- Entwurfsphase: Entwurfsbeschreibung Softwarestudie -

Version: 1.0

Projektbezeichnung	MSP-13 - Integration eines Semantischen Tagging Systems in Microsoft Sharepoint	
Projektleiter	Martin John	
Verantwortlich	Martin John	
Erstellt am	06.04.2013	
Zuletzt geändert	08.04.2013 21:19	
Bearbeitungszustand	X	in Bearbeitung
		vorgelegt
		fertig gestellt
Dokumentablage	C:\Users\Martin\msp-git\Dokumentation\Entwurfsbeschreibung\13-04-06_Leiter_Entwurfsbeschreibung-Studie_E .odt	
V-Modell-XT Version	1.4	

INHALTSVERZEICHNIS

Änderungsverzeichnis

Änderungen			Geänderte Kapitel	Beschreibung der Änderungen	Autor	Zustand
Nr.	Datum	Version				
1	06.04.2013	0.1	alle	Initiale Erstellung	JM	erledigt
2	08.04.2013	0.1	3,4,5	Paketübersicht aktualisiert, Beschreibung Pakete und Testkonzept hinzugefügt	FM, BR	erledigt
3	08.04.2013	1.0	3, 5	Korrekturen	JM	erledigt

Prüfverzeichnis

Die folgende Tabelle zeigt einen Überblick über alle Prüfungen – sowohl Eigenprüfungen wie auch Prüfungen durch eigenständige Qualitätssicherung – des vorliegenden Dokumentes.

Datum	Geprüfte Version	Anmerkungen	Prüfer	Neuer Produktzustand
08.04.2013	0.1	Änderungen aufgezeichnet und Kommentare eingefügt.	RA	In Bearbeitung

1 EINLEITUNG

Die folgende Entwurfsbeschreibung gibt einen Überblick über die geplante und erfolgte Umsetzung der Softwarestudie des Projekt. Ziel der Studie ist die Herstellung der Verbindung zwischen Triple-Store und Sharepoint. Dabei wird zuerst nur in eine Richtung gearbeitet: Die Abfrage des Stores durch die Sharepoint Oberfläche.

2 PRODUKTÜBERSICHT

Das Projekt besteht aus drei Hauptkomponenten:

1. Virtuoso-Datenbank zur Speicherung und Verwaltung der RDF-Triples

2. Farmfeature zur Deaktivierung des Standard-Taggings und Starten des erweiterten Taggings

3. Webfeature als Interface zur Darstellung und Bearbeitung der Tags

3 GRUNDSÄTZLICHE STRUKTUR- UND ENTWURFSPRINZIPIEN

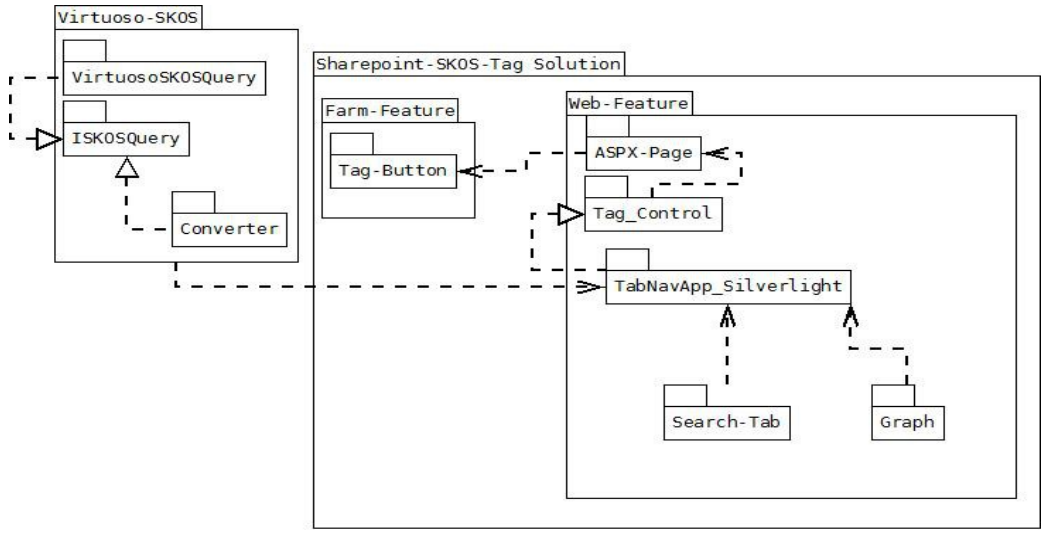
Das System orientiert sich am MVC-Modell. Durch die Nutzung von Sharepoint ist dieses Modell bereits vorgegeben.

Das Modell versorgt die Sharepoint-Features mit den RDF-Triples aus dem Triple Store.

Das View stellt die Daten aus dem Store im Web-Browser dar. Dazu ist eine Silverlight-Komponente erforderlich. Dafür und für die Darstellung dynamischer Inhalte wurde aufgrund der einfachen Kompatibilität auf Microsoft-Silverlight zurückgegriffen.

Die Controller-Komponente überwacht und kontrolliert die eingehenden Daten und Vorfälle und entscheidet, wie dann zu verfahren ist. Diese Komponente ist in der Studie nicht explizit aufgeführt.

Die Hauptkomponenten sind in weitere Pakete unterteilt. Die folgende Darstellung zeigt eine Paketübersicht.



4 STRUKTUR- UND ENTWURFSPRINZIPIEN EINZELNER PAKETE

4.1 Sharepoint Paket (Skos-Tag Solution)

Die Skos-Tag Solution ist in verschiedene Pakete und Module gegliedert, die sich grundsätzlich auf zwei Sharepointfeatures auteilen und zusammen die View- und ViewModel-Schicht einer VMMV-Architektur darstellen.

4.1.1 Feature 1 (Farmfeature)

Dieses Feature hat den Scope 'Farm', d.h. Nach dessen Aktivierung ist es auf der gesamten Serverfarm vorhanden. Es umfasst zur Zeit allein den Button in der oberen rechten Ecke jeder Sharepointseite, welches die standardmäßig vorhanden Buttons für die Sharepoint Social Features ('I Like', 'Tags & Comments') ersetzt. Daher muss dieses Feature auch zuvor deaktiviert werden. (Da dieses Projekt die Funktionen vollkommen ersetzen soll (kann) steht diesem auch nichts im Wege.)

Anders als Controls in den Ribbons einer Seite bei Sharepoint, befindet sich dieser Button auf der Masterpage und muss daher über ein 'Delegate Control' in vordefinierte Platzhalter eingefügt werden.

Etwas so: http://mihirsharepoint.wordpress.com/2012/11/15/create-delegate-control-in-sharepoint/

Dieser Button ruft den Inhalt des zweiten Features in einem Popupfenster auf.

4.1.2 Feature 2 (Webfeature)

Der Scope dieses Featues ('Web' - bedeutet bei Sharepoint: Seite) beschränkt sich auf eine einzelne Seite oder Seitenkollektion.

Daher muss es auch für alle Seiten einzeln aktiviert werden, welche diese Solution einsetzen sollen.

Dies muss zur Zeit noch manuell eingestellt werden, soll aber bei der Endversion automatisch über 'Event Receiver' aktiviert werden.

Dieses Feature basiert auf einer einzelnen aspx-Seite deren einziger Inhalt eine Silverlightanwendung ist, welche alle anderen Bestandteile dieses Features beheimatet.

Silverlight (Version 5.1) kommt an dieser Stelle aus o.g. zum Einsatz:

4.1.3 Silverlight Anwendung (vorläufiger Projektname: TabNavApp)

Diese Anwendung ist zentraler Bestandteil des Web basierten Features und damit auch Kern der gesamten Solution. In ihr vereinigen sich alle Benutzeroberflächeninhalte, sowie die gesamte Logik der Triple-Store Abfragen und die Verarbeitung der von dort erhaltenen Antworten.

Verschiedene Ansichten wie: Dokumentzusammenfassung mit Tag-Vorschlägen, Tag Suchfunktion mit Graphvisiualisierung für den Frontendeinsatz, sowie zusätzliche Backendfunktionen für Administratoren und Ontologiemanager soll über ein einfaches Tab-Control realisiert werden.

Der Inhalt des Vorprojekts befindet sich im Tab 'Search Tags' und bietet den kompletten Funktionsumfang für einzig abfragebasierte Teilaufgaben (Suche, Anzeige von Eigenschaften eines Konzepts (Tag) sowie die Visualisierung des Konzepts in seiner Graphumgebung (broader, narrower, related und sibling Konzepte).

Jedes Sharepoint Projekt wird mit den beiden Dateien 'App.xaml' and 'MainPage.xaml' initialsiert.

Die 'App.xaml' dient zur Ressourcendefinition, während die 'MainPage.xaml' die Definition der Oberfläche der automatisch generierten Startseite der Anwendung beinhaltet.

Wird die Anwendung mit Tab-Control initialisiert kommen die Ordner Assets (hier können verschieden Anzeige Templets hinterlegt werden)und Views (beinhaltet alls Tab Seiten) hinzu.

Allgemein: Hinter jeder .xaml Datei befindet sich eine so genannten 'Code-behind' Datei, z.B. In C#, welche die gesamte Logik für die in der .xaml definierten Oberfläche zur Verfügung stellt und das Bindeglied zur Datenzugriffsschicht darstellt (Model). Silverlight setzt damit auf die von WPF oder HTML5 bekannte Architektur: Model View ViewModel (MVVM) und beinhaltet View und ViewModel.

4.1.4 Search Tags:

Der Tab Search Tags ist wie folgt gegliedert:

a) Tag Suchfunktion mit Ergebnisliste

Setzt die searchTags Methode der VirtuosoSkosQuery.cs um und Zeigt die top-k Suchergebnisse an.

Mit einen Klick auf 'View' wird ein Graphvisiualisierung initialisiert

b) Im oberen Abschnitt werden Eigenschaften des gewählten Konzepts wie Name, Alternativnamen, Beschreibungen und die Mitgliedschaft in Collections angezeigt.

c) Die Graphvisiualisierung ist interaktiv. Mit einem Klick auf ein beliebiges Concept wird dieses zum zentralen Konzept und dessen Umgebungskonzepte angezeigt. Mit Strg+Click soll später ein instant-tagging realisiert werden.

Diese Visiualisierung beruht auf diesem Projekt: https://github.com/lvaleriu/bagotricks, wurde aber an die Bedürfnisse dieses Projekts angepasst und entsprechend verändert.

Die Graph Mechanik findet sich in der SilverlightCompLib.dll, einige Funktionen für das Frontend wurden in die Klasse SearchGrapg.xaml.cs übernommen. (genauere Informationen über die Kommentare dieser Klasse)

4.2 Virtuoso-Skos-Libraries

Dieses Paket umfasst einen Teil der zweigeteilten Datenzugriffsschicht dieses Projektes und stellt den Zugriff auf einen Virtuoso-Triple-Store sicher. Die zweite Datenzugriffsschicht wird von Sharepoint über das Data-Object-Model, das auf einer SQL-Datenbank beruht, zu Verfügung gestellt.

In disem Paket greifen wir auf die umfangreiche API Dot-Net-RDF (https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/User%20Guide) zurück, welche alle Funktionen die zur Kommunikation mit einem RDF-Triple-Store von Nöten sind bietet. Ein Virtuoso-Server mit aktiven SPARQL-Endpunkt bildet die Datenschicht für unser Projekt (http://www.openlinksw.com/wiki/main/) und ist mit dem Skos-Graphen des UNESO-Theasaurus versehen (http://skos.um.es/unescothes/), welches die Grundlage für alle Überlegungen dieses Paketes bildete. Der von Virtuoso eingesetzte SPARQL-Dialekt setzt bereits auf einige Konzepte der Sparql-Version 1.1 (http://www.w3.org/TR/rdf-sparql-query/) aber hat noch nicht alle beschriebenen Konzepte von W3C für SPARQL 1.1 implementiert

(http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtTipsAndTricksSPARQL11FeaturesExamplesCollection). Im Folgenden beschreiben 'Virtuoso' und 'Triple-Store' das selbe.

Zum Zeitpunkt der Vorprojektabgabe bietet dieses Modul allein Unterstützung für Anfragen zum Abfragen von Daten aus dem Triple-Store (nach Vereinbarung). Alle Daten werden über SPARQL-Abfragen abgerufen und deren Ergebnisse als SPARQL-Resultsets empfangen, (DOT-Net-RDF)

Die Bibliotheken haben den folgenden groben Aufbau:

ISkosQuery: ist das definierende Interface welches den aktuellen Funktionsumfang festlegt.

VirtuosoSkosQuery: dessen Implementierung für einen Virtuoso-Sparql-Endpunkt auf Grundlage von Dot-Net-RDF und einem Skos-Graphen der broader, sowie narrower als Prädikate bereitstellt (am Beispiel UNESCO-Theasaurus). Abweichungen von diesen Eckpunkten können durch eigene Implementierungen von IskosQuery begegnet werden.

ConverterClass: Bietet statische Methoden, welche beliebig ergänzt werden können und zur Umwandlung von SPARQL-Resultsets in andere Metadatenformate dienen.

Eine Settingsdatei welche zur externen Konfiguration genutzt werden kann.

5 TESTKONZEPT

Die Komplexität dieses Projekts macht es erforderlich die Implementierungsphase (bzw. Einführungsphase) mit umfangreichen Tests zu begleiten. Hierbei werden mittels Komponententests (unit tests) die korrekte Funktionsweise jeder Methode und Klasse sichergestellt. Integrationstests stellen die richtige Zusammenarbeit der einzelnen Komponenten sicher und Systemtests überwachen das Verhalten der kompletten Software. Zum Schluss kontrolliert der Auftraggeber der Software durch einen Abnahmetest die korrekte Funktionsweise.

5.1 Komponententests

Da die Entwicklung in diesem Projekt mit Microsoft Visual Studio 2010 durchgeführt wird, ist das in die Entwicklungsumgebung tief integrierte Komponententest-Framework von Microsoft unser verwendetes Unit-Test-Tool. Ziel ist es, den Entwickler schnell auf logische Fehler im Quellcode hinzuweisen und Funktionsmerkmale einzelner Komponenten unabhängig vom Gesamtsystem zu testen. Da einzelne Methoden einer Klasse nur eine geringe Komplexität besitzen, können diese mit wenigen Testfällen nahezu vollständig geprüft werden. Das Komponententest-Framework sieht dazu vor, die einzelnen Tests mit Hilfe von sogenannten Test-Methoden zu realisieren. Diese Methoden werden mit speziellen Test-Klassen zusammen gefasst und können somit nach jeder Code-Änderung des entsprechenden Abschnitts getestet und auf ihre Funktionstüchtigkeit geprüft werden. Mit Hilfe der in Visual Studio integrierten Test-Listen lässt sich dieser Prozess übersichtlich organisieren und auch automatisiert durchführen.

5.2 Integrationstests

In dieser Phase werden insbesondere Funktions- und Schnittstellentests mit den zuvor in Komponententests bestätigten Einzelkomponenten durchgeführt und auf ihre Interaktionsfähigkeit im Softwaresystem untersucht. Zur Durchführung dieses Tests werden die einzelnen erstellten Komponententests auf mehrere Komponenten erweitert und anschließend wieder mit Hilfe von Visual Studio ausgewertet.

5.3 Systemtest

In dieser Phase wird das vollständige Produkt in einer Testumgebung, die den Anforderungen des späteren Anwendungsgebietes genügen muss, getestet. Hierbei werden nicht mehr einzelne Elemente betrachtet, sondern die Gesamtfunktion des aus den einzelnen Komponenten zusammengesetzten Produkts überprüft. Dabei wird beachtet, dass dem Benutzer alle vom Pflichtenheft geforderten Funktionen vollständig und fehlerfrei zur Verfügung stehen.

5.4 Abnahmetest

Beim Abnahmetest wird ein letzter Testlauf gemeinsam mit allen Beteiligten durchgeführt. Als Testumgebung dient eine möglichst genaue Simulation der Benutzerumgebung, um Kompatibilitätsprobleme vollständig auszuschließen. Alle Anforderungen des Pflichtenheftes werden der Reihe nach überprüft und auf Vollständigkeit getestet. Am Ende dieser Phase steht ein fehlerfreies und dem Pflichtenheft genügendes Gesamtprodukt.