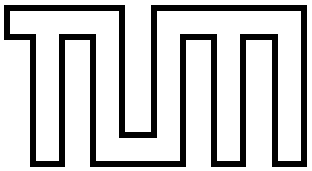


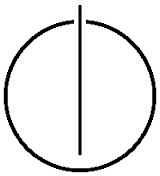
FAKULTÄT FÜR INFORMATIK

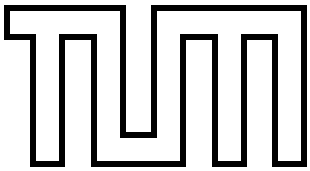
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Unterstützung für teamorientiertes Informationsmanagement mit föderierten Datenquellen

Thomas Reschenhofer
Bernhard Waltl





FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

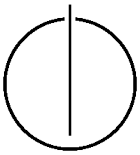
Bachelorarbeit in Informatik

Unterstützung für teamorientiertes Informationsmanagement mit föderierten Datenquellen

Enabling Collaborative Information Management on Federated Data Sources

Analysis, Design and Prototypical Implementation for MS-SharePoint and MS-Exchange

Authors:	Thomas Reschenhofer
	Bernhard Waltl
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Dr. Thomas Büchner
	Christian Neubert
Date:	August 15, 2011



Wir versichern, dass wir diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet haben.

München, den 13. August 2011 Thomas Reschenhofer Bernhard Waltl

Abstract

Nowadays information is an important commodity, especially for IT aided companies. Thus, information management is an essential challenge to be faced. There are many systems which enable operational information management, such as Enterprise Content Management Systems, Database Systems or simple File Systems.

Therefore a single company may consist of a whole system topography, each system containing several types of information. But a large amount of systems leads to a complex situation, in which it is very difficult to find a specific information.

One possibility to face this problem is to introduce a system, which integrates and joins several data sources. This system provides a centralized and uniform entry into the jungle of information in a company. This approach is called data integration.

This thesis covers the theoretical aspects of data integration as well as an prototypical implementation of such an data integration approach. The existing system *Tricia*, a software product from infoAsset, gets the ability to import data from the following federated data sources: Microsoft SharePoint 2010 and Microsoft Exchange 2010. During the prototypical implementation, there were many requirements that needed to be considered, such as effective synchronization or flexibility.

The result of the thesis is a plugin for Tricia, which enhances this Enterprise 2.0 platform to a centralized system amongst a matured system landscape.

Zusammenfassung

Information ist in der heutigen Gesellschaft und speziell im Umfeld IT-gestützter Unternehmen ein bedeutendes Gut, demzufolge ist dessen Management eine wichtige Aufgabe. Viele Unternehmen verdienen ihr Geld damit, Systeme zu entwickeln und zu verkaufen, welche betriebliches Informationsmanagement ermöglichen, z.B. Enterprise Content Management Systeme, Datenbanksysteme oder einfache Dateiablagen.

Nicht selten findet man also in Unternehmen eine ganze Landschaft an Systemen vor, welche verschiedenste Arten von Information verwalten, was vorwiegend dazu führt, dass es für Benutzer schwieriger wird, eine bestimmte Information im Unternehmen zu finden.

Eine Möglichkeit, mit diesem Problem umzugehen, ist es, ein System zu bestimmen bzw. einzuführen, in welchem verschiedene Informationsquellen integriert werden und wodurch dieses zentrale System eine einheitliche Oberfläche für den Einstieg in den Informationsdschungel des Unternehmens bietet.

Diesem Ansatz der Datenintegration widmet sich diese Bachelorarbeit, wobei neben einer theoretischen Sicht auf die Probleme und Herausforderungen der Datenintegration zugleich eine prototypische Implementierung stattfindet. Das zentrale System, dargestellt durch das infoAsset-Produkt *Tricia*, integriert dabei Daten sowohl aus Microsoft SharePoint 2010 als auch aus Microsoft Exchange 2010, wobei auf spezielle Anforderungen wie effiziente Synchronisierung und Flexibilität geachtet wurde.

Das Ergebnis der Arbeit ist ein prototypisches Plugin für Tricia, welches diese Enterprise 2.0 Plattform zu einem zentralen System inmitten einer ausgewachsenen Anwendungslandschaft steigern lässt.

Inhaltsverzeichnis

Übersicht

KAPITEL 1: ZIELE DER ARBEIT UND PROBLEMSTELLUNG

Dieses Kapitel beschäftigt sich mit den konkreten Zielen der Arbeit und bietet eine detaillierte Erläuterung der Problemstellung.

Teil I: Einführende Theorie

KAPITEL 2: KOLLABORATIVES INFORMATIONSMANAGEMENT

Dieses Kapitel bietet eine Einführung in das Thema "Kollaboratives Informationsmanagement", greift besondere Aspekte zu diesem Thema auf und führt anschließend den Begriff "Enterprise 2.0" ein.

KAPITEL 3: DATENINTEGRATION

Kapitel 3 widmet sich dem Begriff der "Föderierten Datenquellen" und zeigt bekannte Probleme und Herausforderungen im Kontext der Datenintegration auf.

KAPITEL 4: DAS KONZEPT DER HYBRIDS

In diesem Abschnitt ist eine Einführung in das Thema "Hybrids" zu lesen, wobei hier zusätzlich auch spezielle Hybrid-bezogene Features aus Tricia wie z.B. Type Tag Definitionen behandelt werden. Diese sind insbesondere im weiteren Verlauf der Arbeit interessant, da diese Technologie zur Repräsentation externer Daten eingesetzt wird (siehe Kapitel 11).

Teil II: Einführung und Analyse verwendeter Systeme

KAPITEL 5: TRICIA

Dieses Kapitel ist eine Einführung in das infoAsset-Produkt Tricia. Dabei handelt es sich um ein Enterprise Content-Managementsystem, das im Rahmen dieser Arbeit um die Fähigkeit zur Datenintegration aus Fremdsystemen erweitert wird. Neben einem allgemeinen Überblick über das System wird insbesondere die Architektur auf der Store-Ebene detailliert erläutert.

KAPITEL 6: MICROSOFT SHAREPOINT 2010

In diesem Abschnitt findet eine Erläuterung von SharePoint 2010, einem Informationsmanagementsystem von Microsoft, statt. Dabei wird zum Einen auf die 6 Kerndienste von SharePoint eingegangen, zum Anderen werden die angebotenen Kommunikationsmöglichkeiten mit Drittsystemen angesprochen, wobei speziell der OData-Service, welcher in der Arbeit verwendet wird, genauer erläutert wird.

KAPITEL 7: MICROSOFT EXCHANGE 2010

Der Exchange Server 2010 ist ein sehr umfangreiches Software Produkt von Microsoft. Neben seinem bekannten Nachrichtendienst hat er sich den letzten Jahren auch als Group-

waresystem etabliert. Er verwaltet nicht nur E-Mail Nachrichten, Kontaktdaten, Termine und Aufgaben, sondern unterstützt Unternehmen auch bei ihren Geschäftsprozessen.

Teil III: Schicht 1: Schnittstelle zu externen Datenquellen

KAPITEL 8: GENERISCHE SCHNITTSTELLE ZU DATENQUELLEN

Kapitel 8 behandelt die Implementierung einer abstrakten Schnittstelle zu externen Datenquellen. Diese verfügen über die gesamte Information, die notwendig ist, um auf ein externes System zuzugreifen. Die Architektur der Schnittstelle soll eine einfache Möglichkeit bieten, neue externe System hinzuzufügen und in Trica verwenden zu können.

KAPITEL 9: SCHNITTSTELLE ZU MS SHAREPOINT: ODATACONTAINER

Dieses Kapitel erläutert die Implementierung der Schnittstelle zu MS SharePoint, wobei SharePoint rein als sog. *OData-Producer* behandelt wird und all dessen Listen und Bibliotheken, also die konkreten Datenquellen in SharePoint, entsprechende OData-Objekte darstellen. Als Client-Framework wird hierbei das odata4j-Framework [19] eingesetzt, welches ebenfalls in diesem Kapitel eingeführt wird.

KAPITEL 10: SCHNITTSTELLE ZU MS EXCHANGE: EXCHANGECONTAINER

Kapitel 10 befasst sich mit der Umsetzung der Schnittstelle zu MS Exchange. Im Zusammenhang mit dieser Implementierung musste eine geeignete API, zum Zugriff auf die Webservices des MS Exchange Server, ermittelt werden. Die freie Microsoft API "EWS Java API" hat sich dabei gegenüber eines kostenpflichtigen Produkts durchgesetzt.

Teil IV: Schicht 2: Repräsentation externer Daten in Tricia

KAPITEL 11: EVALUIERUNG DER MÖGLICHKEITEN ZUR REPRÄSENTATION EXTERNER DATEN

Dieses Kapitel beinhaltet eine Diskussion darüber, welche Möglichkeiten zur Repräsentation von externen Inhalten bestehen, in welchen Aspekten sich diese unterscheiden und welcher Ansatz schlussendlich ausgewählt und realisiert wurde. Als Abschluss bietet dieses Kapitel eine Einführung des sog. External Modells, welches die Grundlage für das weitere Vorgehen lieferte.

KAPITEL 12: EXTERNE DATENQUELLE

In diesem Kapitel geht es um die Implementierung der Repräsentation von externen Datenquellen, dessen Synchronisation mit dem externen System und dessen Oberflächengestaltung. Desweiteren ist die Darstellung und der Abgleich des Schemas externer Datenquellen in Tricia Thema in diesem Abschnitt. Einen wichtigen Part übernehmen hierbei die in Kapitel 4 eingeführten, Hybrid-bezogenen Features wie Type Tag Definitionen und Attribut Definitionen.

1. Ziele der Arbeit und Problemstellung

Ziel der Arbeit ist die Erweiterung des teamorientierten Informationsmanagementsystems Tricia um die Funktionalität der Datenintegration von föderierten Datenquellen. Zu Beginn steht die Ausarbeitung und die Aneignung der notwendigen Grundlagen zu den Themen Informationsmanagement, föderierten Datenquellen und Datenintegration bevor. Dabei wird ein besonderes Augenmerk auf die damit verbundenen Herausforderungen gelegt. Um diese bei der Implementierung besser berücksichtigen zu können, sind die konkreten Probleme ausgearbeitet und in Kapitel 3.2 zusammengefasst.

Um überhaupt eine konkrete Implementierung durchführen zu können, sind entsprechende Systeme notwendig. In unserer Arbeit konzentrieren wir uns dabei besonders auf das Enterprise 2.0 System "Tricia", das von dem Unternehmen infoAsset entwickelt wurde. Ziel dieser Arbeit ist es, in Tricia die Datenintegration zweier konkreter Systeme zu ermöglichen, wobei es durch einen möglichst generischen Ansatz einfach sein soll, weitere Systeme in Tricia zu einzubinden. Bei den konkreten externen Systemen handelt es sich um zwei Produkte, die häufig in Unternehmen eingesetzt werden: Microsoft SharePoint 2010 und Microsoft Exchange Server 2010. Die Softwareprodukte aus dem Hause Microsoft haben sich bereits über viele Jahre in Unternehmen etabliert und sind so zu einem integralen Bestandteil dieser geworden (Eine Erläuterung aller Systeme und deren Kommunikationsmöglichkeiten mit Drittsystemen ist in Teil II zu lesen).

Zur Umsetzung der Datenintegrationsfunktionalität in Tricia ergab sich eine Architektur, welche wir auf zwei Schichten aufgeteilt haben. Dabei beschreibt die erste Schicht (Teil III der Arbeit) die Schnittstelle zu externen Systemen und bietet der zweiten Schicht (Teil IV der Arbeit), welche die Repräsentation der externen Daten in Tricia auf Datenmodell-Ebene und deren GUI behandelt, eine einheitliche Sicht auf deren Daten. Während dieser Implementierung wurden jederzeit die bereits erwähnten Anforderungen an der Datenintegration berücksichtigt.

Teil I.

Einführende Grundlagen

2. Kollaboratives Informationsmanagement

Nachdem bereits der Titel der Arbeit die Bezeichnung "Kollaboratives Informationsmanagement" enthält, wird dieser Begriff im folgenden Kapitel eingeführt. Dieser Abschnitt bildet die Grundlage zu Kapitel 3, welches sich im Wesentlichen mit den theoretischen Grundlagen dieser Arbeit befasst. Des Weiteren werden bestimmte Aspekte des Informationsmanagements erläutert, welche speziell für den weiteren Verlauf der Arbeit entscheidend sind.

2.1. Teamorientiertes Informationsmanagement in Unternehmen

Informationsmanagement an sich ist ein sehr weitläufiger Begriff, welcher in der Fachliteratur über keine eindeutige Definition verfügt. Vielmehr gibt es mehrere kontextabhängige Ansätze, Informationsmanagement zu beschreiben [1]. Einer dieser Ansätze ist das sog. *Persönliche Informationsmanagement* (PIM), welches sich zum Ziel setzt, persönliche Informationsartefakte wie Dokumente und eMails in geeignter Art und Weise zu verwalten und Aktivitäten wie die Suche zu vereinfachen [27].

Eine Erweiterung des persönlichen Informationsmanagements ist das *kollaborative Informationsmanagement* (auch Gruppeninformationsmanagement, [10]). Hierbei sollen Informationen nicht nur für den eigenen Gebrauch verwaltet werden, sondern für alle Mitglieder einer bestimmten Gruppe zugänglich gemacht werden. Mit dem Einsatz von GIM (Gruppeninformationsmanagement) ist es des Weiteren möglich, Aktivitäten von Gruppenmitgliedern verfolgen und gemeinsame Aktivitäten, beispielsweise gemeinsame Termine, verwalten zu können. Viele GIM Systeme erlauben es zudem, die Verfügbarkeit der Gruppenmitglieder zu überprüfen, bzw. über eigene Kommunikationskanäle mit diesen in Kontakt zu treten.

Der Sinn und Zweck von kollaborativem Informationsmanagement in einem Unternehmen ist es, die Zusammenarbeit innerhalb eines Teams, aber auch die Kommunikation zwischen mehreren Teams zu verbessern [4]. So ist es vorstellbar, dass im Kontext einer Abteilung oder eines Projektteams ein gemeinsamer Pool an Informationen definiert wird. Als Paradebeispiel gilt hierfür der gemeinsame Terminkalender zur Vereinbarung von Teammeetings.

2.1.1. Zugriffsberechtigungen

Da bei kollaborativem Informationsmanagement mehrere Personen anstatt eines Individuums betrachtet werden, wird dem Thema Sicherheit eine größere Bedeutung zuteil als es bei einfachen PIM der Fall ist: Jede Person muss entscheiden können, welche seiner Informationen vertraulich sind und welche Informationen öffentlich, d.h. für Teammitglieder

einsehbar sein sollen.

Desweiteren sind mehrere Abstufungen beim Zugriff auf die Daten einer anderen Person möglich, d.h. ein Individuum soll festlegen können, ob seine Informationen nur lesbar oder sogar bearbeitet werden sein sollen.

Erlaubt ein GIM System zusätzlich die Verteilung bestimmter Rechte nur auf bestimmte Personen, so geht mit kollaborativem Informationsmanagement zugleich ein komplexes Benutzer- und Berechtigungsmanagement einher.

2.1.2. Struktur der Information

Insbesondere für die Suche ist es von entscheidender Bedeutung, wie Informationen vorliegen. In diesem Kontext kann Information in folgende Klassen eingeteilt werden:

Unstrukturiert Unstrukturierte Daten folgen keinem bestimmten Format und liegen entweder als Text oder als Binärstrom (z.B. Video oder Bild) vor.

Strukturiert Daten, welche in festen Feldern atomaren Datentyps (z.B. Integer) vorliegen, werden als strukturiert bezeichnet [23]. Die Felder werden oft "Attribute" und die Gesamtheit dieser Felder "Entität" genannt. Ein Beispiel für strukturierte Datenhaltung ist eine Datenbanktabelle. Unstrukturierte Information lässt sich als Entität mit genau einem Attribut (Text oder Binärstrom) darstellen.

Semi-strukturiert Semi-strukurierte Daten zeichnen sich dadurch aus, dass sie über kein fixes Schema verfügen, sondern ihre (meist verschachtelte und heterogene) Struktur implizit aus den Daten selbst hervorgeht [17]. Beispiele für semi-strukturierte Daten sind Informationen, welche in XML oder JSON vorliegen. Oft wird Semi-Strukturiertheit auch als eine Mischform aus Unstrukturiertheit und Strukturiertheit bezeichnet.

Die Suche strukturierter Information ist um ein Vielfaches präziser und ausdrucksmächtiger als die Volltextsuche bei unstrukturierter Textinformation. Ausdrucksmächtig heißt, dass bei der Suche strukturierter Daten spezielle Ausdrücke bzw. Operationen auf den Daten ausgeführt werden können, wie beispielsweise Vereinigungen *Joins*. Entsprechend steigt der Bedarf an einer strukturierten Suche mit wachsendem Datenaufkommen, da die Suche in unstrukturierten Daten zu unpräzise ist.

Da das Schema der Information bzw. eines Teiles davon jedoch nicht vorab bekannt sein muss, ist es oft sehr schwierig, Information vollkommen strukturiert abzulegen. Als Kompromiss zwischen Performanz der Suche/Indizierung und Flexibilität des Schemas wird daher sehr oft Information semi-strukturiert gespeichert.

2.2. Enterprise 2.0

Enterprise 2.0 bezeichnet das Übertragen des Web 2.0 - Gedanken in den unternehmerischen Kontext und kann prinzipiell als Erweiterung des Gruppeninformationsmanage-

ment gesehen werden.

In diesem Zusammenhang ist vor allem die Wandlung des Benutzers vom passivem Beobachter hin zum aktivem Teilnehmer und Mitgestalter von Bedeutung, wobei hier insbesondere das Konzept der Wikis, Podcasts und Blogs zu erwähnen ist. Berühmt geworden durch die Plattform *Wikipedia*, nutzen viele Unternehmen heutzutage Wikis als Enterprise 2.0 - Konzept zur Wissensweitergabe. Ein besonderes Feature, welches Wikis auszeichnet, ist die Möglichkeit zur Verlinkung mehrerer Wiki-Seiten, sodass neben der im Text enthaltenen Daten zusätzliche Information durch Beziehungen generiert werden kann.

Auch Features wie Tagging (auch Social Tagging) und die Möglichkeit des Kommentierens bestimmter Inhalte werden Web 2.0 bzw. im Unternehmensbereich Enterprise 2.0 zugeordnet und erlauben Benutzern solcher Plattformen, aktiv an der Generierung und Manipulation von Information mitzuarbeiten.

Nahezu jedes moderne Informationsmanagementsystem integriert all diese Web 2.0 Features bereits und hat somit eine Wandlung zur Enterprise 2.0 - Plattform vollzogen, so dass Funktionen, wie man sie beispielsweise aus Plattformen wie Facebook kennt, auch in Unternehmensportalen zu finden sind.

3. Datenintegration

3.1. Definition "Föderierte Datenquellen"

Föderation ist ein gängiges Pattern der Datenintegration das in der Industrie seit Mitte der 80er Jahre verwendet wird. Föderation kombiniert Daten, denen ein verschiedenes Schema zugrunde liegt, zu einer gemeinsamen logischen Struktur. Dabei werden die Daten üblicherweise in relationalen Beziehungen dargestellt. Die entsprechenden Datensätze werden jedoch nicht kopiert oder verschoben, es wird vielmehr eine einheitliche Sicht auf die Daten erstellt bzw. entworfen. Es handelt sich also um ein Konzept, dass darauf abzielt über bestehende Datenbanken ein gemeinsames logisches Schema zu stellen, welches sich dann als eine "virtuelle" Datenbank präsentiert [5].

Die Referenz, die der föderierten Schema-Architektur zugrunde liegt, wurde von Sheth und Larson [22] eingeführt. Es handelt sich dabei um eine Fünf-Schichten-Architektur. Die interne und unterste Schicht entspricht dabei der Speicherung der Objekte in der zugrundeliegenden Datenbank. In der zweiten, konzeptionellen Schicht wird das Datenmodell abgebildet. Diese enthält die Beziehungen zwischen den Objekten und Relationen der Datenbank. Die externe Schicht stellt nun die Schnittstelle der Datenquelle zu anderen Systemen dar, sie definiert welche Teilmenge der möglichen Operationen und Funktionen nach außen exportiert werden. Die globale Schicht bzw. das föderierte Schema, spiegelt die Integration der verfügbaren externen Sichten wieder. Sie enthält dabei die notwendigen Informationen, um die Herkunft der Daten und detaillierte Beschreibungen an die darüberliegende Schicht weiterzuleiten und anbieten zu können. Die oberste Schicht des Integrationsmodells, kommuniziert nun ihrerseits mit der globalen Schicht. Sie definiert für spezielle Anwendungen und Anwendungsfälle die Teilmenge der auszuwählenden Objekte und reduziert somit die Komplexität auf das Notwendigste.
In Abbildung 3.1 wurden die fünf Schichten grün markiert. Des Weiteren wurde das allgemeine Modell auf den in der Arbeit vorliegenden Sachverhalt adaptiert.

Im Zusammenhang mit unserer Arbeit sind die Drittsysteme, aus denen Tricia Daten importiert, föderierte Datenquellen (MS SharePoint und MS Exchange). Diese haben weiterhin die Datenhoheit über die zur Verfügung gestellten Informationen. Der Integrationsprozess in Tricia ermittelt aus den verfügbaren Schemainformationen der externen Systeme selbstständig die resultierende Sicht (siehe Kapitel 8.3.1).

3.2. Anforderungen der Datenintegration

Bei der Datenintegration handelt es sich um einen Prozess, der als Eingabe Daten aus verschiedenen, unter Umständen auch verteilten, Quellen erhält und dessen Ausgabe eine

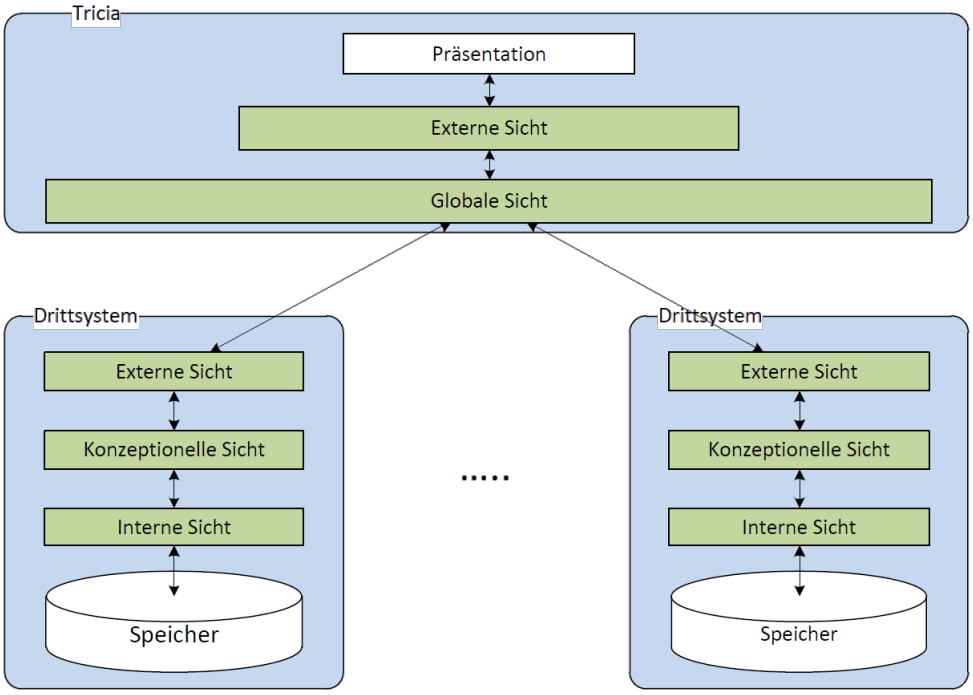


Abbildung 3.1.: Schematische Darstellung des Integrationsmodells föderierter Datenquellen nach Sheth and Larson [22]

einheitliche Sicht auf diese Daten ist. Zum Thema Datenintegration und Enterprise Information Integration existieren zahlreiche Arbeiten, die sich intensiv mit dieser Problematik auseinandergesetzt haben [5] [6] [9]. Im Rahmen dieser Forschungen wurden wesentliche grundlegende Eigenschaften und Anforderungen definiert.

Im Rahmen unserer Aufgabenstellung haben sich die folgenden Anforderungen als besonders wichtig herausgestellt, diese wurden in dem entworfenen Architekturmodell berücksichtigt.

3.2.1. Effizientes Importmanagement

In einem Integrationsprozess gibt es verschiedene Möglichkeiten und Ansätze, wie konkrete Datenbestände übertragen und aus dem Drittsystem importiert werden können. Eine Hauptaufgabe dabei ist es, nur die tatsächlich benötigten Daten zu Übertragen. Um Overhead zu vermeiden sollen nicht benötigte Datensätze sollen schon von der Übertragung ausgeschlossen werden, bevor diese überhaupt erst begonnen hat. Es ist also eine Selektion der Datensätze, auf der Seite des Drittsystems, notwendig.

Datensätze, die bereits importiert wurden, sollten lokal zwischengespeichert werden. Das Importmanagement soll also über eine Art Cache verfügen. Nicht jede Abfrage an ein Fremdsystem soll tatsächlich in einem zeit- und rechenintensiven Aufruf bei dem externen System enden. Es gilt also importierte Daten zu speichern, andererseits soll die Aktualität der lokal vorliegenden Datensätze gewährleistet sein.

3.2.2. Flexibilität

Eine entscheidende und wichtige Anforderung an die Datenintegration betrifft dessen Flexibilität. Dabei kann zwischen zwei Varianten unterschieden werden:

Erweiterbarkeit Neue externe Systeme sollen flexibel und möglichst einfach, das bedeutet ohne großen Aufwand, in die bestehende Applikation zu integrieren sein. Die davon betroffenen Architekturkonzepte sollen daher einen möglichst generischen Ansatz verfolgen, die diese Anpassungen und Erweiterungen unterstützen.

Anpassbarkeit Externe Systeme, aus denen bereits erfolgreich Daten importiert wurden, können sich unter Umständen ändern. Diese Änderungen können selbstverständlich auch das Schema der zu importierenden Daten betreffen. Das Architekturmodell soll gegenüber diesen Änderungen zum einen eine gewisse Robustheit besitzen und zum Anderen soll der Aufwand diese Änderungen in dem zentralen System einzuarbeiten möglichst gering sein.

3.2.3. Benachrichtigungen durch Fremdsysteme

Um Aktualität und Integrität der importierten Daten zu gewährleisten, muss sichergestellt werden, dass diese weder veraltet sind, noch auf dem Drittsystem gelöscht wurden. Dieses Problem ist bekannt und dazu haben sich diverse Lösungsansätze entwickelt.

Eine mögliche Lösung ist die Daten zyklisch aus dem System abzufragen und die lokalen Datensätze mit den aktuellen Datensätzen zu überschreiben. Eine Verbesserung dieser Methode besteht darin, nur die tatsächlichen Änderungen zu erkennen und zu importieren, um dann die korrespondierenden lokalen Datensätze zu aktualisieren. Dieses Konzept wird als Polling-Modell bezeichnet.

Dem gegenüber steht das sogenannte Push-Modell. Dabei erkennt das Drittsystem die Änderung in den Datensätzen und benachrichtigt dann die zugreifenden Applikationen über die abgelaufene Gültigkeit dieser Datensätze. Je nach Technologie und verwendetem System wird die Applikation dazu aufgefordert die aktuellen Daten zu laden. Die Entscheidung, wann die Daten tatsächlich abgerufen werden, obliegt dem Client.

Diese Anforderung ist sehr stark von dem verwendeten Drittsystem ab. Nicht alle möglichen Datenquellen bieten einen derartigen Benachrichtigungs-Dienst an. Oft bleibt keine andere Wahl als die Daten zyklisch abzurufen und seinen lokalen Speicher zu aktualisieren.

3.2.4. Zusätzliche Attribute

Eine weitere Anforderung, die gelegentlich an die importierten Objekte gestellt wird, betrifft das Anheften zusätzlicher Attribute. Die Objekte werden mit definierten Eigenschaften und Methoden vom Drittsystem zur Verfügung gestellt. In einem lokalen System besteht oftmals der Wunsch, diese Eigenschaften zu erweitern. Zu den bereits verfügbaren Feldern soll es möglich sein, neue Eigenschaften zu definieren und diese dann an das Objekt anzuheften.

Je nach Anforderung kann es auch gewollt sein, die festgelegten Eigenschaften zurück in das externe System zu schreiben. Diese Option hängt jedoch sehr stark vom Drittsystem ab. Wie bei der Anforderung aus Kapitel 3.2.3 stellen nur wenige externe System Methoden zur Verfügung, solche neuen Werte an bestehende Objekte anzuheften und diese dann entfernt zu speichern.

3.2.5. Typemapping

Eine wesentliche Anforderung, die grundsätzlich alle Importszenarien betreffen, stellt die Konvertierung unterschiedlicher Datentypen von dem Drittsystem zu dem lokalen Typsystem der Daten dar. Es ist immer eine Herausforderung, die importierten Daten mit ihren spezifischen Datentypen, auf die Datentypen zu konvertieren, die im lokalen System verfügbar sind. Dabei muss darauf geachtet werden, dass so wenig Information wie möglich verloren geht und das die Objekte mit ihren Eigenschaften so importiert werden, wie sie auch tatsächlich im externen System vorliegen.

In der Literatur wird dabei vor allem auf die Problematik hingewiesen, die entsteht, wenn aus verschiedenen Plattformen und Systemen, die zum Teil schon älter sind bzw. eine veraltete Technologie verwenden [5]. Datentypen und Speicheranordnungen können in diesen Fällen zu einem Problem werden, das bei der Datenintegration berücksichtigt werden muss.

3.2.6. Authentifizierung

Informationssysteme verwalten oft unternehmenskritische und vertrauliche Daten. Um zu verhindern, dass die Daten in die Hände Unbefugter geraten, werden diese geschützt. Die meisten Informationssysteme stellen Daten also nur zur Verfügung, wenn sich der Benutzer erfolgreich authentifiziert. Genauso verhält es sich auch, wenn ein anderes System Daten anfordern möchte. Es werden Informationen über die Identität benötigt, um Zugriff auf Informationen zu erhalten.

Die Möglichkeit, Angaben über die eigene Identität bzw. über die zu verwendende Identität zu hinterlegen, muss also gegeben sein. Diese Informationen werden dann an das externe System weitergegeben, das dann auf Basis dieser Benutzerinformation entscheidet, ob überhaupt auf Daten zugegriffen werden darf und welche Datensätze dann eingesehen werden können.

3.2.7. Änderungshistorie

Es ist nicht ausgeschlossen, ja sogar anzunehmen, dass sich die Objekte nach einem erfolgreichen Importszenario ändern. Diese Änderung kann nicht nur den Inhalt der Objekte, also die Werte deren Eigenschaften betreffen, sondern auch das Schema der Objekte. Werden bei dem Objekttyp eines externen Systems Änderungen festgestellt, so sollen diese protokolliert werden. Das lokale System soll in der Lage sein, die betroffenen Felder und Eigenschaften zur Laufzeit zu erkennen und zu verarbeiten. Eine Aufzeichnung und Mitteilung an einen Benutzer sind aber durchaus auch wünschenswert. Idealerweise wird dabei ermittelt welche Eigenschaften sich geändert haben und, soweit vorhanden, auch welche Instanz die Änderung durchgeführt hat. Die gesammelten Änderungen werden gespeichert. Somit ist nachvollziehbar, wie sich ein Objekt und der Inhalt davon seit dem Import im lokalen System verändert hat.

4. Das Konzept der Hybrids

Da Hybrids im weiteren Verlauf (speziell in Teil IV der Arbeit) eine tragende Rolle spielen, wird dieses Konzept in diesem Kapitel eingeführt und dessen Aspekte erläutert. Als Grundlage dient dabei der Artikel "Hybrid Wikis: Empowering Users to Collaboratively Structure Information" von Florian Matthes, Christian Neubert und Alexander Steinhoff [11].

4.1. Einführung

Das Wort Hybrid ist griechischen Ursprungs und bedeutet im wesentlichen "Etwas Gebündeltes, Gekreuztes oder Gemischtes" [25]. Im Kontext des Informationsmanagements und speziell in dieser Arbeit ist dabei das Versehen unstrukturierter Informationsobjekte mit Struktur (zu einem beliebigem Zeitpunkt) gemeint. Diese Struktur kann beispielsweise aus einem Name-Wert-Paar (Attribut) oder einer Typinformation (sog. Type Tag) bestehen, wodurch sich die Semantik des ursprünglichen Objekts ändert. Beispielsweise kann einem Objekt des Built-In-Typs "Person" als Hybrid das Name-Wert-Paar "Abteilung = Marketing" und der Typ "Angestellter" zugewiesen werden.

Zweck des Einsatzes von Hybrids ist die Präzisierung der Suche. Stelle man sich z.B. eine Menge von Projektbeschreibungen realisiert als Wikiseiten vor, welche zwar über Built-In - Attribute wie Titel und RichText-Inhalt enthalten, nicht aber das Startdatum des Projekts, so ist es nicht möglich, genau jene Projekte eines bestimmten Zeitraumes zu finden. Wird die Wikiseite als Hybrid angelegt, können Benutzer beliebige Attribute (z.B. "Startdatum") und Type Tags (z.B. "Projekt") anheften und somit die Wikiseite mit zusätzlicher Struktur versehen.

Attribute

Ein Attribut ist im Kontext von Hybrids ein einfaches Name-Wert-Paar, wobei der Name ein einfacher String und der Wert grundsätzlich beliebigen Typs sein kann. Ein Hybrid besitzt also eine ganze Liste von Attributen, welche ein Benutzer jederzeit erweitern und ändern kann.

Type Tags

Mithilfe von Type Tags können Benutzer bestimmte Objekte klassifizieren, d.h. sie können Objekten einen oder mehrere Typen zuweisen. Eine wichtige Funktion dieser Type Tags ist es, Ähnlichkeiten zwischen Objekten festzustellen um damit Vorschläge für die Angabe von Attributen zu machen.

Sei beispielsweise ein Objekt A mit dem Attribut “Startdatum” gegeben, welches über den Type Tag “Projekt” verfügt. Wird ein neues Objekt B erzeugt und diesem ebenfalls der Type Tag “Projekt” zugewiesen, so wird zwischen den Objekten A und B eine Ähnlichkeit festgestellt und für das Objekt B das Attribut “Startdatum” vorgeschlagen.

Mit diesen Vorschlägen können Benutzer dazu angeregt werden, Objekte zu strukturieren, so dass bei immer größer werdenden Datenaufkommen eine präzise Suche möglich ist.

4.2. Type Tag Definitionen und Attribut Definitionen

In einem Informationssystem wie dem infoAsset-Produkt Tricia (siehe Kapitel 5) wird das Hybrid-Konzept in Form von Hybrid Wikis in den Mittelpunkt gestellt und durch spezielle Features erweitert.

Eines dieser Features von Tricia ist die Möglichkeit von Type Tag Definitionen. Solch eine Type Tag Definition beschreibt einen speziellen Type Tag und enthält eine beliebige Menge an Attribut Definitionen, welche sich auf die Attribute aller hybriden Objekte beziehen, welche mit dem entsprechenden Type Tag versehen sind. Diese Attribut Definitionen charakterisieren einzelne Attribute, indem sie eine Menge von Validatoren festlegen. Diese Validatoren können beispielsweise den Typ eines Attributs oder dessen Multiplizität festlegen. Nicht-valide Attributwerte können entsprechend mit einer Fehlermeldung bzw. Warnung dargestellt werden, bzw. bei der Erzeugung eines Attributs gar nicht als Wert zugelassen werden.

Zusätzlich zu diesen Valdierungen dienen diese Attribut Definitionen wiederum der Generierung von Attribut-Vorschlägen für bestimmte Objekte. Existiert beispielsweise eine Type Tag Definition mit dem Namen “Projekt”, welches über eine Attribut Definition “Startdatum” verfügt (optional auch als Datumstyp), so kann für jedes Objekt, welchem der Type Tag “Projekt” zugeordnet ist, das Attribut “Startdatum” vorgeschlagen werden.

Type Tag Definitionen entsprechen also im Grunde einer Schemabeschreibung, wobei ein Schema jederzeit durch Hinzufügen eines Type Tags auf ein Objekt angewandt werden kann.

Abbildung 4.1 zeigt das Hybrid Wiki Modell nach Matthes, Neubert und Steinhoff, welches den Zusammenhang zwischen hybriden Informationsobjekten, in diesem Fall hybriden Wikiseiten, und den Type Tag Definitionen inklusive deren Attribut Definitionen darstellt.

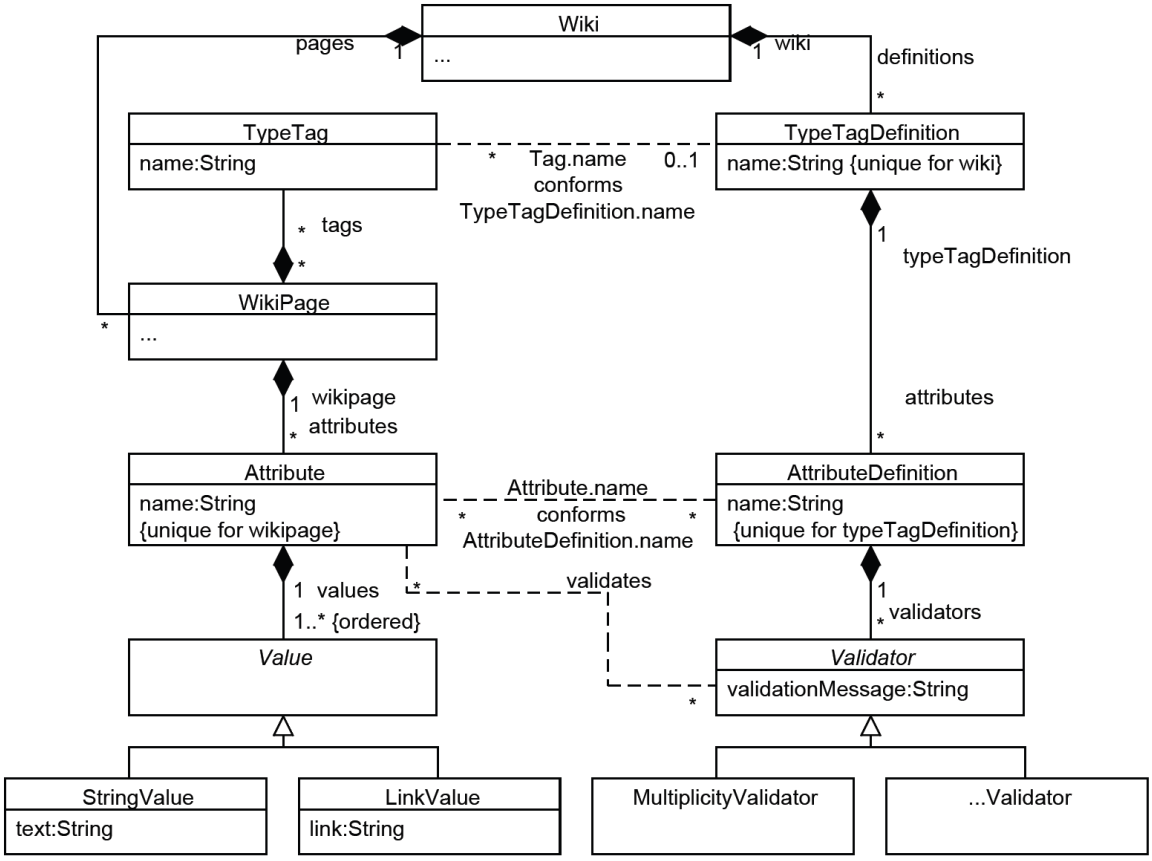


Abbildung 4.1.: Das Hybrid Wiki Modell nach Matthes, Neubert und Steinhoff [11]

Teil II.

Einführung und Analyse verwendeter Systeme

5. Tricia

5.1. Beschreibung

Bei Tricia handelt es sich um eine Java-Plattform, die dazu verwendet werden kann, Enterprise 2.0 Systeme zu realisieren. Die vom Unternehmen InfoAsset AG entwickelte Technologie unterstützt viele gängige Web 2.0 Funktionen in einer integrierte Plattform. Insbesondere Wikis, Blogs, sozial Netzwerke und das Austauschen von Dokumenten [24].

5.2. Architektur

Die Architektur, mit der Tricia entwickelt wurde, stammt ursprünglich aus einem anderen System: der Toro Plattform. Dieses Framework wurde unter Verwendung des Model-View-Controller Architekturmuster entworfen. Dabei werden drei voneinander unabhängige Einheiten definiert, die durch ihre lose Kopplung einen flexiblen Programmentwurf und gute Wiederverwendbarkeit ermöglichen. Des Weiteren bietet dieses Konzept eine gute Möglichkeit auftretende Änderungen und Erweiterungen unkompliziert umzusetzen (siehe Kapitel 5.3).

Bei diesen drei Einheiten handelt es sich um:

- Modell (engl. model)
- Präsentation (engl. view)
- Programmsteuerung (engl. controller)

Modell

Das Datenmodell, der in der Datenbank abgelegten Objekte, wird durch Assets dargestellt. Es handelt sich dabei um ein objektrelationales Mapping. Assets haben Eigenschaften (sog. Properties) die einem bestimmten Typ entsprechen und sie können miteinander in Beziehung stehen (sog. Roles). Jede Asset Klasse wird in der Datenbank abgelegt, wobei dessen Eigenschaften die Attribute eines Elements in der Datenbank definieren. Die Beziehungen werden ebenfalls in der Datenbank erfasst. Es existieren drei verschiedene Typen von Beziehungen:

one-to-one und one-to-many Sind die Assets über diese Beziehungen verknüpft, werden die entsprechenden Primärschlüssel als Fremdschlüssel in der Tabelle erfasst.

many-to-many Diese Art der Beziehungen werden durch Kreuztabellen realisiert. Diese verwalten in einem Datensatz zwei Fremdschlüssel und sind somit in der Lage, die in Beziehung stehenden Einträge eindeutig zu identifizieren.

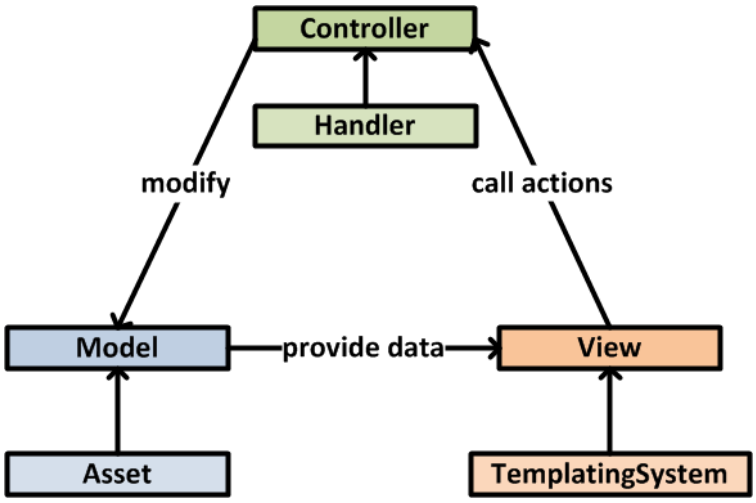


Abbildung 5.1.: MVC - Architekturmuster angewendet auf Tricia [2]

Präsentation

Bei einer Template Engine handelt es sich um eine Software, die vordefinierte Dateien, die üblicherweise HTML - Seiten beschreiben, mit Inhalt füllt. Es müssen Platzhalter definiert sein um die Inhalte in die HTML Seite einzufügen. Die Platzhalter werden durch den zur Laufzeit ermittelten Inhalt ersetzt. In Tricia wird dabei auf das Toro Template System zurückgegriffen. Es handelt sich also um ein technisches Konzept, das es ermöglicht, Webseiten dynamisch, zu generieren.

Die Benutzerinteraktionen finden über Webseiten statt, die Tricia jeweils zur Laufzeit erstellt. Der Inhalt ist vom Anwendungsfall abhängig. Somit ist die Schnittstelle zum Benutzer, die Oberfläche, von der Ebene der Logik entkoppelt. Die Werte, die die Kontrolleinheit oder das Modell zur Verfügung stellen, werden an das Template System weitergegeben, welches dann die definierten Platzhalter durch die tatsächlichen Daten ersetzt. In Tricia sind folgende Platzhalter möglich [2]:

Print Substitutions Dabei handelt es sich um Platzhalter, die durch eine einfache Zeichenkette ersetzt werden.

Conditional Substitutions Diese Art der Platzhalter ermöglichen es, ähnlich zu if-Anweisungen, Bedingungen zu definieren, die darüber entscheiden welcher Wert ausgegeben wird.

List Substitutions Um eine Tabelle, in der jede Zeile die gleiche Form besitzt, auszugeben, werden sogenannte List Substitutions verwendet. Dabei wird eine Liste variabler Länge von Datenobjekten in den Platzhalter eingesetzt.

Template Substitutions Um ganze Templates in Platzhaltern einzufügen kann diese Substitution verwendet werden, wodurch eine beliebige Schachtelung ermöglicht wird.

Programmsteuerung

Aufrufe der Webapplikation werden durch HTTP Requests von Clientbrowsern initiiert. Der Web Server nimmt diese Aufrufe entgegen und leitet sie der Java-Applikation weiter. Die Aufrufe werden dort von sog. Handlern entgegengenommen und verarbeitet. Die HTTP Nachrichten werden ausgewertet und eine korrespondierende Antwort generiert. In den Handlern ist die Logik einer Applikation implementiert. Diese repräsentieren deshalb die Controllereinheit des MVC - Architekturmusters.

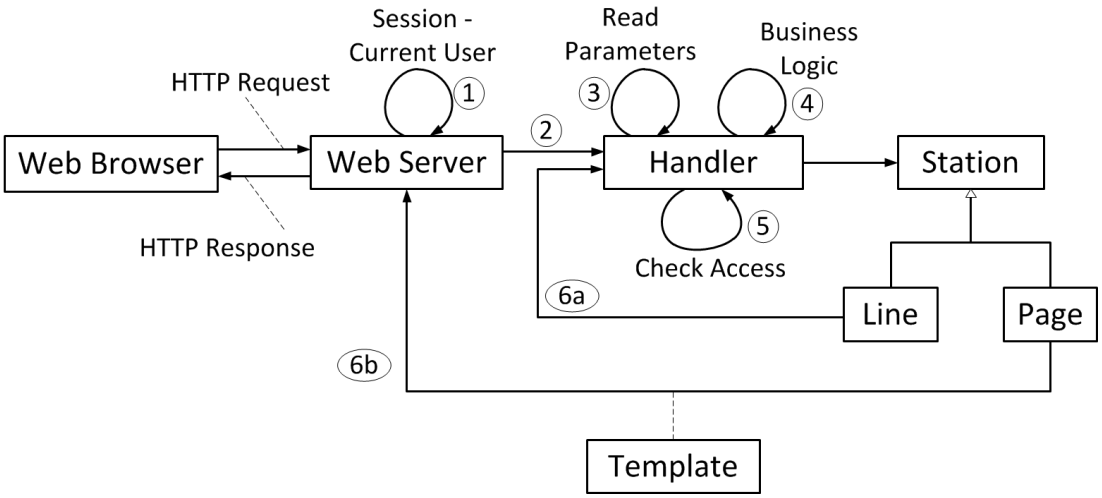


Abbildung 5.2.: Schematischer Ablauf der Verarbeitung einer Clientanforderung

In Abbildung 5.2 ist der schematische Ablauf visualisiert. Es sind verschiedene Schritte notwendig, bis der Server auf einen HTTP Request vom Client antworten kann.
Folgende Schritte werden bei jedem Aufruf durch einen Client durchgeführt:

1. Der HTTPServer erhält einen, vom Clientbrowser initiierten, HTTP Request und ordnet diesem einer bestehenden Session zu oder erstellt eine neue Session.
2. Der HandlerDispatcher ermittelt aus der HTTP Nachricht den zuständigen Handler und leitet die Clientanforderung an diesen weiter.
3. Der Handler verarbeitet die HTTP Nachricht. Er liest alle darin enthaltenen Parameter aus, dazu verwendet er die Methode getParameters(). Die Parameter sind in der URL enthalten.
4. Der Handler überprüft, ob der Client überhaupt die Berechtigung hat diesen Handler aufzurufen. In der Methode checkAccess() werden die notwendigen Routinen durchgeführt, dabei wird die Session des Clients benötigt.
5. Nachdem alle vorausgehenden Schritten erfolgreich verlaufen sind, arbeitet der Handler in der Methode doBusinessLogic() den Aufruf ab. In dieser Methode ist die Logik der Applikation implementiert. Nach erfolgter Bearbeitung wird ein Objekt vom Typ Station generiert.

6. a) Der Handler kann die Bearbeitung an einen anderen Handler weiterleiten. In diesem Fall wird ein Objekt vom Typ Line erzeugt.

   b) Ist das Station Objekt aber vom Typ Page, so erhält der Web Server dieses Objekt, was in der Regel eine Webseite ist, erstellt daraus den HTTP Response und sendet diesen zurück an den Client.

HandlerDispatcher

In Schritt 2 ermittelt der HandlerDispatcher aus einer HTTP - Nachricht den zuständigen Handler, deshalb benötigt er zusätzliche Informationen. Die zur eindeutigen Zuordnung notwendige Information steckt in der URL der HTTP - Nachricht. Die Handler beinhalten ihrerseits HandlerPattern Objekte, in denen spezifiziert wird, welche URLs sie verarbeiten. Der Dispatcher vergleicht die URL aus der HTTP - Nachricht mit den HandlerPattern Objekten aller Handler und leitet den Anforderung dann an den zuständigen Handler weiter. Das HandlerPattern Objekt ermöglicht das Mapping auf generische URLs, da auch sogenannte *regular Expressions* möglich sind.

5.3. Plugins

Die Java-Applikation Tricia besteht aus einem Core und mehreren Plugins, die die möglichen Funktionalitäten, wie Wikis, Blogs, . . . , implementieren. Im Core sind alle Anforderungen umgesetzt, auf denen andere Plugins aufsetzen können. Diese umfassen unter anderem das Verwalten von Benutzern, Gruppen, Login und Registrierung.

Die Plugins können modular zusammengesetzt werden. Ein Plugin kann selbst wieder von anderen Plugins abhängen. So existiert zum Beispiel ein Wiki-Plugin das die Wikifunktionalität in Tricia implementiert und vom File-Plugin abhängt. Das Wiki-Plugin verwendet wiederum die Benutzeridentitäten, die vom Core zur Verfügung gestellt werden, um dokumentieren zu können, welcher Benutzer Änderungen durchgeführt hat.

Dieses Konzept der Plugins ermöglicht also auch eine flexible Erweiterung durch neue Plugins. Diese können fast beliebig in die Applikation eingebunden werden um den Funktionsumfang zu erweitern.

5.4. Bestehende Architektur auf Store Ebene

Eine der Hauptaufgaben eines jeden Enterprise 2.0 Systems ist natürlich das Bereitstellen von Informationen. Zu diesem Zweck muss eine entsprechende Persistierung in einen lokalen Datenspeicher vorhergegangen sein. Diese Aufgaben fallen auch in Tricia an. Ein Konzept und die damit verbundene Architektur ist deshalb auch in Tricia implementiert.

Abbildung 5.3 zeigt den schematischen Überblick der Klassen, die an dem Store-Konzept in Tricia beteiligt sind.

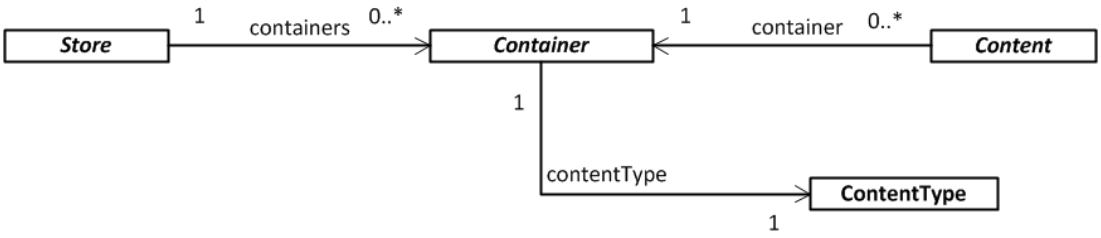


Abbildung 5.3.: Schematischer Überblick über das Store-Konzept in Tricia

Store

Die Klasse Store repräsentiert den direkten Zugriff auf die Datenbank. Sie beinhaltet Informationen über den Datenspeicher, der der gesamten Applikation zugrunde liegt. Des Weiteren ist diese Klasse auch dafür verantwortlich Tabellen in der Datenbank anzulegen und bestehende Tabellen zu löschen. Sie verwaltet auch die Verbindung zur Datenbank.

Die Store-Klasse wurde als Singleton entworfen. Bei diesem Entwurfsmuster handelt es sich um ein Konzept der Softwareentwicklung, das verhindern soll, dass ein Objekt dieses Typs öfter als einmal erzeugt werden kann. Dieses Vorgehen ist vor allem dann üblich, wenn der Zugriff auf eine Ressource nur von einem zentralen Objekt geschehen soll. Dies ist hier der Fall, man erreicht damit, dass der Zugriff auf den Datenspeicher nur von einem zu beginn erzeugten und global verfügbaren Store Objekt vorgenommen wird.

Container

Ein Container Objekt repräsentiert gewissermaßen die Tabellen, die in einem Store vorhanden sind. Daraus ergibt sich auch die Beziehung zum Store Objekt. Die Multiplizität zwischen der Store Instanz und der Container Klasse ist 1 zu N. Das bedeutet, dass jeder Container auch einem (dem einzigen) Store zugeordnet ist. Umgekehrt beinhaltet der Store mehrere, unter Umständen auch kein einziges, Container Objekt.

Das Container-Objekt verfügt auch über die notwendige Informationen des Datenspeichers, der in der Applikation verwendet wird. Es verwaltet den Zugriff und die Bereitstellung der enthaltenen Content-Objekte und stellt diese auch zur Verfügung.

Welche Attribute und Elemente ein Container Objekt beinhaltet wird durch den *ContentType* festgelegt. Die Container Objekte werden beim Start der Applikation erstellt. Dabei wird zur Laufzeit ermittelt, welche Container Objekte angelegt werden müssen.

ContentType

Die bereits erwähnten Container Objekte stellen in dem Datenmodell die Tabellen dar. Die Attribute und der Datentyp der Attribute, also das Schema, wird jedoch nicht im Container-Objekt verwaltet. Diese Aufgabe übernimmt das ContentType-Objekt. Diese Klasse enthält eine Liste mit allen verfügbaren Elementen und speichert zusätzlich noch

die zugeordneten Datentypen.
Mögliche Datentypen sind:

- INT_TYPE
- STRING_TYPE
- DATE_TYPE
- BYTESTREAM_TYPE
- CHARSTREAM_TYPE
- BOOLEAN_TYPE
- TIMESTAMP_TYPE

Diese Datentypen werden vom Store-Objekt ausgelesen. Das Store-Objekt besitzt auch die Information, die notwendig ist, um das Attribut der Tabelle hinzuzufügen. Es kennt die zugrunde liegende Datenbank und weiß somit welchen Datentyp ein Attribut, das z.B. einen Integerwert darstellt, in der Zieldatenbank besitzt.

Da ein ContentType genau einen Container beschreibt und umgekehrt die Attribute eines Containers durch einen ContentType definiert wird, ist die Multiplizität zwischen diesen beiden Objekten offensichtlich. Es handelt sich um eine 1 zu 1 Beziehung (siehe Abbildung 5.3).

Content

Neben den bisher genannten Objekten Store, Container und dem ContentType existiert noch ein viertes Objekt, das einen wesentlichen Beitrag zur Realisierung der Store-Architektur beiträgt: das Content-Objekt. Das Store-Objekt kümmert sich um die Datenbank und den Zugriff darauf, das Container-Objekt stellt gewissermaßen eine Tabelle in der Datenbank dar und der ContentType definiert diese Tabelle mit den Attributen und deren Datentypen. Ein Eintrag in einer Tabellen wird jetzt durch das Content-Objekt repräsentiert. Die Content-Objekte beinhalten also die konkrete Information eines in der Datenbank abgelegten Elements. Es verfügt neben einer eindeutigen ID auch über eine Menge von Attributen und den zugewiesenen Werten. Diese Attribute können gesetzt und abgefragt werden. Des Weiteren trägt jeder Content auch die Information wann der Inhalt zuletzt geändert wurde.

Die Multiplizität ergibt sich nun aus der Architektur. Stellt der Container, wie bereits gezeigt wurde, die Tabelle und der Content die Einträge in dieser Tabelle dar, so ergibt sich für die Beziehung der beiden Objekte die Multiplizität 1 zu N. Eine Tabelle enthält mehrere, unter Umständen auch kein einziges, Content-Objekt. Jedes Content-Objekt ist hingegen immer genau einem Container zugeordnet.

Abstrakte Klassen und davon abgeleitete Klassen

In dem Store-Konzept von Tricia sind mehrere Klassen als abstract definiert.
Diese sind:

- Store
- Container
- Content

Diese Basisklassen definieren die Funktionen, die jede Spezialisierung implementieren muss. Die Datenbank, die die Applikation zum Ablegen der Objekte benutzt, ist nicht zwingend vorgeschrieben. Es handelt sich also nicht um eine unveränderliche Datenbank für alle Instanzen von Tricia Applikationen. Es kann durchaus vorkommen, dass eine Installation ein Datenbanksystem von einem bestimmten Typ nutzt, ein anderes Unternehmen verwendet für Tricia jedoch eine anderes System. Mit diesem Konzept ist eine entsprechende Möglichkeit gegeben. So sind zum Beispiel relationale Datenbanken wie der Microsoft SQL Server, mySQL, HyperSQL, Oracle oder auch die DB2 von IBM als konkrete Store-Objekte verfügbar. Aber auch die MongoDB, eine dokumentenorientierte Datenbank, wäre als Datenspeicher für Tricia denkbar, da die entsprechende Klasse vorhanden ist.

Da aus den Objekten Container und Content ebenfalls ein Zugriff auf die Datenbank erfolgt, sind diese auch entsprechend spezialisiert in Tricia implementiert. Dabei werden vor allem die syntaktischen Eigenheiten der verschiedenen Datenbanken berücksichtigt. Idealerweise merken die darüberliegenden Schichten nichts davon welche Datenbank bzw. welches Datenbanksystem verwendet wird. Die Architektur kapselt die konkreten Aufrufe die an die Datenbank gestellt werden, wie z.B. Erstellen, Aktualisieren, Lesen oder Löschen eines Datensatzes.

5.5. Zusammenfassung

Bei der Java-Applikation Tricia handelt es sich um ein Enterprise 2.0 Web-Kollaborationssystem. Bei dem zur Implementierung verwendeten Architekturmuster handelt es sich um das sogenannte Model-View-Controller Konzept. Dabei werden die einzelnen Komponenten gekapselt. Das bedeutet die Präsentation ist soweit als möglich unabhängig von der internen Applikationslogik und hängt auch nicht vom verwendeten Datenspeicher ab. Genauso verhält es sich auch mit den zwei weiteren Komponenten, diese arbeiten weitgehendst autonom und kommunizieren miteinander ausschließlich über definierte Schnittstellen.

Die zur Persistierung von Objekten verwendete interne Architektur enthält vier Komponenten, die ein objektrelationales Mapping ermöglichen: Der Store, der die Datenbank repräsentiert, die Container, die die Tabellen einer Datenbank darstellen und der Content, der die einzelnen Einträge in den Tabellen abbildet. Ein zusätzliches Objekt, der ContentType, definiert welche Attribute die einzelnen Tabellen besitzen und welche Datentypen

diese haben. In Abbildung 5.4 sind alle beteiligten Klassen detailliert dargestellt, ebenso die Beziehungen und die dazugehörigen Multiplizitäten.

Das Wissen um die in Tricia verwendeten Architekturmuster ist notwendig, um in dieser Arbeit das bestehende System zu erweitern. Unsere Aufgabenstellung sieht eine solche Erweiterung vor. Mit dem Pluginkonzept von Tricia ist es jedoch sehr komfortabel die Funktionalitäten zu implementieren ohne das bestehende System zu verändern bzw. nur minimale Anpassungen vorzunehmen. Es ist also naheliegend die Datenintegration in einem Plugin zu implementieren, welches die Komponenten von Tricia erweitert und die gestellten Anforderungen erfüllt. Die konkrete Umsetzung des Plugins ist ab Teil III beschrieben. Dort werden auch die erforderlichen Objekte und deren Methoden genau erläutert.

Abbildung 5.4.: Klassendiagramm der am Store-Konzept beteiligten Klassen

6. Microsoft SharePoint 2010

Da Microsoft SharePoint 2010 eine der beiden prototypisch zu integrierenden Datenquellen ist, wird dieses System in diesem Kapitel erläutert. Des Weiteren werden die Kommunikationsmöglichkeiten von SharePoint mit Drittsystemen erwähnt und dabei besonders auf den in Kapitel 9 verwendeten OData-Service eingegangen.

6.1. Beschreibung

SharePoint 2010 ist bereits die vierte Generation von Microsofts Business Plattform und laut einer aktuellen Studie [3] immer noch das im Vergleich zur Konkurrenz am stärksten wachsende ECM System (Enterprise Content Management) weltweit.

Microsoft selbst beschreibt sein Produkt als Vereinigung von 6 Kerndiensten [21], zu sehen in Abbildung 6.1.

Abbildung 6.1.: Die 6 Kerndienste von Microsoft SharePoint 2010

6.1.1. Sites

SharePoint 2010 kann als Portallösung zur Erstellung und Verwaltung von Intranet- und Internetanwendungen genutzt werden. Webseiten lassen sich entweder direkt im Browser oder mithilfe einer Clientanwendung (SharePoint Designer 2010) anpassen. Zusätzlich zu normalem Richtext-Inhalt lassen sich mithilfe sog. *Webparts* komplexe Geschäftsdaten ohne spezielle Webentwickler-Kenntnisse in SharePoint-Seiten integrieren.

Ein besonderer Vorteil von SharePoint 2010 im Vergleich zu Konkurrenzsystemen ist die reibungslose Integration der Clientanwendungen der Microsoft Office-Suite. Nachdem

Office, speziell Word, Excel und Outlook, eine bedeutende Rolle in vielen Unternehmen spielen, ein wichtiges Kriterium bei der Wahl einer entsprechenden Portallösung. In diesem Zusammenhang bietet eine SharePoint-Seite eine gewohnte Umgebung, da das seit Office 2007 bekannte Menüband ("RibbonUI") auch in der Weboberfläche von SharePoint Einzug gehalten hat.

6.1.2. Communities

Wie in jedem Informationsmanagementsystem hat auch in SharePoint Web 2.0 seine Spuren hinterlassen, so dass sich SharePoint 2010 zu einer ausgewachsenen Enterprise 2.0-Plattform entwickelt hat.

Wikis sind zwar schon in Vorgängerversionen möglich gewesen, allerdings wurde dessen Funktionen mit der Einführung von Tags und ganzen Begriffstaxonomien bedeutend erweitert. Überhaupt ist die größte Änderung seit SharePoint 2007 die Einführung des Social Computing mit all seinen Facetten. Jedem Mitarbeiter wird es ermöglicht, sein eigenes Profil zu verwalten und sich mit Kollegen zu verbinden, vergleichbar mit den Funktionen in Facebook. Selbst der bekannte "I Like"-Button ist in SharePoint zu finden.

6.1.3. Content

Einer der wichtigsten Rollen von SharePoint ist jene als Enterprise Content Management System.

Zentrale Elemente in SharePoint sind *Listen* und *Bibliotheken*, welche den Großteil der Information eines SharePoint-Portals tragen. Im Wesentlichen handelt es sich dabei um Tabellen mit entsprechenden Interaktionsmöglichkeiten (siehe Abbildung 6.2).

Auch hier profitiert der Endbenutzer wieder von der Integration von SharePoint in Office, da sich viele Aktionen (z.B. Ein- und Auschecken von Dokumenten) alleine über die Clientanwendung (z.B. Word) ausführen lassen.

SharePoint wäre kein ECM System, wenn es keine Geschäftsprozesse abbilden könnte. Solche Workflows sind im Grunde Prozesse, die aufgrund bestimmter Ereignisse (z.B. neues Dokument wurde in Bibliothek hochgeladen) automatisch gestartet und mit allen Elementen in SharePoint interagieren können.

6.1.4. Search

Nachdem SharePoint riesige Datenmengen verwalten kann, ist eine effiziente Suche unabdingbar. Microsoft bietet dazu speziell für umfangreiche SharePoint-Systeme ein eigenes Produkt an (FAST Search Server for SharePoint), welches die in SharePoint standardmäßig vorhandene Suchinfrastruktur erweitert und optimiert.

Die in SharePoint vorhandene Datenintegrationslösung (Business Connectivity Services) erlaubt es sogar, nicht nur Inhalte in SharePoint selbst zu indizieren, sondern Fremdsysteme über SharePoint suchbar zu machen.

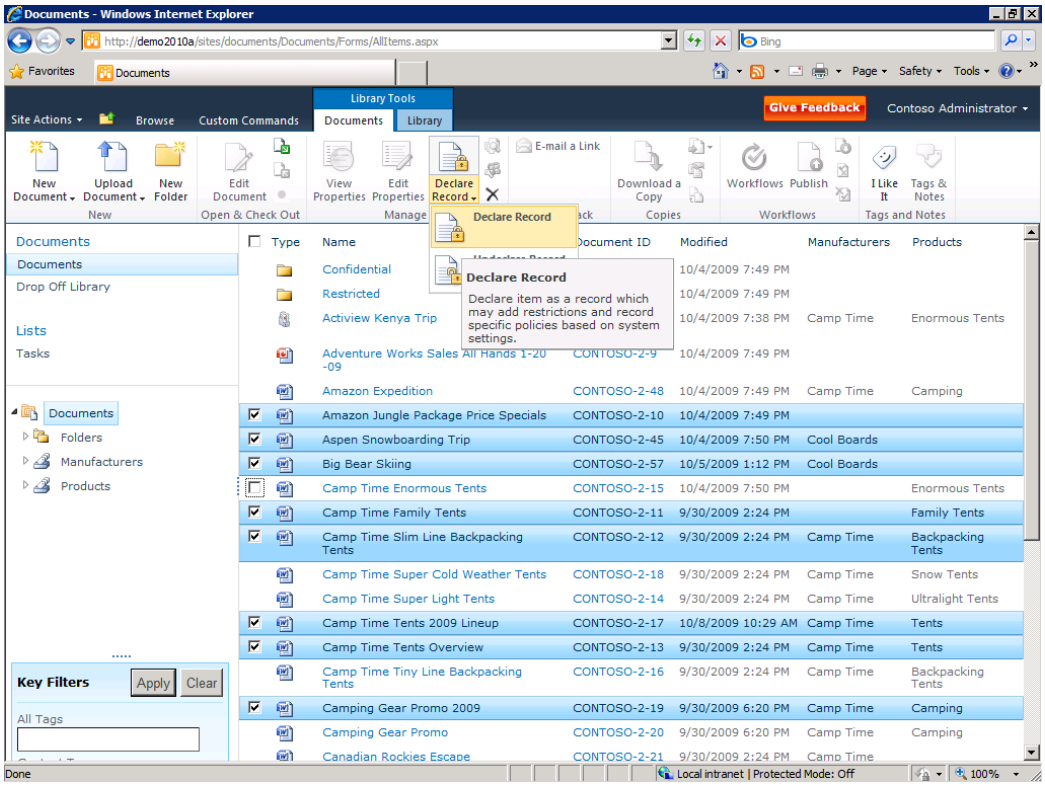


Abbildung 6.2.: Eine beispielhafte Bibliothek in SharePoint 2010 [21]

6.1.5. Insights

Insights meint in SharePoint 2010 jede Menge *Business Intelligence*-Funktionen. Und auch hier hat Office in Form bestimmter Dienste wieder seine Finger im Spiel:

Excel Services Diese Dienste erlauben es, Elemente aus Excel-Sheets (z.B. Diagramme) im Browser anzuzeigen, d.h. browserfähig zu machen, ohne dass der Client selbst die Clientanwendung Excel installiert haben muss. Diese Elemente sind jedoch keine statischen Objekte, lassen sich doch Parameter von der Website in das Excel-Sheet übertragen und somit womöglich die Darstellung eines Diagramms ändern.

Access Services Durch diesen Dienst wird es möglich, eine Access Datenbank in eine SharePoint-Website zu "konvertieren". Alle Tabellen und Sichten werden dabei zu entsprechenden Listen, wobei selbst Beziehungen zwischen diesen Objekten vorhanden bleiben.

Visio Services Um Visualisierungen wie Flowcharts oder ähnliches abzubilden, können die Visio Services genutzt werden. Diese machen Visio-Zeichnungen webbrowserfähig und setzen somit nicht die Anwendung Visio auf dem Client voraus.

Neben diesen Diensten bietet SharePoint auch einfache KPI-Listen (Key Performance Indikatoren), welche über eine Art Ampelsystem den Benutzer grafisch auf eventuelle

Missstände in bestimmten Bereichen hinweisen. Abbildung 6.3 zeigt sowohl eine KPI-Liste als auch Excel Services in Action.

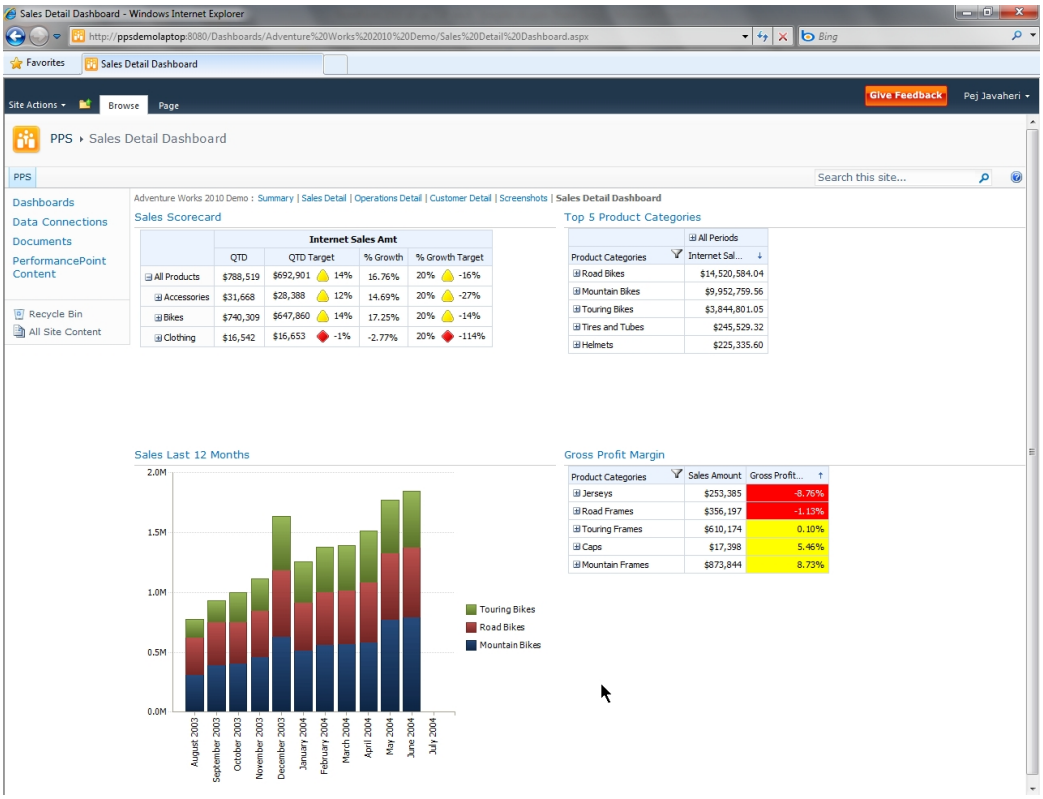


Abbildung 6.3.: SharePoint 2010 Business Intelligence im Überblick [21]

6.1.6. Composites

Als letzter der 6 Kerndienste steht *Composites* vor allem für die Möglichkeit, SharePoint völlig ohne Code zu erweitern.

Für die Implementierung von Lösungen mithilfe der eben genannten Dienste Excel Services, Access Services sowie Visio Services ist kein Software-Entwickler notwendig. Selbst die Business Connectivity Services, welche für die Datenintegration zuständig sind, lassen sich unter bestimmten Bedingungen ohne Fachkenntnisse der Software-Entwicklung bedienen.

Zusätzlich ermöglicht der sog. SharePoint Designer das Anpassen von Design und Layout sowie die Definition benutzerdefinierter Sichten auf vorhandene Listen und Bibliotheken, so dass sich viele Anpassungen als No-Code-Lösungen umsetzen lassen.

6.2. Kommunikation mit Drittsystem

Um anderen Systemen zu ermöglichen, mit SharePoint zu kommunizieren, werden diverse Webservices bzw. Komponenten angeboten. So existiert seit der Version 2010 eine eigene Klassenbibliothek für .NET, Silverlight und ECMA-Skripte, die eine Kommunikation mit SharePoint erlauben. Diese Bibliotheken dienen dabei als Wrapper für von SharePoint angebotene Webservices aller Art (z.B. herkömmliche ASP.NET-Services genauso wie WCF-Services). In der aktuellen Version wurde SharePoint auch als sog. *OData-Producer*, implementiert, so dass speziell Daten aus Listen und Bibliotheken als OData-Entitäten abfragbar und manipulierbar sind.

OData selbst ist ein Webprotokoll zur Abfrage und Manipulation von Datensätzen (OData-Entitäten). Es baut dabei auf bekannten Standards wie HTTP, AtomPub und JSON auf [18]. Nachdem schon sehr viele Systeme OData implementieren, wird in dieser Arbeit auch dieses Protokoll zur Abfrage von Daten in SharePoint verwendet, da durch einfache Adaptionen beliebige OData-Producer als Datenquelle verwendet können und dies somit der Flexibilität der Datenintegration dient.

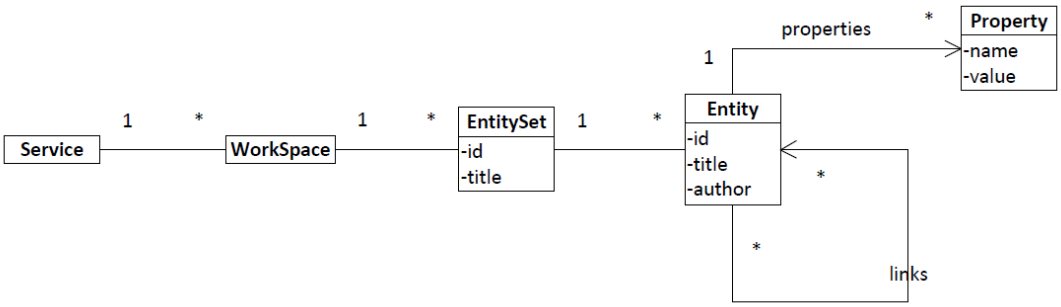


Abbildung 6.4.: Kernobjekte in OData und deren Beziehungen

Eine Liste oder Bibliothek bildet im Kontext von OData ein sog. *EntitySet* (siehe Abbildung 6.4. Listing 6.1 zeigt, wie alle Listen und Bibliotheken einer bestimmten SharePoint-Website ("myspwebsite") über den OData-Service (*http://myspwebsite/_vti_bin/ListData.svc*, im Folgenden nur noch *.../ListData.svc*) als EntitySets (*Collection*) abgefragt werden können.

```
<service ...>
  <workspace>
    <collection href="Lectures">
      <atom:title>Lectures</atom:title>
    </collection>
    <collection href="People">
      <atom:title>People</atom:title>
    </collection>
  </workspace>
</service>
```

Listing 6.1: Alle abfragbaren EntitySets eines OData-Producers im Atom-Format

Standardmäßig ist das Format von OData-Objekten AtomPub (Atom Publishing Protocol). Um OData-Objekte in JSON (JavaScript Object Notation) anzufordern, muss im HTTP-Header explizit das "accept"-Feld auf "application/json" gesetzt werden. Im Folgenden wird zur Anzeige von OData-Objekten jedoch immer AtomPub verwendet.

Um nun Daten einer bestimmten Liste (im Folgenden auch als Synonym für Bibliothek gebraucht) zu selektieren, zu filtern oder zu sortieren, bietet OData bestimmte Möglichkeiten an. So lässt sich beispielsweise mit dem Aufruf

```
.../ListData.svc/People?$select=LastName,FirstName
        &$filter=JobTitle eq 'Student'&$orderby=FirstName asc
```

die Liste in Abbildung 6.5 so abfragen, dass das Ergebnis in Listing 6.2 entsteht. Dazu wird diese Abfrage in eine HTTP-GET-Nachricht mit folgendem Header verpackt und an den entsprechenden OData-Service gesendet:

```
GET .../ListData.svc/People?$select=LastName,FirstName &
        $filter=JobTitle eq 'Student'&$orderby=FirstName asc
Accept: application/xml
Authorization: Basic ...
User-Agent: odata4j.org
```

Zur Abfrage eines Eintrages mit der ID XXX wird eine HTTP-GET-Nachricht an die URL

```
.../ListData.svc/People(XXX)
```

abgesetzt.

Die OData-Entries selbst bestehen aus herkömmlichen Properties, welche einfache Name-Wert-Paare sind, und Links als Referenzen auf andere OData-Entitäten (siehe Abbildung 6.4). Der Wert der herkömmlichen Properties kann dabei einen der folgenden Datentypen besitzen:

- Binary
- Boolean
- Byte, SByte
- DateTime, Time, DateTimeOffset
- Decimal, Single, Double
- Guid (General Unique ID)
- Int16, Int32, Int64
- String

Diese Datentypen gelten sowohl für AtomPub als auch für JSON.

```
<feed ...>
  <title type="text">People</title>
  <id>.../listdata.svc/People</id>
  <updated>2011-07-14T07:39:25Z</updated>
  <link rel="self" title="People" href="People" />
  <entry m:etag="W/&quot;1&quot;">
    <id>.../ListData.svc/People(17)</id>
    <title type="text">Waltl</title>
    <updated>2011-06-21T12:33:12+02:00</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="PeopleItem" href="People(17)" />
    <category term="Microsoft.SharePoint.DataService.PeopleItem" ... />
    <content type="application/xml">
      <m:properties>
        <d:LastName>Waltl</d:LastName>
        <d:FirstName>Bernhard</d:FirstName>
      </m:properties>
    </content>
  </entry>
  <entry m:etag="W/&quot;17&quot;">
    <id>.../ListData.svc/People(11)</id>
    <title type="text">Reschenhofer</title>
    <updated>2011-06-21T11:04:28+02:00</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="PeopleItem" href="People(11)" />
    <category term="Microsoft.SharePoint.DataService.PeopleItem" ... />
    <content type="application/xml">
      <m:properties>
        <d:LastName>Reschenhofer</d:LastName>
        <d:FirstName>Thomas</d:FirstName>
      </m:properties>
    </content>
  </entry>
</feed>
```

Listing 6.2: Beispielhafte Abfrage der People-Liste

Zum Erzeugen, Aktualisieren und Löschen eines OData-Eintrages werden entsprechende HTTP-POST/PUT/DELETE-Nachrichten abgesetzt:

Create Zum Erzeugen eines People-Eintrages wird eine HTTP-POST-Nachricht mit dem Header

```
POST .../ListData.svc/People
Accept: application/xml
Authorization: Basic ...
User-Agent: odata4j.org
```

und dem in Listing 6.3 gezeigten HTTP-Body abgesetzt. Die Antword des Servers ist das neu erzeugte OData-Element.

Update Zur Aktualisierung eines People-Eintrages mit der ID XXX wird eine HTTP-PUT-Nachricht mit dem Header

```
PUT .../ListData.svc/People(XXX)
Accept: application/xml
Authorization: Basic ...
If-Match: *
User-Agent: odata4j.org
```

und dem in Listing 6.3 gezeigten HTTP-Body abgesetzt. Die Antword des Servers ist das aktualisierte OData-Element.

Delete Zum Löschen eines People-Eintrages mit der ID XXX wird eine HTTP-PUT-Nachricht mit leerem Body und folgendem Header abgesetzt:

```
DELETE .../ListData.svc/People(XXX)
Accept: application/xml
Authorization: Basic ...
User-Agent: odata4j.org
```

```
<entry ...>
  <title type="text"></title>
  <updated>2011-07-14T08:18:43Z</updated>
  <author>
    <name></name>
  </author>
  <content type="application/xml">
    <m:properties>
      <d:FirstName>Max</d:FirstName>
      <d:JobTitle>CEO</d:JobTitle>
      <d:LastName>Mustermann</d:LastName>
      ...
    </m:properties>
  </content>
</entry>
```

Listing 6.3: Der Body einer HTTP-POST/PUT-Nachricht zur Erzeugung/Aktualisierung eines OData-Entries

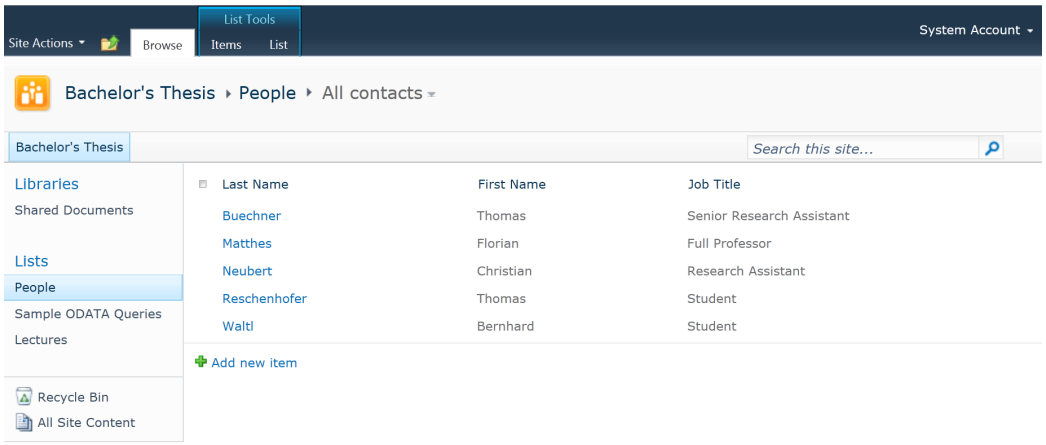


Abbildung 6.5.: Beispielhafte SharePoint Liste mit 5 Einträgen

Die konkrete Kommunikation von Tricia mit SharePoint wird später in Kapitel 9 behandelt.

7. Microsoft Exchange 2010

7.1. Beschreibung

Bei dem Exchange Server 2010 handelt es sich um ein sehr umfangreiches Produkt des Unternehmens Microsoft, das als Nachrichten- und Groupwaresystem fungieren kann. Die Software wird in vielen Infrastrukturen von großen, aber auch bei kleinen und mittelständischen Unternehmen eingesetzt. Als etabliertes Nachrichtensystem kann es den E-Mail Verkehr abwickeln, Kontaktdaten pflegen, Aufgabenlisten verwalten, Kalendereinträge und Zeitpläne erstellen. In der aktuellen Version stellt der Exchange Server auch einen sogenannten Web-Access zu Verfügung. Dabei handelt es sich um eine Möglichkeit, die auf dem Server gespeicherten Informationen über das Internet (z.B: mit mobilen Endgeräten) abzurufen.

Die verfügbaren Funktionen haben sich seit der ersten Version 1.0, im Jahre 1996, ständig vermehrt. In der aktuellen Version, Microsoft Exchange Server 2010, deckt die Software, unter Umständen, die kompletten Intranet-Ansprüche eines Unternehmens ab. Neben den bereits genannten Funktionalitäten existieren noch zahlreiche weitere Features, die in dieser Arbeit aber nicht ansatzweise benötigt werden bzw. nicht erforderlich sind. Im Rahmen dieser Arbeit ist es ausreichend, wenn man den Exchange Server als eine autonome Datenquelle bzw. Datensenke betrachtet.

In vielen Unternehmen ist es der Fall, dass Daten bereits durch Prozesse in diesem bestehenden System verwaltet werden (siehe Kapitel 2).

Da es sich beim Exchange Server um eine serverseitige Technologie handelt, bietet diese mehrere Varianten um auf die gespeicherten Daten zuzugreifen. Die weit verbreitete, clientseitige Applikation Microsoft Outlook benutzt dazu ein RPC-Protokoll [15]. Der Microsoft Exchange Server bietet aber noch weitere Schnittstellen, um Zugriff auf die Daten aus Client-Applikationen zu ermöglichen. Unter anderem stellt er auch Web-Services zur Verfügung [12]. Unter Zuhilfenahme dieser Web-Services können Objekte auf dem Server abgerufen und verwaltet werden. Für den Aufruf werden die SOAP Versionen 1.1 und 1.2 unterstützt. Bei SOAP handelt es sich um ein XML-basiertes Protokoll, welches eingebettet in HTTP Nachrichten zwischen dem Client und dem Server ausgetauscht wird. Die Exchange Web Services (EWS) stellen die notwendige Funktionalität zur Verfügung die von Clientapplikationen benötigt wird um mit dem Exchange Server zu kommunizieren. Da der Exchange Server, im Rahmen dieser Arbeit, als föderierte Datenquelle gesehen wird, sind die Operationen auf Objekte (sog. *Item Operations*) von besonderem Interesse. Diese Operationen ermöglichen die Erstellung, die Aktualisierung, das Löschen, das Kopieren, das Abfragen, das Verschieben und das Senden von Objekten.
Diese Objekte umfassen:

Nachrichten E-Mail Nachrichten werden im Exchange Server als XML formatiert im Internet Message Format [20] dargestellt.

Kontakte Bei Kontakt Objekten handelt es sich um eine Sammlung an Eigenschaften, die zum Einen in dem Active Directory des Servers und zum Anderen im privaten Kontaktspeicher eines Benutzers gespeichert sein können. Diese Unterscheidung ist notwendig, da auf die unterschiedlichen Kontakte jeweils andere Operationen verfügbar sind. Unser Implementierungsansatz konzentriert sich auf den privaten Kontaktspeicher eines bestimmten Benutzers.

Aufgaben Jeder Benutzer kann durch Outlook oder durch den Exchange Web Access eine Aufgaben- Liste erstellen. Diese Einträge werden durch diese Objektkategorie repräsentiert.

Kalender Die Kalendereinträge umfassen eine Vielzahl an Objekten, wie Ort, Ressourcen, Teilnehmer und Zeit. Diese Einträge können nicht nur erstellt und verwaltet, sondern auch noch weitergeleitet werden. Benutzer können entsprechende Einladungen annehmen oder ablehnen.

Für die konkrete Implementierung legten wir uns auf zwei Objekte fest: Nachrichten und Kontakte. Alle weiteren Objekte sind durch unseren generischen Ansatz mit reinem Programmieraufwand zu bewältigen. Die Architektur ist davon in keinster Weise betroffen.

Neben vielen weiteren Konzepten verfügt der Exchange Server auch über sogenannte *Notification Operations*. Diese Operationen ermöglichen Client-Systemen über Ereignisse und Änderungen in den Ordnern eines Exchange Servers informiert zu werden. Um solche Benachrichtigungen aber überhaupt erhalten zu können, muss sich die Client-Applikation beim Server registrieren. Dabei muss die Client-Applikation, über entsprechende Webservices, beim Server einen Endpunkt (TCP Listener oder Webservice) angeben, der der Callback Methode entspricht. Desweitern muss das Ereignis angegeben werden, für das der clientseitige Webservice dann aufgerufen wird. Tritt das spezifizierte Ereignis auf, so wird dem Endpunkt eine SOAP Nachricht gesendet.

Folgende Ereignisse werden vom Notification Dienst des Exchange Servers registriert, diese können also eine Benachrichtigung auslösen:

- Eintreffen einer E-Mail Nachricht
- Objekterzeugung
- Löschen eines Objekts
- Änderungen an einem Objekt
- Verschieben oder Kopieren eines Objekts

Es existieren verschiedene Ansätze um Änderungen in einem externen System mitzubekommen. Die zwei für die Kommunikation mit dem Exchange Server relevanten sind

polling und *pushing* (siehe Kapitel 3.2.3). Der Vorteil des Push-Modells liegt darin, dass nur dann Daten übertragen werden, wenn tatsächlich eine Änderung stattgefunden hat.

Das in Kapitel 3.2.4 angeführte Problem, dass zu den verfügbaren Objekten unter Umständen zusätzliche Attribute und Eigenschaften angeheftet werden, kann mit Methoden des Microsoft Exchange Servers 2010 behoben werden. Der Exchange Server erlaubt es, zu jedem seiner Objekte sogenannte ExtendedProperties zu definieren. Es wäre zum Beispiel denkbar, an einen bestehenden Kontakt die Information anzuheften, wann das letzte persönliche Treffen mit dieser Person stattgefunden hat. Diese Attribute werden dann in der Datenbank des Serves abgespeichert. Wird nun ein Objekt, an dem ExtendedProperties angefügt wurden, abgerufen, so sind diese zusätzlichen Attribute verfügbar. Im Rahmen dieser Arbeit wurde das Problem behandelt, aus zeitlichen Gründen jedoch nicht in die Implementierung mitaufgenommen.

7.2. Kommunikation mit Drittsystem

Die bereits erwähnten Web Services die vom Microsoft Exchange 2010 zur Verfügung gestellt werden, ermöglichen die Kommunikation und den Datenaustausch zwischen Tricia und dem Server. Die Web Services werden vom Server durch den IIS (Internet Information Service) veröffentlicht.

Initiiert eine Client-Applikation nun einen Aufruf eines angebotenen Web Services, dass heißt es werden Daten vom Exchange Server angefordert, so wird eine XML - Nachricht generiert, die dem SOAP Standard entspricht, und zum Server gesendet. Nach dem erhalt der Nachricht prüft der Server, ob die mitgesendete Benutzerinformationen gültig sind. Falls dies der Fall ist, wird die XML Nachricht weiterverarbeitet. Der Server bearbeitet die Anfrage und sendet seinerseits ein Ergebnis (HTTP-Response) zurück. Die Client-Applikation erhält also die Antwort als in einer HTTP-Nachricht eingebettete XML-Nachricht. Die gesamte Infrastruktur ist in Abbildung 7.1 abgebildet.

Die Web Services des Microsoft Exchange Server 2010 decken in der aktuellen Version schon fast den gesamten Funktionsumfang, den der Server auf die Exchange Mailboxes bietet, ab. Die Abbildung 7.1 verdeutlicht den Umfang. Die Web Services ermöglichen bereits alle Operationen die überhaupt auf den Exchange Ordnern verfügbar sind. Dies spiegelt sich auch in der großen Anzahl der angebotenen Web Services wieder. Alleine in der WSDL des Exchange Servers 2010 werden über 200 verschiedene Funktionen deklariert. Diverse Internetforen [1] [2] und allen voran die offizielle Microsoft Seite [3] empfehlen daher die Verwendung einer API [13]. Die Schwierigkeit, die sich dabei ergibt betrifft jedoch die verwendete Programmiersprache. Die verfügbare API wurde für das .NET Framework entwickelt und zielt daher auch auf die Sprachen des .NET Frameworks ab. Tricia

[1] http://www.aljoscha-rittner.de/blog/archive/2010/12/15/javadev-microsoft-exchange-api-fuer-java/, August 2011

[2] http://www.msxfaq.de/code/ews.htm, August 2011

[3] http://msdn.microsoft.com/en-us/library/dd633709.aspx, Juli 2011

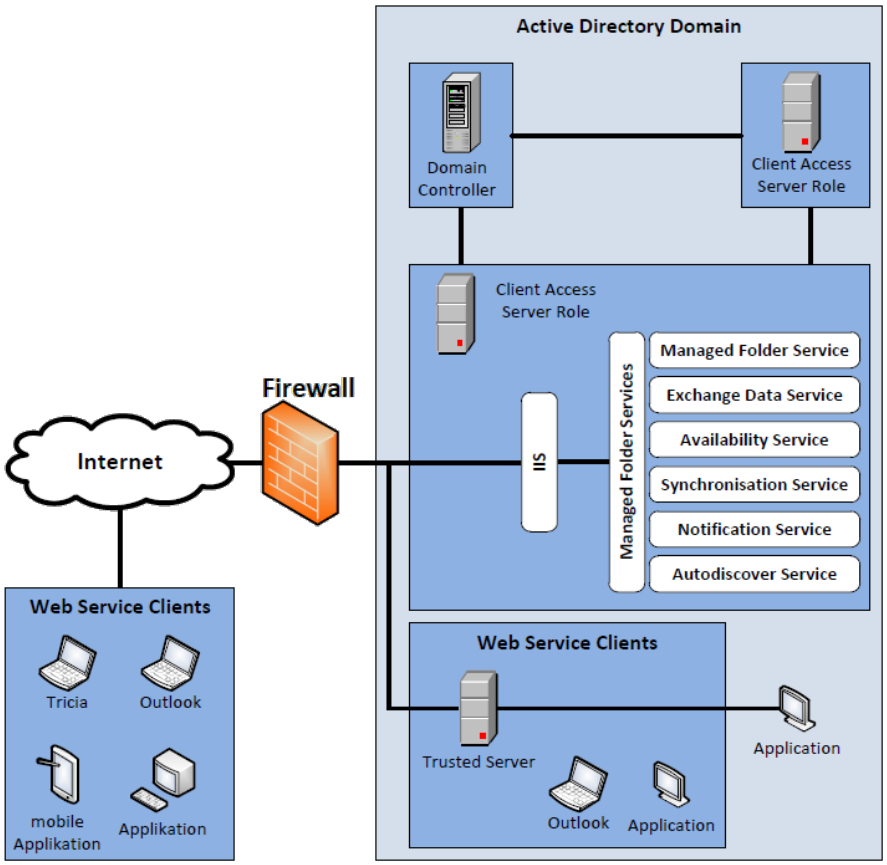


Abbildung 7.1.: Client-Server Interaktion [13]

ist jedoch vollständig in Java geschrieben, eine Verwendung der offiziellen EWS Managed API ist also nicht möglich. Für unsere Anwendung existieren jedoch zwei mögliche APIs für den Zugriff auf die Web Services.

Dies ist zum Einen eine freie Microsoft API, die EWS Java API 1.1 [4], und zum Anderen wurde von dem deutschen Unternehmen Independentsoft [5] eine kostenpflichtige API entwickelt. Die "JWebServices for Exchange" API von Independentsoft unterstützt die Microsoft Exchange Versionen 2007 und 2010. In diesem Zusammenhang musste also evaluiert werden, welche der beiden APIs für unsere Anwendung besser geeignet ist. Der notwendige Vergleich wird im Kapitel 10.1 vorgenommen.

[4] http://archive.msdn.microsoft.com/ewsjavaapi, Juli 2011
[5] http://www.independentsoft.de/, Juli 2011

Teil III.

Schicht 1: Schnittstelle zu externen Datenquellen

8. Generische Schnittstelle zu Datenquellen

Schicht I der Datenintegrations-Architektur stellt die Schnittstelle zu externen Systemen dar. Datensätze aus Datenquellen externer Systeme werden dabei importiert und als lokale Entitäten abgelegt. Schicht II (siehe Teil III der Arbeit) kümmert sich um die Präsentation der Daten und die Interaktion mit dem Benutzer. Schicht I erweitert dabei die bestehende Store-Architektur (siehe Kapitel 5.4), ohne sie zu verändern. Ziel ist eine Architektur, die die wesentlichen Anforderungen an eine Datenintegration (siehe Kapitel 3.2) erfüllt.

8.1. Allgemein

Um eine Datenintegration aus einem der vorgestellten Drittsysteme ermöglichen zu können, muss die bestehende Architektur (siehe Kapitel 5) erweitert werden. Bei dieser Erweiterung sollen alle Anforderungen, die in Kapitel 3 gestellt wurden, soweit als möglich abgedeckt werden. Außerdem ist es wünschenswert, dass Tricia weitgehend unberührt bleibt, d.h., es soll weder im *Core* noch in anderen Plugins Änderungen an der Architektur bzw. an dem Code vollzogen werden müssen. Das Ergebnis dieser Arbeit besteht aus einer erweiterten Architektur, dass die gestellten Forderungen erfüllt.

Die Erweiterungen befinden sich in einem eigenen Plugin, welches zu Tricia hinzugefügt werden kann. Das *External* Plugin ist von dem Plugin *hybridWiki* abhängig. Es besteht somit auch eine transitive Abhängigkeit zu anderen Plugins, auf die hier aber nicht näher eingegangen wird.

Das Hauptaugenmerk der Architektur zur Datenintegration liegt in der Schnittstelle zu den externen Systemen bzw. zu den föderierten Datenquellen. Dabei muss die Kommunikation von dieser Schnittstelle übernommen werden. Unter der Berücksichtigung der gestellten Anforderungen ergab sich nun eine Architektur, die in den folgenden Kapiteln näher erläutert wird.

8.2. Architektur

Die Abbildung 8.1 zeigt das Klassendiagramm und das Architekturmodell der Schnittstelle zwischen Tricia und den externen Systemen. In einem gestrichelten Rahmen ist die bereits bestehende Architektur von Tricia zusammgefasst (siehe Kapitel 5). Außerhalb des Rahmens sind die Klassen abgebildet, die durch das External Plugin neu hinzugekommen sind. Diese sind notwendig, um die Datenintegration zu ermöglichen, sie stellen also die Schnittstelle zu den Datenquellen dar.

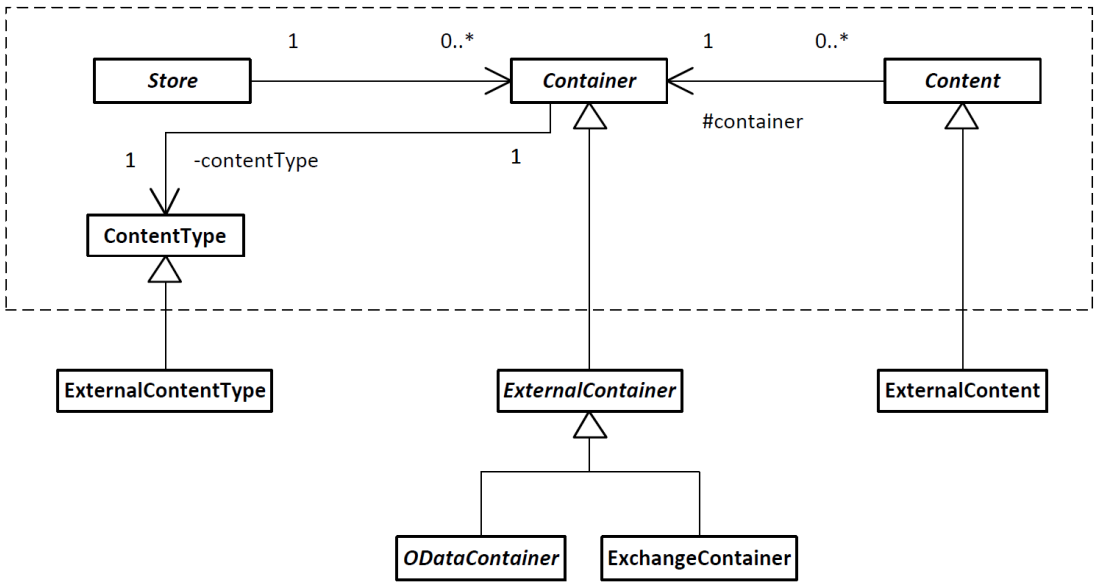


Abbildung 8.1.: Erweiterung der bestehenden Architektur durch das *External* Plugin

In Kapitel 5.4 wurden die Konzepte der bestehenden Architektur, die einzelnen Klassen und deren Aufgaben bereits ausführlich erläutert. Durch das External Plugin wurden jetzt im Wesentlichen vier neue Klassen eingeführt, die den Datenimport auf der Store-Ebene ermöglichen.

8.2.1. ExternalContentType

Die Klasse ExternalContentType leitet von der Klasse ContentType ab. Sie übernimmt alle Aufgaben, die die ContentType-Klasse bei der vorhandenen Store-Architektur übernimmt. Wie bereits in Kapitel 5.4 erwähnt wurde, definiert diese das Schema einer Entität. Die Schemainformation einer externen Entität wird von dieser Klasse verwaltet. Die Information wird bei der Erstellung von der *ExternalTypeTagDefinition* übernommen.

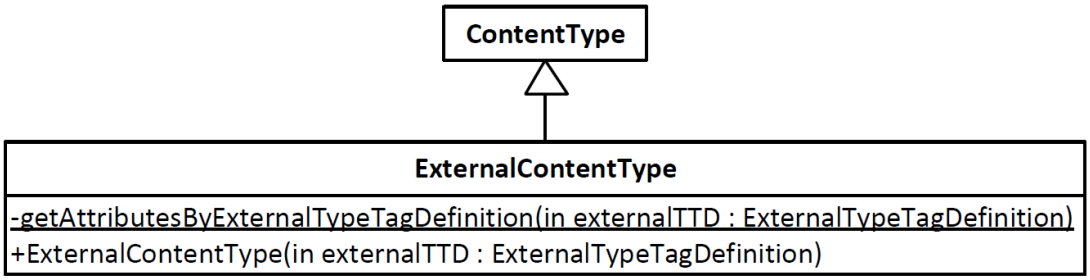


Abbildung 8.2.: Klassendiagramm der ExternalContentType Klasse

Abbildung 8.2 zeigt nun die Klasse in detaillierter Form. Bei der Erstellung wird im Konstruktor die Schemainformation des externen Objekts unter der Verwendung der Ex-

ternalTypeTagDefinition bekanntgegeben. Diese Informationen werden also in einer lokalen Liste abgelegt. Im Konstruktor werden also alle verfügbaren Attribute aus der TypeTagDefinition ausgelesen und an den Konstruktor der Basisklasse weitergeleitet, der dann alle Attribute in seiner lokalen Liste mitaufnimmt und dort verwaltet. Über die entsprechenden Methoden (*getAttributes()* und *getAllAttributes()*) können diese dann abgefragt werden.

ParameterAttribute

Bei einem ParameterAttribute Objekt handelt es sich um eine Wrapperklasse für das StringAttribute Interface. Es besteht aus einem einfachen Attribut, welches nur aus einem Namen besteht und keinen Wert besitzt. Dies ist notwendig, da es Zeitpunkte gibt, an denen für eine Entität noch kein zugeordnetes Hybrid besteht. Die Schemainformation ist allerdings in der TypeTagDefinition vorhanden. Um die Schemainformation also in einer Entität zu speichern, werden die Namen der Attribute als *ParameterAttribute* abgelegt.

8.2.2. ExternalContent

Die neu erstellte ExternalContent Klasse leitet von der bestehende Content Klasse ab. Die Content Klasse stellt, wie in Kapitel 5.4 erläutert, einen Eintrag in einer Datenbanktabelle dar und entspricht somit einer konkreten Entität. Die ExternalContent Klasse übernimmt genau diese Aufgabe jetzt für eine externe Entität.

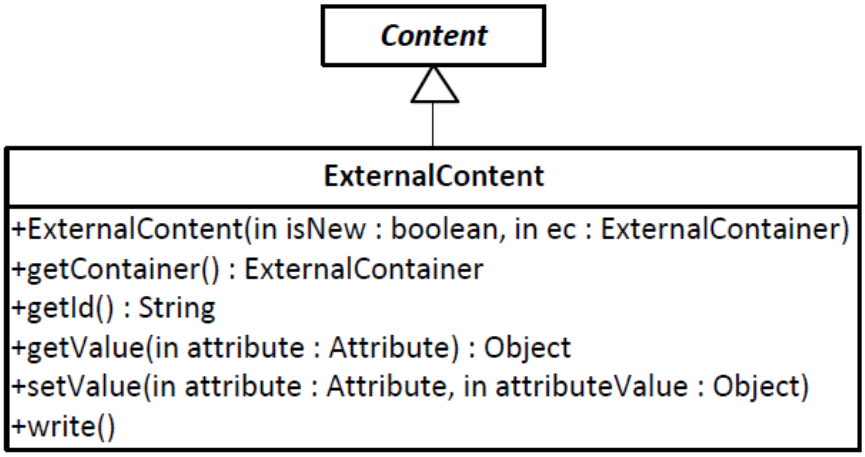


Abbildung 8.3.: Klassendiagramm der ExternalContent Klasse

Die entscheidenden Methoden dabei sind:

setValue In dieser Methode werden die Inhalte einer Entität gesetzt. Es muss zur Laufzeit unterschieden werden, welches Attribut gesetzt wird.

```
@Override
public void setValue(Attribute attribute,
        Object attributeValue) {
  if (attribute instanceof HybridProperty) {
    content.put(
          Utilities.trim(attribute.getAttributeName(),'\''),
          attributeValue.toString().trim());
  } else if (attribute instanceof ParameterAttribute) {
    setString((StringAttribute) attribute,
          attributeValue.toString());
  } else {
    super.setValue(attribute, attributeValue);  }
 }
```

Listing 8.1: Die setValue() - Methode in der ExternalContent Klasse

getContainer Diese Methode liefert für einen ExternalContent den zugeordneten ExternalContainer. Handelt es sich bei dem Attribut um ein HybridProperty Objekt, so wird der Wert direkt in der content - Liste gesetzt. Andernfalls muss sich aus dem Objekt der Attributname und der Wert ermittelt werden, welcher dann gesetzt wird.

getValue Diese Methode ist gewissermaßen das Pendant zu der Methode setValue(). In dieser Routine werden die Werte einer Entität abgerufen. An dieser Stelle muss wieder unterschieden werden, welches Attribut ausgelesen wird. Je nach angeforderten Typ wird dabei entweder direkt auf die content - Liste zugegriffen oder zuerst der Name des betreffenden Attributs ermittelt, welches dann zurückgeliefert wird.

```
@Override
public Object getValue(Attribute attribute) {
  if (attribute instanceof HybridProperty) {
        return content.get(Utilities.trim(
                attribute.getAttributeName(),'\''));
  } else if (attribute instanceof ParameterAttribute) {
        return getString((StringAttribute) attribute);
  } else {
        return super.getValue(attribute);
  }
}
```

Listing 8.2: Die getValue() - Methode in der ExternalContent Klasse

8.3. ExternalContainer

Die bisher vorgestellten Klassen (ExternalContentType und ExternalContent) unterstützen den Zugriff auf die Daten für höhere Schichten in der Tricia Architektur wie Hybrids und Wikis. Sie stellen die notwendige Information über das Schema dar oder repräsentieren einen bereits integrierten Datensatz aus einem Drittsystem der in dem lokalen Speicher

vorliegt. Die wesentliche Aufgabe des Datenimports wird nun von einer dritten Klasse übernommen: dem ExternalContainer.

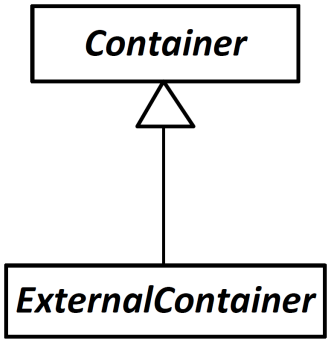


Abbildung 8.4.: Klassendiagramm der ExternalContainer Klasse

Diese Klasse leitet von der bekannten Container Klasse ab und übernimmt einen sehr wichtigen Teil für die Datenintegration. In Kapitel 5.4 wurde bereits erläutert, dass die Klasse Container alle vorhandenen Entities eines Typs verwaltet, sie besitzt einen Namen und verwaltet die zugehörigen Datensätze. Jedes Entity in dem Container steht dabei für einen Eintrag in der Datenbank. Die ExternalContainer - Klasse stellt nun Methoden und Funktionen zur Verfügung um diese zu verwalten. Dabei arbeitet sie nicht auf dem lokalen Datenspeicher, sondert greift auf das jeweilige Drittsystem zu. Bei den Einträgen, die von dieser Klasse verwaltet werden, handelt es sich um externe, aus Drittsystemen importierte, Einträgen.

In den Methoden der Klasse ExternalContainer findet also der Zugriff auf die externe Datenquelle statt. Diese Klasse bzw. die davon abgeleiteten Klassen verfügen über die Informationen, die notwendig sind, um konkrete Datensätze aus einem fremden System zu importieren. Dieser Import kann über verschiedene Wege erfolgen.

Im Rahmen dieser Arbeit erfolgt der Zugriff auf zwei externe Systeme. Die Kommunikation mit dem Microsoft SharePoint wird über REST - Webservices und dem OData - Protokoll realisiert und auf die Datensätze des Microsoft Exchange Servers 2010 wird ebenfalls über Webservices zugegriffen, allerdings nicht mit dem OData - Protokoll.

8.3.1. Schemainformation

Um mit externen Daten in Tricia arbeiten zu können ist neben den konkreten Datensätzen auch das Schema der importierten Daten notwendig. Die ExternalContainer - Klasse stellt die Schnittstelle zu den Drittsystemen dar und somit das einzige Objekt, dass mit dem externen System kommuniziert. Die Attribute und die entsprechenden Datentypen der zu importierenden Datensätze müssen also von dieser Klasse angeboten werden. Höhere Schichten fordern zur Laufzeit die aktuelle Schemainformation an. Dabei werden Methoden benötigt, die das Ergebnis generieren und zurückliefern (siehe Kapitel 12.3).

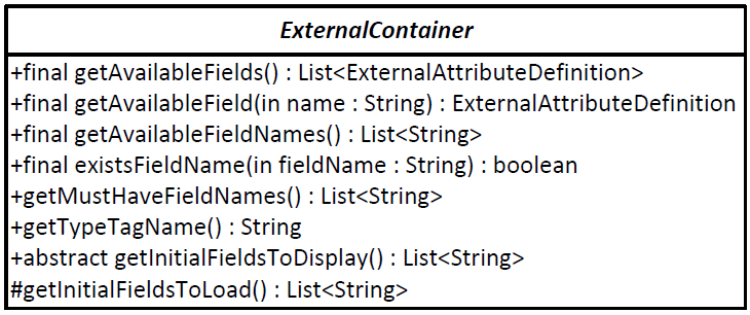


Abbildung 8.5.: Die ExternalContainer Klasse und die Methoden zum Zugriff auf das Schema

Abbildung 8.5 zeigt einen Ausschnitt aus den von der ExternalContainer Klasse angebotenen Methoden. Die dargestellten Methoden ermöglichen das Anbieten der verfügbaren Schemainformationen an die höheren Schichten.

getAvailableFields Liefert eine Liste aller verfügbaren Attribute zurück. Ein Attribut wird dabei als ein Objekt der Klasse *ExternalAttributeDefinition* repräsentiert.

getAvailableField Diese Operation liefert nur ein Attribut zurück, dessen Name dem übergebenen Parameter entspricht. Existiert kein Attribut mit dem Namen wird *null* zurückgeliefert.

getAvailableFieldNames Das Ergebnis dieser Methode ist eine Liste von Strings, die die Namen aller Attribute beinhaltet.

existsFieldName Liefert *true*, falls ein Attribute mit dem übergebenen Namen existiert, andernfalls *false*.

getMustHaveFieldNames Definiert die Attribute, die von der externen Datenquelle unbedingt geladen werden müssen. Aus Performancegründen werden ja nicht alle Attribute angefordert diese Methode legt fest, welche Attribute bei den Datensätzen vorausgesetzt werden.

getTypeTagName Jeder externen Datenquelle wird ein TypeTag zugeordnet. Welcher Name dabei verwendet wird, wird von der Klasse selbst festgelegt. Dies ist üblicherweise der Name der Klasse.

getInitialFieldsToDisplay Jeder importierte Datensatz wird durch ein Hybrid repräsentiert (siehe Kapitel 13). Um das Hybrid dem Benutzer anzeigen zu können, muss festgelegt werden, welche Attribute in der Oberfläche sichtbar sind. Diese Methode definiert eine Liste aller Attribute, die angezeigt werden.

getInitialFieldsToLoad Die Methode legt alle Parameter fest, die zu Beginn geladen werden. Es handelt sich dabei um die Parameter, die von der Methode *getInitialFieldsToDisplay* festgelegt werden und zusätzlich noch die Parameter, die zur Anzeige des

externen Datensatzes benötigt werden. Jede Klasse definiert in einem Template wie die einzelnen Datensätze dargestellt werden, die dabei benötigten Parameter müssen also auch zu Beginn geladen werden (siehe Kapitel 12.3).

ExternalAttributeDefinition

Ein Attribut wird durch die Klasse *ExternalAttributeDefinition* repräsentiert. In Abbildung 8.6 ist die Klasse dargestellt. Sie beinhaltet neben einem Namen und einer Beschreibung auch die Information über ihre Multiplizität. Diese gibt an wieviele Werte ein Attribute besitzen darf. Es sind die folgenden Werte möglich:

- *ExactlyOne*
- *MaximalOne*
- *AtLeastOne*
- *Any* (Standard)

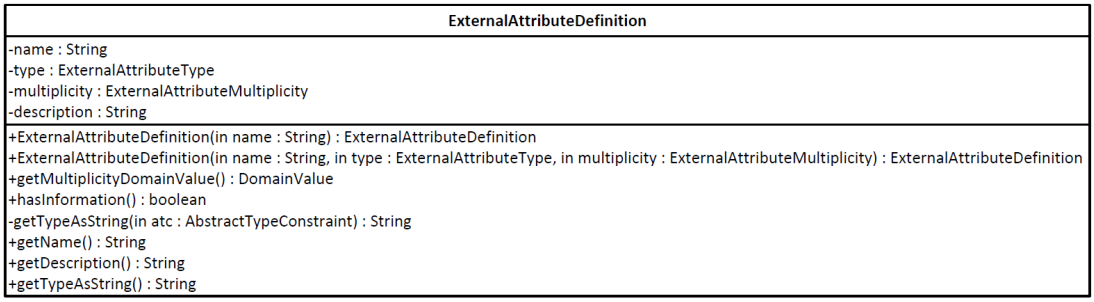


Abbildung 8.6.: Die ExternalAttributeDefinition Klasse

In der Methode *getMultiplicityDomainValue()* wird die Multiplizität, die in der *ExternalAttributeDefinition* als Enumeration definiert ist, zurückgeliefert. Die Multiplizitäten die in Hybrids verfügbar sind, sind vom Typ DomainValue. Ein entsprechendes Mapping von dem Enumerationstyp *ExternalAttributeMultiplicity* in den Typ *DomainValue* wird ebenfalls vorgenommen.

Jedes Attribut besitzt auch einen Datentyp. Dabei ist für ein Attribut aus einer externen Datenquelle einer der folgenden Datentypen möglich:

- Text
- Enumeration
- Date
- Boolean

8.3.2. CRUD - Methoden

Neben dem Zugriff auf die Schemainformation ist die ExternalContainer Klasse auch für die Integration der konkreten Datensätze aus einer externen Datenquelle zuständig. Um eine Verwaltung dieser Daten durchführen zu können, werden spezielle Methoden benötigt, die den geforderten Zugriff ermöglichen. Es handelt sich dabei um die sogenannten CRUD - Methoden:

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

Die CRUD - Methoden werden von der vorhandenen Container Klasse bereits implementiert. Diese Funktionen arbeiten jedoch auf der bestehenden Store Architektur. Die Erweiterung durch die *ExternalContainer* Klasse ermöglicht nun den Zugriff auf externe Systeme und deren Datenquellen.

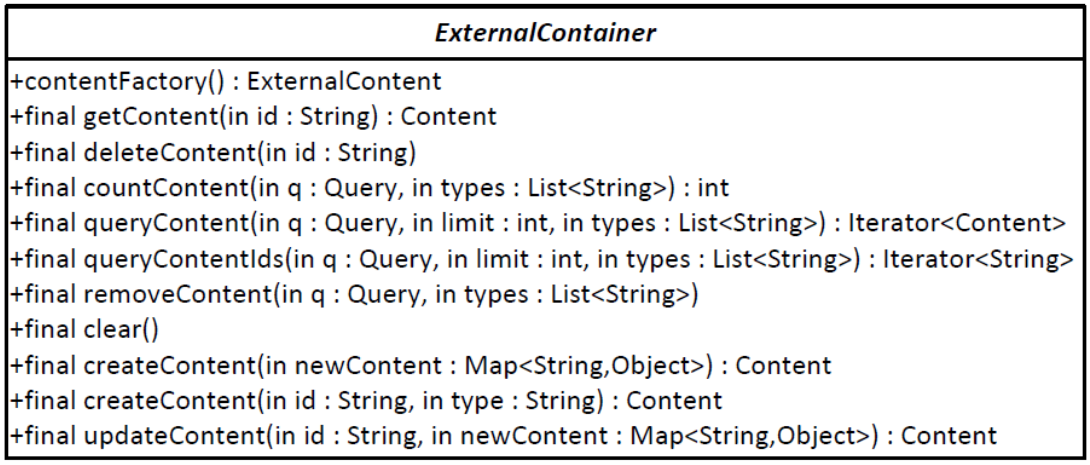


Abbildung 8.7.: Die CRUD Methoden des ExternalContainers

Abbildung 8.7 zeigt die CRUD - Methoden der implementierten ExternalContainer Klasse. Es existieren für die einzelnen Operationen zum Teil mehrere Überladungen, die sich bei den verschiedenen Anwendungsfällen als hilfreich und notwendig erwiesen haben. Einige Methoden unterstützen auch den Aufruf mit einem Query Objekt, welches einen Filter angibt das die Objekte die dann verarbeitet werden spezifiziert.

Die CRUD - Methoden sind allesamt als final deklariert, das bedeutet, dass sie in den spezialisierten Klassen des ExternalContainer nicht mehr überschrieben werden können. Es handelt sich, wie bereits erwähnt, um die Methoden, die von den höheren Schichten in Tricia aufgerufen werden. Es stellt gewissermaßen die Schnittstelle zu den höheren Schichten dar. Die Funktionen selbst sind sehr einfach und bestehen oftmals nur aus einer Zeile (siehe Listing 8.3). Diese Methoden dürfen in Spezialisierungen auch nicht mehr

überschrieben werden, da sonst nicht mehr sichergestellt werden kann, dass mit dem Aufruf der Importmanager eingebunden wird. In Kapitel 8.4 wird erläutert, warum dieser Aufruf, vor allem bei der Verwendung von Cachingstrategien, unbedingt erforderlich ist.

```
public final Content createContent(Map<String, Object> newContent) {
  return ImportManager.INSTANCE().createContent(newContent, this);
}
```

Listing 8.3: Die createContent() - Methode in der ExternalContainer Klasse

Auffällig dabei ist, dass der Aufruf direkt an den ImportManager weitergeleitet wird. Diese übernimmt den Aufruf der zuständigen Methoden und kümmert sich um Cache - Strategien, um zukünftige Importszenarien effizienter zu gestalten (siehe 8.4).

8.3.3. Live - Methoden

Neben den CRUD Methoden, die die Schnittstelle zu den höheren Schichten darstellen, sind auch Methoden vorhanden, die die Aufrufe und den Datenimport aus einem Fremdsystem realisieren: die sogenannten Live - Methoden. Die Bezeichnung "Live" wurde gewählt, da diese Methoden direkt Routinen des externen Systems aufrufen.

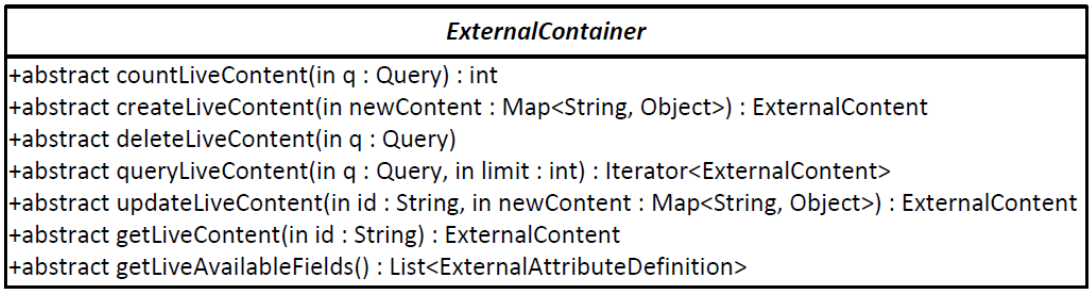


Abbildung 8.8.: Die Live Methoden des ExternalContainers

Da sich die Information über die tatsächlichen Aufrufe nur in dem ExternalContainer Objekt befindet, müssen die Live Methoden auch als "abstract" deklariert werden. Jede Spezialisierung dieser Klasse, die instanziiert werden kann, muss diese also konkret implementieren. Dabei wird bei bestimmten Methoden auch das Query Objekt als Parameter übergeben. Das ermöglicht eine Selektion der Datensätze, auf welche dann die Operation angewendet wird.

countLiveContent Zählt alle externen Datensätze, die das Filterkriterium erfüllen und liefert als Resultat die Anzahl zurück.

createLiveContent Erstellt ein neues externes Element und liefert dieses auch zurück. Der Inhalt des Elements wird durch eine Liste, bestehend aus Attributname und Wert, spezifiziert.

deleteLiveContent Datensätze in externen Datenquellen werden gelöscht. Die zu löschenden Datensätze müssen das Filterkriterium erfüllen.

queryLiveContent Es werden Datensätze aus dem externen System abgerufen und als Resultat zurückgeliefert. Dabei kann ein Filterkriterium spezifiziert werden und zusätzlich kann auch eine maximale Obergrenze an zurückzuliefernden Datensätzen angegeben werden.

updateLiveContent Aktualisiert einen externen Datensatz. Die zu aktualisierenden Attribute sowie die neuen Werte werden in einer separaten Liste übergeben.

getLiveContent Liefert als Ergebnis einen Datensatz mit der angegebenen ID.

getLiveAvailableFields Liefert eine Liste mit allen verfügbaren Attributen zurück. Die Attribute werden zur Laufzeit ermittelt. Ein Attribut wird als *ExternalAttributeDefinition* repräsentiert (siehe Abbildung 8.6).

8.4. ImportManagement

Um die Anforderungen an ein effizientes Importmanagement (siehe Kapitel 3.2) erfüllen zu können, sind auf der Seite von Tricia zusätzliche Methoden und Konzepte notwendig. Effizientes Importmanagement beinhaltet unter anderem intelligente Ladestrategien für externe Datensätze sowie das Vorhalten bestimmter Daten in einem lokalen Cache. Nicht jede Anfrage an einen Datensatz aus einer externen Datenquelle soll automatisch in einem aufwändigen Aufruf des externen Systems enden. Um Importmanagement ermöglichen zu können, ist eine zusätzliche Erweiterung an der Schnittstelle zwischen Tricia und den externen Systemen notwendig.

Wie bereits erwähnt befindet sich die notwendige Information zum Zugriff auf die Daten von externen Systemen in den Live-Methoden der *ExternalContainer* Klasse. Die darüberliegenden Schichten fordern Datensätze aus einer bestimmt Datenquelle über die CRUD-Methoden des entsprechenden ExternalContainers an. Wie das Listing 8.3 zeigt, leitet eine CRUD Methode diesen Aufruf an die ImportManager Klasse weiter. Die ImportManager Klasse weiß nun ob die Daten im Speicher vorliegen oder ob eine Anfrage an das externe System gestellt werden muss. Falls die Daten nicht lokal vorhanden sind, muss das externe System bzw. die entsprechende Live - Methode im ExternalContainer aufgerufen werden.

Abbildung 8.9 zeigt den Ablauf einer Datenanfrage die von einem Benutzer initiiert wurde. Der ExternalContainer ruft in der CRUD - Methode die zuständige Methode des Importmanagers auf. Dieser entscheidet dann, ob die Daten in einem lokalen Cache vorliegen oder nicht. Daraufhin wird die Anfrage direkt an die Live - Methoden des ExternalContainers weitergeleitet oder das Ergebnis aus dem Cache zurückliefert. Die Informationen die von den Live - Methoden zurückgeliefert werden, werden in den lokalen Cache mitaufgenommen und als Ergebnis an die aufrufende Instanz weitergeleitet. Weitere Anfragen auf dieselben Daten können zukünftige direkt aus dem lokalen Speicher bearbeitet werden.

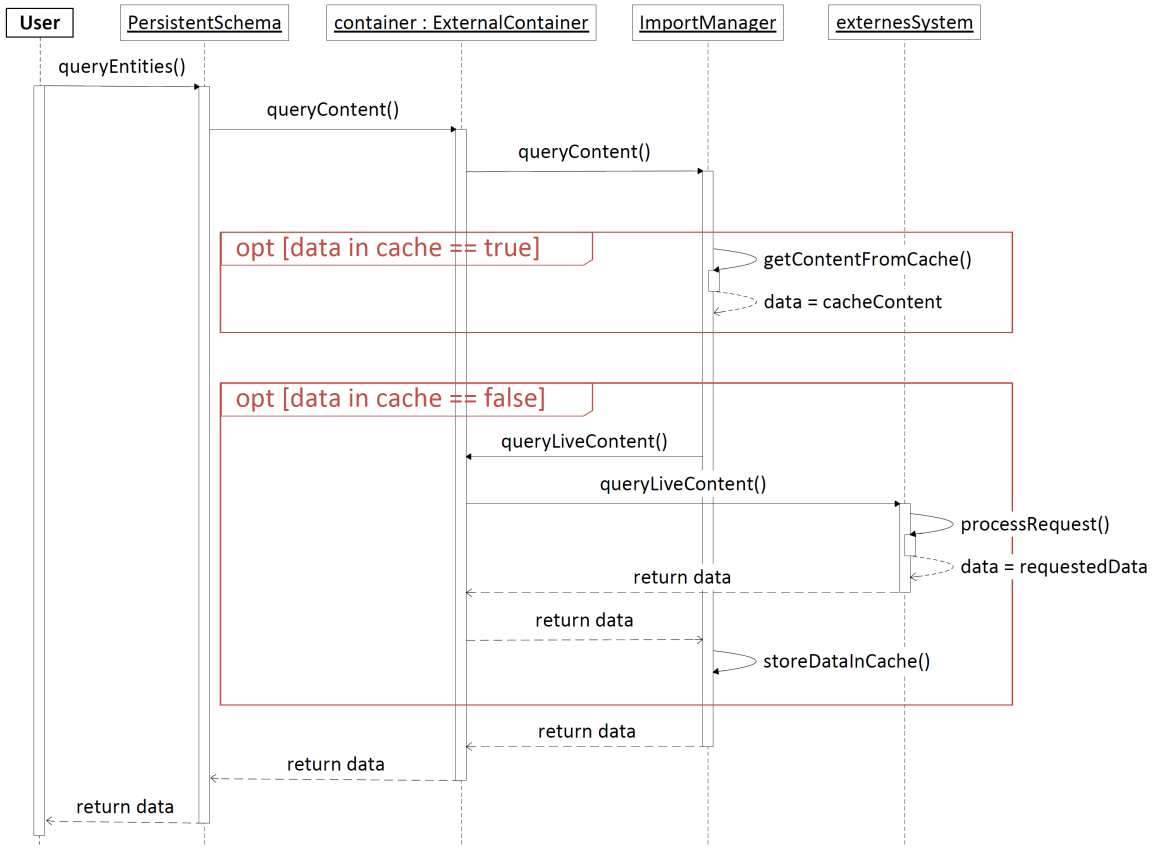


Abbildung 8.9.: Schematischer Ablauf einer Datenabfrage über das Importmanagement

8.4.1. Design

Aus den gestellten Anforderungen, ergibt sich nun eine ImportManager-Klasse, die in Abbildung 8.10 dargestellt ist. Es handelt sich dabei um eine Singleton Implementierung. Das bedeutet, dass die ImportManager Klasse bzw. die Spezialisierungen davon nur ein einziges Mal instanziiert werden kann.

Die abstrakte Basisklasse implementiert bereits die zyklischen Aktualisierungen der Daten aus den externen Datenquellen. Diese Aufgabe wird von der Methode *startAllExternalContainerTimerTasks()* wahrgenommen. Ein Timerdienst lädt in periodischen Zeitabständen alle Daten und die zugehörige Schemainformation aus den externen Datenquellen. Die Zykluszeit kann von den Methoden *setPeriod()* und *setDelay()* gesetzt werden. Die Methoden sind auch in der Konfiguration von Tricia verfügbar.

Für jede CRUD-Methode existiert im ImportManager ein abstrakter Methodenrumpf um sicherzustellen, dass für jede Methode des ExternalContainers ein entsprechendes Pendant im ImportManager vorhanden ist. Die CRUD - Methoden sind für das Erstellen, Abfragen, Aktualisieren und Löschen von Datensätzen zuständig. Neben den Datensätzen ist es vor allem die Schemainformationen einer externen Datenquelle, die sehr häufig benötigt

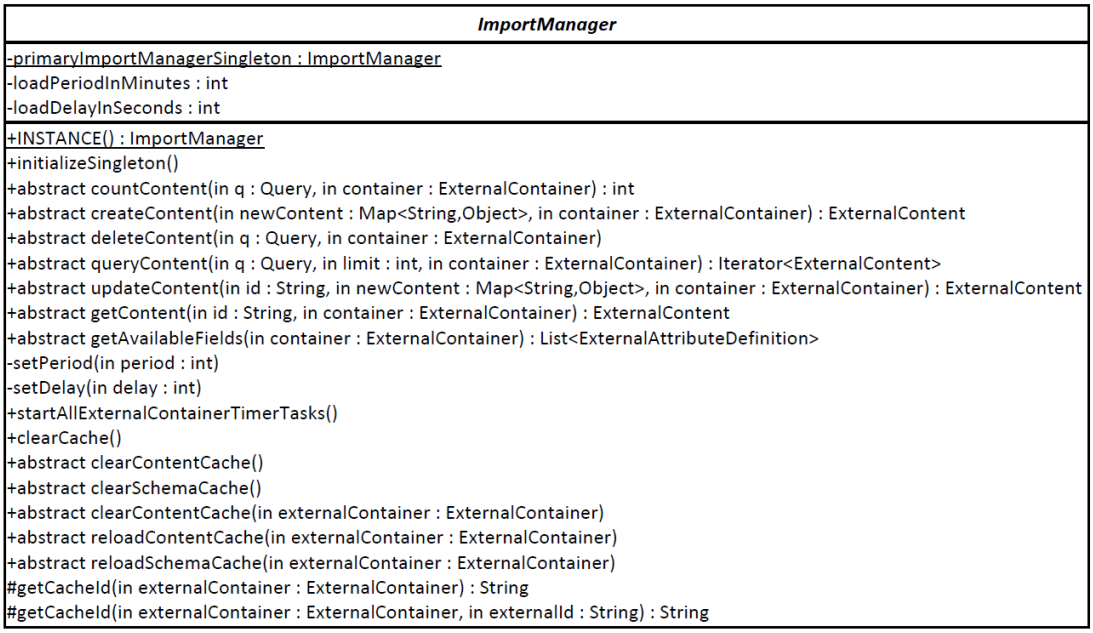


Abbildung 8.10.: Klassendiagramm der ImportManager Klasse

wird. Es bietet sich also an auch diese lokal zwischenzuspeichern. Die dafür zuständigen Methoden *clearSchemaCache()* und *reloadSchemaCache()* sind in der Klasse vorhanden.

Die Basisklasse *ImportManager* stellt eine abstrakte Klasse dar, von der jeder konkrete ImportManager ableiten muss. Durch das Singleton Entwurfsmuster ist sichergestellt, dass es in der gesamten Java Applikation Tricia nur einen ImportManager gibt. Welche Ladestrategien dieser umsetzt bzw. wie der Import von Datensätzen im Hintergrund abläuft bleibt vor der restlichen Applikation verborgen. Im Rahmen dieser Arbeit sind zwei einfache ImportManager Klassen enthalten, die im Kapitel 8.4.2 erläutert werden.

8.4.2. Implementierung

Im Rahmen dieser Arbeit wurden zwei einfache ImportManager entwickelt. Beide Varianten beinhalten noch keine Zwischenspeicherung von tatsächlichen Datensätzen. Ansätze mit intelligenten Ladestrategien und effizientem Caching sind komplex und bei der Implementierung mit viel Zeitaufwand einzuplanen.

ProxyImportManager

Bei diesem Ansatz handelt es sich um die einfachste Methode einen ImportManager zu implementieren. Es erfolgt noch keine Zwischenspeicherung. Jede Anfrage wird direkt an das externe System bzw. an die Live-Methoden des ExternalContainers weitergeleitet. Eine Methode ist in Listing 8.4 dargestellt. Dabei wird die Anzahl der verfügbaren Datensätze ermittelt und direkt auf den Inhalt der externen Datenquelle zugegriffen.

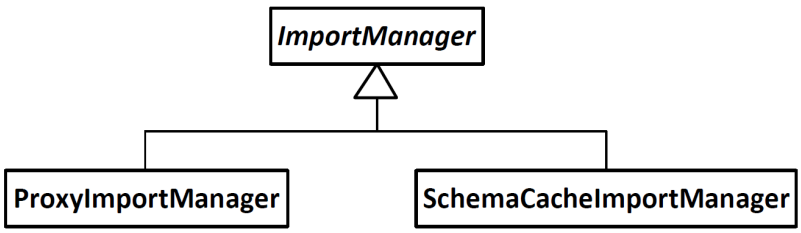


Abbildung 8.11.: ImportManager Klassenhierarchie

```
@Override
public int countContent(Query q, ExternalContainer container) {
  return container.countLiveContent(q);
}
```

Listing 8.4: Die countContent() - Methode in der ProxyImportManager Klasse

SchemaCacheImportManager

Die Klasse SchemaCacheImportManager speichert ebenfalls noch keine konkreten Datensätze zwischen, sie legt allerdings bereits die Schemainformation einer externen Datenquelle in einem lokalen Speicher ab. Die Pendants der CRUD - Methoden sind, wie beim ProxyImportManager, einfach und rufen nur die Live - Methoden des ExternalContainers auf (siehe Listing 8.4).

Eine Abfrage, die das Schema der externen Datenquelle betrifft, wird jedoch aus dem lokalen Speicher beantwort, sofern dieser bereits geladen ist (siehe Listing 8.5). Dabei wird die Liste mit den Attributinformationen aus dem Speicher für die Schemainformationen abgerufen. Über die Id eines Containers kann die Liste, die die Information beinhaltet, eindeutig zugeordnet werden. Ist keine Liste vorhanden, so wird sie geladen und kann dann aus dem Speicher gelesen werden.

```
@Override
public List<ExternalAttributeDefinition> getAvailableFields(
        ExternalContainer container) {

  Object attributes = Services.get(schemaCache,
        getCacheId(container));

  if (attributes == null) {
    reloadSchemaCache(container);
    attributes = Services.get(schemaCache, getCacheId(container));
  }
  return (List<ExternalAttributeDefinition>) attributes;
}
```

Listing 8.5: Die countContent() - Methode in der ProxyImportManager Klasse

9. Schnittstelle zu MS SharePoint: ODataContainer

In diesem Abschnitt findet eine Erläuterung darüber statt, wie die Inhalte verschiedener Listen aus SharePoint in Tricia integriert werden können. Wieso es dabei von Nutzen ist, den OData-Service von SharePoint zu nutzen, ist ebenfalls Thema dieses Abschnitts.

Wie in Kapitel 6.2 zu lesen war, wurde SharePoint in der Version 2010 als OData-Producer implementiert, d.h. SharePoint stellt die Listen aller Websites als Entity-Sets und deren Elemente als OData-Entitäten bereit.

Nun ist es vorteilhaft, statt einer konkreten SharePoint-Schnittstelle eine etwas generellere OData-Schnittstelle (ODataContainer) als Kommunikationsmittel mit SharePoint umzusetzen, da mit diesem ODataContainer alle möglichen OData-Producer als Datenquelle verwendet werden können. Darunter fallen z.B. folgende Systeme [18]:

- SAP NetWaver Gateway
- IBM WebSphere
- Microsoft SQL Azure & Windows Azure Table Storage
- Facebook
- ebay

Sollten in Zukunft weitere Informationssysteme auf dem Markt als OData-Producer fungieren, könnten diese ebenfalls über den ODataContainer in Tricia integriert werden könnten.

9.1. odata4j

Das Gegenstück zum OData-Producer ist der *OData-Consumer*. Solche Konsumenten von OData-Services können zum Einen bekannte Anwendungen wie Webbrowser, RSS-Feed-Reader oder Microsoft Excel (mit *PowerPivot for Excel 2010*) sein, zum Anderen existieren entsprechende Frameworks für die wichtigsten Plattformen (darunter Java, .NET und PHP), so dass die Entwicklung benutzerdefinierter OData-Consumer sehr stark vereinfacht wird.

Das entsprechende Framework für Java trägt den Namen “odata4j” [19],ist ein Open-Source-Projekt und liegt aktuell in der Version 0.4 vor, zugleich auch die für die Implementierung des ODataContainers verwendete Version. Abbildung 9.1 zeigt die Architektur dieses Frameworks, welches im Folgenden noch etwas genauer erläutert wird.

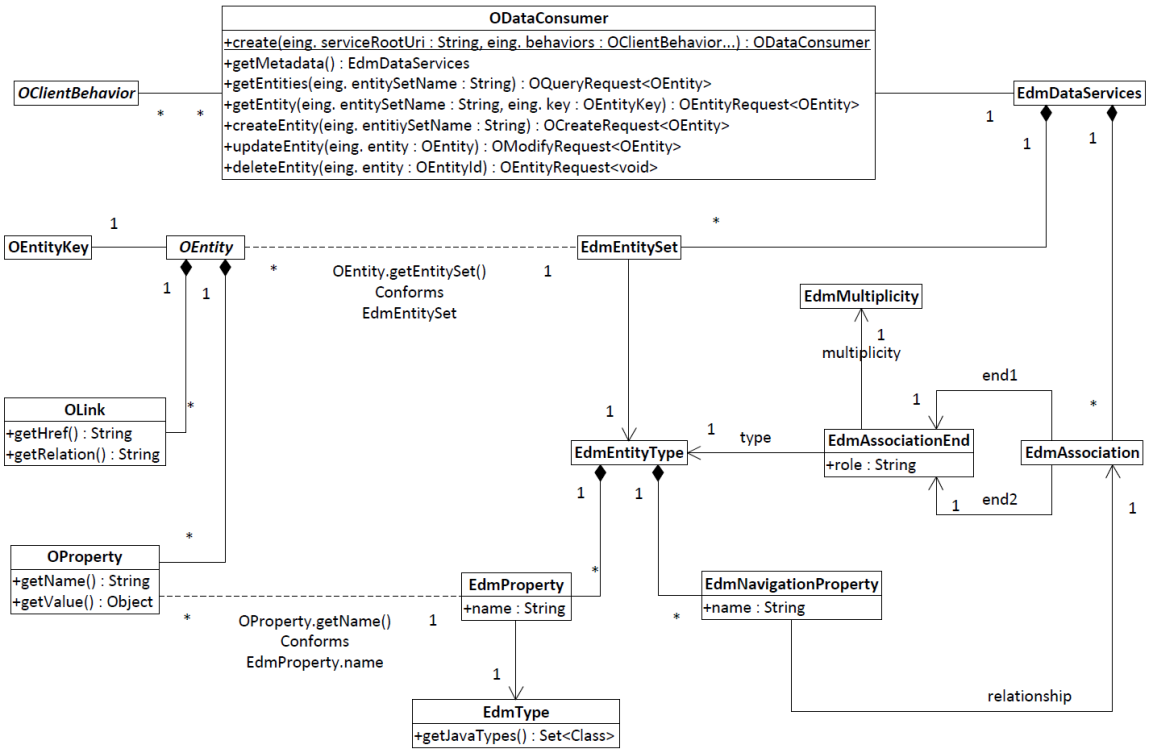


Abbildung 9.1.: Zentrale Elemente des odata4j-Frameworks und deren Beziehungen

Der Ausgangspunkt jeder Interaktion mit dem OData-Producer ist das *ODataConsumer*-Objekt, welches unter Angabe der URL des OData-Service mit der statischen *create*-Methode der ODataConsumer-Klasse erzeugt werden kann. Zusätzlich zur Service-URL ist die Übergabe von sog. *OClientBehavior*-Objekten möglich, welche der benutzerdefinierten Konfiguration des HTTP-Request-Headers dienen. Das odata4j-Framework bietet dabei bereits 4 vordefinierte OClientBehavior-Objekte an:

OClientBehaviors.basicAuth Erzeugt unter Angabe von Benutzernamen und Passwort ein OClientBehavior-Objekt, welches im HTTP-Header ein entsprechendes Authentifizierungsfeld einfügt und somit eine HTTP Basis Authentifizierung ermöglicht

OClientBehaviors.allowSelfSignedCerts Erzeugt ein Objekt, welches eine Authentifizierung per X.509-Zertifikat ermöglicht

OClientBehaviors.azureTables Erzeugt unter Angabe eines Windows Azure-Kontos ein OClientBehavior-Objekt, welches der Authentifizierung beim Cloud-Service von Microsoft dient.

OClientBehaviors.methodTunneling Das durch diese statische Methode erzeugte Objekt sorgt dafür, dass alle als Parameter übergebenen HTTP-Methoden durch eine HTTP-POST-Nachricht getunnelt werden.

9.1.1. Abfragen und Manipulieren des Contents

Über das ODataConsumer-Objekt können alle CRUD-Methoden (*Create-Read-Update-Delete*) auf OData-Entitäten ausgeführt werden. Dabei erzeugt jede dieser Methoden ein *Request*-Objekt, welches alle für das Absetzen der Abfrage benötigten Informationen enthält. Mit dem Aufruf der *execute*-Methode des Request-Objekts wird der HTTP-Request generiert und an den angegeben OData-Service-Host gesendet. Ergebnis der execute-Methode sind die angeforderten OData-Entries bzw. der angeforderte OData-Entry (*getEntities* bzw. *getEntity*), ein neu erzeugtes OData-Element (*createEntity*) oder ein Flag, welches über den Erfolg der Operation informiert (*updateEntity*). Die Ausführung von *deleteEntity* liefert keinen Rückgabewert.

Das umfangreichste Request-Objekt ist das *OQueryRequest*-Objekt als Ergebnis der *getEntities*-Methode. Dieses besitzt neben einer execute-Methode weitere Funktionen, welche bekannte Abfragemöglichkeiten wie Filterung, Sortierung und Selektion anbieten. Um die OData-Abfrage

```
.../ListData.svc/People?$select=LastName,FirstName
        &$filter=JobTitle eq 'Student'&$orderby=FirstName asc
```

exakt nachzubilden und gleichzeitig abzusetzen, wäre der in Listing 9.1 zu sehende Code notwendig. Die People-Liste bzw. das Ergebnis der Abfrage sind in Abbildung 6.5 bzw. Listing 6.2 zu sehen.

```
ODataConsumer c = ODataConsumer.create("../ListData.svc");

Enumerable<OEntity> entities = c.getEntities("People")
                    .select("LastName,FirstName")
                    .filter("JobTitle eq 'Student'")
                    .orderBy("FirstName asc")
                    .execute();
```

Listing 9.1: Abfrage der People-Liste mithilfe von odata4j

Wie in Abbildung 9.1 zu sehen ist, besitzen OData-Einträge (OEntity-Objekte) mehrere OProperties und OLinks. Ein OProperty-Objekt stellt ein Name-Wert-Paar dar, während ein OLink einen Verweis auf eine andere Entität enthält. Dieser Verweis auf eine Entität ist dabei lediglich die URL auf die entsprechende OData-Darstellung des referenzierten Elements.

Die Information, welche Properties eine Entität besitzt und welchen Typs diese sind, kann aus dem Schema seines EntitySets ausgelesen werden.

9.1.2. Abfragen des Schemas

Das Schema eines EntitySets bezeichnet grundsätzlich die Menge und Typen seiner Attribute, worunter sowohl herkömmliche als auch referenzierende Properties fallen. Zusätzlich erfolgt eine Beschreibung der Assoziationen zwischen den EntitySets (siehe Abbildung 9.1).

Die Beschreibung von EntitySets besteht im Wesentlichen aus einer Typangabe (EdmEntityType), wobei dieser Typ wiederum aus mehreren *EdmProperty*- und *EdmNavigationProperty*-Objekten besteht. Die OData-Typen der EdmProperty-Objekte lassen sich dabei jeweils auf einen entsprechenden Java-Typ mappen. Listing 9.2 zeigt, wie mithilfe von odata4j alle Attribute eines bestimmten EntitySets abgerufen werden können.

```
ODataConsumer c = ODataConsumer.create("../ListData.svc");

EdmEntitySet entitySet = c.getMetadata().findEdmEntitySet("People");

List<EdmProperty> properties=entitySet.type.properties;
```

Listing 9.2: Abfrage der Attributmenge der People-Liste mithilfe von odata4j

Während die Beziehung zwischen konkreten OData-Entitäten aus einer einfachen URL besteht (wie in Kapitel 9.1.1 zu lesen war), ist die schematische Beschreibung der Assoziation schon etwas komplexer (siehe Abbildung 9.1): Das *EdmDataServices*-Objekt als Repräsentation für den gesamten OData-Service enthält neben einer Menge an EntitySet-Beschreibungen (*EdmEntitySet*) auch eine Menge an Assoziations-Beschreibungen (*EdmAssociation*), welche von entsprechenden EdmNavigationProperty-Objekten referenziert werden können. Eine EdmAssociation beschreibt dessen zwei Endpunkte (*EdmAssociationEnd*), wobei jedes dieser Enden wiederum auf einen EdmEntityType verweist sowie über eine Multiplizität (*ONE, MANY, ZERO-TO-ONE*) verfügt.

9.2. Implementierung des ODataContainers

Der ODataContainer soll als generische Schnittstelle zu beliebigen OData-Producern dienen (siehe Abbildung 9.2). Die einzigen Angaben, die zur Implementierung eines konkreten ODataContainers notwendig sein sollen, sind:

- Absolute URL des OData-Services
- Name des EntitySets
- Benutzername und Passwort zur Anmeldung per Basisauthentifizierung

Entsprechend enthält der abstrakte ODataContainer die 4 abstrakten Methoden *getServiceUrl*, *getEntitySet*, *getUsername* und *getPassword*. Außerdem werden die beiden abstrakten Methoden *getInitialFieldsToDisplay* und *getSourceUrlTemplate* im ODataContainer nicht überschrieben, so dass auch diese vom Entwickler eines konkreten ODataContainers umgesetzt werden müssen.

Listing 9.3 zeigt, wie die schon mehrmals erwähnte beispielhafte People-Liste (siehe Abbildung 6.5) über einen ODataContainer in Tricia integriert werden kann.

Aufgabe des ODataContainers ist es also, die vom abstrakten ExternalContainer geerbten Methoden (darunter auch die CRUD-Methoden) zu implementieren, sowie einen Query-Parser für OData-Abfragen zur Verfügung zu stellen.

```
class PeopleContainer extends ODataContainer {

    @Override
    protected String getServiceUrl() {
        return "http://myspwebsite/_vti_bin/listdata.svc";
    }

    @Override
    protected String getEntitySetName() {
        return "People";
    }

    @Override
    protected String getUsername() {
        return "mustermann";
    }

    @Override
    protected String getPassword() {
        return "1234";
    }

    @Override
    public List<String> getInitialFieldsToDisplay() {
       return Lists.newArrayList("Title","FirstName");
    }

    @Override
    protected String getSourceUrlTemplate() {
        return "http://myspwebsite/Lists
                /People/DispForm.aspx?ID={<ID>}";
    }

    @Override
    public String getTypeTagName() {
        return "contact";
    };
}
```

Listing 9.3: Implementierung eines *PeopleContainer* als Konkretisierung des ODataContainers

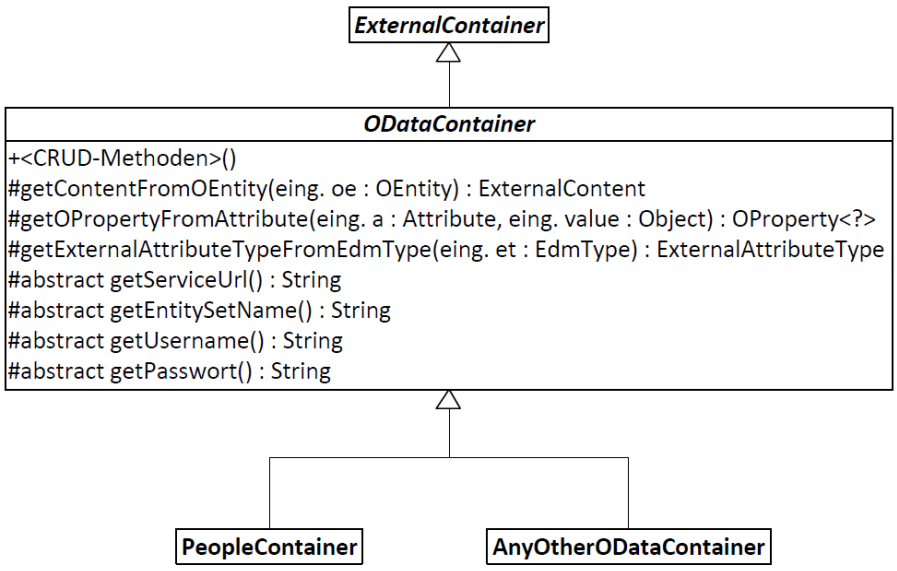


Abbildung 9.2.: Der abstrakte ODataContainer als Basis für Schnittstellen zu OData-Producern

Der Query-Parser hat die Aufgabe, Query-Objekte aus Tricia in ein OData-konformes Format umzuwandeln, d.h. in einen String, welcher der *filter*-Methode eines OQueryRequest-Objekts übergeben werden kann. So soll aus dem Query-Objekt

```
Query q = new QueryEquals(lastName,"Mustermann");
```

der String "lastName eq Mustermann" generiert werden. Dabei werden alle von OData unterstützen Vergleichsoperatoren (z.B. *startswith*, *gt* ("Greater than") oder *and* (Logisches Und)) entsprechend implementiert.

Abstrakte ODataContainer-Methoden wie *queryLiveContent*, *updateLiveContent* und *getLiveAvailableFields* werden mithilfe des in Kapitel 9.1 bereits eingeführten und erläuterten odata4j-Frameworks realisiert, weswegen detailliertere Erläuterungen zu diesen Methoden ausgelassen werden.

Noch besonders erwähnenswerte Methoden des ODataContainers sind folgende:

getContentFromOEntity Diese Methode erzeugt für eine übergebene OData-Entität ein für die weitere Verarbeitung notwendiges Content-Objekt. Dabei werden die Werte aller Attribute des aktuellen Content-Types aus der OData-Entität in das Content-Objekt übertragen. Im Falle einer Datumsangabe wird dieses in das entsprechende Format umgewandelt.

getOPropertyFromAttribute Als Gegenstück zu *getContentFromOEntity* dient diese Methode dazu, Werte aus einem Tricia-Attribut auszulesen und daraus ein OProperty-Objekt entsprechenden Typs zu erzeugen.

getExternalAttributeTypeFromEdmType Diese Methode ist für das Mapping der OData-Datentypen auf die Tricia-Datentypen zuständig (siehe Listing 9.4). Wie später in Kapitel 11 noch zu lesen sein wird, gibt es auf Seiten von Tricia 4 Datentypen: *DateTime*, *Boolean*, *Enumeration* und *Default (String)*, wobei OData selbst keine Enumerationen als Datentyp anbietet, sodass lediglich 3 Datentypen unterschieden werden müssen. Ein weiterer Typ wäre ein Link auf eine Tricia-Entität, was es erlauben würde, in OData existierende Beziehungen in Tricia zu integrieren (siehe dazu auch Kapitel 15.9).

```
protected ExternalAttributeType
        getExternalAttributeTypeFromEdmType(EdmType et) {

    if (et.equals(EdmType.DATETIME) || et.equals(EdmType.TIME)) {
        return ExternalAttributeType.getDateType();
    }

    if (et.equals(EdmType.BOOLEAN)) {
        return ExternalAttributeType.getBooleanType();
    }

    // Default type
    return null;
}
```

Listing 9.4: Die *getExternalAttributeTypeFromEdmType*-Methode des ODataContainers

10. Schnittstelle zu MS Exchange: ExchangeContainer

In diesem Kapitel wird die konkrete Implementierung der Schnittstelle zum Microsoft Exchange Server 2010 beschrieben. Die in Kapitel 7 beschriebenen Objekte und der Zugriff über die Exchange Web Services spielen dabei eine zentrale Rolle.

10.1. Client-Library für Exchange

In Kapitel 7 wurde die umfangreiche Sammlung an Funktionen und Operationen, die der Exchange Server mit seinen Web Services anbietet, bereits erwähnt. Auch auf die Problematik der Komplexität, die sich durch die vielen Web Services (über 200 verschiedene Funktionen) ergibt, wurde hingewiesen. Im Rahmen unseres Projektes nahmen wir die Aufforderung von Microsoft war und bedienten uns einer geeigneten API, die den Zugriff auf die Web Services für unsere Anforderungen entsprechend kapselt und aufbereitet.

10.1.1. Vergleich der möglichen Bibliotheken

Eine Internetrecherche liefert uns zwei verschiedene Bibliotheken, die eine API für Java-Applikationen implementieren.

EWS Java API [1] Diese Bibliothek wurde am 14. Dezember 2010 erstmals von Microsoft veröffentlicht. Bei der Veröffentlichung wurde bekanntgegeben, dass diese Bibliothek dieselbe Funktionalität anbietet, wie die EWS Managed API 1.1, die nur für das .NET Framework verfügbar ist. Die API wurde entwickelt, um die Entwicklung von plattformunabhängigen Applikationen, die auf den Exchange Server zugreifen wollen voranzutreiben. Auf der offiziellen Internetseite der Bibliothek steht auch der Quellcode zur Verfügung, dieser darf jedoch aus lizenzrechtlichen Gründen nicht modifiziert werden. Die Verwendung der Bibliothek ist hingegen kostenlos [2].

JWebServices for Exchange [3] Die zweite Bibliothek, die wir betrachteten, wurde von Independentsoft, einem deutschen Unternehmen, entwickelt. Diese Bibliothek ist kostenpflichtig. Sie arbeitet auf den Web Services der Microsoft Exchange Versionen 2007 und 2010. Die API bietet, laut Internetpräsenz, ebenfalls einen kompletten Zugriff auf die Funktionalitäten des Servers.

[1] http://archive.msdn.microsoft.com/ewsjavaapi, Juli 2011
[2] http://archive.msdn.microsoft.com/ewsjavaapi/Project/License.aspx, Juli 2011
[3] http://www.independentsoft.de/jwebservices/index.html, Juli 2011

Um die Unterschiede zu eruieren, fragten wir direkt bei dem Unternehmen Independentsoft an. Wir erhofften zu erfahren, in welchen Punkten sich die kostenpflichtige Bibliothek von der Bibliothek von Microsoft unterscheidet und ob es dazu detailliertere Dokumentation gab, denn auf der Internetseite war kaum Dokumentation verfügbar. Wir wurden auf die Tutorials und Codeausschnitte auf der Homepage verwiesen. Des Weiteren wurden wir darüber aufgeklärt, dass das Unternehmen das Microsoft Pendant zur ihrem Produkt nicht kennt, deshalb sei man sich auch der Unterschiede nicht bewusst.

Zu der EWS Java API von Microsoft hingegen gab es eine Dokumentation. Dabei wurde die Verwendung der Bibliothek und die verfügbaren Funktionen erläutert. Auf der Internetseite gibt es einen Bereich für Kommentare, bei dem Benutzer Fragen und Bugs posten können, eine Community ist also vorhanden und die Entwickler antworten auch auf Anfragen. Auch eine ständige Weiterentwicklung ist anzunehmen, so sind alleine in dem Zeitraum dieser Arbeit zwei neue Versionen veröffentlicht worden.

Nach einer genaueren Betrachtung der angebotenen Funktionen fielen mehrere Gemeinsamkeiten aber auch Unterschiede auf.

Beide Bibliotheken unterstützen komplexe Abfragen und das Filtern von Objekten. Dabei wird eine baumartige Struktur und Verschachtelung von Abfragebedingungen ermöglicht. Diese Eigenschaft ist von entscheidender Bedeutung, da beim Import immer beachtet werden muss, dass nicht benötigte Datensätze nicht unnötigerweise abgerufen und übertragen werden.

Ein Unterschied der beiden Funktionen betrifft das Konzept der Notifications (siehe Kapitel 3.2.3). So bietet die JWebService Bibliothek zwar die Möglichkeit, sich für eine Pull-Benachrichtigung einzutragen, bei der dann zyklisch die geänderten bzw. die neu erstellten Objekte abgerufen werden. Eine vom Exchange Server initiierte Benachrichtigung scheint die Bibliothek jedoch nicht zu unterstützen. Dieses Konzept der Push-Notification wird in der Microsoft API hingegen unterstützt. Dabei kann ein Web Service angegeben werden, der dann vom Server aufgerufen wird, falls sich ein Objekt geändert hat.

10.1.2. Resultat

Nach der Gegenüberstellung der beiden Bibliotheken musste eine Entscheidung getroffen werden, welche man für den Zugriff auf den Exchange Server und für die Implementierung in Tricia verwendet.

Unter Abwägung der Informationen, die wir gesammelt hatten, entschieden wir uns die freie Bibliothek von Microsoft zu verwenden. Diese unterstützt nicht nur alle unsere Anforderungen, auch das Entwicklerteam erweckt den Anschein, regelmäßiger auf Anfragen und Bugs zu reagieren. Insgesamt scheint die Bibliothek auch über eine aktivere Benutzer- und Interessengruppe im Internet zu verfügen.

10.1.3. Exchange Web Service Java Managed API

Die EWS Java API besteht aus einer einzigen Java Bibliothek. Sie benötigt jedoch vier Bibliotheken damit die korrekte Funktionsweise sichergestellt ist:

- Apache Commons HttpClient 3.1
- Apache Commons Codec 1.4
- Apache Commons Logging 1.1.1
- JCIFS 1.3.15

Die Objekttypen des Exchange Servers werden durch eine Klassenhierarchie in der Bibliothek dargestellt (siehe Abbildung 10.1).

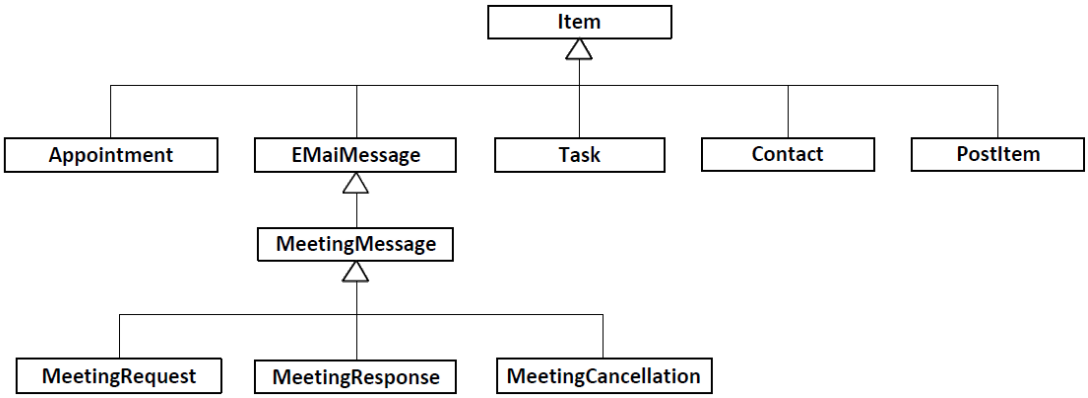


Abbildung 10.1.: Klassenhierarchie der EWS Java API [14]

Für jedes Objekt im Exchange Server existiert also auch das entsprechende Objekt auf der Clientseite in Java. Es wird an dieser Stelle auch deutlich, welche Objekte vom Server abgefragt werden können. Im Exchange Server werden die Objekte in logischen Ordnern abgelegt. Diese Ordner und die zugehörige Ordnerstruktur können ebenfalls über die API verwaltet werden. Sie erweitern den Zugriff auf die Objekte und stellen ein Containerelement für die Objekte dar. Jedes Objekt und jeder Ordner erhält vom System eine ID, mit der sie innerhalb des Exchange Servers eindeutig identifiziert werden können. Die Klassen dazu lauten: *ItemId* und *FolderId*. Kennt man die ID eines Objekts bzw. eines Ordners so kann man detailliertere Informationen abrufen.

10.2. Implementierung

Bei der Implementierung der Schnittstelle zum Exchange Server 2010 muss zwischen den möglichen Items unterschieden werden. Wie bereits weiter oben erwähnt setzt diese Arbeit die Integration für die beiden Itemtypen ”Contacts” und ”EMailMessage” um. Es wird also von der allgemeinen Basisklasse ”Exchange Container” abgeleitet (siehe Abbildung 10.2). Da der Zugriff über Web Services erfolgt, ist es notwendig die vollständige URL zu

den Exchange Web Services anzugeben. Es müssen folgende Parameter angegeben werden:

- Die vollständige URL zu dem Exchange Web Services
- Der Benutzername und das Passwort des zu verwendenden Exchange Kontos

Diese Parameter reichen aus, um eine Integration der Items vom Exchange Server in den Store von Tricia durchzuführen. Bei der vollständigen URL ist darauf zu achten, dass die Adresse das Protokoll *https* verwendet.

```
new ExchangeContactContainer() {
        @Override
        protected String getServiceUrl() {
                return "https://myEXServer/EWS/Exchange.asmx"; }

        @Override
        protected String getUsername() {
                return "mustermann"; }

        @Override
        protected String getPassword() {
                return "1234"; }

        public String getName() {
                return "ExchangeContacts";  };

        @Override
        public List<String> getInitialFieldsToDisplay() {
                List<String> defaultFieldNames =
                                new ArrayList<String>();
                defaultFieldNames.add("Surname");
                defaultFieldNames.add("GivenName");
                defaultFieldNames.add("EmailAddresses");

                return defaultFieldNames;
        }

        protected String getItemNameTemplate() {
                return "{Surname} {GivenName}"; }

        @Override
        protected String getSourceUrlTemplate() {
                return "https://myEXServer/owa";  }
```

Listing 10.1: Implementierung eines konkreten ExchangeContainer Objekts für Contact Items

Die Web Services werden vom Exchange Server nur über den damit verbunden Port 443 also über eine sichere SSL Verbindung angeboten. Zu der Ordner ID ist zu sagen, dass die IDs der Standardordner als Enumeration in dem Objekt *WellKnownFolderName* hinterlegt ist. Die ID ist nicht für jeden Itemtyp erforderlich, so gibt es für Contact Items nur einen Ordner für E-Mail Nachrichten hingegen gibt es viele verschiedene Ordner, die diese Items enthalten können.

Der ExchangeContainer implementiert, wie alle anderen Spezialisierungen des ExternalContainers, die CRUD - Operationen. Besonders bemerkenswert dabei ist, dass die Abfrage von Items aus dem Exchange Server in zwei Schritten erfolgt.

1. Zuerst wird die ID von jedem Item aus dem angegebenen Ordner, der mit der Methode getFolderId() spezifiziert ist, ermittelt.

```
ExchangeService service = getEXService();

FindItemsResults<Item> allItems = service.findItems(
        this.getFolderId(),
        getFilterFromQuery(q),
        new ItemView(limit));
```

Listing 10.2: Abfragen der Item IDs aus dem Exchange Server

2. Mit den erhaltenen IDs muss dann jedes zugehörige Item einzeln aus dem Exchange abgefragt werden.

```
T it = (T) Item.bind(getEXService(), item.getId());
```

Listing 10.3: Laden der konkreten Items aus dem Exchange Server

In Listing 10.2 wird auch die Abfrage, die Tricia erzeugt hat, konvertiert. Ein Query-Parser wird benötigt, da effiziente Importstrategien eine deutliche Verbesserung der Performance mit sich bringt (siehe Kapitel 3.2.1). Dabei wird die Abfrage in die für den Exchange Server benötigte Form gebracht und die Datensätze auf der Serverseite bereits gefiltert. Über das ItemView Objekt wird außerdem noch spezifiziert, wieviele Objekte maximal in der Ergebnisliste zurückgegeben werden können. Die Methode getEXService() erstellt aus den angegebenen Parametern (URL, Username, Passwort) das Objekt, das den Zugriff auf den Exchange Server ermöglicht. Das ExchangeService Objekt ermöglicht die Abfrage von Elementen, Daten und Eigenschaften des Exchange Servers.

Zwei Methoden, die in der Klasse ExchangeContainer besonders auffallen, sind zuständig für das Auslesen von Eigenschaften bzw. Attributen von Spezialisierten Items wie Contact oder EMailMessage. Dabei handelt es sich um folgende Methoden:

- processSpecific
- setSpecific

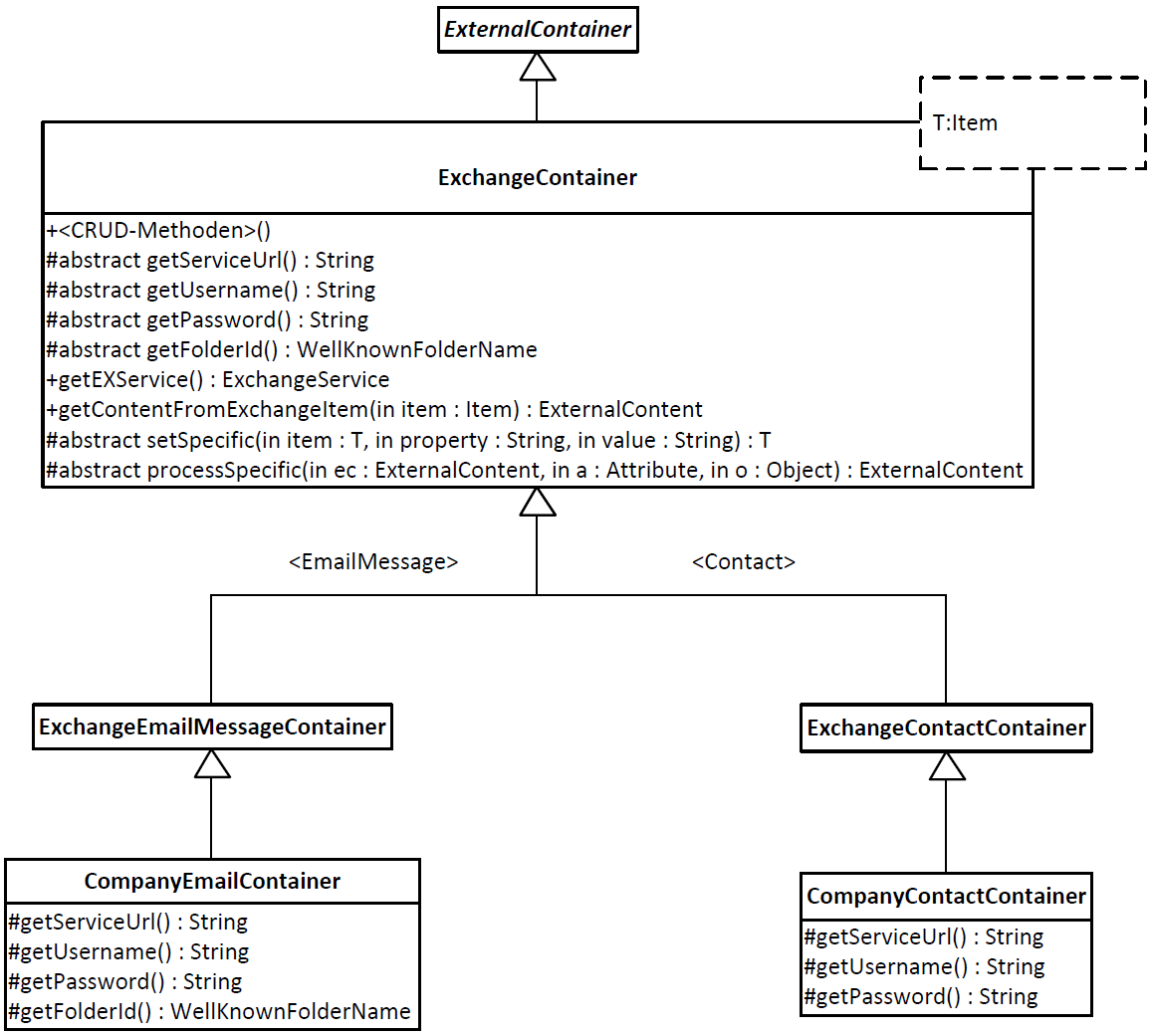


Abbildung 10.2.: Der abstrakte ExchangeContainer als Basis für Schnittstellen zu konkreten Item-Objekten im Exchange Server 2010

Beim Auslesen aller Attribute aus einem Item muss unterschieden werden, um welchen Itemtyp es sich handelt, da die verschiedenen Itemtypen auch verschiedene Attribute anbieten. So sind in den Kontaktentitäten diverse Informationen zu einer Person gespeichert (z.B. Telefonnummern, Adressen, Beruf, ...). Diese Informationen sind in einer E-Mail jedoch nicht enthalten. Die vorgestellten Methoden lesen jetzt genau die Attribute aus den Entitäten aus, die in dem vorliegenden Item vorhanden sind. Somit ist sichergestellt, dass jede verfügbare Information auch tatsächlich ausgelesen wird und in Tricia verfügbar ist.

Teil IV.

Schicht 2: Repräsentation externer Daten in Tricia

11. Evaluierung der Möglichkeiten zur Repräsentation externer Daten

Schicht II der Datenintegrations-Architektur befasst sich mit der Repräsentation der integrierten Daten in Tricia, während in Schicht I (Teil III der Arbeit) die Schnittstelle zu den externen Datenquellen behandelt wurde. Dabei spielt besonders die Repräsentation einzelner Datensätze (siehe Kapitel 13) sowie die Abbildung des Schemas (siehe Kapitel 12) der externen Datenquelle im Datenmodell von Tricia eine wichtige Rolle.

Die Aufgabe besteht darin, eine Entität aus einer externen Datenquelle adäquat auf Ebene des Datenmodells zu repräsentieren. Eine Entität stellt dabei einen konkreten Datensatz der externen Datenquelle dar, welcher über mehrere Attribute verfügen kann. Jede Entität ist dabei eine konkrete Ausprägung eines Entitätstyps, welcher die schematische Beschreibung der Attribute darstellt. Stelle man sich beispielsweise eine Datenbanktabelle vor, so ist die Beschreibung der Tabellenspalten der Entitätstyp, wohingegen jede Zeile der Tabelle einer Entität entspricht.

In diesem Kapitel findet nun eine Evaluierung zweier Ansätze zur Repräsentation einer Entität sowie dessen Entitätstyps auf Ebene des Datenmodells in Tricia statt: Darstellung einer Entität mithilfe von Built-In-Typen oder als Hybrid.

11.1. Ansatz 1: Built-In-Typen

Built-In-Typen sind im Wesentlichen Klassen und deren Membervariablen stellen die Attribute des Entitätstyps dar. Entitätstypen werden also durch die Klasse selbst und Entitäten durch Instanzen davon dargestellt. In Tricia existiert dafür eine spezielle Klasse namens *PersistentEntity*, welche das Persistieren der Entität und derer Attribute, welche im Kontext dieser Tricia-Entitäten "Properties" heißen, erlaubt. Die Klasse ist dabei ein *Built-In-Typ*, d.h., eine kompilierte Klasse und damit abgeschlossene Einheit, so dass das Schema zur Laufzeit nicht veränderbar ist. Entsprechend sind Built-In-Entitäten Instanzen dieser Built-In-Typen.

Der Ansatz der Built-In-Typen würde also vorsehen, einen konkrete Datensatz einer externen Datenquelle in Tricia als persistente Entität und die Attribute des Datensatzes als Properties in Form von Membervariablen verschiedenen Typs darzustellen. Eine externe Entität (konkreter Datensatz einer externen Datenquelle) mit den Attributen "Fullname", "Birthday" und "MailAddress" würde dabei wie in Abbildung 11.1 dargestellt werden.

Instanzen persistenter Entitäten stellen in Tricia auf Store-Ebene Content-Objekte dar, welche Teil von Container-Objekten sind (Details dazu sind in Kapitel 5.4 zu lesen). Nach

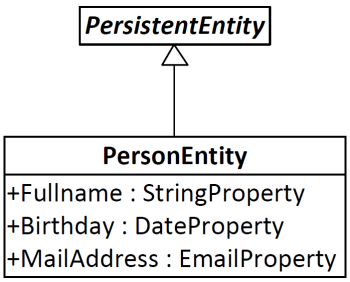


Abbildung 11.1.: Klasse als Darstellung eines Entitätstyps mit den Attributen “Fullname”, “Birthday” und “MailAddress”

dem Ansatz der Built-In-Typen entsprechen also Entitäten, welche Elemente externer Datenquellen darstellen, ExternalContent-Objekten, die wiederum Teil von ExternalContainer-Objekten sind (siehe Kapitel 8). Um die Datenintegrations-Anforderung der zusätzlichen Attribute zu erfüllen (siehe Kapitel 3.2.4), wäre dabei jedoch eine Mischform notwendig, d.h., eine Tricia-Entität stellt auf Store-Ebene 2 Content-Objekte dar, wobei eines davon ein ExternalContent-Objekt mit zugehörigem ExternalContainer und das andere ein “lokales” Content-Objekt darstellt, welcher die an die externe Entität angeheftete Information enthält.

Das Schema der externen Datenquelle würde in Ansatz 1 also direkt in die Klassendefinition der entsprechenden Tricia-Entität übertragen und dort in Form von Properties verschiedener Typen dargestellt werden, wohingegen die Inhalte der externen Datenquelle als Ausprägungen des Built-In-Typs realisiert werden würden.

11.2. Ansatz 2: Hybrids

Das Konzept der Hybrids wurde bereits in Kapitel 4 eingeführt. Demnach ist eine hybride Entität ein Objekt, an welches (auch zur Laufzeit) HybridProperties (einfache Name-Wert-Paare) und Typen (als Type Tags) angeheftet werden können. Mithilfe der Type Tag Definitionen können zusätzlich Attribut Definitionen erzeugt werden, welche die Angabe bestimmter Validatoren erlauben, z.B. Einschränkungen hinsichtlich des Typs. In Tricia heißen diese Attribut Definitionen *HybridPropertyDefinition*, und deren Typ-Valdiatoren sind dabei auf folgende Datentypen beschränkt:

- Date
- Boolean
- Enumeration
- Link (auf andere Hybrids)
- Default (keine Typangabe, einfacher String)

Generell sind Properties von Hybrids so konzepiert, dass eine Property über mehrere Werte verfügen kann, d.h. der Wert einer solchen Property ist eine Liste von Objekten. HybridPropertyDefinitions können dazu bestimmte Validatoren besitzen, welche Aussagen über die Multiplizität der Property treffen (Genau 1 Wert, mindestens 1 Wert, maximal 1 Wert).

In Tricia werden Hybrids mithilfe von sog. *Mixins* realisiert. Das Mixins-Konzept ersetzt im Grunde die Fähigkeit der Mehrfachvererbung. Enthält eine Tricia-Entität in dessen *Mixin-Auflistung* (eine Liste von Mixin-Klassen, von denen "abgeleitet" wird) ein Objekt des Typs "Hybrid", so wird die Entität selbst *Hybrid* genannt und dazu befähigt, über beliebig viele Properties sowie Type Tags zu verfügen. Abbildung 11.2 zeigt dazu einen beispielhaften Auszug zur Realisierung des Mixin-Konzepts in Tricia.

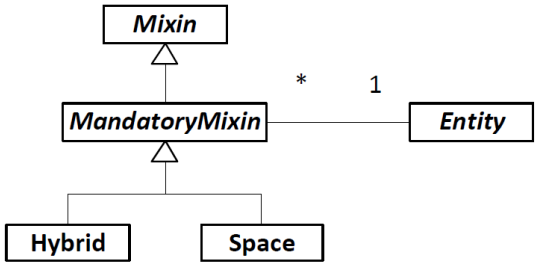


Abbildung 11.2.: Beispielhafter Auszug aus der Realisierung des Mixin-Konzepts in Tricia

Der zweite Ansatz zur Repräsentation externer Daten sieht also die Erweiterung der Hybrids vor, sodass die Werte deren Properties nicht mehr aus einem lokalen Speicher stammen, sondern aus externen Datenquellen abgerufen werden.

In diesem Ansatz würde der externe Entitätstyp (also das Schema der externen Datenquelle) auf Seiten von Tricia als Type Tag Definition realisiert werden, d.h. für jedes Attribut des Entitätstyps wird eine HybridPropertyDefinition erzeugt, optional mit entsprechenden Typvalidatoren und Multiplizitäten. Die konkreten Datensätze würden als Instanzen hybrider Entitäten implementiert werden, wobei dessen konkreten Attribute als HybridProperties umgesetzt werden würden.

Wollte man beispielsweise einen Entitätstyp mit den Attributen "Fullname", "Birthday" und "MailAddress" in Tricia mithilfe dieses Ansatzes darstellen, so würde dieser wie in Abbildung 12.2 dargestellt werden.

11.3. Evaluierung

Die beiden Ansätze unterscheiden sich in einigen Aspekten, von denen einige die Möglichkeit zur Abdeckung bestimmter Anforderungen an der Datenintegration betreffen (siehe Kapitel 3.2).

11.3.1. Datentypen

Das Mapping von Datentypen externer Datenquellen auf Datentypen von Tricia ist eine der Anforderungen an der Datenintegration (siehe Kapitel 3.2.5) und ein Aspekt, in welchem sich die beiden Ansätze unterscheiden. Dabei ist hier weniger von Bedeutung, wie die Datentypen zugeordnet werden, sondern vielmehr, welche Datentypen in den beiden Ansätzen überhaupt verfügbar sind.

Ansatz 1

Tricia bietet standardmäßig bereits eine ganze Menge an verschiedenen Property-Typen an (siehe Abbildung 11.3). Außerdem können neue Typen ganz einfach durch die Implementierung neuer Property-Klassen eingeführt und anschließend in Entitäten genutzt werden, vorausgesetzt, diese neuen Typen sind Erweiterungen bestehender Propertytypen. Die Implementierung sehr spezieller Propertytypen hätte weitreichende Änderungen im Kern von Tricia zur Folge.

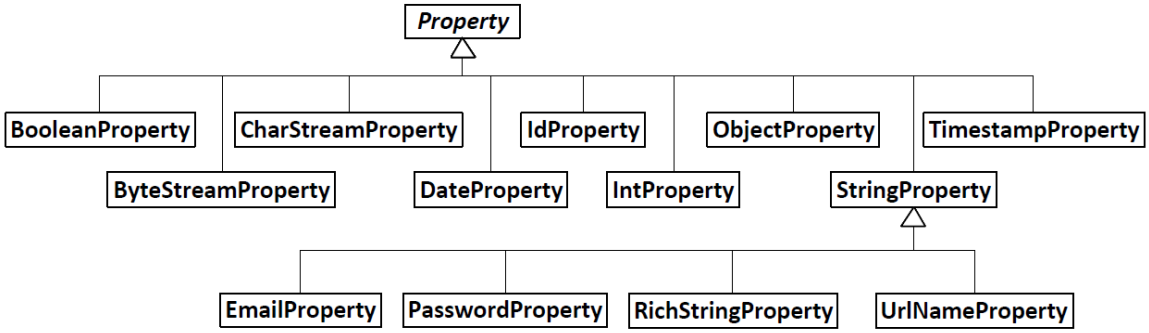


Abbildung 11.3.: Verschiedene Typen von Tricia-Properties

Ansatz 2

HybridProperties, wie die Properties von Hybrids in Tricia genannt werden, bieten streng genommen gar keine Datentypen an, bzw. sind immer entweder vom Typ String oder ein Link auf eine andere Entität. Allerdings lassen sich spezielle Einschränkungen (*Constraints*) definieren, um ein bestimmtes Format des Wertes einer HybridProperty vorzugeben. (siehe Abbildung 11.4). Beispielsweise würde ein *BoolanConstraint* dazu führen, dass der String-Wert einer HybridProperty entweder “true” oder “false” ist bzw. das der Benutzer zumindest darauf hingewiesen wird, dass der aktuelle Wert nicht der Definition durch die HybridPropertyDefinition entspricht.

Im Gegensatz zu den Property-Typen lassen sich jedoch nicht so einfach neue Typ-Einschränkungen implementieren, ohne weitere Änderungen im Kern von Tricia vorzunehmen (Dies betrifft die *applyParameters*-Methode der Klasse *HybridPropertyDefinition*). Unter anderem besteht der Code zur Ermittlung einer bestimmten Typ-Einschränkung anhand dessen Namens aus einer Reihe von Typabfragen, sodass bei der Implementierung

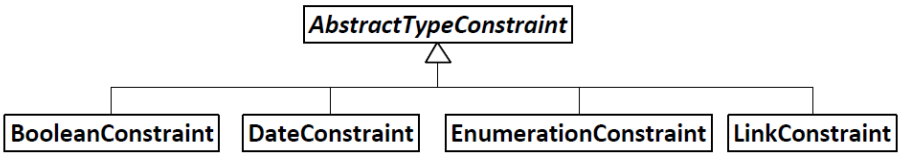


Abbildung 11.4.: Verschiedene Typen von Constraints als Pseudodatentyp von HybridProperties

einer neuen Typ-Einschränkung eine Änderung dieses Codes notwendig wäre.

Vergleich

In Bezug auf Datentypen bietet Ansatz 1 bedeutende Vorteile: Die Datentypenvielfalt der Properties erlaubt es, viele der Datentypen der externen Datenquelle auf analoge Propertytypen von Tricia abzubilden, wohingegen Hybrids lediglich 4 Typ-Einschrängungen bieten. Ebenso ist eine Erweiterung durch die Einführung neuer Datentypen im ersten Ansatz deutlich einfacher, sofern es sich dabei nicht um ganz spezielle Datentypen handelt.

11.3.2. Flexibilität

Ändert sich das Schema der externen Datenquelle, so wird von Tricia gefordert, entsprechend darauf zu reagieren. Ebenso soll der Aufwand zur Integration weiterer externer Datenquellen ein möglichst geringer sein. Dies sind zentrale Anforderungen an der Datenintegration (siehe Kapitel 3.2.2), welche beide Ansätze unterschiedlich gut abdecken können.

Ansatz 1

Flexibilität ist in Ansatz 1 eigentlich gar nicht vorhanden. Da das Schema der Entität in Form einer Klasse mit deren Properties realisiert wird, kann zur Laufzeit weder die Menge der Properties noch deren Datentyp geändert werden.

Des Weiteren ist für jede einzelne externe Datenquelle eine neue Klasse, welche das Schema der entsprechenden Datenquelle darstellt, zu implementieren. Das Hinzufügen neuer Datenquellen ist also ein nicht zu vernachlässigender Aufwand.

Ansatz 2

Bei einer Änderung des Schemas der externen Datenquelle kann zur Laufzeit die entsprechende Type Tag Definition angepasst werden, indem diese die Menge der HybridPropertyDefinitions an das neue externe Schema anpasst. Zusätzlich müssen alle Instanzen der Hybrids ihre ExternalHybridProperties aktualisieren.

Zur Integration weiterer externer Datenquellen ist kein Zusatzaufwand notwendig, da dafür einfach neue Type Tag Definitionen erzeugt werden können, welche wiederum das

externe Schema mithilfe der HybridPropertyDefinitions in Tricia abbilden.

Alles in Allem ermöglicht dieser Ansatz also eine sehr gute Anpassung an jegliche Änderungen des externen Schemas sowie eine einfache Integration weiterer externer Datenquellen, da hier genau ein hybrider Entitätentyp reicht, welcher alle möglichen externen Daten repräsentieren kann.

Vergleich

In Punkto Flexibilität ist der Ansatz der fest-programmierten Entitäten weniger gut geeignet, da diese Art der Entitäten in keinster Weise auf Änderungen des Schemas externer Systeme oder der Integration weiterer externer Datenquellen reagieren kann.

Im Gegensatz dazu bieten Hybrids eine bessere Abdeckung der Flexibilitätsanforderung an Datenintegrationslösungen.

11.3.3. GUI der Entitäten

Während bisher unter der Repräsentation von Entitäten dessen Abbildung im Datenmodell von Tricia gemeint war, behandelt dieser Abschnitt die Darstellung von Entitäten auf Ebene der Benutzeroberfläche, also dessen Präsentation dem Endbenutzer gegenüber. Dies ist zugleich auch ein Aspekt der Datenintegration, in welchem sich die beiden Ansätze zur Repräsentation externer Daten sehr stark unterscheiden.

Ansatz 1

Für jede eine externe Datenquelle repräsentierende Tricia-Entität sind entsprechende Views und Handler zur Anzeige und Manipulation der externen Daten zu erstellen, was bedeutet, dass bei der Integration weiterer externer Datenquellen ein zusätzlicher Mehraufwand anfällt.

Listing 11.1 zeigt ein beispielhaftes View-Template zur Anzeige einer Entität, welche die schon mehrfach erwähnte People-Liste aus Abbildung 6.5 repräsentiert. Im Listing ist zu sehen, dass jede Property explizit angegeben werden muss, was den 1. Ansatz zur Repräsentation externer Daten wiederum sehr unflexibel in Bezug auf Änderungen des Schemas macht.

```
<h1>$LastName.show()$</h1>
$FirstName.show()$
$JobTitle.show()$
```

Listing 11.1: Beispielhaftes View-Template zur Anzeige dreier Properties

Als Alternative ließe sich hier eine ähnliche Funktionalität implementieren, wie sie gleich für Ansatz 2 erwähnt wird, jedoch wäre dies mit einem nicht unerheblichen Programmieraufwand verbunden. Dazu müsste man alle Properties einer Entität als Collection abrufen können und entsprechende Views realisieren, welche diese Properties auf geeignete Art

und Weise anzeigen, eine Möglichkeit zur Manipulation derer Werte ermöglichen sowie Property-eigene Aspekte beachten (z.B. Validatoren).

Ansatz 2

Der zweite Ansatz macht sich die bereits Vorhandenen Views des *hybridWiki*-Plugins zu Nutze, welches u.a. ein Template für die sog. *Hybrid-Tabelle* enthält. Diese Tabelle zeigt alle HybridProperties sowie alle Type Tags einer hybriden Entität in tabellarischer Form an und bietet zusätzlich Möglichkeiten an, deren Werte in-place zu editieren. Um die HybridProperties einer hybriden Entität anzuzeigen (siehe Abbildung 11.5), reicht folgendes Stück Code:

```
$hybridTable()$
```

Dadurch, dass diese Funktionalität bereits vorhanden ist und vom Kontext der Wikiseiten einfach auf externe Inhalte repräsentierende Entitäten übertragen werden kann, fällt für die Erzeugung von Ansichten keinerlei Aufwand an.

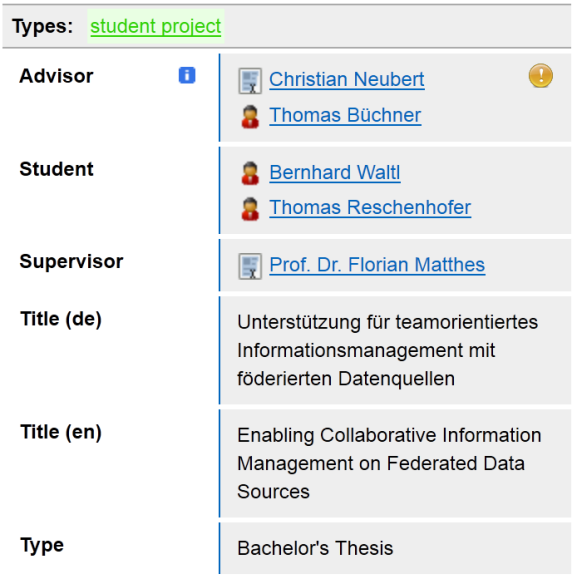


Abbildung 11.5.: Die Hybrid-Tabelle aus dem hybridWiki-Plugin

Vergleich

Nachdem in Ansatz 2 zur Anzeige externer Inhalte auf Tricia-Webseiten kein Mehraufwand anfällt, da die vorhandene Funktionalität aus dem hybridWiki-Plugin genutzt werden kann, hat dieser Ansatz im Vergleich zu Ansatz 1 erhebliche Vorteile bezüglich des Programmieraufwands.

Built-In-Entitäten erfordern hingegen entweder für jede einzelne externe Datenquelle eigene Views oder die Implementierung eines ähnlichen Mechanismus, wie ihn die Hybrid-Tabelle für HybridProperties bietet.

11.4. Ergebnis der Evaluierung

Speziell aufgrund der Vorteile bezüglich der Flexibilität wurde der zweite Ansatz zur Repräsentation externer Daten in Tricia ausgewählt. Zwar bieten Hybrids nicht die Fülle an Datentypen wie Built-In-Typen, jedoch wiegen die beiden Vorteile der Flexibilität und Verwendbarkeit der Hybrid-Tabelle zur Anzeige der Daten deutlich schwerer als der Nachteil der eingeschränkten Datentypenmenge. Diese Flexibilität hat auch zur Folge, dass die Implementierung und Registrierung eines ExternalContainers der einzige Aufwand ist, welcher bei der Integration einer neuen Datenquelle anfällt, d.h., in Schicht II der Architektur ist keine Erweiterung notwendig.

Die Auswahl hybrider Entitäten zur Repräsentation erfordert gezielte Anpassungen des Hybrid Wiki Modells aus Abbildung 4.1. Dabei finden keine gravierenden Änderungen der Tricia-Architektur statt, sondern lediglich Erweiterungen bestehender Klassen durch die Erzeugung neuer von den existierenden ableitenden Klassen.

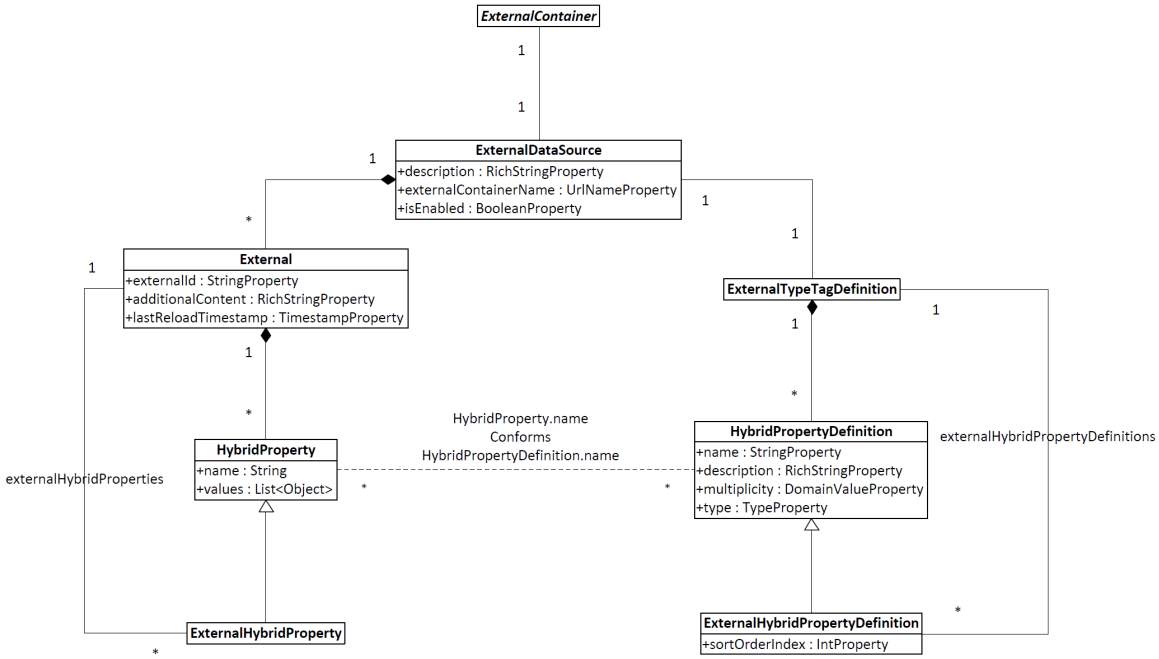


Abbildung 11.6.: Das *External Modell* als Modifikation des Hybrid Wiki Models nach Matthes, Neubert und Steinhoff [11]

In Abbildung 11.6 ist das sog. *External Modell* als Modifizierung des Hybrid Wiki Modells zu sehen.

Den Platz des Wikis nimmt in diesem Modell die externe Datenquelle ein, welche zum Einen Informationen zum Schema (über die *ExternalTypeTagDefinition*) und zum Anderen den Inhalt der Datenquelle (als *External*) bereitstellt.

Das External als Repräsentation eines einzelnen Eintrages aus einer externen Daten-

quelle besitzt sowohl *ExternalHybridProperties*, welche externe Daten darstellen, als auch herkömmliche HybridProperties, welche lokal (in Tricia) persistiert werden und die Anheftung zusätzlicher Attribute ermöglichen, ohne diese an das externe System zu übertragen.

Analog dazu besitzt eine ExternalTypeTagDefinition neben *ExternalHybridPropertyDefinitions*, welche das Schema der externen Datenquelle darstellen, auch herkömmliche HybridPropertyDefinitions zur Charakterisierung herkömmlicher HybridProperties.

Im Vergleich zum Hybrid Wiki Modell nehmen ExternalHybridProperties und HybridProperties den Platz der Attribute ein und ExternalHybridPropertyDefinitions und HybridPropertyDefinitions den Platz der Attribute Definitionen ein.

12. Externe Datenquelle

Ein ExternalContainer-Objekt bietet im Wesentlichen drei Arten der Information einer externen Datenquelle an: Metainformation (z.B. Name oder Beschreibung), Schemainformation (angebotene Attribute) und konkrete Daten (Abfrage und Manipuliation von Datensätzen). Genau wie in Schicht I der Architektur, beschrieben in Teil III der Arbeit, existiert für jeden dieser Informationstypen ein entsprechendes Objekt, welche allesamt im External Modell (siehe Abbildung 11.6) abgebildet sind:

- *ExternalDataSource* als Beschreibung der Datenquelle selbst (Metainformationen)
- *ExternalTypeTagDefinition* als Schemabeschreibung
- *External* als Darstellung eines einzelnen Datensatzes

Dieses Kapitel beschäftigt sich mit ersteren beiden, während letzteres in Kapitel 13 behandelt wird.

12.1. Objekte zur Repräsentation der Datenquelle und dessen Schemas

Im Folgenden findet eine Erläuterung zu Abbildung 12.1 statt, welche einen Teil des External Modells detaillierter wiedergibt.

12.1.1. ExternalDataSource

Die Datenquelle selbst wird dabei als persistente Tricia-Entität dargestellt und verfügt über folgende drei Properties:

externalContainerName Über diese Property wird die Beziehung zum ExternalContainer hergestellt, welcher als Schnittstelle zu jener externer Datenquelle dient, welche durch die Tricia-Entität dargestellt wird.

description Dieses Feld beinhaltet eine RichText-Beschreibung der Datenquellen

isEnabled Womöglich soll nicht jede Datenquelle, welche mithilfe eines ExternalContainers anbindbar wäre, d.h., für welche ein entsprechender ExternalContainer registriert wurde, auch tatsächlich in Tricia integriert werden. Über dieses boolsche Feld lässt sich dieser Umstand steuern.

Jedes ExternalDataSource-Objekt verfügt über genau ein zugehöriges ExternalTypeTagDefinition-Objekt als Beschreibung des entsprechenden Schemas. Diese ExternalTypeTagDefinition ist eine Spezialisierung der TypeTagDefinition aus dem Hybrid Wiki Modell von Abbildung 4.1.

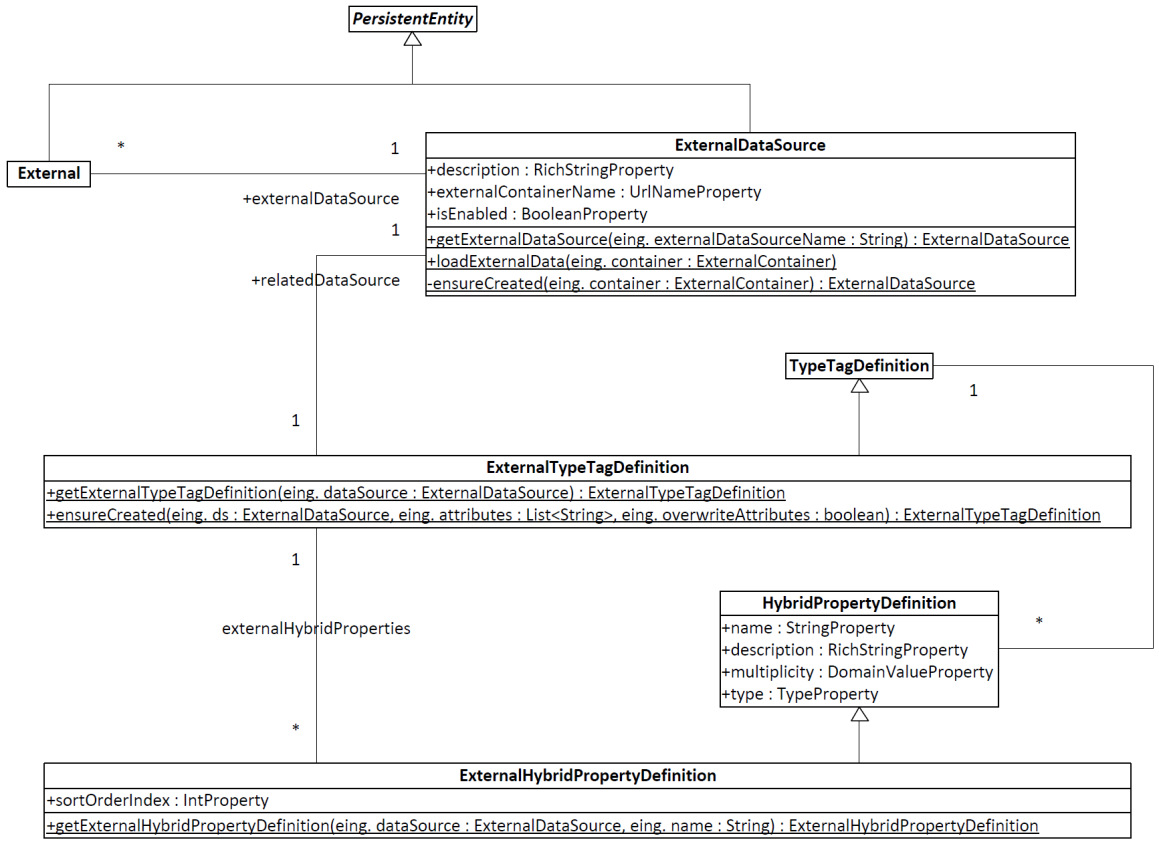


Abbildung 12.1.: Detaillierterer Ausschnitt aus dem External Modell, welcher sich mit der Datenquelle selbst und dessen Schemabeschreibung befasst

12.1.2. ExternalTypeTagDefinition

Zur Darstellung und Persistierung des externen Schemas in Tricia wird für jedes Attribut der externen Datenquelle ein dieses Attribut repräsentierendes ExternalHybridProperty-Definition-Objekt erzeugt. Diese können neben dem Namen des Attributs zusätzliche Informationen wie Typ, Multiplizität und Beschreibung enthalten. Nähere Informationen zu Typ und Multiplizität sind unter Kapitel 11.2 zu lesen. Zusätzlich enthält ein External-HybridPropertyDefinition-Objekt die Property *sortOrderIndex*, mit welcher es möglich ist, eine spezielle Reihenfolge bei der Anzeige der HybridProperties eines externen Datensatzes zu definieren.

Wie solch eine Schemabeschreibung mithilfe der ExternalTypeTagDefinition aussehen kann, soll ein Beispiel zeigen. Ausgangspunkt soll dabei eine Datenquelle “Personen” sein, welche folgende 3 Attribute anbietet:

Fullname Ein normales Textattribut, welches mit Ausnahme einer Beschreibung keine weiteren Angaben enthält

Birthday Ein Attribut vom Typ DateTime

MailAddresses Ein Attribut mit Mehrfachwert, wobei zumindest 1 Wert angegeben sein muss

Existiert zu dieser Datenquelle ein registrierter ExternalContainer und ist die entsprechende ExternalDataSource-Entität aktiviert, so wird aus dem genannten Schema die in Abbildung 12.2 Objektkonstellation generiert werden.

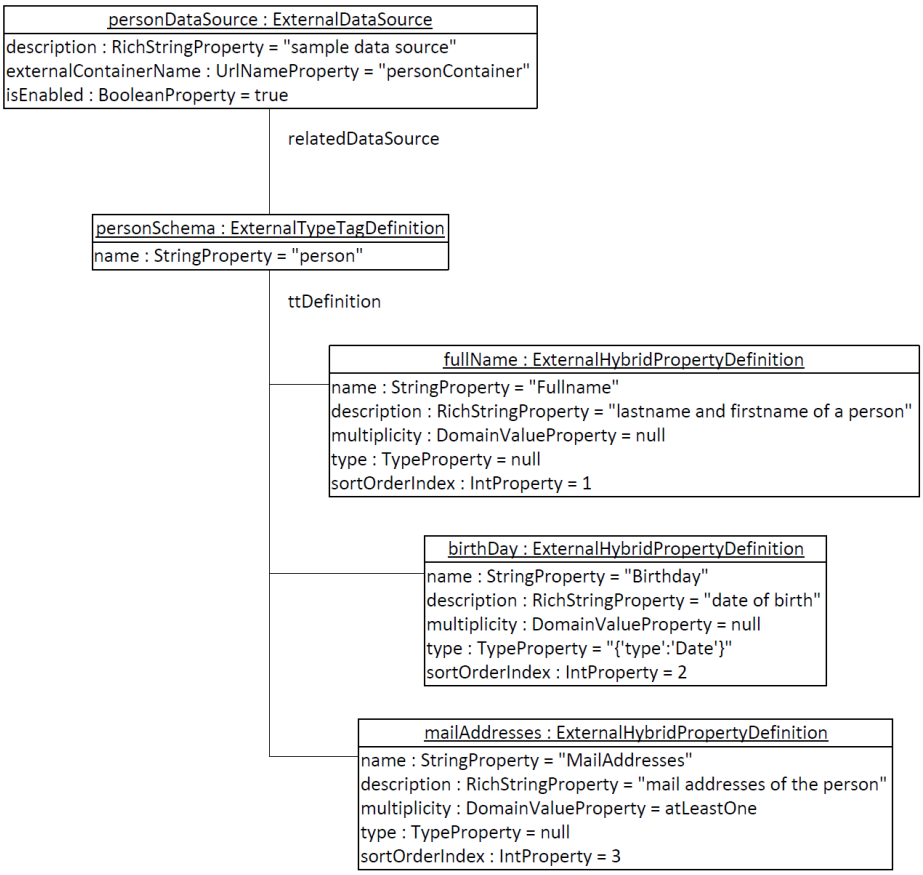


Abbildung 12.2.: Beispielhafte Ausprägung des Schemas einer externen Datenquelle in Tricia mithilfe eines ExternalTypeTagDefinition-Objekts

Genau jene Attribute, die mithilfe eines ExternalHybridPropertyDefinition-Objekts dargestellt werden, werden in den einzelnen Datensätzen in der Hybrid-Tabelle angezeigt, sodass sich über diese die Ansicht steuern lässt. Genau genommen repräsentiert die ExternalTypeTagDefinition also nicht das exakte Schema der externen Datenquelle, sondern eine für eine konkrete Anwendung optimierte Version davon, da nur noch jene Attribute geladen werden, welche auch tatsächlich angezeigt werden sollen.

12.1.3. ExternalHybridPropertyDefinition

Den Namen des ExternalTypeTagDefinition-Objekts liefert die Methode *getTypeTagName* des ExternalContainers, während die Methode *getAvailableFields* eine Liste von ExternalAttributeDefinition-Objekten liefert, aus welchen entsprechende ExternalHybridProperty-

Definition-Objekte generiert werden.

Der Name ist das einzige Feld, das in der ExternalTypeDefinition angegeben werden muss und auch genauso in die passende Property der ExternalHybridPropertyDefinition übertragen wird. Alle anderen Angaben sind optional.

Die Typangabe der ExternalTypeDefinition wird als String in JSON Notation in der *type*-Property des ExternalHybridPropertyDefinition-Objekts abgelegt. Ist der Typ des Attributs beispielsweise eine Enumeration mit den Werten “eins”, “zwei” und “drei”, so lautet der entsprechende Typ-String folgendermaßen:

```
{"type":"Enumeration","elements":["eins","zwei","drei"]}
```

Aus diesem Typ-String generiert Tricia passende Constraint-Objekte, welche für die Validierung der Attributwerte sorgen. Invalide Hybridproperties werden beispielsweise mit einer entsprechenden Validierungsnachricht angezeigt.

Die Multiplizität wird in der ExternalAttributeDefinition als Enumeration mit den Werten *exactlyOne*, *maxinmalOne* und *atLeastOne* dargestellt, wobei als vierte Option keine Multiplizität angegeben werden muss. Diese Werte werden in der ExternalHybridPropertyDefinition auf entsprechende *DomainValue*-Werte umgelegt, welche wiederum für die Validierung der Attributwerte sorgen.

Die *sortOrderIndex*-Property wird als einzige Information nicht aus der Schemainformation der externen Datenquelle ausgelesen, sondern kann über das User-Interface (siehe Kapitel 12.4.4) gesteuert werden. Dieser Index bestimmt die Reihenfolge der entsprechenden Attribute bei der Anzeige in der Hybrid-Tabelle.

Zusätzlich zu ExternalHybridPropertyDefinition-Objekten kann eine ExternalTypeTagDefinition auch herkömmliche HybridPropertyDefinitions besitzen, welche nichts mit dem Schema der externen Datenquelle zu tun haben. Der Grund dafür ist, dass das später in Kapitel 13 erwähnte *ExternalHybrid* nicht nur externe Daten darstellende HybridProperties zulässt, sondern auch zusätzliche, herkömmliche HybridProperties zur Anheftung zusätzlicher Information erlaubt, welche lokal persistiert werden. Zu diesen “lokalen” HybridProperties können eben die normalen HybridPropertyDefinition-Objekte definiert werden.

12.2. Laden externer Daten

Zur Synchronisation mit der externen Datenquelle existieren mehrere Strategien:

Zyklisch Beim Start der Webanwendung wird für jede registrierte externe Datenquelle ein TriciaTimerTask gestartet, welcher sich um eine regelmäßige Synchronisation mit der entsprechenden Datenquelle kümmert.

Per Benachrichtigung Manche externe Datenquellen verfügen über einen Notification-Service, welcher eine explizite Synchronisation anfordern kann.

Manuell Das User-Interface bietet entsprechende Aktionen an, mit welchen die gesamte Datenquelle bzw. einzelne Datensätze synchronisiert werden können.

Unabhängig von der Strategie wird zum Laden der externen Daten und des externen Schemas die statische Methode *loadExternalData* der *ExternalDataSource*-Klasse mit dem die Datenquelle repräsentierenden ExternalContainer als Parameter aufgerufen.

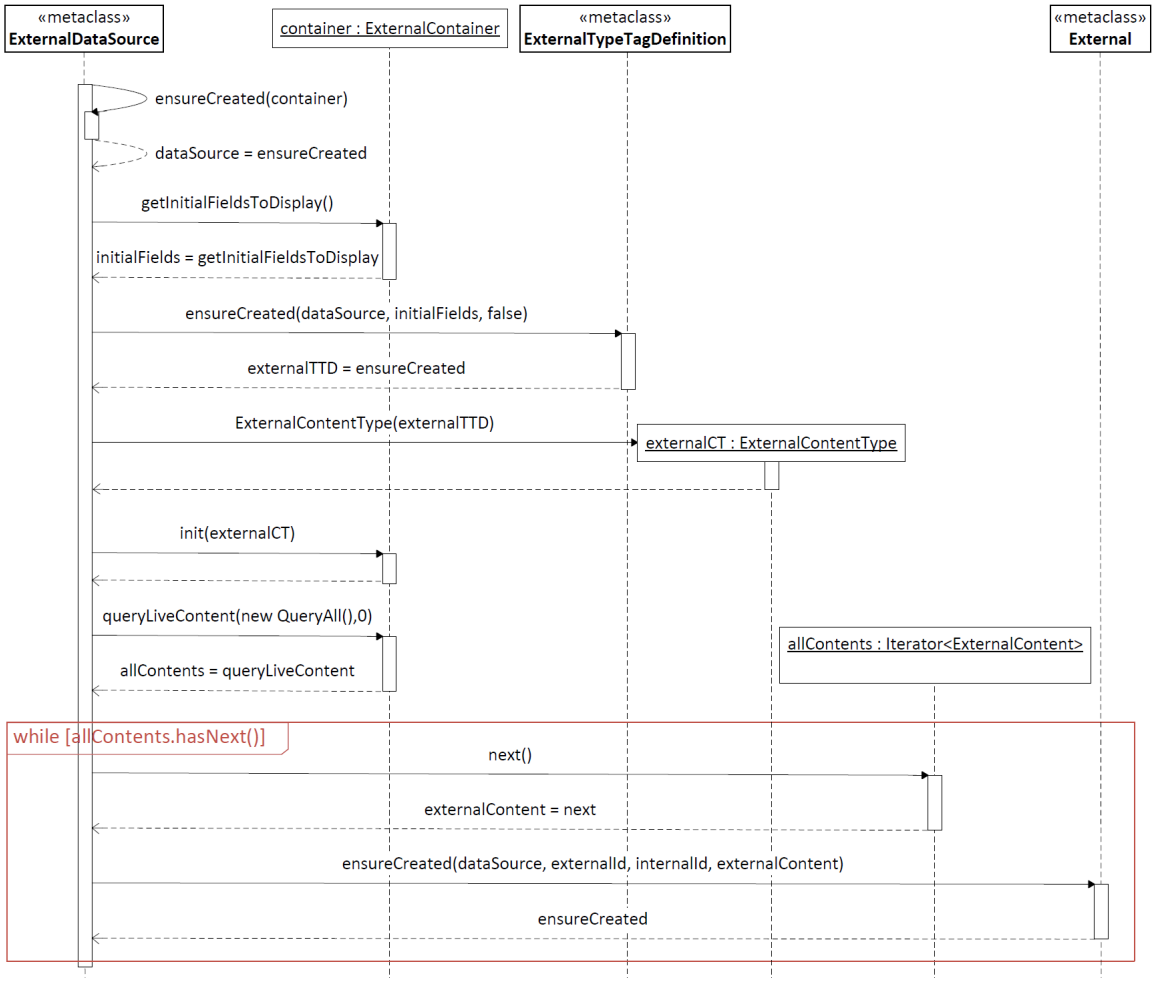


Abbildung 12.3.: Ein UML-Sequenzdiagramm als Visualisierung der loadExternalData-Methode der ExternalDataSource-Klasse

Folgende Angaben beziehen sich auf die in Abbildung 12.3, welche die Methode *loadExternalData* in Form eines UML-Sequenzdiagramms zeigt. Diese Methode lädt alle Datensätze der externen Datenquelle und aktualisiert bereits bestehende Tricia-Entitäten.

Zuerst wird das den übergebenen ExternalContainer repräsentierende ExternalDataSource-Objekt ermittelt, welches unter Umständen erst erzeugt werden muss. Sollte die Datenquelle noch nicht aktiviert worden sein, bzw. seit dem letzten Aufruf deaktiviert worden sein, so werden alle aus dieser Datenquelle geladenen Daten und Schemainformationen

auf Seiten von Tricia entfernt.

Ist die Datenquelle aktiviert, wird die Existenz der zugehörigen ExternalTypeTagDefinition sichergestellt. In der dafür zuständigen Methode *ensureCreated* der ExternalTypeTagDefinition wird das Standardschema der Datenquelle geladen, sofern noch keine ExternalTypeTagDefinition existiert (siehe auch Kapitel 12.3). Das Standardschema wird mithilfe der Methode *getInitialFieldsToDisplay* des ExternalContainers bestimmt.

Aus dem ExternalTypeTagDefinition-Objekt kann ein ExternalContentType erzeugt werden, mit welchem der ExternalContainer initialisiert wird. Dies ist für die folgende Abfrage der Datensätze der externen Datenquelle notwendig, da über diesen ContentType die zu ladenden Attribute festgelegt werden.

Für jedes resultierende Element der Abfrage wird eine entsprechende External-Entität erzeugt bzw. aktualisiert (siehe Kapitel 13.2). Alle existierenden External-Entitäten, welche Objekte repräsentieren, die seit dem letzten Abgleich in der externen Datenquelle gelöscht wurden, werden auf Seiten von Tricia ebenfalls nachträglich entfernt.

Die in Abbildung 12.3 vorkommenden Aufrufe der Methoden *ExternalTypeTagDefinition.ensureCreated* und *External.ensureCreated* werden im weiteren Verlauf ebenfalls als Sequenzdiagramm dargestellt (Abbildungen 12.4 und 13.2).

12.3. Abgleich des Schemas

Bei der Aktivierung einer Datenquelle wird dessen Standardschema geladen, welches durch die Methode *getInitialFieldsToDisplay* des ExternalContainers bestimmt wird. Diese Methode liefert die Namen von Attributen der externen Datenquelle, welche standardmäßig ausgewählt sein sollen. Zusätzlich werden jene Attribute geladen, welche als Parameter im Template für die Vorlage des Elementnamens enthalten sind. Lautet das Template beispielsweise “{Lastname}, {Firstname}”, so ergeben sich die zu ladenden Attribute aus dem Ergebnis der getInitialFieldsToDisplay-Methode und den Attributen “Lastname” und “Firstname”.

Das Schema aktivierter Datenquellen kann über eine spezielle Ansicht (erläutert in Kapitel 12.4.4) konfiguriert werden, d.h. die in der Hybrid-Tabelle der einzelnen Datensätze angezeigte Menge von Attributen kann angepasst sowie deren Reihenfolge festgelegt werden.

Bei einer Aktualisierung des externen Schemas in Tricia (z.B. ein Attribut soll in die Anzeige aufgenommen werden) wird die Methode *ensureCreated* der ExternalTypeTagDefinition-Klasse aufgerufen, welche in Abbildung 12.4 in Form eines UML-Sequenzdiagramms abgebildet ist. Deren Parameter sind:

datasource Die Datenquelle, für welche das Schema erstellt, bzw. aktualisiert werden soll.

attributes Die Namen der Attribute, aus welchen die das Schema beschreibenden ExternalHybridPropertyDefinition-Objekte erzeugt werden sollen.

overwriteExistingAttributes Angabe, ob die Attribute, die eine bestehende ExternalTypeTagDefinition beschreibt, durch die übergebenen Attribute ersetzt (überschrieben) werden sollen. Sollte noch keine ExternalTypeTagDefinition existieren, wird das Schema unabhängig von dieser Angabe aus den übergebenen Attributnamen erstellt

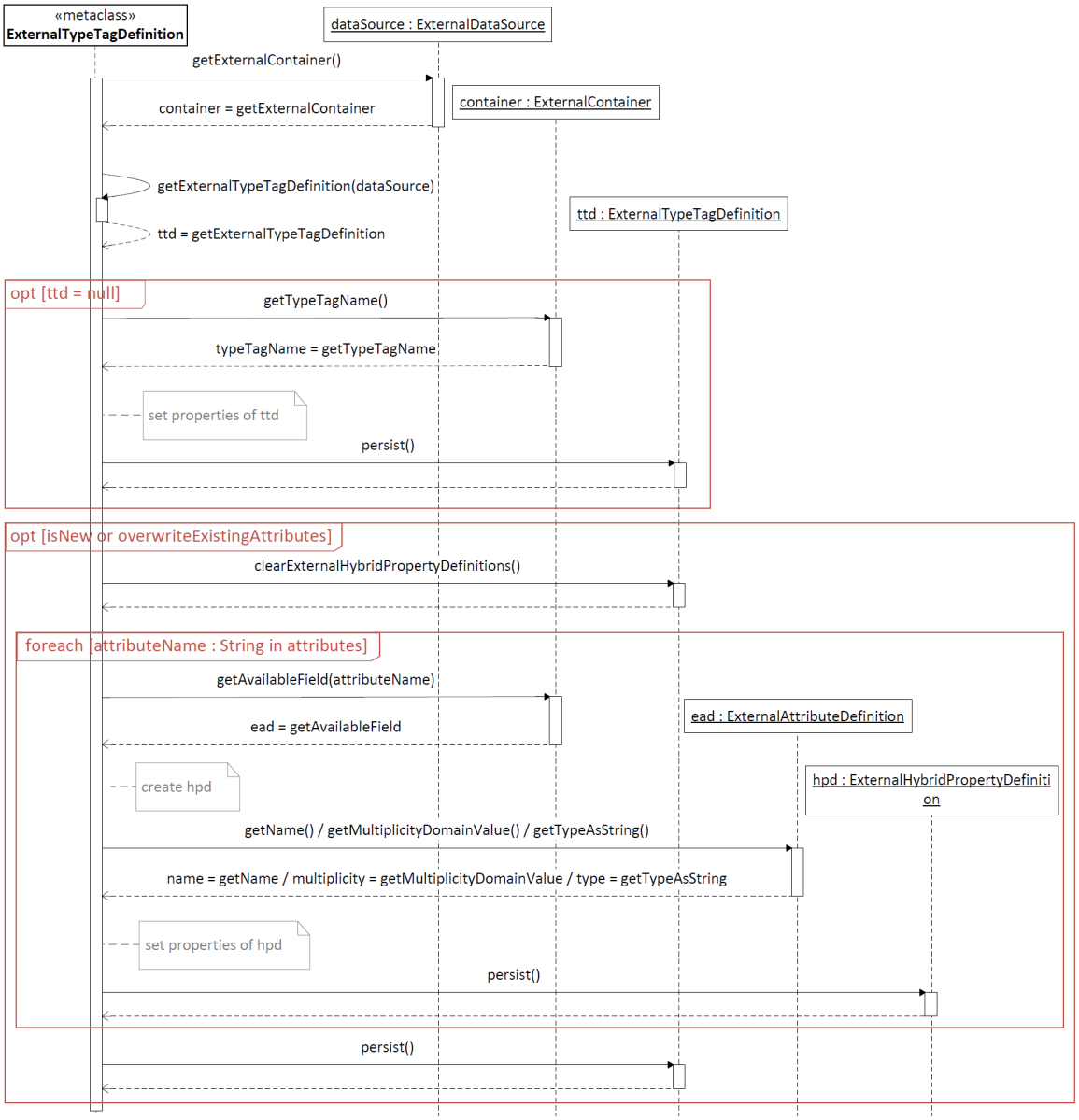


Abbildung 12.4.: Ein Sequenzdiagramm als Visualisierung der ensureCreated-Methode der ExternalTypeTagDefinition-Klasse

Ebenso findet in der zuvor erwähnten loadExternalData-Methode ein Aufruf von ensu-

reCreated statt.

Sollte die übergebene Datenquelle gar nicht aktiviert sein, so wird die Methode vorzeitig beendet. Im Falle einer aktivierten Datenquelle wird erst versucht, eine bereits existierende ExternalTypeTagDefinition abzufragen. Sollte diese noch nicht exisiteren, so wird aus der Datenquelle das entsprechende ExternalContainer-Objekt ermittelt und mit dessen Informationen eine neue ExternalTypeTagDefinition erzeugt.

Sollte die ExternalTypeTagDefinition als neu erzeugt worden sein oder sollen die existierenden ExternalHybridPropertyDefinition-Objekte durch neue ersetzt werden, so werden alle bereits bestehenden ExternalHybridPropertyDefinition-Entitäten gelöscht und anhand der übergebenen Attributnamen neu erzeugt.

Für jeden Attributnamen in der übergebenen Liste wird nun mithilfe des zuvor ermittelten ExternalContainer-Objekts die dazugehörige ExternalTypeDefinition geladen, aus welcher Informationen wie Typ und Multiplizität des Attributs ausgelesen und daraus ExternalHybridPropertyDefinition-Entitäten erzeugt werden können. Zusätzlich werden diesen ExternalHybridPropertyDefinition-Objekten sortOrder-Indizes verpasst, so dass diese später in der Hybrid-Tabelle in derselben Reihenfolge angezeigt werden wie sie hier als Parameter übergeben werden.

12.4. Views der externen Datenquelle

Zur Konfiguration einiger Parameter der externen Datenquelle bzw. dessen Schemas werden einige Views, d.h. Webseiten in Tricia angeboten. Damit lassen sich beispielsweise die Menge der Attribute konfigurieren oder die Datenquelle aktivieren und deaktivieren.

12.4.1. Übersicht über alle verfügbaren externen Datenquellen

Alle Datenquellen, die via ExternalContainer und einer ExternalContainersExtension in einem Tricia-Plugin registriert wurden, können unter der URL

`/externalDataSource/list`

übersichtsweise angezeigt werden, unabhängig davon, ob diese aktiviert oder deaktiviert sind.

In der tabellarisch aufgebauten Übersicht werden zu jeder Datenquelle folgende Parameter angezeigt:

Enabled Gibt an, ob die Datenquelle bereits aktiviert ist

Related Type Tag Zeigt einen Link zur das entsprechende Schema repräsentierenden TypTagDefinition an.

# of Entries Zeigt die aktuelle Anzahl jener Tricia-Entitäten an, welche den Inhalt der Datenquelle darstellen.

Abbildung 12.5 zeigt eine beispielhafte Übersichtsseite mit insgesamt fünf registrierten Datenquellen, von denen drei aktiviert sind.

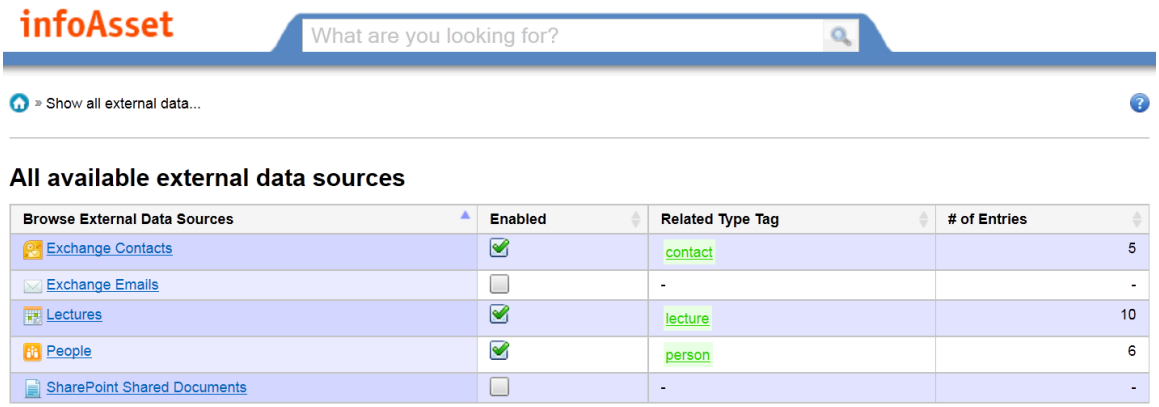


Abbildung 12.5.: Eine beispielhafte Übersichtsseite mit fünf registrierten Datenquellen

12.4.2. Ansicht einer externen Datenquelle

Jede Datenquelle besitzt eine eigene Ansicht, auf welcher dessen Eigenschaften angezeigt und spezielle Parameter konfiguriert werden können. Die URL dazu ist von folgendem Format:

`/externalDataSource/view/{externalDataSourceName}`

Hier werden neben den Paramtern, welche auch auf der Übersichtsseite angezeigt werden, zusätzlich die aktuell ausgewählten externen Attribute abgebildet.

Es ist zudem möglich, jeder externen Datenquelle über das RichText-Feld *Description* einen beliebigen RichText-Inhalt anzuheften um damit etwa die externe Datenquelle zu beschreiben.

Ist der Besucher der Website als Tricia-User angemeldet, so sind außerdem folgende Aktionen sichtbar:

Enable/Disable Hier kann die Datenquelle aktiviert und deaktiviert werden (siehe Kapitel 12.4.3).

Show all external data sources Hier gelangt man zur zuvor erwähnten Seite, welche einen Überblick über alle registrierten Datenquellen anbietet (siehe Kapitel 12.4.1).

Ist die Datenquelle zusätzlich auch noch aktiviert, erweitert sich die Auswahl der Aktionen um folgende beiden Punkte:

Configure attributes Hier kann die Menge der Attribute, welche in der Hybrid-Tabelle der einzelnen Datensätzen angezeigt werden sollen, eingestellt werden (siehe Kapitel 12.4.4).

Reload all entries Mit dieser Aktion kann die Methode loadExternalData der ExternalDataSource-Klasse (siehe Kapitel 12.2) explizit aufgerufen werden.

In Abbildung 12.6 ist die Ansicht einer beispielhaften Datenquelle mit 6 Einträgen zu sehen.

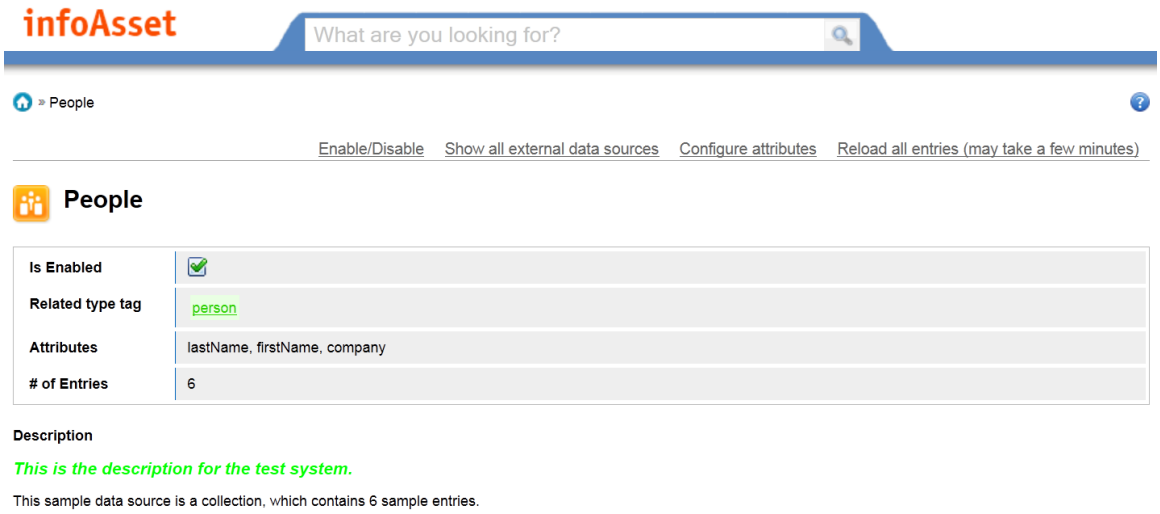


Abbildung 12.6.: Eine beispielhafte Ansicht einer Datenquelle mit sechs Einträgen

12.4.3. Aktivierung/Deaktivierung einer Datenquelle

Sollen die Daten einer Datenquelle geladen und damit nutzbar gemacht werden, muss diese erst aktiviert werden.

Wird die aktuelle Auswahl mit einem Klick auf “Save” bestätigt, so wird die Datenquelle entsprechend aktiviert oder deaktiviert. Beim Aktivieren wird die Methode loadExternalData (siehe Kapitel 12.2) ausgeführt, welche für eine Synchronisation mit der externen Datenquelle sorgt. Beim Deaktivieren werden hingegen alle Datensätze, welche Inhalte der deaktivierten Datenquelle repräsentieren, unwiderruflich gelöscht, sodass zusätzliche Informationen, welche an die Datensätze angeheftet worden sind, verloren gehen. Ebenso wird die entsprechende ExternalTypeTagDefinition gelöscht.

Wird die Property *isEnabled* gesetzt, so sorgt dies für die Ausführung des in Listing 12.1 abgebildeten ChangeListeners. Dieser sorgt eben dafür, dass bei einer Änderung die loadExternalData-Methode ausgeführt wird, welche entweder eine Synchronisation mit der externen Datenquelle durchführt oder alle entsprechenden Daten auf Seiten von Tricia entfernt. Außerdem wird abhängig davon, ob die Datenquelle aktiviert oder deaktiviert wird, die ExternalContainer-Methode *onEnabled* oder eben *onDisabled* aufgerufen, so dass an dieser Stelle beispielsweise Notification-Services registriert oder deregistriert werden können.

```
final ChangeListener reloadExternalDataChangeListener =
        new DeferredChangeListener() {

        @Override
        public void change(List<Change> changes) {
        if (changes.size() > 0) {
            final ExternalContainer ec = getExternalContainer();
            if (ec != null) {
                loadExternalData(ec);

                if (isEnabled.get()) {
                    ec.onEnabled();
                } else {
                    ec.onDisabled();
                }
            }
        }
    }
};
```

Listing 12.1: Der ChangeListener für die isEnabled-Property der ExternalDataSource

12.4.4. Konfiguration des Schemas

Unter der URL

`/externalDataSource/edit/{externalDataSourceName}`

lässt sich die Menge und Reihenfolge der Attribute, welche in der Hybrid-Tabelle aller Datensätze der externen Datenquelle angezeigt werden sollen, konfigurieren (siehe Abbildung 12.7).

Einige der Attribute einer Datenquelle sind dabei mit "(must be selected)" markiert, da das Template für den Elementnamen eines Datensatzes dieses Attribut als Parameter enthält. Beispielsweise müsste ein Attribut mit dem Namen "Lastname" zwingend ausgewählt sein, sollte die Vorlage des Elementnames den Parameterstring "{Lastname}" enthalten.

Wird die Menge oder Reihenfolge der Attribute verändert, so wird die zuvor erläuterte Methode ensureCreated der Klasse ExternalTypeTagDefinition (siehe Kapitel 12.3) mit der ausgewählten Attributmenge aufgerufen. Diese sorgt dafür, dass die das Schema repräsentierende ExternalTypeTagDefinition und deren ExternalHybridPropertyDefinitions aktualisiert werden.

Die Datensätze selbst werden jedoch noch nicht aktualisiert. Erst wenn solch ein Datensatz ausgewählt wird und dieser durch einen Vergleich seines *LastTimeReloaded*-Zeitstempels mit dem *LastModification*-Zeitstempels der ExternalTypeTagDefinition bemerkt, dass das

Schema seit seinem letzten Reload aktualisert wurde, stößt dieser einen automatischen Reload seiner Daten an.

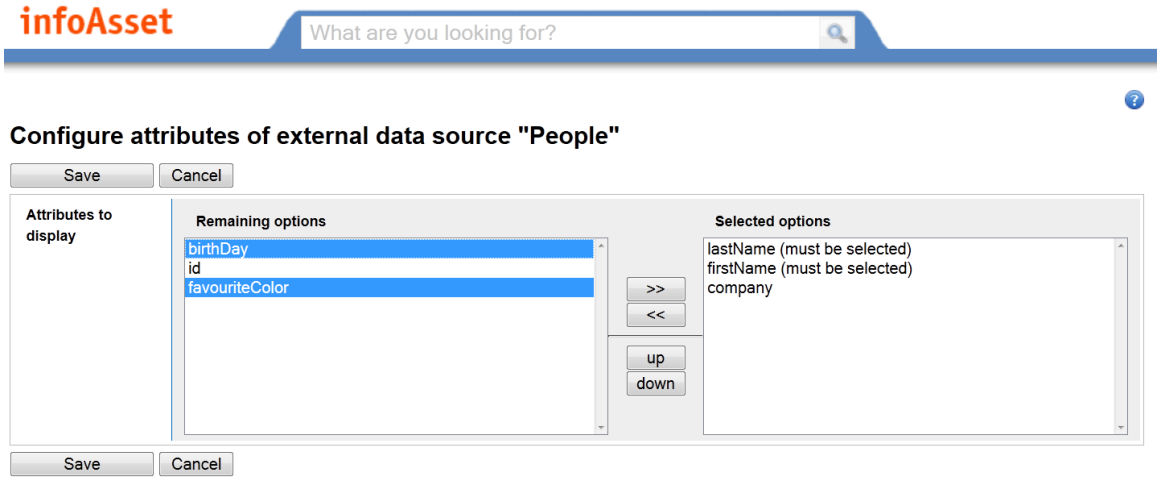


Abbildung 12.7.: Ansicht zur Konfiguration des Schemas einer beispielhaften Datenquelle

13. Inhalt externer Datenquellen

Das sog. *External* repräsentiert genau einen einzelnen Datensatz einer externen Datenquelle als Tricia-Entität, welche in folgendem Abschnitt Thema sein wird. Darstellungen der Datenquelle selbst und dessen Schemas wurden bereits in Kapitel 12 behandelt.

13.1. Objekte zur Repräsentation der Datensätze externer Datenquellen

Zur Darstellung externer Datensätze werden aufgrund der Flexibilität Hybrids verwendet (siehe Kapitel 11), so dass die Entität über beliebig viele Name-Wert-Paare verfügen kann. Unter diesen Name-Wert-Paaren werden sich im Fall der External-Entität nun auch "externe Name-Wert-Paare" befinden, d.h. Daten, die ursprünglich aus einer externen Datenquelle stammen. Abbildung 13.1 zeigt dazu einen detaillierten Ausschnitt aus dem External Modell von Abbildung 11.6.

13.1.1. External

Das External besitzt vier feste Properties bzw. Rollen:

externalDataSource Diese Rolle realisiert eine 1:N-Beziehung zur externen Datenquelle, aus welcher die durch das External dargestellten Datensätze stammen.

externalId Als Mapping einer External-Entität zu dessen Pendant in der externen Datenquelle besitzt das External die Property *externalId*. Diese externe ID stellt einen beliebigen eindeutigen Identifikator in der externen Datenquelle dar. D.h., zu einem Tupel aus externer Datenquelle und externer ID existiert genau 1 Datensatz, welcher durch das External dargestellt wird. Um ein External anhand dieses Tupels abzufragen, kann die statische Methode *getExternal* verwendet werden.

additionalContent Diese RichText-Property kann beliebigen zusätzlichen Inhalt enthalten, welcher an den externen Datensatz angeheftet wird. Der Inhalt dieser Property wird nicht in die externe Datenquelle übertragen.

lastReloadTimeStamp Dieser Zeitstempel enthält den Zeitpunkt der letzten Synchronisation des durch das External dargestellten Datensatzes. Mit der Methode *isReloadedSinceLastAttributeChange* lässt sich zudem abfragen, ob der Datensatz seit der letzten Schemaaktualisierung synchronisiert wurde.

Zusätzlich zu diesen festen Properties kann ein External über ExternalHybridProperties sowie herkömmliche HybridProperties verfügen.

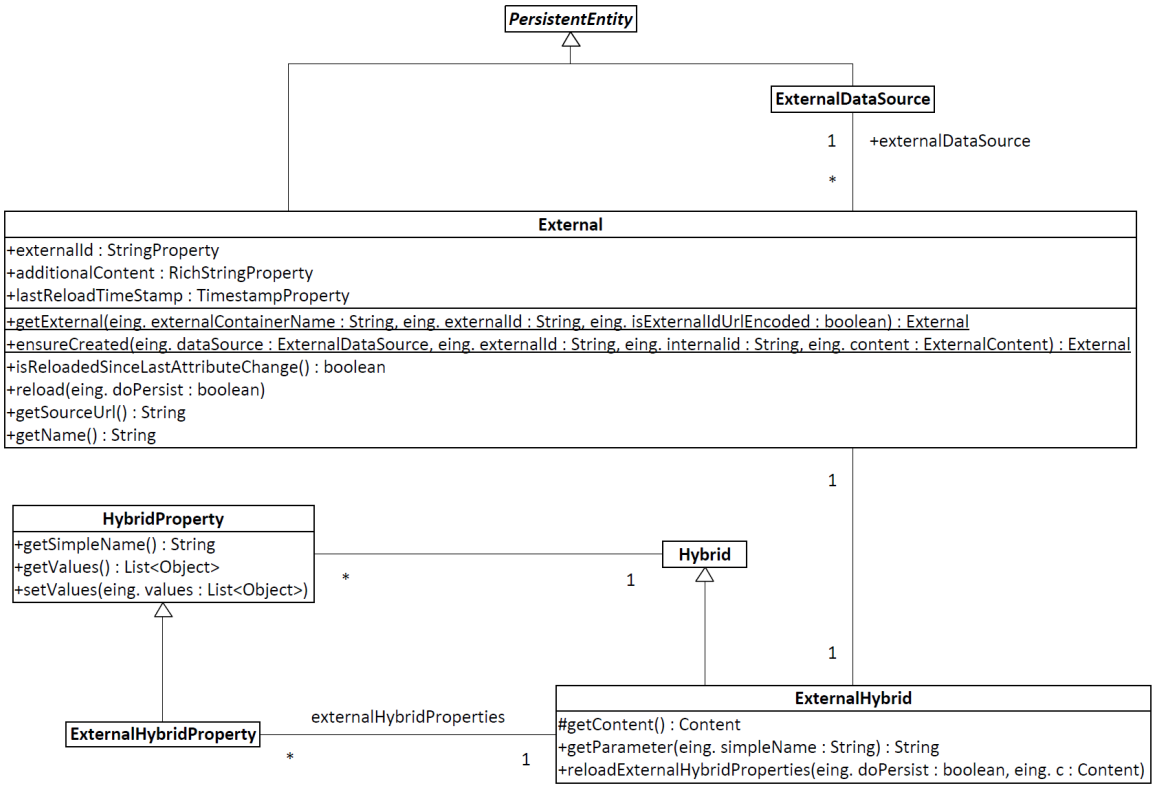


Abbildung 13.1.: Detaillierterer Ausschnitt aus dem External Modell, welcher sich mit der Darstellung externer Datensätze (External) befasst

Die Methode *getSourceUrl* liefert eine URL zum Datensatz der externen Datenquelle, welcher durch das External repräsentiert wird. Dazu wird das vom ExternalContainer angebotene Source Url Template genutzt und dessen Parameter mit konkreten Werten ersetzt. Analog dazu funktioniert die Methode *getName*, welche den Namen des Externals darstellt (z.B. in der Suchbox) und entsprechend das Item Name Template des External-Containers verwendet.

Das External wird zur hybriden Entität, weil es in der Mixin-Collection ein Objekt des Typs ExternalHybrid enthält (siehe Listing 13.1).

Nachdem die ExternalHybrid-Klasse abstrakte Methoden besitzt, wird das ExternalHybrid als Objekt einer anonymen Klasse realisiert, in welcher die Methoden *getExternalId*, *onReload* und *getExternalDataSource* implementiert werden (siehe Listing 13.1).

```
protected MandatoryMixin[] mandatoryMixins() {
    return new MandatoryMixin[] {
        new ExternalHybrid() {

            @Override
            protected String getExternalId() {
                return externalId.get();
            }

            @Override
            protected void onReload() {
                lastReloadTimeStamp.setToNow();
            }

            @Override
            protected ExternalDataSource getExternalDataSource() {
                return externalDataSource.get();
            };
        },
        ...
    };
}
```

Listing 13.1: Das ExternalHybrid als Element der Mixin-Collection der External-Entität, wodurch diese zur hybriden Entität wird

13.1.2. ExternalHybrid

Das ExternalHybrid selbst besitzt keine Properties, es bezieht seine Informationen vorwiegend aus seiner External-Entität. Es erweitert jedoch das herkömmliche Hybrid unter anderem um folgende Methoden:

getContent Diese Methode sorgt für die Abfrage der Daten jenes Datensatzes, welcher durch das External repräsentiert wird. Dabei wird der entsprechende ExternalContainer erst mit einem ExternalContentType initialisiert und anschließend der Datensatz mit der gegebenen externen ID abgefragt.

getParameter Der ExternalContainer bestimmt spezielle Vorlagen für den Elementnamen und der Source URL eines Datensatzes, wobei diese Templates Paramter enthalten können. Diese Parameter sind grundsätzlich Namen externer Attribute und sollen durch deren Werte ersetzt werden. Nachdem die Attribute, welche als Parameter in einem Template auftauchen, nicht zwingend durch das Schema erfasst werden müssen (zumindest jene des Source URL Templates nicht), müssen diese Parameter auch nicht als HybridProperties des ExternalHybrids vorliegen. Aus diesem Grund ist für solche Paramter ein besonderes Vorgehen notwendig, welches durch die Methode *getParameter* ermöglicht wird. Diese Methode überprüft, ob der Paramter als HybridProperty vorliegt und gibt diesen, sofern vorhanden, zurück. Sollte dies nicht

der Fall sein, so werden die Werte, welche für den Paramter notwendig ist, vom externen System abgefragt.

reloadExternalHybridProperties Diese Methode existiert in 2 verschiedenen Varianten, wobei in einer der zu ladende Inhalt übergeben wird und in der anderen dieser erst von der externen Datenquelle abgefragt werden muss. Kapitel 13.2 geht näher auf diese Methode ein.

getExternalHybridProperties Da das External neben ExternalHybridProperties auch herkömmliche besitzen kann, ist eine Unterscheidung dieser notwendig. Darum liefert diese Methode nur jene HybridProperties, welche auch Daten des externen Datensatzes repräsentieren. Dazu wird im Wesentlichen von der ExternalDataSource-Entität das aktuelle Schema abgerufen und für jedes Attribut eine ExternalHybridProperty erzeugt. Die Methode *getHybridProperties* liefert hingegen alle HybridProperties, d.h. es werden sowohl herkömmliche als auch ExternalHybridProperties zurückgegeben.

13.2. Laden externer Datensätze

Für das Laden einzelner Datensätze aus der externen Datenquelle ist die statische Funktion *ensureCreated* der External-Klasse zuständig. Diese Funktion wird sowohl bei einem kompletten Reload (siehe auch Kapitel 12.2) der gesamten Datenquelle als auch bei der Synchronisation eines einzelnen Datensatzes aufgerufen und sorgt sowohl dafür, dass eine External-Entität existiert, welche den übergebenen Datensatz darstellt und dass die HybridProperties des Externals aktualisiert werden. Durch das Persistieren des Externals in Tricia werden dessen Daten indiziert, wodurch der persistierte Datensatz in der Suche verfügbar wird. Abbildung 13.2 zeigt die Methode als UML-Sequenzdiagramm.

Abhängig davon, ob das External mit der übergebenen internen ID (also der in Tricia verwendeten ID) bereits existiert, wird die entsprechende External-Entität entweder abgefragt oder erstellt. Anschließend werden dessen Properties gesetzt, die Werte des Datensatzes in die HybridProperties geladen und der das Schema darstellende Type Tag dem Hybrid zugeordnet.

Für das Laden der Werte eines Datensatzes in die entsprechenden HybridProperties ist die Methode *reloadExternalHybridProperties* des ExternalHybrids zuständig. Dieser wird sowohl ein Flag übergeben, welches explizit angibt, ob das External persistiert werden soll, als auch der externe Datensatz in Form eines Content-Objekts, mit dessen Daten die HybridProperties befüllt werden sollen. Alternativ lässt sich diese Methode ohne Content-Objekt aufrufen (also mit nur einem Übergabeparameter), was zur Folge hat, dass das entsprechende Content-Objekt implizit von der Datenquelle abgefragt wird.

Da sich seit dem letzten Reload des Datensatzes die Menge der anzuzeigenden HybridProperties, also die das Schema darstellende ExternalTypeTagDefinition, geändert haben könnte, werden alle aktuell gespeicherten ExternalHybridProperties entfernt. Anschließend wird für jede im Schema definierte HybridProperties der entsprechende Wert aus

dem Content-Objekt gelesen und die HybridProperty mit diesem Wert belegt.

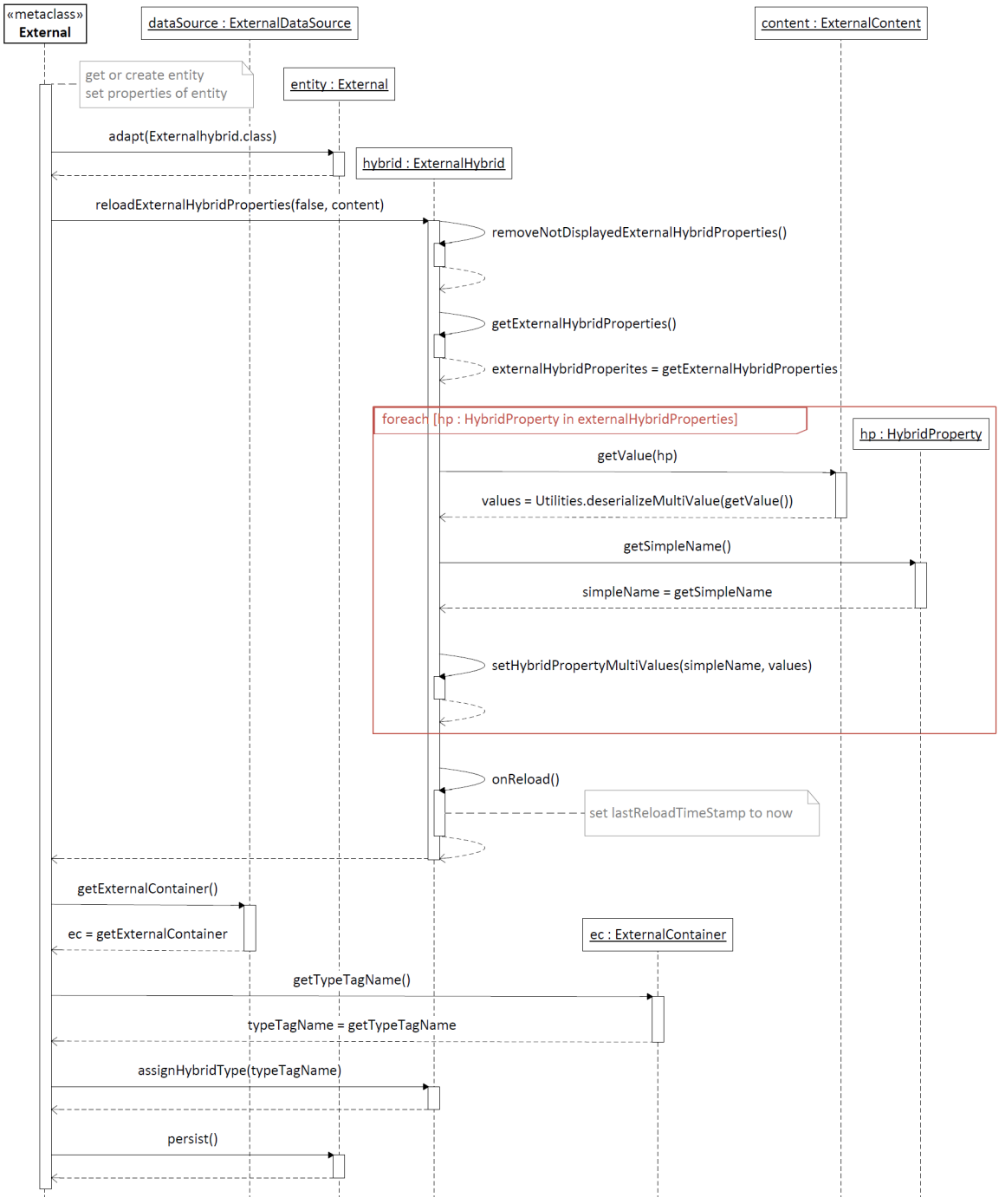


Abbildung 13.2.: Ein Sequenzdiagramm als Visualisierung der ensureCreated-Methode der External-Klasse

Der folgende Aufruf der *onReload*-Methode hat zur Folge, dass der *lastReloadTimeStamp*

des Externals auf den aktuellen Zeitpunkt gesetzt wird. Abhängig vom Wert des Übergabeparameters *doPersist* wird das External noch persistiert.

13.3. View eines externen Datensatzes

Zur Anzeige und Manipulation eines Datensatzes einer externen Datenquelle wird eine spezielle Webseite angeboten, abrufbar unter folgender URL:

`/external/view/{externalDataSourceName}?eid={urlEncodedExternalId}`

Abbildung 13.3 zeigt dazu die Ansicht eines beispielhaften Datensatzes.

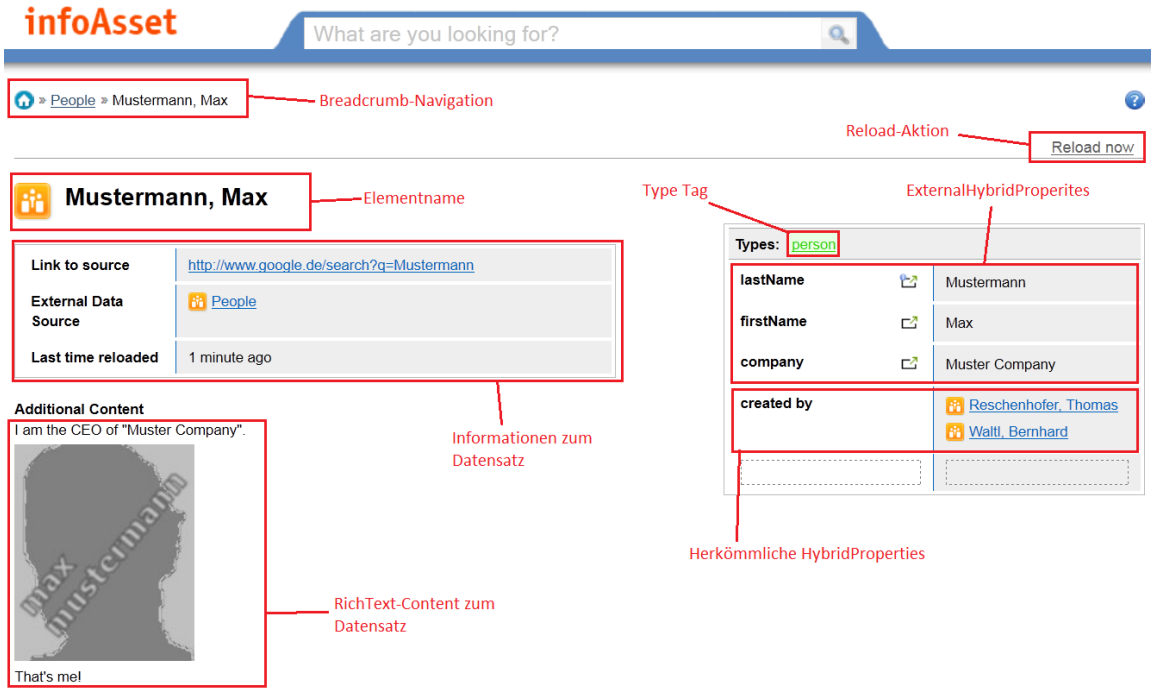


Abbildung 13.3.: Die Ansicht eines beispielhaften Datensatzes mit einer Beschreibung der einzelnen Komponenten

Im linken Bereich befindet sich eine Übersicht über bestimmte Informationen zum externen Datensatz:

Link to source Hier wird ein Link zur Repräsentation des Datensatzes im externen System angezeigt.

External Data Source An dieser Stelle ist ein Link zur externen Datenquelle zu sehen, zu welcher der aktuelle Datensatz gehört. Diese Information ist auch in der Breadcrumb-Navigation ablesbar.

Last time reloaded Hier ist der Zeitpunkt der letzten Synchronisation des Datensatzes zu sehen. Ein Datensatz wird immer genau dann geladen, wenn

- die gesamte Datenquelle neu geladen wird, z.B. bei dessen Aktivierung oder durch einen Timer-Task
- die Ansicht des Datensatzes aufgerufen wird und die das Schema repräsentierende ExternalTypeTagDefinition seit dem letzten Reload des Datensatzes verändert worden ist
- ein nicht-anonymer Benutzer einen expliziten Reload anfordert, indem er die Aktion "Reload now" auswählt.

Additional Content In diesem Bereich befindet sich ein an den externen Datensatz angehefteter RichText-Inhalt, welcher nur lokal persistiert und nicht auf die externe Datenquelle übertragen wird

Im rechten Bereich der Ansicht befindet sich die Hybrid-Tabelle, welche alle Hybrid-Properties der External-Entität anzeigt. Außerdem wird im oberen Bereich der Tabelle der Type Tag angezeigt, welcher das Schema der externen Datenquelle darstellt. Die External-HybridProperties sind mit einem entsprechenden Icon markiert, wobei jene Properties, in dessen Icons ein Info-Symbol zu sehen ist, zusätzliche Informationen wie Typ oder Multiplizität besitzen. Die in Abbildung 13.3 zu sehende HybridProperty "created by" ist dabei eine lokale HybridProperty, d.h., deren Wert wird nur lokal persistiert und nicht in die externe Datenquelle übertragen.

Teil V.

Fazit

14. Ergebnis

14.1. Abgedeckte Herausforderungen

Im Rahmen dieser Arbeit wurde ein "External" Plugin (siehe Kapitel 5.3) erstellt. Die an der Datenintegration gestellten Anforderungen (siehe Kapitel 3.2) wurden dabei großteils abgedeckt:

Effizientes Importmanagement Durch das Einführen des Importmanagers, der die Zugriffe auf das externe System verwaltet, ist es möglich, Cachingstrategien und intelligentes Laden umzusetzen.

Flexibilität Das Hinzufügen von neuen Datenquellen ist über die entworfene Architektur einfach möglich. Für OData - Producer existiert bereits eine implementierte Basisklasse. Um Daten aus neuen OData - Datenquellen zu importieren, reicht es aus, die URL zum REST - Webservice, den Namen der zu einzubindenden Entität und Benutzerinformationen anzugeben. Des Weiteren ist es auch einfach möglich, das Schema der externen Datenquelle zu aktualisieren. Die Darstellung eines externen Datensatzes in der Benutzeroberfläche kann ebenfalls gesteuert werden. Aus den verfügbaren Attributen können die angezeigten Attribute ausgewählt bzw. abgewählt werden.

Zusätzliche Attribute Da die importierten Daten intern als Hybrids repräsentiert werden, ist das Anheften zusätzlicher Attribute möglich. Es kann dabei strukturierte Information als Name-Wert Paar hinzugefügt werden. Aber auch das Zuweisen nichtstrukturierter Information in Form von Freitext ist möglich.

Typemapping Beim Aktivieren einer Datenquelle in Tricia wird die verfügbare Schemainformation ausgelesen. Im Rahmen dieses Prozesses wird auch die Datentypinformation der Attribute verarbeitet. Dabei werden die in Tricia verfügbaren Attributtypen berücksichtigt.

Authentifizierung Viele Datenquellen verlangen eine Authentifizierung, bevor sie Daten zur Verfügung stellen. Diese Information ist in dem ExternalContainer hinterlegt. Ob es sich dabei um einen Benutzernamen und ein Kennwort oder um eine gültiges Zertifikat handelt muss, wird in der Implementierung berücksichtigt.

14.2. Nichtabgedeckte Herausforderungen

Anforderungen an Datenintegration, die nicht abgedeckt wurden:

Benachrichtigung durch Fremdsysteme Die Benachrichtigung durch ein Fremdsystem hängt von dem externen System ab. Nicht alle Systeme bieten einen derartigen Dienst

an. Da sich die Information über den Zugriff auf eine Drittsystem in dem External-Container - Objekt befindet, ist es auch möglich, eine entsprechende Spezialisierung zu implementieren, welche einen Benachrichtigungsdienst verwendet. Im Rahmen dieser Arbeit wird diese Anforderung jedoch nicht umgesetzt.

Änderungshistorie Ändert sich ein bereits importierter Datensatz, wird dieser zwar aktualisiert, es wird jedoch nicht protokolliert, dass er geändert wurde. Die aufgetretenen Änderungen seit dem Import der Datensätze sind somit nicht nachvollziehbar. Die Änderungen an einem externen Schema werden ebenfalls nicht aufgezeichnet.

14.3. External Plugin

Die Architektur für das Importszenario wurden in einem eigenen Plugin *External* erstellt. Es umfasst viele verschiedene Klassen und diverse Templates für den Zugriff, die Verwaltung und die Anzeige der Datensätze und der externen Datenquellen. Eine neue Datenquelle wird über die sogenannte *ExternalContainerExtension* hinzugefügt. Es reicht dabei aus, den ExternalContainer einer Datenquelle als anonyme Klasse in der ExternalContainer Liste zu instanziieren. Das Plugin ist von dem *Hybrid* Plugin abhängig.

14.4. Tricia

In der Java-Applikation Tricia ist die Möglichkeit des Imports von Daten aus einem externen System gegeben. In einer Übersicht erhält man einen Überblick über alle verfügbaren Datenquellen (siehe Abbildung 14.1).

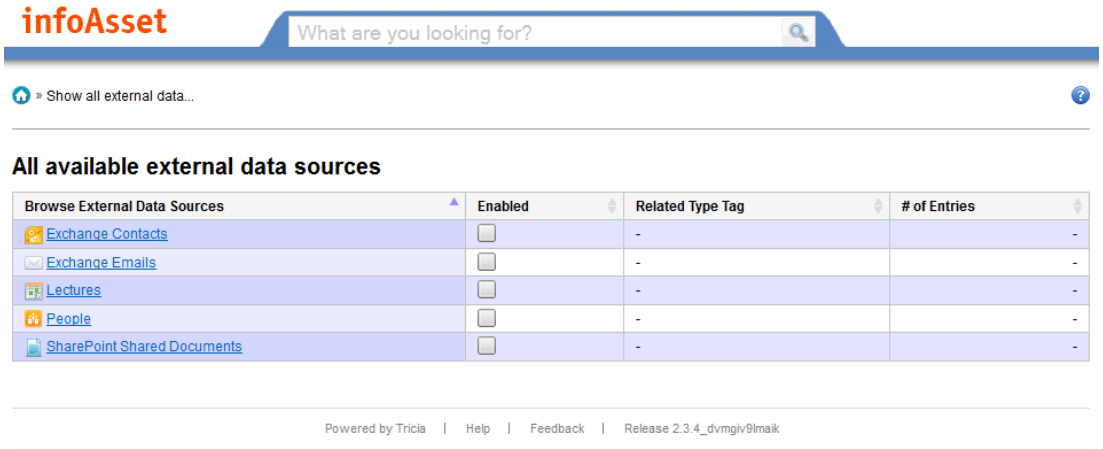


Abbildung 14.1.: Eine Liste aller verfügbaren externen Datenquellen

Man kann jede einzelne Datenquelle aktivieren. Man wählt aus der Liste die gewünschte Datenquelle aus und erhält eine detailliertere Ansicht über das gewählte Element.

Für die Datenquelle sind nach der Aktivierung mehrere Aktionen verfügbar:

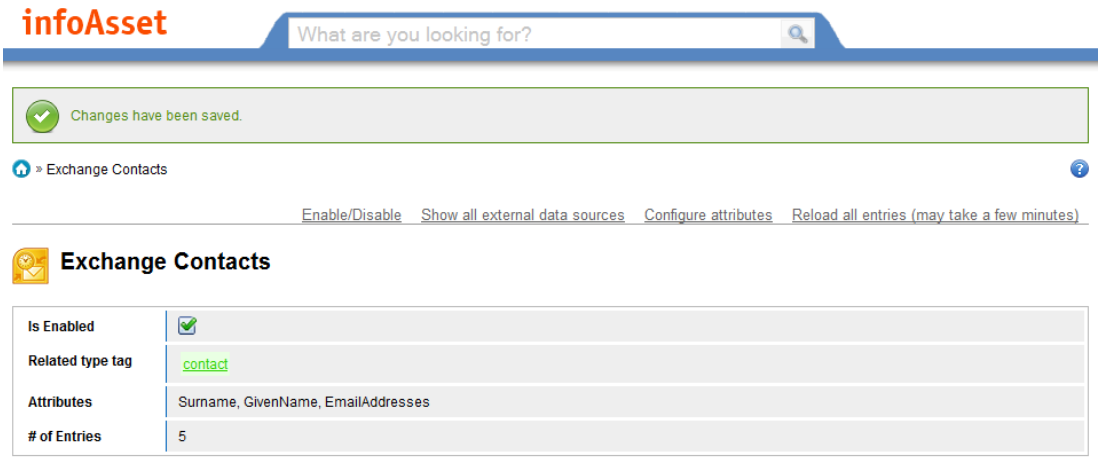


Abbildung 14.2.: Detaillierte Ansicht einer Datenquelle

Enable/Disable Diese Aktion ermöglicht das Aktivieren bzw. das Deaktivieren einer Datenquelle. Bei der Aktivierung werden alle verfügbaren Datensätze sowie das Schema der Datensätze importiert.

Show all external data sources Über diesen Button wird man an die Übersichtsseite über die verfügbaren Datenquellen weitergeleitet (siehe Abbildung 14.1).

Configure attributes Nachdem die Schemainformation geladen wurde und lokal vorliegt ist es möglich die Attribute zu spezifizieren, die angezeigt werden sollen (siehe Abbildung 14.3).

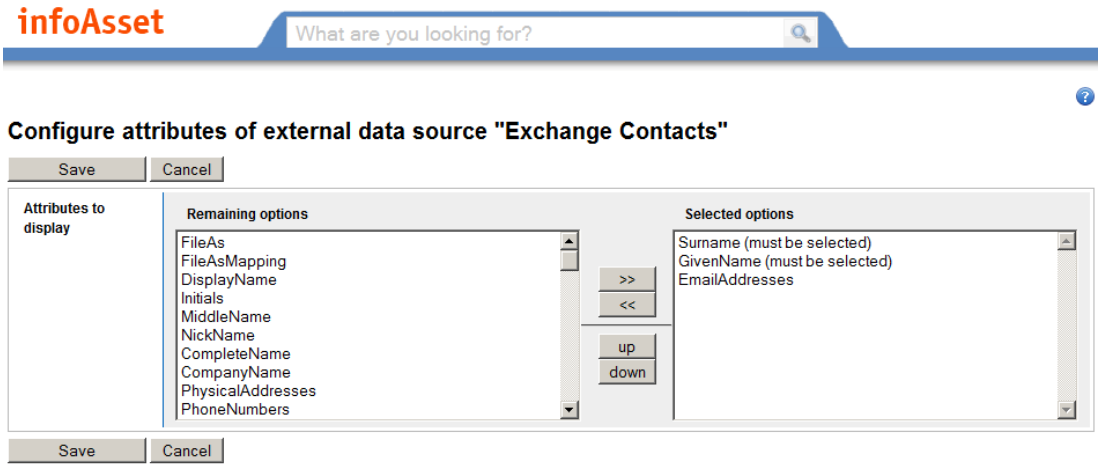


Abbildung 14.3.: Konfiguration der angezeigten Attribute für die Elemente der Datenquelle

Reload all entries Diese Aktion importiert alle Einträge aus der externen Datenquelle. Dieser Vorgang wird standardmäßig zyklisch ausgeführt, kann jedoch über diesen Link manuell angestoßen werden.

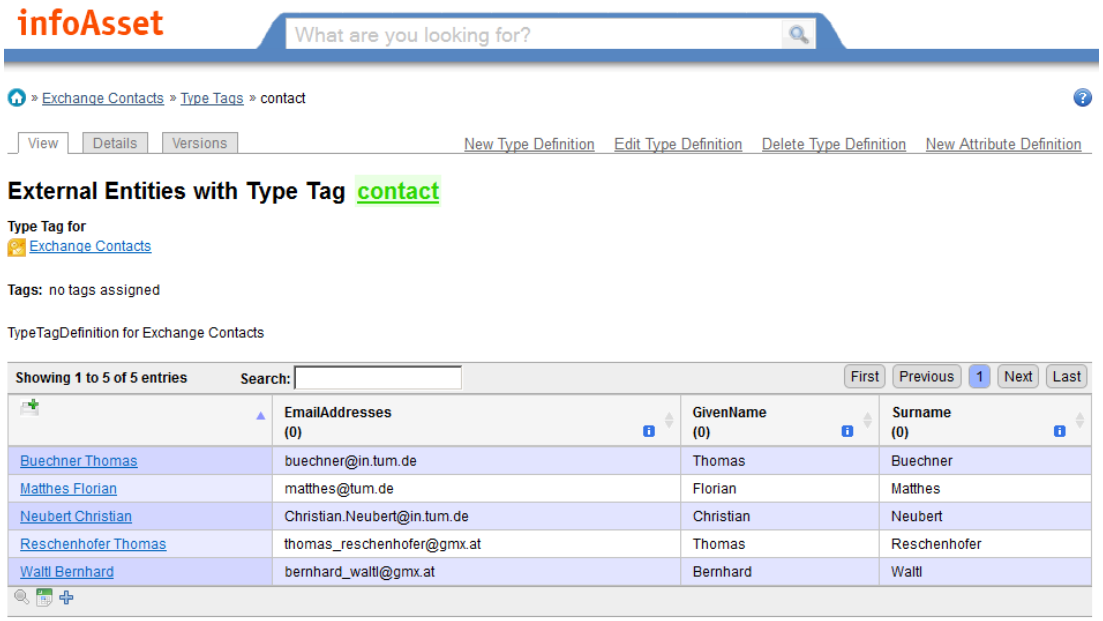


Abbildung 14.4.: Übersicht über die Objekte denen dasselbe TypeTag zugeordnet ist

Mit jeder Datenquelle wird auch ein Type Tag erstellt. In der TypeTagDefinition dieses Type Tags werden die Attribute gespeichert, die angezeigt werden und daher auch aus der externen Datenquelle geladen werden (siehe Kapitel 12). Wird eine TypeTagDefinition bzw. ein Type Tag ausgewählt, werden alle Datensätze und die spezifizierten Attribute in einer Liste dargestellt (siehe Abbildung 14.4).

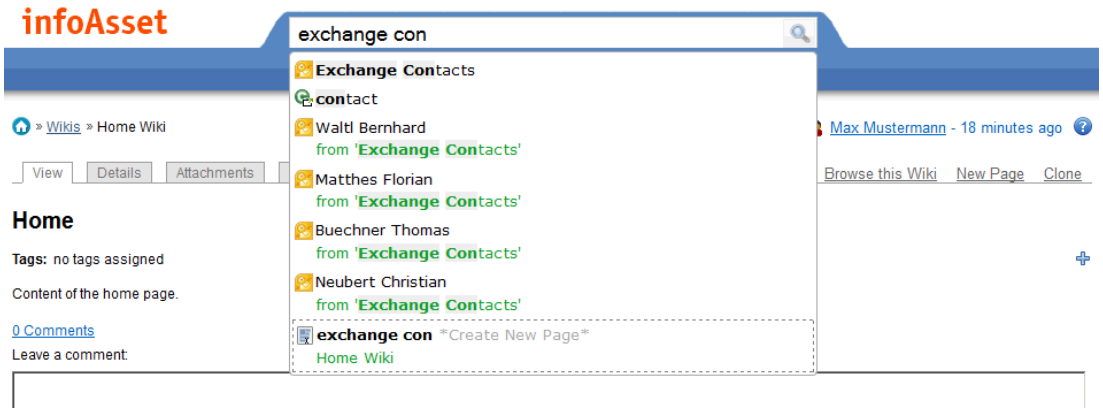


Abbildung 14.5.: Die Volltextsuche indiziert alle verfügbaren externen Datensätze

Ein Hybrid enthält alle Attributen und die zugehörigen Werte eines externen Datensatzes. Alle Attribute werden indiziert und sind somit in das Suchsystem von Tricia integriert. Es ist dabei möglich, auf Datensätze direkt über die Suchbox zuzugreifen (siehe Abbildung 14.5). Aber auch in der facettierten Suche, die auch eine Filterung ermöglicht, sind die Type Tags vorhanden und somit als Teil eines Filterkriteriums verfügbar (siehe Abbildung 14.6).

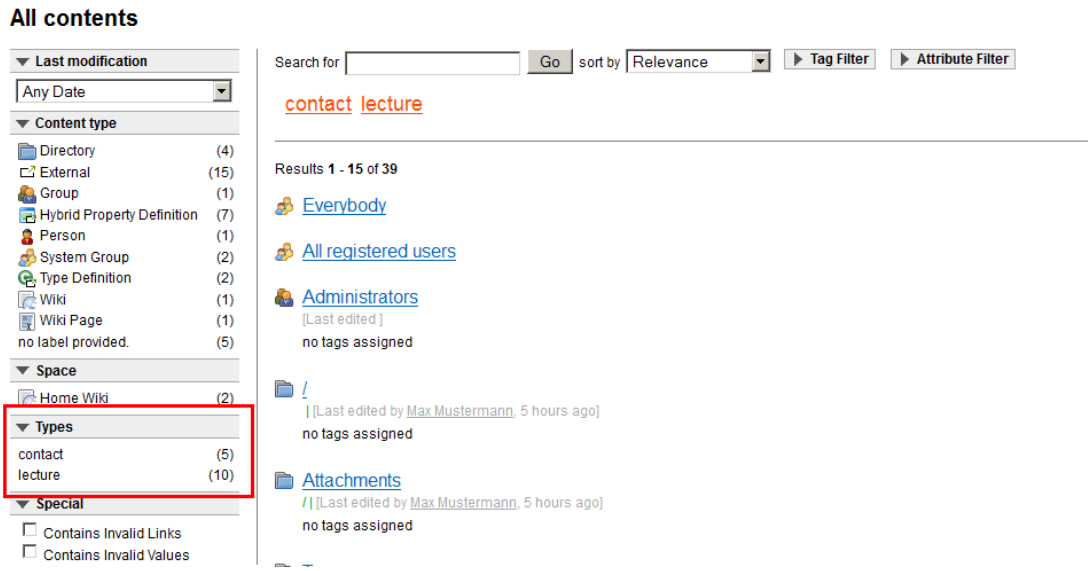


Abbildung 14.6.: Die Suchmaske enthält die verfügbaren TypeTagDefinitionen (rote Markierung).

Die Schemainformationen und der Inhalt konkreter Datensätze sind in Tricia verfügbar. Diese werden in Tricia als ExternalHybrids repräsentiert (siehe Kapitel 13) und deshalb auch in der Benutzeroberfläche als Hybrids dargestellt. Abbildung 14.7 zeigt die Darstellung eines importierten Datensatzes. Im rechten Bereich sind die Name-Wert Paare aufgelistet, dass diese aus einer externen Datenquelle stammen wird durch ein Symbol (kleines Kästchen mit Pfeil) gekennzeichnet. Neben den Attributen ist als zusätzliche Information noch die Zeit seit dem letzten Reload und der Link zu dem externen Datensatz verfügbar.

Diese Entitäten können auch miteinander in Beziehung gebracht werden. Das bedeutet, dass man in den Attributen eines Datensatzes andere, unter Umständen auch externe Daten verwenden kann. Es ist möglich, in der lokal gespeicherten Entität zusätzliche Informationen anzuheften. In Abbildung 14.8 ist die Entität aus 14.7 abgebildet und mit zusätzlichen Informationen erweitert. Neben einem Freitext in dem Bereich "Additional Content" sind auch zusätzliche Name-Wert Paare angefügt worden. Bei den neu hinzugekommenen Elementen handelt es sich um MS Exchange Kontaktdaten, die ebenfalls importiert wurden.

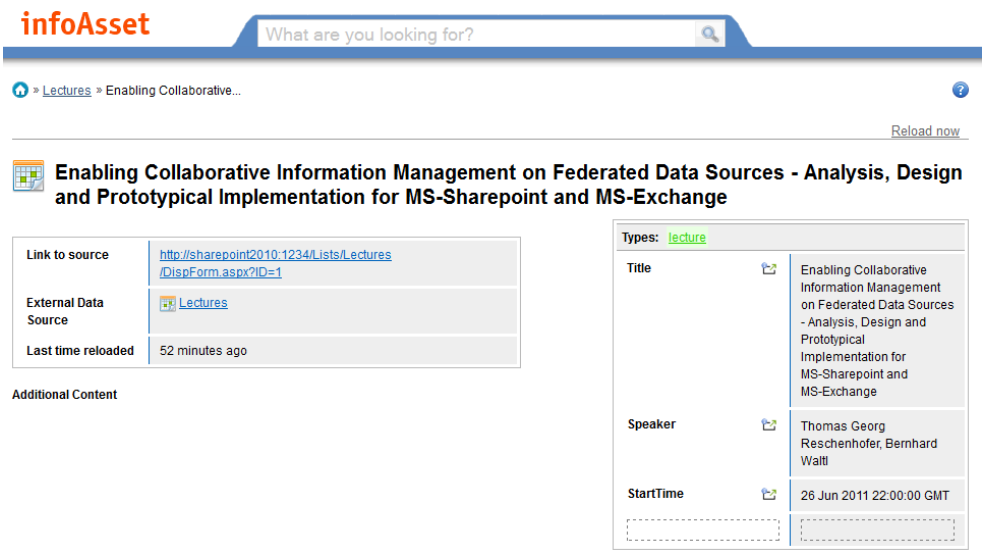


Abbildung 14.7.: Ein importierter Datensatz aus einer Sharepoint Liste

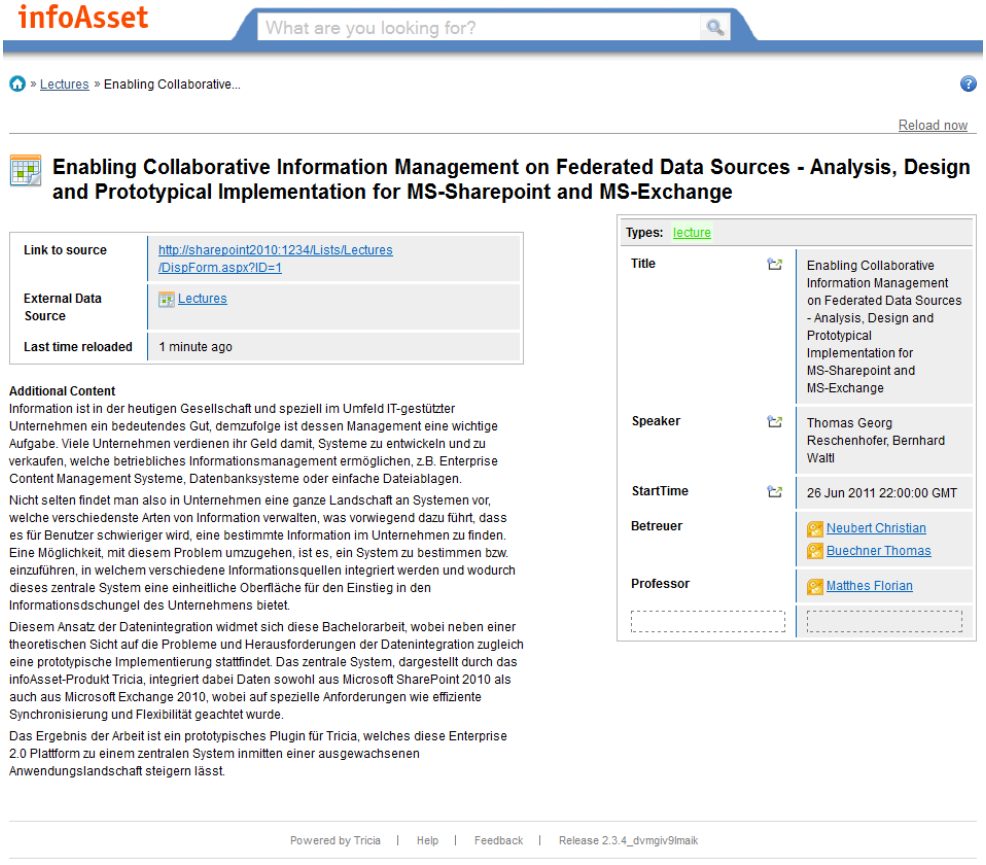


Abbildung 14.8.: Ein importierter Datensatz mit zusätzlichen Informationen

15. Ausblick

Der aktuelle Projektstand erlaubt es bereits, externe Datenquellen wie Microsoft SharePoint 2010 und Microsoft Exchange 2010 in Tricia zu integrieren und deren Daten suchbar und sogar manipulierbar zu machen.

Nichtsdestotrotz gibt es einige Aspekte und Ideen, welche im Rahmen der Arbeit zeitlich nicht umzusetzen waren. Einige der folgenden Punkte stammen unter anderem aus Quervergleichen mit anderen Systemen. Speziell SharePoint, da dieses System eben auch über eine Datenintegrationslösung verfügt, dient nicht selten als Vorlage bzw. Ideengeber.

Im folgenden Kapitel werden diese Ideen als Ausblick dafür genannt, welche Möglichkeiten sich mit dieser Arbeit als Grundlage bieten und wie diese womöglich umgesetzt werden können.

15.1. Single Sign-On

Für die Authentifizierung bei einer externen Datenquelle ist der ExternalContainer zuständig, da dieser die Schnittstelle von Tricia zur Datenquelle bildet. Die zu diesem Zeitpunkt implementierten ExternalContainer gehen dabei so vor, dass die Anmeldeinformation durch das Überschreiben bestimmter Methoden (*getUsername* und *getPassword*) festgelegt werden kann. Dabei authentifiziert sich der ExternalContainer beim externen System immer mit der selben Anmeldeinformation, völlig unabhängig davon, als welcher Benutzer eine Person bei Tricia angemeldet ist.

Das sog. “Single Sign-on” setzt sich zum Ziel, dass sich ein Benutzer an einem Arbeitsplatz nur einmal zu authentifizieren hat und anschließend entsprechenden Zugriff auf alle Systeme erhält.

Im Kontext dieser Arbeit meint Single Sign-on, dass sich eine Person bei Tricia als Tricia-Benutzer anmeldet und dieser entsprechend auf Identitäten externer Systeme “gemappt” wird. Damit wäre es beispielsweise möglich, einen Tricia-Benutzer auf sein eMail-Konto des Exchange Servers abzubilden, damit jeder Benutzer nur genau seine eMail-Nachrichten lesen kann.

Als Vorbild für solch eine Single Sign-on Funktion könnte SharePoint 2010 dienen: In SharePoint existiert ein sog. “Secure Store”, welcher das angesprochene Mapping der Identitäten beinhaltet. Dieser Secure Store liefert dabei für ein Tupel aus SharePoint Benutzer/-Gruppe und der Zielanwendung eine entsprechende Anmeldeinformation, mit welcher eine Authentifizierung am angegebenen externen System möglich ist. Jede im Secure Sto-

re abgelegte Information ist dabei speziell verschlüsselt, so dass ein SharePoint Benutzer nur die für ihn relevanten Anmeldeinformationen auslesen kann. Der vom Secure Store angebotene Dienst heißt auch "Secure Store Service" [16].

15.2. Berechtigungsverwaltung

Die Daten externer Datenquellen sind in der aktuellen Implementierung für jedermann lesbar. Um eine Datenquelle jedoch zu konfigurieren, d.h. diese zu aktivieren oder dessen Schema anzupassen, muss die Person sich als Tricia-Benutzer authentifizieren, wobei es belanglos ist, als welchen Benutzer man sich ausweist. Ebenso für einen expliziten Reload eines einzelnen Datensatzes ist es notwendig, ein nicht-anonymer Tricia-Benutzer zu sein.

Um bestimmen zu können, welcher Benutzer die Daten bestimmter externer Datenquellen lesen bzw. konfigurieren kann, könnte ein Berechtigungsmanagement eingeführt werden, wie es beispielsweise auch bei den in Tricia bestehenden Wikis der Fall ist.

Dabei kann für jedes Wiki ein Pool an Benutzern definiert werden, welche das Wiki selbst und deren Wikiseiten bearbeiten darf (*Writers*), und ein weiterer Pool an Benutzern, welche das Wiki und deren Wikiseiten nur lesen darf (*Readers*). Andere Benutzer haben gar keinen Zugriff auf die entsprechenden Ressourcen. Die Wikiseiten erben dabei die Berechtigungen der Wikis, können die vererbten Berechtigungen aber auch durch explizite, eigene Berechtigungen ersetzen.

Legt man dieses Konzept in den Kontext der Datenintegration um, so nimmt den Platz des Wikis die externe Datenquelle und den Platz der Wikiseite der externe Datensatz ein.

15.3. Erzeugen und Löschen externer Datensätze

Zur Anzeige externer Datensätze wird die Hybrid-Tabelle aus dem hybridWiki-Plugin verwendet. Diese erlaubt es, eine beliebige Menge an Name-Wert-Paaren (HybridProperties) anzuzeigen.

Die HybridProperties sind in der Hybrid-Tabelle "In-Place"-editierbar, d.h. deren Werte sind direkt in der Seite per Overlay bearbeitbar. Dies gilt auch für ExternalHybridProperties, wobei im Gegensatz zu herkömmlichen HybridProperties deren Name nicht verändert werden kann. Die Änderungen, die an den Werten von ExternalHybridProperties vorgenommen werden, werden auch direkt in die externe Datenquelle übertragen.

Von den CRUD-Methoden sind zum aktuellen Zeitpunkt also die *Read*- und die *Update*-Methode implementiert. Bei der Implementierung der restlichen beiden CRUD-Methoden *Create* und *Delete* müsste man das Userinterface und Schicht II der Architektur (beschrieben in Teil IV der Arbeit) entsprechend erweitern.

Die Architekturschicht I (Teil III der Arbeit), speziell der ExternalContainer, bietet bereits Methoden zum Erstellen und Löschen von Datensätzen an, so dass hier keine Erweiterung

notwendig wäre.

15.4. Erzeugen und Löschen externer Attribute

Der ExternalContainer erlaubt es, die verfügbare Menge an Attributen einer externen Datenquelle abzufragen. Eine entsprechende ExternalTypeTagDefinition ermöglicht es, aus der Menge der verfügbaren Attribute eine Untermenge der anzuzeigenden Attribute auszuwählen. Diese Attribute werden in der Hybrid-Tabelle jedes Datensatzes als HybridProperties angezeigt.

Das Userinterface der Hybrid-Tabelle erlaubt außerdem das Hinzufügen weiterer HybridProperties, welche jedoch als lokale Properties angelegt werden, d.h. diese Properties werden nicht in die externe Datenquelle geschrieben, sondern lediglich lokal persistiert.

Nun könnte man das Userinterface entsprechend erweitern, um herkömmliche HybridProperties in ExternalHybridProperties umzuwandeln, d.h. in der externen Datenquelle ein entsprechendes Attribut anlegen und die Ausprägung des Datensatzes in die externe Datenquelle übertragen. Das Löschen eines Attributs ist analog dazu denkbar.

Im Gegensatz zum Erstellen und Löschen externer Datensätze (Kapitel 15.3) ist hier eine aufwendigere Erweiterung in Schicht I der Architektur (siehe Teil III der Arbeit) notwendig, da der ExternalContainer und all dessen abgeleitete Klassen weder über eine Methode zum Hinzufügen eines Attributs noch zum Löschen eines solchen verfügen.

15.5. No-Code-Erweiterbarkeit

Um eine neue externe Datenquelle zu integrieren, ist es notwendig einen entsprechenden ExternalContainer zu implementieren. Dieser bietet Methoden an, um Daten abzufragen, zu erstellen, zu aktualisieren und zu löschen, sowie das aktuelle Schema der Datenquelle zu ermitteln.

Ein implementierter ExternalContainer muss anschließend in Tricia registriert werden, indem in einer beliebigen Plugin-Klasse ein ExternalContainersExtension-Objekt eingeführt wird und in dessen *getExternalContainers*-Methode den neuen ExternalContainer als Teil eines ExternalContainer-Arrays zurückliefert.

Sollte es sich bei der zu integrierenden Datenquelle beispielsweise um einen OData-Producer handeln, so kann der ODataContainer durch Ableitung erweitert werden, so dass nur mehr spezielle Parameter wie die URL des OData-Services oder der Name des entsprechenden EntitySets angegeben werden müssen. Da nun speziell bei der Implementierung dieser Container nur mehr bestimmte Parameter angegeben werden müssen, wäre hier denkbar, dass man all diese Parameter über das Userinterface von Tricia steuern lässt. So ließe sich beispielsweise die Service URL als StringProperty anlegen und jederzeit verändern.

Damit könnte man eine Möglichkeit schaffen, über das Userinterface neue ODataContainer zu erschaffen und zu registrieren, ohne dabei auch nur eine Zeile Code schreiben zu müssen. Selbiges wäre für den ExchangeContainer denkbar, bzw. für alle ExternalContainer-Klassen, welche eine passende Abstraktion schaffen, so dass nur mehr einfache Parameter zur konkreten Implementierung benötigt werden.

15.6. Benutzerfreundliche Attributnamen

Über den ExternalContainer lässt sich die Menge der von der externen Datenquelle angebotenen Attribute inklusive Name, Typinformation, Multiplizität und Beschreibung abrufen. Nicht selten entstammen die Namen der Attribute einem sehr technischen Kontext, d.h. sie sind nicht sehr benutzerfreundlich. Des weiteren sind die Namen der Attribute einer Datenquelle womöglich in einer anderen Sprache definiert als es erwünscht ist. Hat etwa ein Attribut den Namen “lastName”, so würde man in einer deutschen Umgebung stattdessen gerne “Nachname” anzeigen.

Andere Systeme wie z.B. SharePoint unterscheiden aus diesem Grund sehr oft zwischen dem Namen des Attributs und dessen Titel, welcher eine benutzerfreundliche Darstellung des Namens bildet.

Um in Tricia nun benutzerfreundlichere Attributnamen anzuzeigen, wären mehrere Strategien denkbar:

- Die naheliegendste, wenn auch unpraktikabelste Lösung ist es, die Attribute direkt in den externen Systemen umzubenennen um dadurch benutzerfreundliche Titel zu schaffen.

- Viele Datenquellen bieten für deren Attribute zusätzlich zu deren tatsächlichen Namen bereits benutzerfreundliche Titel an. So könnte man also die ExternalAttribute-Definition-Klasse um das Feld “Title” erweitern und den Attribut-Titel ebenfalls aus der externen Datenquelle übernehmen.

- Für jede einzelne externe Datenquelle könnte man ein Mapping zwischen dem tatsächlichen Namen eines Attributs und seinem benutzerfreundlichen Pendant definieren. Bei der Anzeige eines Attributs würde dieses in diesem Mapping seinen benutzerfreundlicheren Namen ermitteln und diesen stattdessen anzeigen.

15.7. Datenquellen-abhängige Aktionen

Auf externen Datensätzen kann bisher nur eine Aktion ausgeführt werden, nämlich ein expliziter Abgleich mit der externen Datenquelle.

SharePoint erlaubt es in diesem Zusammenhang, für jedes integrierte System benutzerdefinierte Aktionen zu definieren, welche im Wesentlichen aus einer parametrisierten

URL besteht. Parameter sind hierbei die Namen von Attributen, welche durch die Attributausprägungen des entsprechenden Datensatzes ersetzt werden.

Dieses Konzept könnte man genau so auch in Tricia einführen, indem man den ExternalContainer entsprechend um eine Methode erweitert, welche alle Aktionen für dessen Datensätze liefert. Parametrisierte URLs werden im ExternalContainer übrigens bereits bei der Angabe des Templates für die Source URL eines Datensatzes verwendet.

Mit diesen Datenquellen-abhängigen Aktionen wäre es beispielsweise möglich, für einen Datensatz einer Personendatenquelle (z.B. Exchange Kontakt) eine Aktion "Send Mail" zu definieren, welche den Benutzer sofort zu einer Eingabemaske für die Parameter einer eMail-Nachricht weiterleitet, wobei Ziel- und Quelladresse bereits ausgefüllt sind.

15.8. Datenquellen-abhängige Ansichten

Ein großer Vorteil der hybriden Entitäten als Repräsentation externer Inhalte ist deren Erweiterbarkeit: Soll eine neue Datenquelle integriert werden, muss nur ein entsprechender ExternalContainer implementiert und registriert werden. In Schicht II der Architektur und somit auch im Userinterface muss also keine Erweiterung stattfinden.

Dies hat natürlich zur Folge, dass die Ansichten von Datensätzen aus verschiedenen Datenquellen die selben sind, was womöglich nicht immer erwünscht ist. So sind beispielsweise einige Attribute (Speziell jene mit RichText- oder HTML-Inhalt) nicht dafür geeignet, in der Hybrid-Tabelle angezeigt zu werden.

Ähnlich wie bei den Datenquellen-abhängigen Aktionen (Kapitel 15.7) wäre es also denkbar, den ExternalContainer um eine Methode zu erweitern, welche die zu verwendende View liefert. In dieser View könnte man auf die speziellen Attribute der Datenquelle eingehen und etwaige RichText-Inhalte entsprechend anzeigen. Ohne spezielle Angabe einer View würde dann eben die Standardansicht verwendet werden, sodass eine Datenquellen-abhängige View nur optional wäre.

15.9. OData: Abfrage von Beziehungen

Der ODataContainer macht es relativ einfach, weitere OData-Producer in Tricia zu integrieren. Allerdings liest der ODataContainer, so wie er aktuell implementiert ist, nur die Properties von Entitäten aus, nicht aber deren Beziehungen.

Nachdem es Tricia jedoch ganz einfach ermöglicht, Entitäten zu verlinken, würde es sich anbieten, auch diese Beziehungen zwischen den OData-Entitäten über den ODataContainer zu konsumieren. Voraussetzung ist dabei natürlich, dass beide an der Beziehung beteiligten OData-EntitySets entsprechend in Tricia integriert werden. Wichtig ist dabei auch die Reihenfolge des Ladens der externen Daten, da Entitäten, auf welche verlinkt wird, bereits auf Seiten von Tricia erzeugt worden sein müssen.

Die Metadaten eines OData-Services beschreiben zwar dessen Beziehungen zwischen den EntitySets sehr genau (siehe auch Abbildung 9.1), in den einzelnen Ausprägungen der OData-Entitäten werden Links jedoch nur als URL dargestellt, welche zur OData-Repräsentation der verlinkten OData-Entität führt. Aus diesem Link muss also erst das entsprechende EntitySet ermittelt und überprüft werden, ob dieses überhaupt über einen ExternalContainer in Tricia integriert wird. Wenn ja, so kann das den verlinkten Datensatz repräsentierende External ermittelt und dieses als Wert für die entsprechende HybridProperty angegeben werden, so dass eine Tricia-konforme Beziehung realisiert wird.

15.10. Exchange: Weitere Element-Typen

Die Schnittstelle zu Exchange-Elementen besteht aus mehreren Abstraktionsebenen (siehe auch Abbildung 10.2):

- Abstrakter ExternalContainer als allgemeine Schnittstelle zu beliebigen Datenquellen
- Abstrakter ExchangeContainer als allgemeine Schnittstelle zu einem beliebigem Exchange-Element
- Abstrakter ExchangeItemContainer als Schnittstelle zu einem speziellen Exchange-Element (z.B. Kontakt, eMail-Nachricht, Event)
- Konkreter ExchangeItemContainer als Schnittstelle zu Exchange-Elementen eines konkreten Exchange Servers

Aktuell existieren die beiden abstrakten ExchangeItemContainer *ExchangeContactContainer* und *ExchangeEmailMessageContainer* als Schnittstelle zu Kontakten und eMail-Nachrichten eines Exchange Servers.

Um weitere Elementtypen eines Exchange Servers wie z.B. Events abzufragen, muss erst ein entsprechender, abstrakter ExchangeItemContainer (z.B. *ExchangeEventContainer*) implementiert werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Ulrike Baumöl. *Informationsmanagement, Aufgaben*, August 2011. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Informationsmanagement/Informationsmanagement--Aufgaben-des.

[2] Jakob Class. Integration eines single-sign-on-verfahrens in ein java-basiertes web-kollaborationswerkzeug. 2009.

[3] Billy Cripe, Kimberly Negaard, and Jason Lamon. *Fishbowl Solutions E2.0, ECM & Information Management 2011 Trends Report*, July 2011. http://www.fishbowlsolutions.com/fishbowl/groups/public/documents/white_papers/028685.pdf.

[4] Thomas Erickson. *From PIM to GIM: Personal Information Management in Group Contexts*, August 2011. http://www.visi.com/~snowfall/FromPIMtoGIM_CACM06.html.

[5] Anthony David Giordano. *Data integration: blueprint and modeling techniques for a scalable and sustainable architecture*. IBM Press, 2011.

[6] Christoph Koch. Data integration against multiple evolving autonomous schemata. 2001.

[7] Alber Kooinman. Juli 2011. http://www.msxfaq.de/code/ews.htm.

[8] Alber Kooinman. *Exchange Web Services (EWS)*, Juli 2011. http://www.msxfaq.de/code/ews.htm.

[9] Maurizio Lenzerini. Data integration: a theoretical perspective. pages 233–246, 2002.

[10] Wayne G. Lutters, Mark S. Ackerman, and Xiaomu Zhou. *Group Information Management*, July 2011. http://www.eecs.umich.edu/~ackerm/pub/06e08/PIM-lutters-ackerman-zhou.prepress.pdf.

[11] Florian Matthes, Christian Neubert, and Alexander Steinhoff. *Hybrid Wikis: Empowering Users to Collaboratively Structure Information*, July 2011. http://wwwmatthes.in.tum.de/file/attachments/wikis/sebis/ma11a-hybrid-wikis-empowering-users-to-co/Ma11a.pdf.

[12] Microsoft. *Exchange Server 2010 SP1 Web Services SDK*, July 2011. http://msdn.microsoft.com/en-us/library/dd877012%28v=EXCHG.140%29.aspx.

[13] Microsoft. *Exchange Web Services Architecture*, July 2011. http://msdn.microsoft.com/en-us/library/aa579369%28v=EXCHG.140%29.aspx.

[14] Microsoft. *Getting Started with the EWS Java API 1.1.2*, 1.1.2 edition, Mai 2011. http://archive.msdn.microsoft.com/ewsjavaapi.

[15] Microsoft. *Microsoft Outlook MAPI Reference*, July 2011. http://msdn.microsoft.com/en-us/library/cc765775.aspx.

[16] *Secure Store Service*, July 2011. http://msdn.microsoft.com/en-us/library/ee557754.aspx.

[17] Svetlozar Nestorov. *Semistructured Data: Definition*, August 2011. http://infolab.stanford.edu/~evtimov/Defense/sld036.htm.

[18] *Open Data Protocol*, July 2011. http://www.odata.org/home.

[19] *odata4j*, July 2011. http://code.google.com/p/odata4j/.

[20] P. Resnick. *Internet Message Format*, July 2011. http://www.faqs.org/rfcs/rfc2822.html.

[21] *SharePoint 2010 - Leistungsvermögen*, July 2011. http://sharepoint.microsoft.com/de-at/product/capabilities/Seiten/default.aspx.

[22] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22:183–236, September 1990.

[23] *structured data computer definition*, August 2011. http://computer.yourdictionary.com/structured-data.

[24] Christian Neubert Thomas Büchner, Florian Matthes. Data model driven implementation of web cooperation systems with tricia. 2010.

[25] *Hybrid*, July 2011. http://de.wikipedia.org/wiki/Hybrid#Software.

[26] Informationsmanagement. http://de.wikipedia.org/wiki/Informationsmanagement, July 2011.

[27] *Personal Information Management*, July 2011. http://en.wikipedia.org/wiki/Personal_information_management.

Appendix

Aufteilung der Arbeit

Kapitel		Autor
1	Ziele der Arbeit und Problemstellung	Bernhard Waltl
2	Kollaboratives Informationsmanagement	Thomas Reschenhofer
3	Datenintegration	Bernhard Waltl
4	Das Konzept der Hybrids	Thomas Reschenhofer
5	Tricia	Bernhard Waltl
6	Microsoft SharePoint 2010	Thomas Reschenhofer
7	Microsoft Exchange 2010	Bernhard Waltl
8	Generische Schnittstelle zu Datenquellen	Bernhard Waltl
9	Schnittstelle zu MS SharePoint: ODataContainer	Thomas Reschenhofer
10	Schnittstelle zu MS Exchange: ExchangeContainer	Bernhard Waltl
11	Evaluierung der Möglichkeiten zur Repräsentation externer Daten	Thomas Reschenhofer
12	Externe Datenquelle	Thomas Reschenhofer
13	Inhalt externer Datenquellen	Thomas Reschenhofer
14	Ergebnis	Bernhard Waltl
15	Ausblick	Thomas Reschenhofer