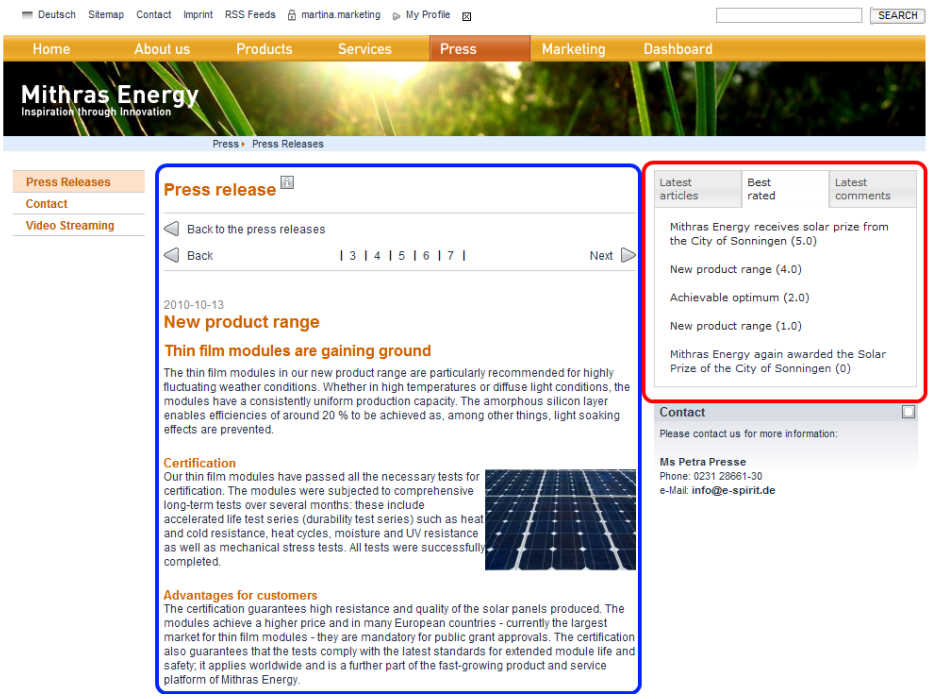

Abbildung 5: News-Szenario

Die Anforderungen an die Website sind:

a) Eine neue oder geänderte News muss innerhalb von 1 Minute auf der Website erscheinen

b) Sehr hohe Auslieferungsgeschwindigkeit der News-Übersichtsseite und News-Detailseite

c) Das Widget-Update (am besten bewertete News) muss sofort erfolgen, wenn neuer UGC eingestellt wird

Die Anforderungen a) und b) lassen sich abdecken, indem statische HTML-Seiten von FirstSpirit über die Standardmechanismen erzeugt werden. Für Anforderung c) eignet sich die UX-Bridge sehr gut (siehe Abbildung 6: Hybrides News-Szenario)

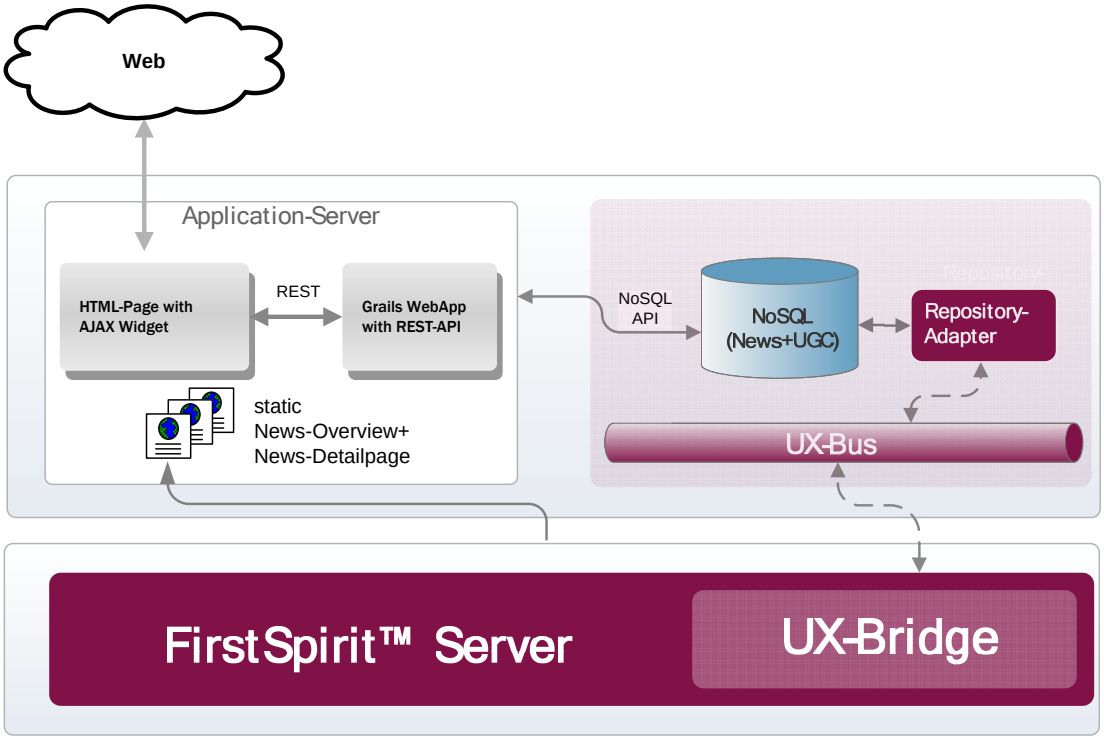

Abbildung 6: Hybrides News-Szenario

In diesem Fall wird eine NoSQL-Datenbank (z.B. MongoDB) ausgewählt, in der sowohl die relevanten (d.h. dynamisch im Teaser angezeigten) Teile der News, als auch der UGC für Kommentare und Bewertungen gespeichert werden. Die NoSQL-DB wird hier gegenüber einer klassischen relationalen DB bevorzugt, da die UGC-Daten sehr einfach strukturiert sind und die Datenbank einen besonders einfachen Zugriff per REST-API bietet. Ohne den UGC-Anteil wäre eine relationale DB ebenfalls eine gute Option.

Zum Auslesen der News und zur Verwaltung (Lesen+Schreiben) des UGC wird hier eine Standalone-Webapplikation mit einem MVC-Webframework (hier Grails) entwickelt. Die Wahl fällt in diesem Beispiel auf Grails (siehe http://grails.org), da es fertige Plugins zum Zugriff auf NoSQL-DBs gibt und die Anwendungsentwicklung für den beschriebenen Anwendungsfall sehr effizient möglich ist. Zudem ist es mit Grails einfach möglich JSONP über eine REST-Schnittstelle zur Verfügung zu stellen.

Die dynamische News-Teaser-Box ist direkt in den statischen News-Seiten per JavaScript eingebunden. Die Box ruft bei jedem Seitenaufruf per AJAX die Grails-WebApp auf und bindet die Ergebnisse in die Website ein.

2.2.2 Realtime-Update der Website

Moderne Webseiten sollen häufig mit einem „Realtime-Update“ ausgestattet werden.

Gemeint ist eine Aktualisierung der Inhalte, ohne dass der Benutzer der Website die Seite (manuell) erneut aufrufen muss.

Technisch kann dafür der gleiche Architekturansatz wie im vorherigen Kapitel verwendet werden. Die einzige notwendige Erweiterung ist ein „Poll-Mechanismus“ im News-Widget. Das Widget fragt in diesem Fall regelmäßig (alle x Sekunden) per AJAX die News-Webapplikation, ob neue Daten für die aktuelle Ansicht vorhanden sind. Ist das der Fall, lädt das Widget die neuen Daten per AJAX nach und aktualisiert sich selbst. Alternativ kann auch ein Pull-Mechanismus, z.B. über Websockets verwendet werden.

2.2.3 Produkt-Konfigurator

Im Beispiel „Produkt-Konfigurator“ ist eine komplexe Webanwendung zu entwickeln, die einem Website-Benutzer die Möglichkeit gibt sich selbst ein Produkt zu konfigurieren, welches aus verschiedenen Varianten und Ausstattungsmerkmalen besteht (z.B. ein Auto). Die Website besteht im Wesentlichen nur aus dieser Anwendung. Vom CMS soll der komplette „Rahmen“ kommen, d.h. das Layout und der Inhalt vom Kopf und Fußbereich. Sämtliche Produktinformationen (Varianten, Ausstattung, etc.) sollen ebenfalls redaktionell komplett über FirstSpirit pflegbar sein.

Zentrales Element der Website ist somit die zunächst unabhängig von FirstSpirit entwickelte Webanwendung des Konfigurators. Da der Webanwendung ein komplexeres Datenmodell zugrunde liegt, wird in diesem Fall eine relationale Datenbank als Repository gewählt. Ebenfalls relevant für diese Auswahl ist die Tatsache, dass moderne Webframeworks ein zum Objektmodell der Webanwendung passendes Relationenmodell automatisch erzeugen, was eine sehr effiziente Entwicklung ermöglicht.

Eine Abstimmung zum FirstSpirit-Entwickler ist bei der Definition des Datenmodells im Live-Repository allerding notwendig, damit auch die Struktur der FirstSpirit Datenquellen auf das gewünschte Repository-Modell abbildbar ist. Die Abbildung wird wiederum im Repository-Adapter ausimplementiert, was entweder ein Webentwickler oder ein FirstSpirit-Entwickler übernehmen kann.

Damit ergibt sich das folgende Architekturbild (siehe Abbildung 7: Produkt-Konfigurator).

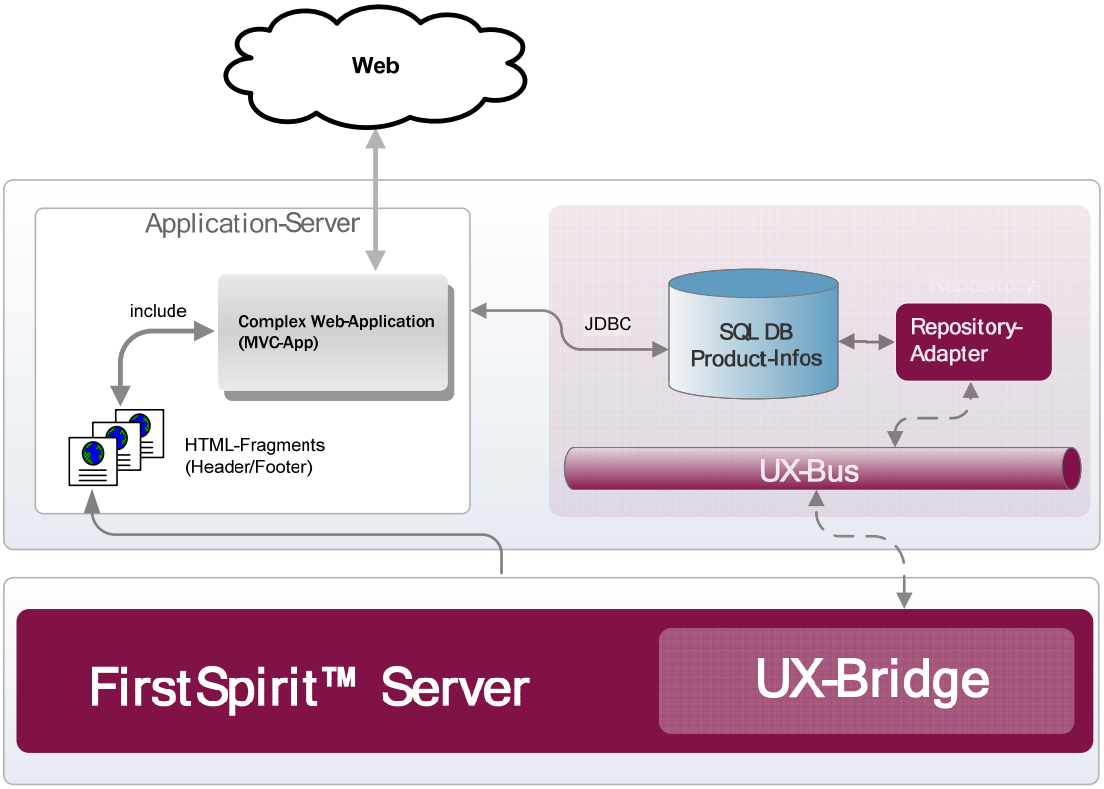

Abbildung 7: Produkt-Konfigurator

FirstSpirit erzeugt einmalig oder zyklisch die passenden HTML-Fragmente für den „Rahmen“ der Webanwendung (Header/Footer). Die WebApp inkludiert die erzeugten Fragmente bei jedem Aufruf und arbeitet ansonsten nur auf dem Live-Repository (hier einer relationalen SQL-Datenbank). Bei jeder Änderung von Produktdaten im FirstSpirit erzeugt die UX-Bridge die passenden Änderungen in der Produktdatenbank.

2.2.4 Filialsuche

Eine Filialsuche ist ein beliebter Anwendungsfall für eine integrierte Webanwendung, die direkt in eine redaktionell gepflegte Webseite eingebunden werden soll. Das Eingabeformular ist eher einfach (z.B. Postleitzahl oder Adressabfrage), ebenso die Ergebnisliste (Auflistung der nächstgelegenen Filialen).

Um die Webanwendung an beliebiger Stelle im Webauftritt einbetten zu können (z.B. in einer Marginalspalten auf der Homepage), bietet sich die Entwicklung einer JSP-Taglib zur Ergebnisvisualisierung sowie eines passenden Java Servlets an, welches die eigentliche Suchlogik abbildet (siehe Abbildung 8: Eingebettete WebApp).

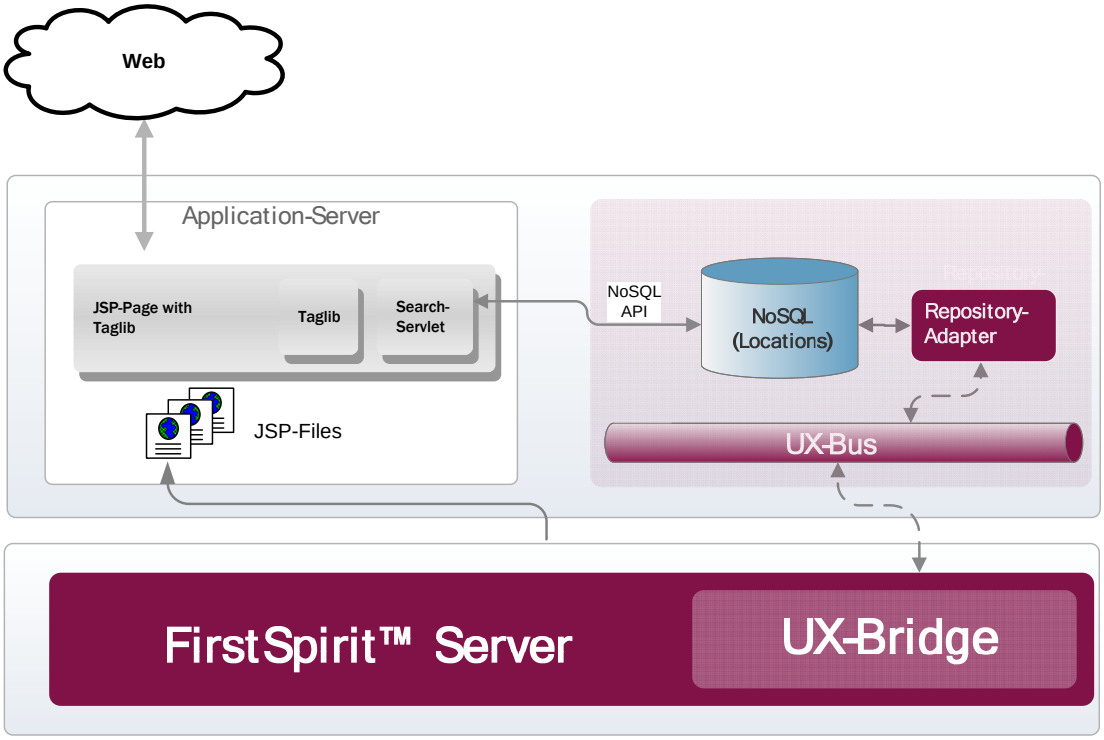

Abbildung 8: Eingebettete WebApp

FirstSpirit erzeugt in diesem Fall JSP-Dateien, die redaktionelle Inhalte, das Suchformular sowie die Ergebnisdarstellung als JSP-Logik enthalten.

Als Repository wird hier eine NoSQL Datenbank gewählt (MongoDB), da diese über fertige Mechanismen zur Umkreissuche (Geospatial) verfügt. Das Such-Servlet spricht diese Geo-Services direkt über die MongeDB-API an, während die notwendige redaktionelle Befüllung der Geo-Daten direkt über FirstSpirit erfolgt (z.B. über eine Google-Maps Integration).

2.2.5 Strukturierte Ansprechpartnersuche

Die Anwendungslogik innerhalb von Webapplikationen ist sehr häufig mit dem Thema „Suche“ verbunden. Neben einer normalen Volltextsuche existiert vielfach der Wunsch nach einer „strukturierten Suche“, d.h. die zu suchenden Objekte besitzen eine Menge von Attributen, nach denen gefiltert werden soll, ggf. sogar hierarchisch für ein Drill-Down.

Ein konkreter Anwendungsfall ist z.B. die Partnersuche auf der e-Spirit Website (siehe Abbildung 9: Suche von Ansprechpartnern). Dort hat jeder Ansprechpartner Attribute wie Region, Branche oder Partner-Level.

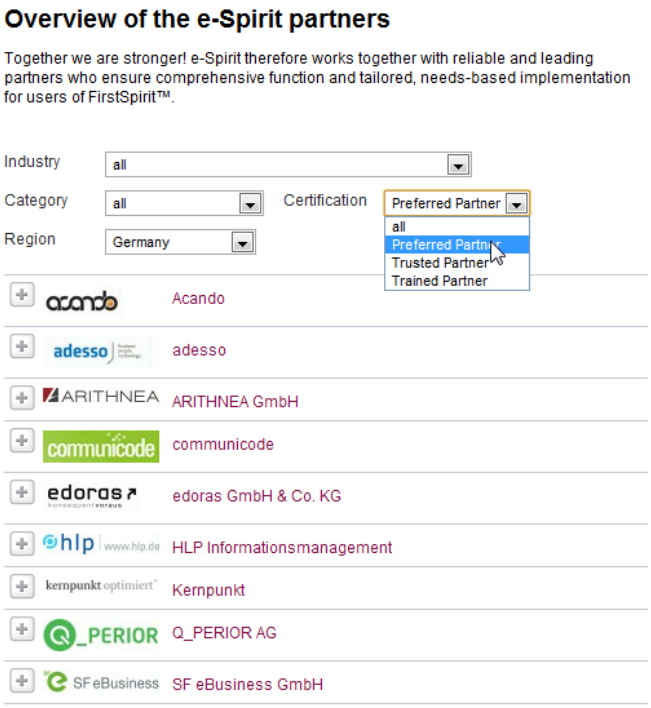

Abbildung 9: Suche von Ansprechpartnern

Bei dieser Art von Anwendung bietet es sich an, einen Index als Live-Repository zu verwenden (z.B. Lucene oder Solr), insbesondere wenn sehr große Datenmengen vorhanden sind, die extrem schnell durchsucht werden müssen (siehe Abbildung 10: Indexbasiertes Repository). FirstSpirit aktualisiert bei neuen oder geänderten Inhalte direkt den Index des Solr-Servers und hält diesen damit immer konsistent zu den Inhalten auf der Website.

Die anderen Architekturkomponenten sind in diesem Beispiel identisch mit der Filialsuche aus Kapitel 2.2.4. Die WebApp spricht über die Solr-API mit dem Live-Repository um die Suchanfragen zu beantworten.

An dieser Stelle lässt sich bereits gut erkennen, dass es sinnvoll ist die UX-Bridge als technische Basis für „höherwertige“ Module wie z.B. Enterprise-Search zu verwenden. In zukünftigen Versionen der FirstSpirit Laufzeitmodule wird dieser Weg vermehrt beschritten werden.

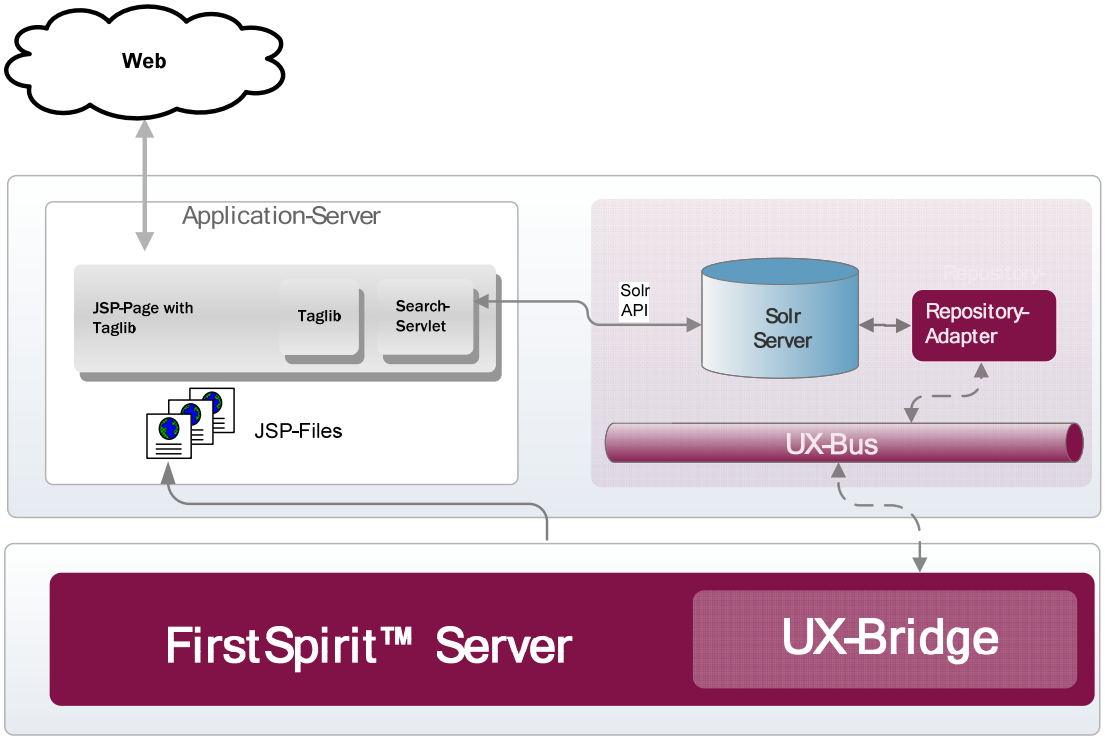

Abbildung 10: Indexbasiertes Repository

2.2.6 Mobiles Newsportal

Die Bereitstellung von Content auf mobiles Devices ist ein immer häufiger anzutreffender Anwendungsfall. In diesem Beispiel soll ein mobiles Newsportal entworfen werden, welches die redaktionell gepflegten News an eine Smartphone-App überträgt. Wichtige funktionale Anforderungen sind hier:

- Offline-Fähigkeit der App:
  Auf mobilen Geräten ist die Netzwerkanbindung immer mal wieder unterbrochen. Der Benutzer der App soll diese temporären Ausfälle nicht bemerken, d.h. die App soll „Offline-fähig“ sein. Sobald Netz vorhanden ist, soll sich der lokale Datenspeicher der App automatisch mit dem zentralen Repository synchronisieren.

- Bereitstellung des Content als „HTML-Fragment“:
  Der App-Entwickler möchte die News gerne als schon vorgerenderte HTML-Fragmente erhalten. Dies erlaubt eine sehr hohe Flexibilität, falls das Layout der Inhalte häufiger geändert wird. In diesem Fall muss die App nicht angepasst, sondern lediglich die Content-Fragmente neu erzeugt werden.

Eine intelligente Lösung für die Realisierung einer Offline-fähigen Content-App ist in

Abbildung 11 zu sehen. Als Content-Repository kommt hier die NoSQL Datenbank CouchDB zum Einsatz (siehe http://www.couchdb.org/). Großer Vorteil dieser Architektur ist, dass das komplexe Thema der Content-Synchronisierung komplett durch die CouchDB erfolgt. Zu diesem Zweck kommt auf dem Smartphone eine lokale/mobile Version der CouchDB zum Einsatz, die über passende Synchronisierungsfähigkeiten verfügt. Eine komplexe Programmierung der Synchronisierungslogik ist hier also nicht notwendig.

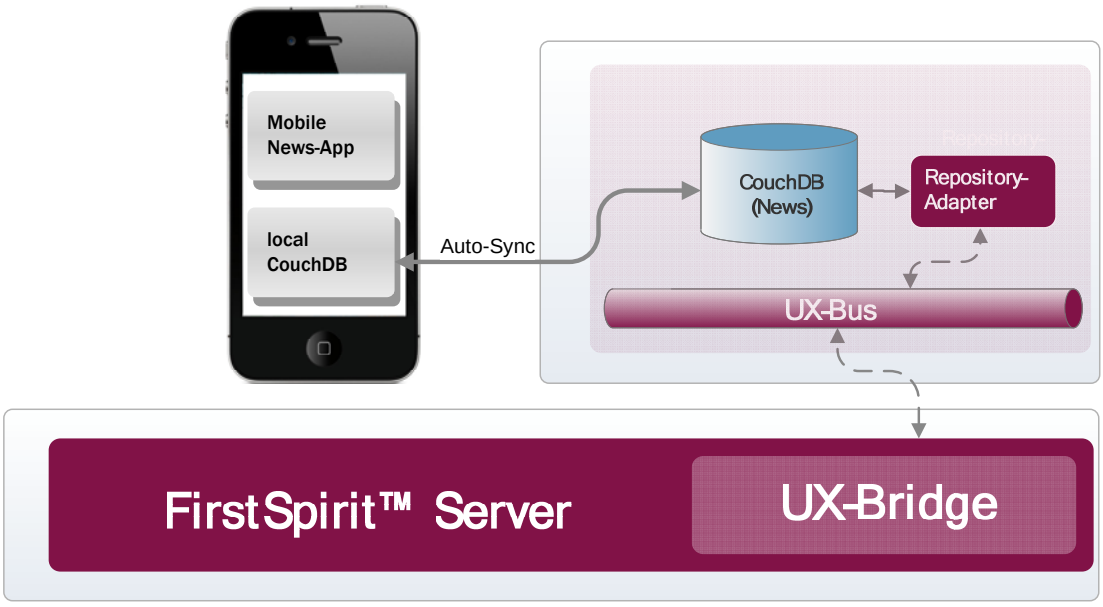

Abbildung 11: Mobilisierung über UX-Bridge

Diese Architektur ist ein gutes Beispiel dafür, dass die geschickte Auswahl des Content-Repositories häufig zu einer Reduzierung von Implementierungsaufwänden führt.

Abschließend kann man bei der Betrachtung der aufgeführten Anwendungsbeispiele festhalten, dass unterschiedliche Anwendungen auch unterschiedliche Architekturvarianten mit sich bringen.

Die UX-Bridge liefert in jeden Fall die notwendige Flexibilität, um auch in sehr unterschiedlichen Szenarien eine angemesse Lösung realisieren zu können.

Im nächsten Kapitel werden mögliche Architekturvarianten noch einmal auf andere Art und Weise klassifiziert. Dabei werden auch noch neue Varianten erwähnt, die in den beschriebenen Beispielen nicht enthalten waren.

2.3 Architekturvarianten

Es existieren verschiedene Architekturvarianten um Web-Applikationen, die webserver-seitig auf die UX-Bridge zugreifen, in einen Webauftritt zu integrieren. Die üblichen Varianten werden in diesem Kapitel noch einmal kurz zusammengefasst und kategorisiert.

2.3.1 Client-seitige Dynamik über Widgets

Eine sehr leichtgewichtige Variante zur Abbildung von nicht allzu komplexen Webanwendungen ist die Implementierung einer JavaScript-Logik, die komplett im Browser innerhalb einer HTML-Seite abläuft (siehe Anwendungsbeispiel in Kapitel 2.2.1).

Sinnvolle Einsatzszenarien sind:

- Leichtgewichtige Widgets
- Überschaubare Komplexität/Logik
- Einfache Einbindung an beliebige Stellen im Webauftritt

2.3.2 Server-seitige Dynamik

Eine server-seitige Dynamik ist immer dann sinnvoll, wenn auch komplexere Applikationslogik notwendig ist oder das Thema „Sicherheit“ eine besondere Rolle spielt.

Es existieren verschiedene Untervarianten dieser Applikationskategorie, wobei neben den aufgezählten Typen sicher noch weitere Varianten denkbar sind.

2.3.2.1 JavaServer Pages (JSP)

Das nahtlose Einbinden von server-seitiger Logik erfolgt im Java-Kontext klassisch über Java Server-Pages (JSP). In der einfachsten Variante (JSP Model 1) ist der JSP-Code (bzw. die JSP-Taglibs) direkt in die Seite eingebunden.

Sinnvolle Einsatzszenarien sind:

- Einfache server-seitige Logik

- Einfache Einbindung an beliebige Stellen im Webauftritt

2.3.2.2 MVC-Frameworks

Für komplexere Webanwendungen mit komplizierterer server-seitiger Logik bietet sich der Einsatz von Model-View-Controller (MVC) Frameworks an. Über diesen Ansatz lassen sich gut wart- und erweiterbare Anwendungen entwickeln.

Sinnvolle Einsatzszenarien sind:

- Komplexe Webanwendungen
- Stand-alone Entwicklung der Webanwendung, ggf. ganz unabhängig von FirstSpirit

2.3.2.3 Portlets

Die Portaltechnologie bietet eine standardisierte Infrastruktur, um einzelne Webanwendungen in Form von Portlets unabhängig von anderen Anwendungen in einen Webauftritt zu integrieren. Dazu können beliebige redaktionelle Inhalte ergänzt werden, welche ebenfalls über (Content-) Portlets eingebunden werden.

Dieser Ansatz ist besonders interessant, da FirstSpirit schon fertige Integrationen zu bekannten Portalservern bereitstellt (SAP Netweaver, IBM Websphere Portal, Microsoft Sharepoint oder Liferay).

Der Einsatz eines Portalservers stellte eine grundlegende Architekturentscheidung dar, die unter verschiedenen Gesichtspunkten getroffen werden muss. Hat man sich einmal für einen konkreten Portalserver entschieden, können aber alle Vorteile einer standardisierten Integrationsplattform genutzt werden.

2.3.2.4 Anbindung von Drittsystemen

Neben der Entwicklung eigener Applikationen gibt es häufiger den Anwendungsfall, dass existierende Drittanwendungen mit redaktionellen Inhalten versorgt bzw. redaktionell aus FirstSpirit heraus parametrisiert und gesteuert werden sollen.

Konkrete Beispiele für diese Art von Anwendungen sind:

- eCommerce Anwendungen / Shop-Systeme

- Recommendation Engines
- Community-Software
- Collaboration Plattformen

Für jeden dieser Anwendungstypen ist auf Basis der fachlichen Anforderungen zu entscheiden, ob eine Vorgenerierung von passenden HTML/XML-Fragmenten ausreichend ist, oder ob alternativ bzw. ergänzend auch die UX-Bridge zum Einsatz kommen soll.

3 FAQ

Beim Einsatz der UX-Bridge im konkreten Projekt ergeben sich häufig wiederkehrende Fragen, die im Folgenden beantwortet werden.

3.1 Fragen zur Dynamisierungstiefe

[Q] Kann ich über die UX-Bridge auch den kompletten Webauftritt mit allen Inhalten und Strukturen ins Live-Repository überführen?
[A] Theoretisch ja, aber dies ist in der Regel keine gute Architektur. Nur in den allerwenigsten Fällen ist wirklich der komplette Webauftritt voll dynamisch. In allen anderen Fällen sollte von der Vorgenerierung Gebrauch gemacht werden.

3.2 Fragen zum UX-Bus

[Q] Kann man den UX-Bus auch auf dem FirstSpirit-Server betreiben, um möglichst wenig Infrastruktur auf der Live-Seite installieren zu müssen?
[A] Ein Betrieb des UX-Buses auf dem FirstSpirit-Server ist technisch möglich. Dabei wird ActiveMQ im integrierten Jetty auf dem FirstSpirit-Server betrieben. Sofern der UX-Bus nur für die Befüllung der Live-Repositories genutzt wird, sollte dieses Szenario für die meisten Anwendungsfälle ausreichend sein. Muss der UX-Bus hochverfügbar sein, so sollte der UX-Bus vom FirstSpirit entkoppelt werden. Weitere Hinweise zur Architektur finden Sie in unserem Adminhandbuch zur UX-Bridge.

[Q] Ist ActiveMQ die einzige unterstützte Messaging Infrastruktur?
[A] Es können beliebige Messaging-Systeme eingesetzt werden, allerdings gibt es nur für ActiveMQ auch Betriebserfahrung bei e-Spirit.

[Q] An wen kann ich mich wenden, wenn es Probleme mit UX-Bridge gibt?
[A] Bei Problemen mit der UX-Bridge wenden Sie sich bitte, wie bei anderen Modulen auch, bitte an unseren Helpdesk. Beratung bzgl. Einsatzszenarien und Architektur ist über unseren Professional Service verfügbar.

[Q] Wie erfolgt ein Sizing für ActiveMQ?
[A] Das Sizing für ActiveMQ hängt im Wesentlichen von der geforderten Verfügbarkeit ab, die durch geeignetes Clustering und Failover-Mechanismen erreicht werden kann. Einen weiteren Faktor stellen die zu erwarteten Nachrichten pro Sekunde dar. Für belastbare Aussagen sollte ein Performance-Test im Projekt durchgeführt werden. Genauere Hinweise finden Sie unter

http://activemq.apache.org/performance.html. Konfigurationsempfehlungen finden Sie in unserem UX-Bridge Adminhandbuch.

3.3 Fragen zur Repository-Wahl

[Q] Was ist das „Standard“ Live-Repository, welches von e-Spirit empfohlen wird?
[A] e-Spirit empfiehlt kein Standard-Repository. Ähnlich wie bei der Auswahl einer konkreten Datenbank oder eines Betriebssystems hängt das „richtige“ Repository sehr stark von den fachlichen Anforderungen ab.

[Q] Ich habe keine besonderen Anforderungen an eine Live-Repository. Welches Live-Repository soll ich dann nehmen?
[A] Wenn es fachlich nur absolut minimale Anforderungen gibt (sehr einfaches Datenmodell, wenig Traffic auf der Website, wenig Updates, etc.) sollte ein Repository gewählt werden, bei dem schon möglichst viel Know-How (im Bereich Betrieb und Verwendung) beim Umsetzungspartner vorliegt. Generell sollte das Know-How des Implementierers bei der Auswahl immer mit berücksichtigt werden.

3.4 Fragen zur Infrastruktur

[Q] Wie wird eine 2- oder 3-stufige Systemlandschaft (D/Q/P) mit der UX-Bridge aufgesetzt?
[A] Wir empfehlen hier, wie bei anderen Modulen auch, die notwendigen Komponenten auf jedem System (Entwicklung, Testsystem, Produktion) zu installieren, um eine strikte Trennung zwischen den einzelnen Systemen zu gewährleisten und auch die korrekte Konfiguration testen zu können. Es ist aber auch möglich durch entsprechendes Routing die Nachrichten aller Systeme über einen UX-Bus zu verteilen.

[Q] **K**ann die UX-Bridge in einem Hochverfügbarkeitsszenario verwendet werden?
[A] Den Kern der UX-Bridge stellt die Messaging-Infrastruktur, basierend auf ActiveMQ, dar. ActiveMQ wurde speziell für Hochverfügbarkeitsszenarien entwickelt. Genauere Informationen zur Fehlertoleranz und Clustering finden Sie in unserem Adminhandbuch.

[Q] Wie sieht ein Cluster-Betrieb der UX-Bridge aus?
[A] Für den Clusterbetrieb sind eine Reihe von Topologien möglich. Details finden Sie in unserem Adminhandbuch.

[Q] Wo erhalte ich kommerziellen Support für ActiveMQ?
[A] Es gibt eine ganze Reihe von Anbietern, die kommerziellen Support für

ActiveMQ anbieten. Eine Liste finden Sie unter
http://activemq.apache.org/support.html

3.5 Sonstige Fragen

[Q] Wie kann die UX-Bridge in die FirstSpirit Vorschau integriert werden?
[A] Für die Integration in die FirstSpirit Vorschau ist nur die Webapplikation maßgeblich. Wie diese in die Vorschau integriert werden kann, hängt von der verwendeten Technologie ab. Bei konkreten Projektanforderungen gehen Sie bitte auf den Professional Services Bereich von e-Spirit zu.

[Q] Wie geht man mit Medien (z.B. Bilder in einem Produktkatalog) um? Sollen diese mit ins Live-Repository eingefügt werden?
[A] Generell ist es technisch möglich auch Binärdateien mit ins Repository zu überführen. In den meisten Anwendungsfällen ist aber stattdessen ein vorgenerierendes Deployment sinnvoll. Wird z.B. eine Produktdetailseite vorgeneriert, so sind die enthaltenen Medien schon Teil des normalen Deployments. Werden bestimmte Auflösungen benötigt (z.B. für verschiedene Produkt-Teaser), bietet sich ein gentrennten Medien-Deployment aller zugehörigen Produktbilder an.

[Q] Hilft mir die UX-Bridge beim Thema „Delta-Deployment“?
[A] Mit Delta-Deployment ist ein Mechanismus gemeint, der nur die Seiten/Objekte generiert und deployed, die sich seit dem letzten Deployment geändert haben. Die UX-Bridge stellt hierfür keine neuen Mechanismen zur Verfügung, sondern verwendet die von FirstSpirit zur Verfügung gestellten Mechanismen (Arbeitsabläufe, Revisions-API, Teilgenerierungen und ab FirstSpirit 5 auch die DeltaGenerierung).

[Q] Brauche ich beim Einsatz der UX-Bridge noch eine Volltextsuche?
[A] Hier muss zwischen zwei Szenarien unterschieden werden: Sofern die Inhalte nicht nur über die UX-Bridge, sondern auch als statische Datei erzeugt werden, so ist keinerlei Sonderbehandlung notwendig. Die Inhalte sind dann ganz normal über die BasicSearch bzw. EnterpriseSearch durchsuchbar. Vergleiche dazu auch Kapitel 2.2.1 News-Szenario. Sind die Inhalte nur im Live-Repository persistiert ist eine zusätzliche Konfiguration bzw. Implementierung notwendig. Das EnterpriseSearch Modul bietet eine Reihe von Konnektoren um externe Repositories anzubinden. Alternativ kann der der Polyglot-Persistance Ansatz verwendet werden, um die Inhalte über die UX-Bridge auch in einen Suchindex zu schreiben (siehe Kapitel 2.2.5).