Entwicklung eines Prototyps zur Visualisierung eines unternehmensspezifischen SoNBO (Knowledge Graphen) am Beispiel von CRM-Informationen

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Wirtschaftsinformatik

Vorgelegt von

Simon Meier

Immatrikulationsnummer: 215202886
E-Mail: simeier@uni-koblenz.de

Fachbereich 4: Informatik

Institut für Wirtschafts- und Verwaltungsinformatik

Universität Koblenz-Landau

Betreuer:

Prof. Dr. Petra Schubert

Berit Gebel-Sauer

Koblenz, März 2019

Erklärung

Ich versichere,

dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. Der Veröffentlichung dieser Arbeit im Internet stimme ich nach Rücksprache mit den Betreuern zu.

Simon Meier

Koblenz, März 2019

Abstract

Social Network of Business Objects (SoNBO) ist ein Konzept, um im Unternehmen in heterogenen Systemlandschaften verteilte Informationen zu aggregieren und gesammelt auf einer Benutzeroberfläche zur Verfügung zu stellen. Die zentrale Idee ist dabei, die Unternehmensinformationen als Netzwerk (auch: Graph) zu verstehen. Es gibt bereits einen SoNBO-Explorer, der die Informationen eines Customer Relationship Management-Systems (CRM-System) integriert. Die Herausforderung bei der Konfiguration einer solchen Anwendung besteht darin, das Unternehmensnetzwerk zu identifizieren, und auf diese Weise heraus zu finden, wie die gespeicherten Daten im Unternehmen verknüpft sind. Dafür ist ein Tool hilfreich, das das Unternehmensnetzwerk visualisieren kann. In dieser Arbeit wird ein selbstentwickeltes Tool (SoNBO-Graph-App) als Prototyp vorgestellt, das diese Visualisierung ermöglicht. Mit dieser Anwendung kann die Konfiguration des Netzwerks im SoNBO-Explorer bestehend aus den zusammengeführten Daten unterstützt werden, indem diese Konfiguration auf graphischer Ebene durchgeführt wird. Der Prototyp ist an zwei verschiedenen Datenbanken eines Customer Relationship Management (CRM) Systems angebunden und ermöglicht die Aggregation dieser Daten, sodass sie in einer Übersicht zusammenhängend als Graph dargestellt werden. Dadurch erhält der Anwender einen besseren Überblick und ein Verständnis über den Zusammenhang der unterschiedlichen Daten. Diese Arbeit ist Teil des Langzeitforschungsprojekts SoNBO, dessen Ziel ein Konzept zur Integration von Informationen verschiedener geschäftlicher Anwendungssysteme ist.

Abstract (English)

Social Network of Business Objects (SoNBO) is a concept for aggregating information distributed in heterogeneous system landscapes and making it available via a single user interface. The central idea is to understand company information as a network (graph). There is already a SoNBO-Explorer which integrates the information of a customer relationship management system (CRM system). The challenge in configuring such an application is to identify the corporate network and thus find out how the stored data is linked within the company. A tool that can visualize the corporate network is helpful for this. In this thesis a self-developed tool (SoNBO-Graph-App) is presented as a prototype, which realizes this visualization. With this application the configuration of the network in the SoNBO Explorer consisting of the merged data can be supported by carrying out that configuration on a graphical level. The prototype is connected to two different databases of a Customer Relationship Management (CRM) system and allows the aggregation of these data so that it is displayed as a graph in an overview. This gives the user a better insight and understanding of the relationship between the different data. This work is part of the long-term research project SoNBO, whose goal is a concept for the integration of information from different business application systems.

Danksagung

Mein Dank richtet sich an die Personen, die mich bei der Entwicklung und beim Schreiben der Thesis unterstützt haben. Zuerst möchte ich mich bei Prof. Dr. Petra Schubert und Berit Gebel-Sauer bedanken, die diese Arbeit betreut und mich mit Begeisterung gefördert haben. Ein weiteres Dankeschön geht an die Korrekturleser dieser Arbeit, Klaus und Otmar. Ein spezieller Dank richtet sich an Julian und Flemming, die mich mit ihrem Wissen bei der Entwicklung des Prototyps sehr unterstützt haben. Außerdem danke ich allen nicht namentlich erwähnten Personen, die auf irgendeine Weise diese Arbeit positiv beeinflusst haben.

Inhaltsverzeichnis

Abkürzungen

API: Application programming interface

BO/SBO: (Social) Business Object

CRM: Customer Relationship Management

CSJS: client-side JavaScript

ERP: Enterprise Resource Planning

JSF: Java Server Faces

JSON: JavaScript Object Notation

KPI: Key Performance Indicator

SC: Social Concept

SoNBO: Social Network of Business Objects

SoNC: Social Network of Concepts

SoNSC: Social Network of Subconcepts

SSC: Social Subconcept

SSJS: server-side JavaScript

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Wissensmanagement ist eine Herausforderung, die in Unternehmen bewältigt werden muss (Probst, Raub, & Romhardt, 2010). Der Wettbewerbsfaktor Wissen kann kaum mehr vernachlässigt werden. Daher ist es von Bedeutung, dass der Mitarbeiter mit dieser Ressource arbeiten kann (Urbach & Ahlemann, 2016). Dazu kann es notwendig sein, Informationen aus mehreren verschiedenen Enterprise-Systemen zu benutzen. Dabei treten Herausforderungen auf, die bei der Entwicklung dieser Systeme hätten gelöst werden können (Markus & Tanis, 2000). Der Zugriff auf diese verteilten und möglicherweise redundanten Informationen kann sich als schwierig erweisen, wenn komplexe Datenbankabfragen notwendig sind, um benötigte Daten zu erhalten (Kharlamov u. a., 2013). Ein bedeutsamer Punkt, den Urbach & Ahlemann (2016) dabei aufführen, ist Benutzerfreundlichkeit. *Social Network of Business Objects* (SoNBO) ist ein Konzept zur Informationsintegration, das es Anwendern erlauben soll, in einer heterogenen Systemlandschaft, unkomplizierten Zugang zu benötigten Informationen zu erhalten (Gewehr, Gebel-Sauer, & Schubert, 2017). Die Inhalte aus möglicherweise verschiedenen Quellen (zum Beispiel CRM-Systemen, ERP-Systemen, …) werden als *Business Objects* betrachtet, die mit anderen Objekten in Beziehung gebracht werden. Dadurch entsteht ein Netzwerk aus Social Business Objects, wodurch neben den Inhalten der Objekte auch deren Beziehungen untereinander für den Anwender sichtbar gemacht werden (vgl. Abbildung 1).

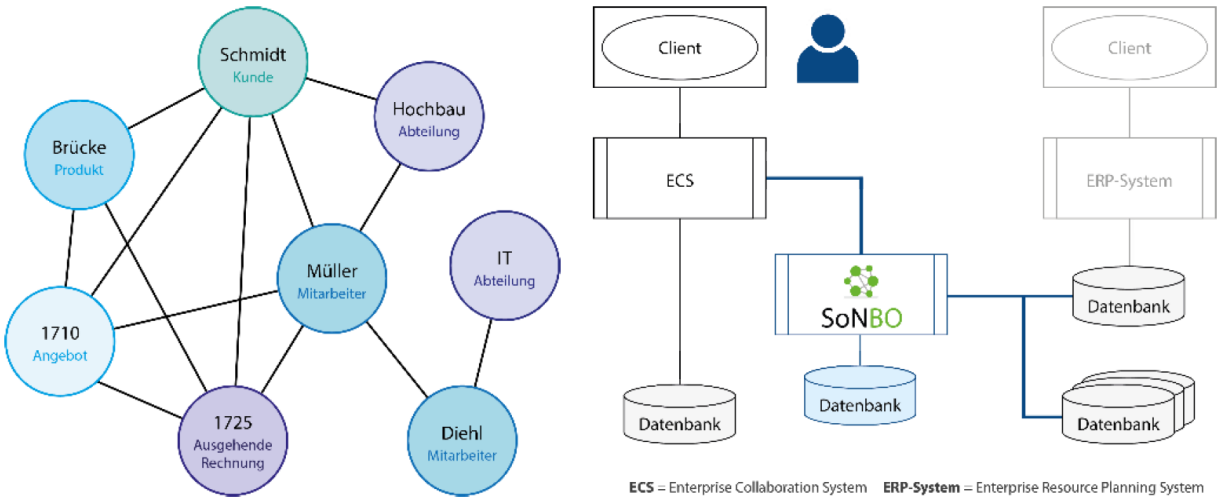


Abbildung 1: SoNBO-Ansatz (Gebel-Sauer & Schubert, 2019)

Diese Informationen können in einer Anwendung, die auf die verschiedenen Datenbanken zugreift, zusammengeführt werden. In einer solchen Anwendung wird ein Business Object aus dem Netzwerk angezeigt. Die aggregierten Informationen werden in dieses Object integriert und in der Anzeige dargestellt (vgl. Abbildung 2).

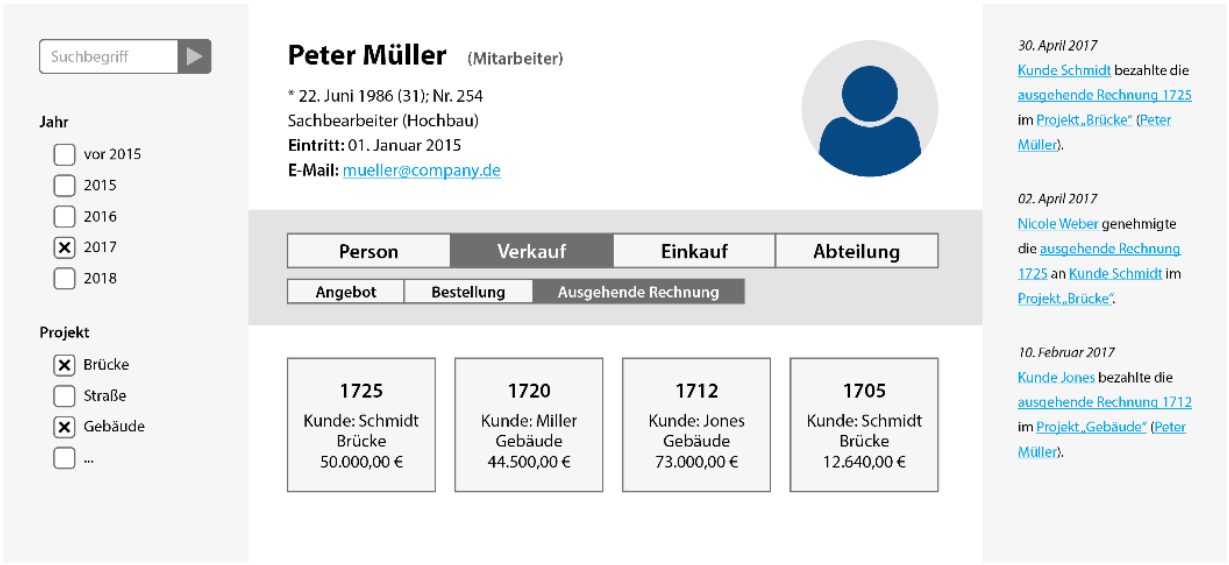


Abbildung 2: Mock-up einer SoNBO-Anwendung (Gebel-Sauer & Schubert, 2019)

Der SoNBO-Ansatz ist prototypisch am Beispiel eines CRM-Systems umgesetzt worden (Götz, 2018). Im dabei entstandenen „SoNBO-Explorer" kann der Anwender sich Informationen zu allen CRM-Objekten anzeigen lassen und durch diese Objekte navigieren. Dadurch kann der Nutzer an einer Stelle auf alle vorhandenen Informationen zugreifen und mit diesen arbeiten.

1.1 Problemstellung und Motivation

Der eingangs beschriebene SoNBO-Explorer kann an verschiedene Notes-Datenbanken angeschlossen werden. Dazu müssen die Anbindungen an die Datenbanken entsprechend konfiguriert werden, weshalb unter anderem Kenntnisse über die Relationen der Daten untereinander notwendig sind. Der Explorer bietet keine direkte Hilfestellung bei dieser Problematik.

Dabei sieht der Konfigurationsprozess wie folgt aus: Zunächst wird ein *Social Network of Concepts* (SoNC) ermittelt (vgl. Abbildung 3). Ein Element aus dem SoNC, ein *Social Concept* (SC), ist mit einer Klasse aus dem Bereich der objektorientierten Programmierung zu vergleichen. Dieses Netzwerk stellt dar, welche SCs wie miteinander verbunden sind. Die Zusammenhänge sind anhand eines Beispiels in Kapitel 3.4 genauer erklärt. Nachdem das unternehmensspezifische Social Network of Concepts ermittelt ist, muss die Konfiguration in den SoNBO-Explorer übernommen werden. Dieses Vorgehen wird in Kapitel 3.5.3 erläutert. Zur Laufzeit wird aus der Konfiguration das *Social Network of Business Objects* (SoNBO) generiert. Die Elemente dieses SoNBOs sind *Business Objects* (BOs), welche aus den Social Concepts erzeugt werden und Instanzen derselben sind.

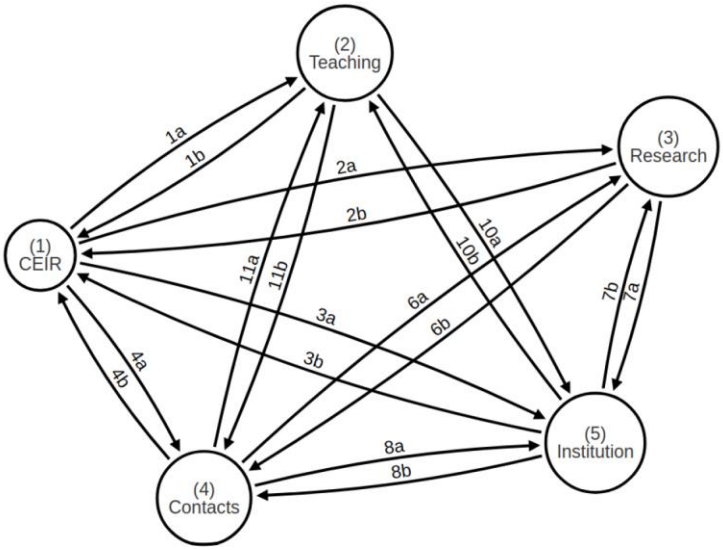


Abbildung 3: Social Network of Concepts (Götz, 2018, S. 49)

Das Ziel des Prototyps, der mit dieser Thesis entstanden ist, ist die Unterstützung dieses Konfigurationsprozesses. Der Anwender soll einen Graphen erstellen können, mit dem er die Beziehungen der Business Objects als Social Network of (Sub)Concepts darstellen kann. Außerdem soll aus diesem Social Network of (Sub)Concepts ein Social Network of Business Objects generiert werden, auf dem die Konfiguration angewendet wird. Hierbei soll im Gegensatz zum SoNBO-Explorer der Fokus nicht auf der Darstellung eines Objekts und dessen Nachbarn liegen, sondern auf der des gesamten Netzwerks. Diese visuelle Aufbereitung soll zum Verständnis beitragen, wie die Daten miteinander verknüpft sind. Somit wird eine Grundlage geschaffen, auf der die weitere Arbeit mit den Daten und die damit verbundene Generierung von Wissen aufbauen kann.

Mit diesem Ziel im Blick sind für die Konfiguration eines SoNBO-Explorers folgende Schritte notwendig:

1. Identifizierung des Netzwerks/Graphen
2. Visualisierung des Netzwerks/Graphen
3. Konfiguration des Netzwerks/Graphen im SoNBO-Explorer

Es gibt noch kein Tool, das den 2. Schritt – die Visualisierung des Netzwerks als Graph – umsetzt. Das Ergebnis dieser Arbeit ist ein Artefakt, das diese Visualisierung ermöglicht.

1.2 Aufbau der Thesis

Zunächst wird das Forschungsvorgehen genauer erklärt, an dem sich die Entwicklung des Prototyps orientiert hat. Danach folgen Informationen zum theoretischen Hintergrund, nämlich zu CRM-Systemen, Graphentheorie und dem SoNBO-Konzept. Zudem wird der SoNBO-Explorer genauer beschrieben. Als nächstes wird auf die Anforderungen eingegangen, aus welchen der Prototyp hervorgegangen ist. Es folgt der Kern der Arbeit, indem der Entwicklungsprozess der Anwendung behandelt

wird. Unter anderem geht es um die Architektur des Prototyps, die Systemlandschaft, in die die Anwendung integriert wurde, die Umsetzung der Visualisierung der Graphen, den Aufbau der Benutzeroberfläche, sowie um Herausforderungen und Probleme bei der Entwicklung. Im letzten Kapitel der Thesis werden die Ergebnisse zusammengefasst, außerdem wird ein Ausblick über weitere Entwicklungs- und Forschungsmöglichkeiten gegeben.

2 Forschungsvorgehen

Das Forschungsvorgehen dieser Thesis ist angelehnt an die Methodologie „Design Science Research“ (Vaishnavi & Kuechler, 2007). Da das Ziel dieser Arbeit die Entwicklung einer Anwendung ist, wurde diese in der gestaltungsorientierten Wirtschaftsinformatik übliche Methodik ausgewählt. Design Science ist im Forschungsgebiet Informationssysteme seit den 1990er-Jahren zunehmend erheblicher geworden (Peffers u. a., 2006). March & Smith (1995) beschreiben Design Science als technologieorientierte Herangehensweise, mit der neue Lösungen entwickelt werden können. Dabei steht primär die Erzeugung eines Artefaktes im Vordergrund und nicht wissenschaftliche Ansätze zum Erklären von Problemen (Vaishnavi & Kuechler, 2007). Hevner u. a. (2004) sagen zusätzlich, dass theoretische Ansätze über Einfluss und Auswirkungen des Artefakts nach dessen Entwicklung und Gebrauch entstehen. Es steht daher nicht von vorneherein fest, ob ein Mehrwert entstehen wird. Dennoch wird ein wissenschaftlicher Nutzen erwartet, weshalb Behavioral Science Methoden verwendet werden können, um eine Veränderung der Situation festzustellen, welche durch das Artefakt hervorgerufen wurde (March & Smith, 1995).

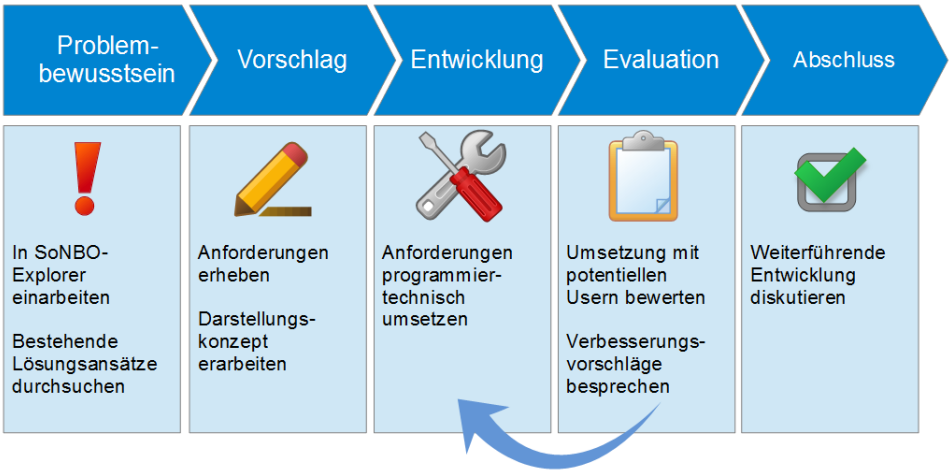


Abbildung 4: Forschungsvorgehen, angelehnt an Vaishnavi & Kuechler (2007) (eigene Darstellung)

Abbildung 4 zeigt das in dieser Arbeit verwendete Vorgehen, welches an das von Vaishnavi & Kuechler (2007) angelehnt ist. Zunächst erfolgte eine Analyse des bestehenden SoNBO-Explorer, um die genaue Funktionsweise zu eruieren. Ein erheblicher Punkt hierbei war das Verstehen der Konfigurationsdokumente des SoNBO-Exlorers. Die zentralen Dokumente im Bezug auf diese Thesis sind die, welche die Verbindung der Datenbankobjekte konfigurieren. Dann erfolgte die Einarbeitung in die Implementierung der Datenbankanbindungen. Parallel dazu hat das Durchsuchen von bestehenden Lösungsansätzen stattgefunden. Dabei wurde recherchiert, welche technischen Möglichkeiten es zur Umsetzung eines Knowledge-Graphen gibt. Nachdem ein Überblick hinsichtlich der Ausgangssituation vorlag und auf welche Weise eine Realisierung des Knowledge-Graphen möglich ist, konnten Anforderungen an

den Prototyp erhoben werden. Zudem wurde mit den recherchierten Ansätzen ein Konzept erarbeitet, wie die visuelle Aufbereitung dargestellt werden soll. Nach diesem Schritt folgte die eigentliche Entwicklung des Prototyps. Dabei wurde in regelmäßigen Abständen bewertet, ob der Prototyp den Anforderungen und dem SoNBO-Konzept nach Gewehr u. a. (2017) entspricht. Diese Bewertung konnte dann in die weitere Entwicklung einfließen. Die Schritte Entwicklung und regelmäßige Evaluation wurden so lange wiederholt, bis die Entwicklungsziele erreicht wurden. Als letzter Schritt wurde diskutiert, wie die weitere Entwicklung dieses Prototyps oder die einer weiteren Applikation aussehen kann, um das Langzeitprojekt SoNBO zu fördern.

2.1 Forschungsfragen

Die Forschungsfragen, welche die Basis für diese Thesis begründen, sind folgendermaßen den Schritten des Forschungsvorgehens zuzuordnen:

Problembewusstsein:

1. Wie ist der aktuelle Stand, auf dem der Prototyp aufbaut?

Vorschlag:

2. Inwiefern erweitert die Anwendung den aktuellen Stand?

Entwicklung:

3. Wie kann die Visualisierung sinnvoll umgesetzt werden?
4. Wie kann ein Anwender die Anwendung konfigurieren?

Evaluation:

5. Wie ist der Prototyp bezüglich der Ziele zu bewerten?

Abschluss:

6. Welche Möglichkeiten zur Weiterentwicklung gibt es?

2.2 Auswahl der Technologie

Die Auswahl der Technologie orientiert sich an den bereits bestehenden Anwendungen. Das SoNBo-Konzept wurde erstmals auf IBM Domino-Datenbanken angewendet (Gewehr u. a., 2017), außerdem ist der SoNBO-Explorer ebenfalls an IBM Domino-Datenbanken angeschlossen (Götz, 2018). Dies sind dieselben Datenbanken, an die auch die SoNBO-Graph-App angebunden ist. Daher wurde für den Prototyp ebenfalls IBM-Technologie verwendet. IBM XPages ist eine Erweiterung der Java Server Faces (JSF)-Technologie, die für die Entwicklung von Webanwendungen und für die Verwendung mehrerer Programmier- und Auszeichnungssprachen (Java, JavaScript, LotusScript, Formular language, HTML, CSS) entwickelt wurde (Donnelly, Wallace, & McGuckin, 2017). Mit der Auswahl dieser Technologie

kann von den bestehenden Datenbankanbindungen Gebrauch gemacht werden, weshalb die Entwicklung einer neuen Schnittstelle (API) nicht notwendig ist.

2.3 Anwendungsdesign und Implementierung

Nach Vaishnavi & Kuechler (2007) ist die Voraussetzung zur Erstellung eines Artefakts das Ergebnis einer (funktionalen) Anforderungsanalyse. Zu Beginn der Arbeit wurden handskizzierte Mockups entworfen (vgl. Anhang: Mock-ups SoNBO-Graph-App), welche die Grundlage hierfür waren. Die erhobenen Anforderungen sind ausformuliert worden und werden in Kapitel 4 beschrieben. Diese stammen hauptsächlich aus den Gesprächen mit den Betreuern der Thesis. Während der Entwicklung des Prototyps ist die Anforderungsliste mehrmals überarbeitet worden, da aus den Zwischenevaluationen gewonnene Erkenntnisse mit in die weitere Entwicklung eingeflossen sind.

Die programmiertechnische Umsetzung kann als agiles Projekt bezeichnet werden, da die Gesamtaufgabe in kleine Teilaufgaben untergliedert worden ist. Dadurch ist der Entwicklungsprozess flexibler und neue oder geänderte Anforderungen können umgesetzt werden (Martin, 2003). Die regelmäßigen Treffen, bei denen der aktuelle Stand der Entwicklung diskutiert wurde, sind ebenfalls Bestandteil einer agilen Entwicklung.

2.4 Evaluation

Evaluation “involves comparing the objectives of a solution to actual observed results from use of the artifact in the demonstration” (Peffers u. a., 2006, S. 92). Daher wurde bei der Evaluation regelmäßig der Entwicklungsstand mit den Anforderungen verglichen. Diese Bewertung der Anwendung hat ausschließlich mit den Betreuern der Thesis stattgefunden, da die Zielgruppe, welche die Graph-App benutzen soll, Kenntnisse über die Struktur der Daten bzw. den Aufbau der angeschlossenen Datenbanken besitzen muss. Nach der Fertigstellung des Prototyps wurde das Netzwerk der Forschungsgruppe Betriebliche Anwendungssysteme, das im SoNBO-Explorer konfiguriert ist, in die Graph-App eingegeben. Dadurch wurde die Funktionalität der Anwendung gezeigt und konnte ebenfalls von den Betreuern beurteilt werden.

3 Theoretischer Hintergrund

In diesem Kapitel werden die für die Arbeit relevanten theoretischen Grundlagen erläutert. Dabei liegt der Fokus zum einem auf CRM-Systemen, aus denen die zu integrierenden Informationen stammen, und zum anderen auf dem graphenbasierten Konzept zur Informationsintegration. Außerdem wird der bestehende SoNBO-Explorer erläutert.

3.1 CRM-Systeme

Die SoNBO-Graph-App sowie der SoNBO-Explorer sind an zwei Datenbanken eines CRM-Systems angeschlossen. Daher wird im Folgenden erläutert, was ein CRM-System ist.

“CRM systems accumulate, store, maintain, and distribute customer knowledge throughout the organization” (Chen & Popovich, 2003, S. 677). Außerdem geht es im Kundenbeziehungsmanagement darum, eine möglichst langlebige, gegenseitige Beziehung aufzubauen. In größeren Unternehmen haben verschiedene Mitarbeiter aus verschiedenen Abteilungen beim Ein- bzw. Verkaufsprozess Kundenkontakt. Der Aufbau langfristiger Kundenbeziehungen ist daher Teil der Strategieentwicklung eines Unternehmens und sollte alle Ebenen des Unternehmens einbeziehen (Håkansson & Snehota, 1995). Bedeutsame Informationen zu Angelegenheiten aus der Vergangenheit können entscheidend für die Zukunft der Kooperation mit einem Kunden sein. Deswegen werden in einem CRM-System Daten bezüglich der Kommunikation mit einem Kunden (zum Beispiel Adressen, Mitarbeiter, Status der Beziehung, …) gespeichert, außerdem operationale und statistische Informationen (Torggler, 2008).

Hippner & Wilde (2001) unterteilen CRM in drei Aufgabenbereiche:

- operatives CRM
- kommunikatives CRM
- analytisches CRM

Operatives CRM schließt alle Anwendungen ein, die auf den direkten Kundenkontakt ausgelegt sind. Diese CRM-Front-Office-Systeme sind auf die Unterstützung der Mitarbeiter eines Unternehmens ausgerichtet und sind häufig der wichtigste Bestandteil einer CRM-Umgebung (Amberg & Schumacher, 2002). Im *kommunikativen CRM* geht es um die „gesamte Steuerung und Unterstützung sowie die Synchronisation aller Kommunikationskanäle zum Kunden“ (Hippner & Wilde, 2001, S. 14). Diese Kommunikation zwischen Kunden und Unternehmen soll bidirektional möglich sein. Operatives und kommunikatives CRM unterstützen somit unmittelbar die Geschäftsprozesse eines Unternehmens. *Analytisches CRM* hingegen umfasst nach Hippner & Wilde (2001) die Aufzeichnung und Auswertung von Kundenreaktionen und zielt auf die Optimierung der kundenbezogenen Geschäftsprozesse ab.

Amberg & Schumacher (2002) unterteilen CRM-typische Funktionalitäten in folgende Kategorien:

- Basisfunktionalitäten
- Marketingmodule
- Vertriebs- oder Sales-Module
- Service-Module

Zu den *Basisfunktionalitäten* gehören z. B. die Bereitstellung einer Nutzeroberfläche, die Verwaltung von Kundendaten oder Aktivitätenmanagement. Diese Funktionalitäten bilden die Basis von CRM-Systemen, auf denen weitere Module aufbauen. *Marketingmodule* unterstützen „Aktivitäten der Planung, Vorbereitung, Durchführung und Kontrolle von Marketingkampagnen“ (Amberg & Schumacher, 2002, S. 29). Dazu zählen beispielsweise Analyse- und Prognosefunktionalitäten. Vertriebs- oder Sales-Module helfen Mitarbeitern im Innen- sowie Außendienst bei der Planung und Umsetzung von Verkaufsaktivitäten. Zu dieser Kategorie gehören z. B. Produktkonfiguratoren oder Systeme zur Bestellaufnahme. In der vierten Kategorie *Service-Module* steht die Einbeziehung des Kunden im Vordergrund. Amberg & Schumacher (2002) nennen Funktionalitäten für das Beschwerdemanagement oder für die Personalisierung für bzw. Individualisierung der Kunden als Beispiele für diese Kategorie.

Die SoNBO-Anwendungen sind an Datenbanken des CRM-Systems *GEDYS IntraWare 8* angebunden. Der Aufbau und die Inhalte dieser Datenbanken werden in Kapitel 5.2 behandelt. GEDYS unterstützt die oben genannten Funktionalitäten, jedoch werden für die Forschungsgruppe Betriebliche Anwendungssysteme der Universität Koblenz-Landau ausschließlich Basismodule verwendet. Daher greifen auch der SoNBO-Explorer und die SoNBO-Graph-App auf diese Module des CRM-Systems zu.

3.2 Graphentheorie

Mit der SoNBO-Graph-App lassen sich Graphen erstellen und generieren. Daher folgt ein kurzer Einblick in die Graphentheorie. „A *graph* is a way of specifying relationships among a collection of items“ (Easley & Kleinberg, 2010, S. 21). Weiterhin wird beschrieben, dass es eine Menge von Objekten gibt, die *Knoten* genannt werden, von denen es Paare gibt, die mit *Kanten* verbunden werden. Nach Bondy & Murty (1976) ist ein Graph G ein geordnetes Tripel aus einer nichtleeren Menge von Knoten V, einer davon disjunkten (elementfremd, kein gemeinsames Element) Menge von Kanten E und einer Funktion f, welche jede Kante einem Paar von Knoten zuordnet:

G = (V(G), E(G), f(G))

Wenn die Richtung, in der man eine Kante überquert, keine Rolle spielt, ist der Graph *ungerichtet*. Ein Graph, dessen Kanten kein Gewicht haben und als einzige Information das Vorhandensein von Beziehungen zwischen den Knoten widerspiegeln, wird *ungewichtet* genannt. Die Graphen, welche in dem Prototyp erstellt werden, sind sowohl ungerichtet als auch ungewichtet. Sie müssen jedoch nicht *vollständig* sein, es kann also Knoten geben, welche mit keinem anderen Knoten verbunden sind. Wenn ein Knoten B über eine Kante von einem Knoten A aus erreichbar ist, so ist B *Nachbar* von A.

3.3 Semantic Web

Im Forschungsgebiet Semantic Web existieren Ansätze, welche auf Graphen basieren und Konzepte zur graphischen Darstellung auf die Anwendung in Unternehmen übertragen: *Linked Enterprise Data* (Auer, Pietzsch, & Unbehauen, 2014) und *Enterprise Knowledge Graph* (Galkin, Auer, & Scerri, 2016). Diese Ansätze bilden die Grundlage für nachfolgend vorgestellte Taxonomie. Gebel-Sauer & Schubert (2019) führen für das SoNBO-Konzept neue Begriffe ein, wobei die *Ontologie* als *Social Network of Concepts (SoNC)* und der *Knowledge Graph* als *Social Network of Business Objects* bezeichnet werden (vgl. Tabelle 1).

Tabelle 1: Taxonomie für FGBAS-SoNBO (verändert nach Gebel-Sauer & Schubert, 2019)

Semantic Web	**SoNBO**		
Bezeichnung des Graphen	**Bezeichnung des Graphen**	**Bezeichnung der Knoten**	**Beispiel**
Ontologie	Social Network of Concepts	Social Concept	Mitarbeiter
Ontologie	*Social Network of Subconcepts*	*Social Subconcept*	*Professor*
Knowledge Graph = Ontologie & Faktenbasis	Social Network of Business Objects	Social Business Object	Prof. Dr. Anna Schäfer

Die Eingabe in den SoNBO-Explorer bzw. die Graph-App ist die Ontologie, der Knowledge Graph wird aus dieser zuzüglich der Daten aus GEDYS IntraWare 8 generiert. In Abbildung 3 ist ein Beispiel für eine Ontologie zu sehen.

Im Rahmen dieser Arbeit wird eine zusätzliche Ebene zwischen SoNC und SoNBO eingeführt: *Das Social Network of Subconcepts* (SoNSC). Diese Ebene dient dazu, die Spezifikationen der Inhalte und Beziehungen der Social Business Objects zu definieren. Auf der SoNC-Ebene werden im Gegensatz dazu die Existenz der Objekte und die der Verbindungen festgelegt. Diese Zusammenhänge werden im folgenden Kapitel 3.4 erklärt und anhand eines Beispiels verdeutlicht (vgl. Abbildung 5).

3.4 Social Network of Business Objects (SoNBO)

Da die Graph-App sich nach dem Konzept „Social Network of Business Objects“ (SoNBO) richtet, wird dieses Konzept genauer erklärt. Der Ursprung von SoNBO liegt bei der ersten Umsetzung, die von Gewehr u. a. (2017) als „innovative approach for information integration“ (S. 911) beschrieben wird. Götz & Gebel-Sauer (2018) definieren das Konzept zusätzlich als integrierte und plattformunabhängige Lö-

sung für den digitalen Arbeitsplatz. Eine weitere bedeutsame Eigenschaft von SoNBO nach Gewehr u. a., (2017) ist die Verarbeitung der Daten ohne redundantes Abspeichern von Informationen. Das heißt, dass ein Social Network of Business Objects lediglich Daten aus den Quellen liest, und daher an keine eigene Datenbank angebunden ist, in der Nutzdaten abgespeichert sind.

Die Knotenelemente, aus denen ein SoNBO besteht, sind die Social *Business Objects (BOs)* (vgl. Tabelle 1). Ein solches Object ist die Instanz einer Subklasse. Diese Subklassen sind die Knotenelemente des Social Network of Subconcepts und werden als *Social Subconcepts (SSCs)* bezeichnet. Außerdem gehört jedes SSC einem im Social Network of Concepts vorkommendem *Social Concept (SC)* an. Ein Business Object besteht aus Attributen, wobei ein Attribut ein Key-Value Paar ist. Die Menge der Attribute wird im zugehörigen Social Subconcept definiert. Wenn beispielsweise „Professor“ ein Social Subconcept mit den Attributen Name, Forschungsgebiet, Lehrveranstaltung und Email-Adresse ist, kann es ein Business Object (Prof. Dr. Ben Müller, CRM, Vorlesung CRM-Systeme, bmueller@uni.de) geben. Ein Business Object (Prof. Dr. Anna Schäfer, ERP, Vorlesung ERP-Systeme, aschaefer@uni.de) ist ein weiteres Beispiel, welches ebenfalls eine Instanz dieses Social Subconcepts ist. Social Concepts aggregieren zusammengehörige Social Subconcepts. So kann das Social Subconcept „Professor“ beispielsweise einem Social Concept „Mitarbeiter“ zuzuordnen sein. Ein anderes Beispiel ist ein Social Subconcept „Doktorand“, welcher ebenfalls „Mitarbeiter“ sein kann. Dieser muss allerdings nicht die gleichen Attribute wie „Professor“ haben, da diese nicht im Social Concept „Mitarbeiter“ festgelegt sind, sondern im jeweiligen Social Subconcept. Nach Gewehr u. a. (2017) sind neben einem Mitarbeiter auch eine Organisationseinheit, ein Projekt, ein Vertrag, ein Lieferant, ein Produkt, eine Urlaubsanforderung, eine Arbeitszeiterfassung, eine Ausgangsrechnung oder ein Kundenkontakt Beispiele für Social Concepts.

Als Nächstes wird erklärt, wie die Business Objects miteinander in Verbindung stehen, sodass mit ihnen ein Netzwerk abgebildet werden kann. Die Struktur des Netzwerks wird im Social Network of Concepts definiert (vgl. Abbildung 3). Sind zwei Social Concepts A und B per Definition miteinander verbunden, so sind auch alle Social Subconcepts von A mit denen von B verbunden. Auf welche Weise diese miteinander verbunden sind, wird im Social Network of Subconcepts festgelegt. Ein Beispiel für ein weiteres Social Concept ist „Lehre“, welche das Social Subconcept „Vorlesung“ mit den Attributen Name, Raum und Dozent hat. Es kann ein zugehöriges Business Object mit (Vorlesung ERP-Systeme, A 301, Prof. Dr. Anna Schäfer) geben. Wenn nun „Mitarbeiter“ und „Lehre“ im Social Network of Concepts miteinander verbunden sind, so sind auch die zugehörigen Social Subconcepts im Social Network of Subconcepts miteinander verbunden, also „Professor“ mit „Vorlesung“ sowie „Doktorand“ mit „Vorlesung“. Wenn im SoNSC entsprechend festgelegt ist, wie zwei SSCs verbunden sind, zum Beispiel wenn in einer Vorlesung als Dozent ein Professor eingetragen ist, können so die Business Objects im Social Network of Business Objects miteinander verbunden sein. In diesem Beispiel wäre daher Prof. Dr. Anna Schäfer mit der Vorlesung ERP-Systeme verbunden.

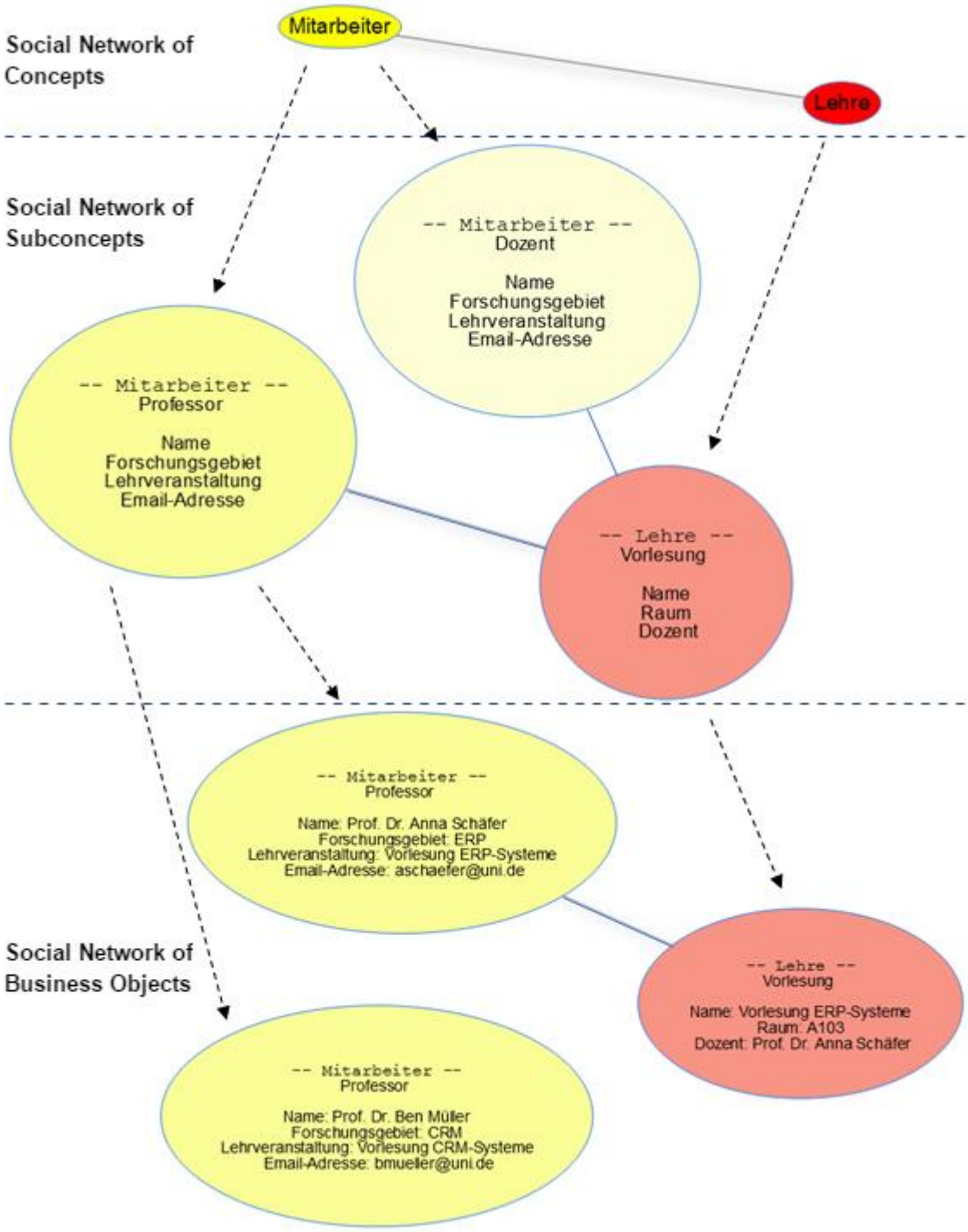


Abbildung 5: Zusammenhang der Social Networks (eigene Darstellung)

Als Unterschied eines Business Objects gegenüber einem einfachen Datensatz führen Gewehr u. a. (2017) Elemente von Social Software auf. Jedes Business Object ist einem Profil zugeordnet, welches die angezeigten Attribute und die Darstellung des Objekts bestimmt. Außerdem gibt es einen activity stream, welcher Ereignisse und Nachrichten der Business Objects auflistet. Zusätzlich können auf jedem Profil Leistungskennzahlen (Key Performance Indicators (KPIs)) und Berichte angezeigt werden. Daher ist der SoNBO-Ansatz nicht auf den Informationszugriff limitiert, sondern trägt auch zur Informationssuche, Entscheidungsfindung und Umsetzung notwendiger Maßnahmen bei und ist als ganzheitliches Konzept für den digitalen Arbeitsplatz zu betrachten.

3.5 SoNBO-Explorer

Der SoNBO-Explorer ist ein Prototyp, welcher im Rahmen einer Masterarbeit an der Universität Koblenz-Landau entstanden ist (Götz, 2018). Im Folgenden wird diese IBM XPages-Anwendung genauer erklärt. Zum einen ist sie eine Grundlage für die SoNBO-Graph-App, zum anderen ist ein Hauptziel der Graph-App, den Konfigurationsprozess des Explorers zu unterstützen.

3.5.1 Benutzerschnittstelle

Mit dem SoNBO-Explorer navigiert man durch ein Social Network of Business Objects (vgl. Abbildung 6). Dabei sieht der Anwender Daten des betrachteten Business Objects (1) und alle Nachbarn dieses Objects (3). Die Nachbarn können gefiltert werden, sodass entweder alle oder nur solche eines besimmten Social Concepts bzw. eines Social Subconcepts angezeigt werden (2). Die Anzahl der gefilterten Ergebnisse wird links neben der Auswahl angezeigt. Eine globale Suchfunktion war ursprünglich geplant, diese ist jedoch noch nicht umgesetzt worden (Götz, 2018, S. 54). Ein Navigationswidget zeigt die zuletzt besuchten Business Objects und bietet die Funktion, zum Ausgangspunkt zurückzuspringen (5). Die Kacheln, welche die Nachbarn anzeigen, können anhand eines Attributes in lexikographischer Ordnung sortiert werden (6). Außerdem ist es möglich, die Nachbarn nach bestimmten Attributen zu filtern (7).

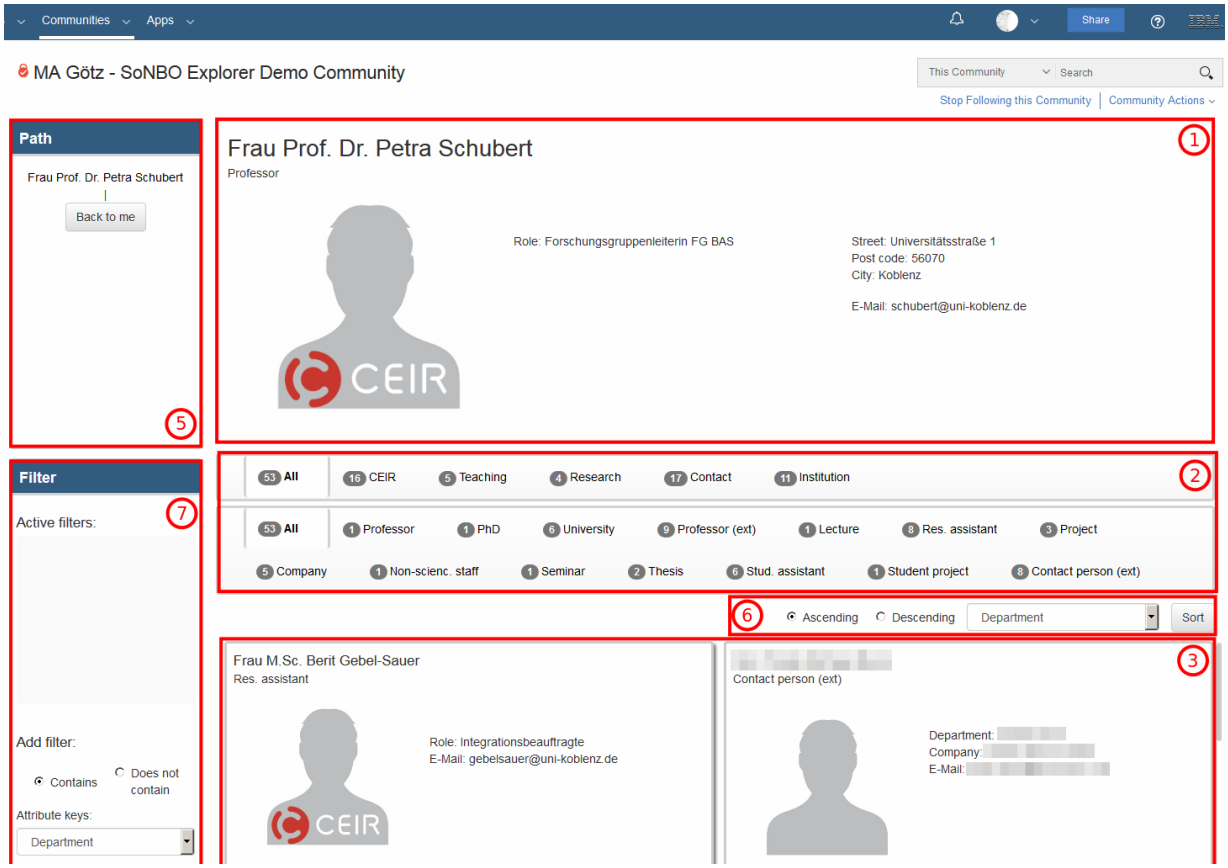


Abbildung 6: SoNBO-Explorer Benutzerschnittstelle (Götz, 2018, S. 55)

Wie in Kapitel 1 eingangs beschrieben, liegt der Fokus auf dem aktuell ausgewählten Business Object und dessen Nachbarn. Mit einem Klick auf eines der verbundenen Business Objects wird dieses ausgewählt und dessen Nachbarn angezeigt. Auf diese Weise kann der Anwender durch das gesamte Netzwerk navigieren. Es ist nicht möglich, eine Gesamtübersicht des Netzwerks zu erhalten.

3.5.2 Systemlandschaft

Der SoNBO-Explorer ist eine IBM Domino-Anwendung, welche zwischen dem CRM-System GEDYS IntraWare 8 und IBM Connections platziert ist (Götz, 2018) (vgl. Abbildung 7). Er ist über den Webbrowser als Communitywidget in IBM Connections aufrufbar. Der Explorer integriert zwei Datenbanken von GEDYS (*contacts.nsf* und *georga.nsf*), außerdem speichert er Konfigurationsdokumente in einer eigenen Datenbank. In georga.nsf werden CRM-Informationen zur eigenen Organisation gespeichert, in contacts.nsf stehen Informationen zu Projekten und externen Kontakten. Die den Datenbanken zugehörige Applikationen, mit denen die Inhalte verwaltet werden, werden über den Notes-Client oder einen Webbrowser aufgerufen. Die Programmierschnittstelle zur Integration der Datenbanken in den SoNBO-Explorer ist individuell angepasst, da GEDYS IntraWare 8 keine eigene Schnittstelle zur Verfügung stellt.

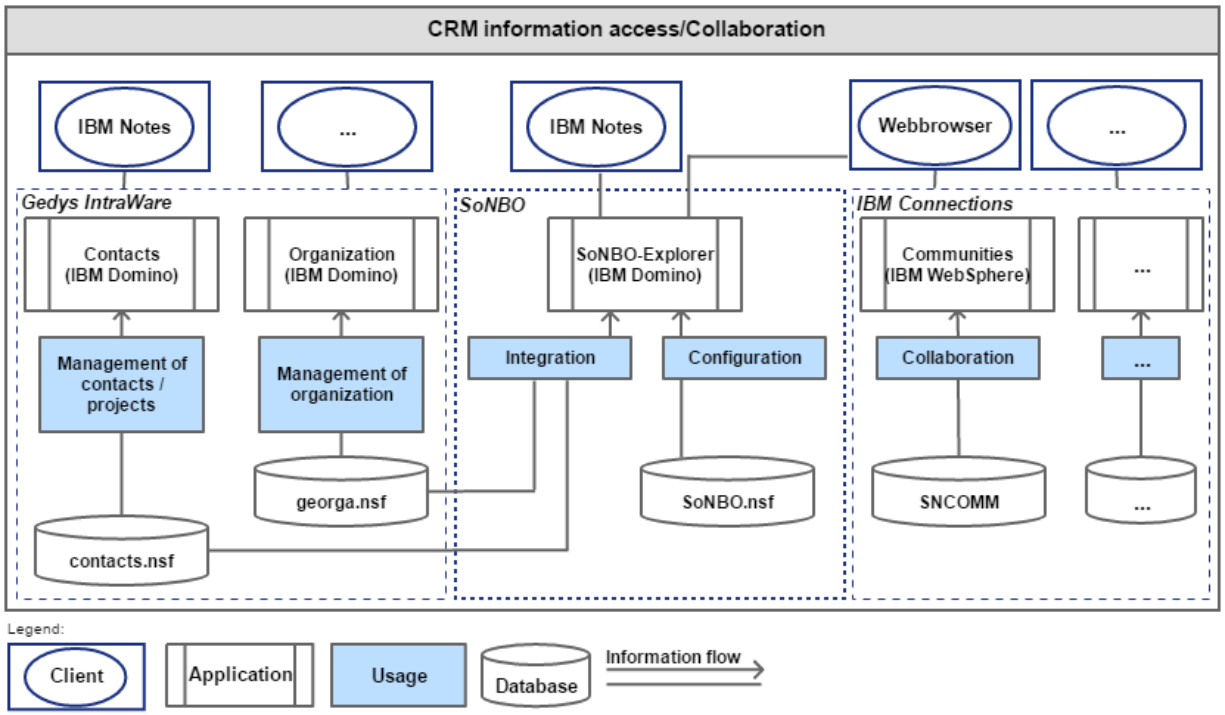


Abbildung 7: Systemlandschaft SoNBO-Explorer in der Forschungsgruppe Betriebliche Anwendungssysteme der Universität Koblenz-Landau (verändert nach Götz & Gebel-Sauer, 2018, S. 2001)

Der SoNBO-Explorer legt in der eigenen Datenbank *SoNBO.nsf* keine Nutzdaten ab, da nach dem SoNBO-Konzept Redundanz vermieden wird.

3.5.3 Administrationsschnittstelle

Es wurde eine Administrationsschnittstelle erstellt, um den SoNBO-Explorer zu konfigurieren. Diese Schnittstelle besteht aus verschiedenen IBM Notes Formularen bzw. Views (Götz, 2018). In den Formularen wird die SoNBO-Konfiguration eingetragen (und in den Views angezeigt). Diese Konfiguration ist der Prozess, welcher durch die SoNBO-Graph-App visuell unterstützt wird.

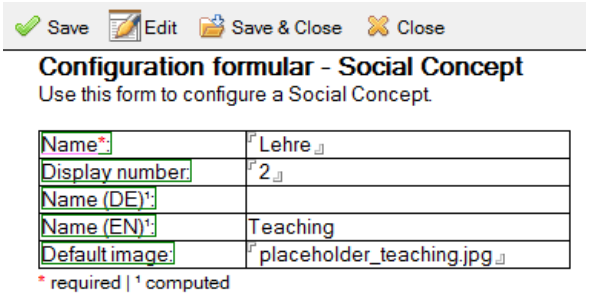


Abbildung 8: Neues Social Concept im SoNBO-Explorer (eigene Darstellung)

In einem Formular werden neue *Node type categories* (Knotentypen) angelegt (vgl. Abbildung 8), was nach der neuen SoNBO-Terminologie von Gebel-Sauer & Schubert (2019) die Social Concepts sind. Hierfür trägt man den Namen des Social Concepts ein, außerdem die Stelle, an der es in der Filterleiste (2) aufgelistet wird, und ein Standardbild, welches angezeigt wird (1).

In einem weiteren Formular werden zuvor erstellte Knotentypen konfiguriert (vgl. Abbildung 9). Diese Konfiguration entspricht der Erstellung von Social Subconcepts. Hierzu werden wie bei den Social Concepts Name und Anzeigebereich in der Filterleiste angegeben. Zusätzlich muss ein Attribut als Identifikator (ID) angegeben werden. Außerdem ist ein Key-Value-Paar notwendig, um den Social Subconcepts entsprechende Business Objects zuordnen zu können. Der Benutzer gibt an, in welchem Feld der Datenbank welcher Wert stehen muss, damit der entsprechende Datensatz diesem Social Subconcept zuzuordnen ist. Im abgebildeten Beispiel ist ein Business Object genau dann ein „Professor“, wenn in dem Feld „fdRoleNames“ in der Datenbank der Wert „Professor“ eingetragen ist. Zuletzt muss noch das Attribut festgelegt werden, welches als Titel angezeigt werden soll und das zugehörige Social Concept ausgewählt werden.

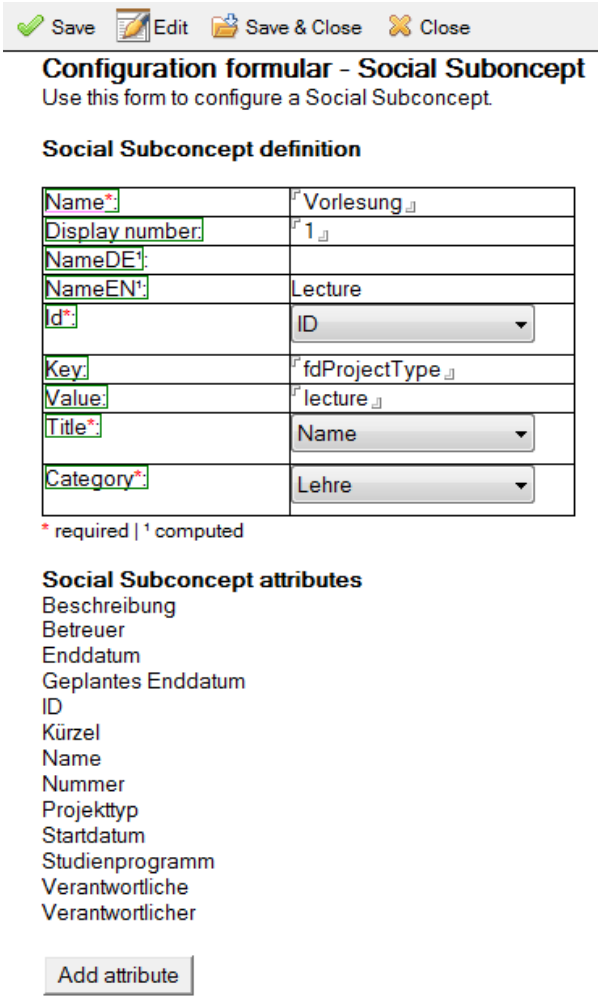


Abbildung 9: Neues Social Subconcept im SoNBO-Explorer (eigene Darstellung)

Unterhalb der Eingabe für die Daten des Social Concepts werden alle Attribute aufgelistet, welche dieses besitzt. Attribute kann der Anwender über eine weitere Eingabemaske hinzufügen (vgl. Abbildung 10). Einem neuen Attribut werden zunächst ein Social Subconcept und ein Name zugeordnet. Als Nächstes folgen die Datenquelle und die Abfrage (Query), mit der das Attribut aus der Datenbank gelesen wird. Die Abfrage liefert einen JSON-String (JavaScript Object Notation) (Bray, 2017) zurück, aus dem ein Wert eines bestimmten Feldes ausgelesen wird. Dieses Key-Value Paar, bestehend aus Feldname und Wert, wird dem Business Object zugeordnet. Der Datentyp des Feldes muss festgelegt werden, zudem noch die Position in (1), an der das Attribut auf der Benutzeroberfläche angezeigt werden soll. Die zwei unteren Eingabefelder bestimmen, ob nach dem Attribut im Filter-Widget (7) gefiltert werden kann und ob das Attribut bei den Nachbarknoten als Information in der Vorschau (3) mitaufgeführt werden soll.

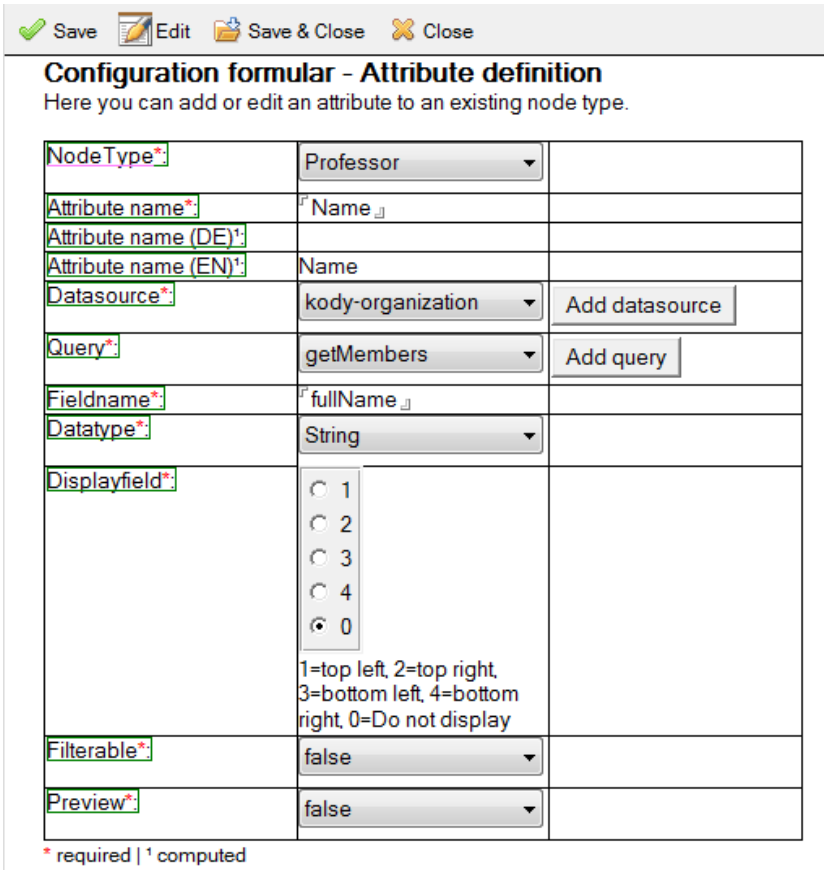
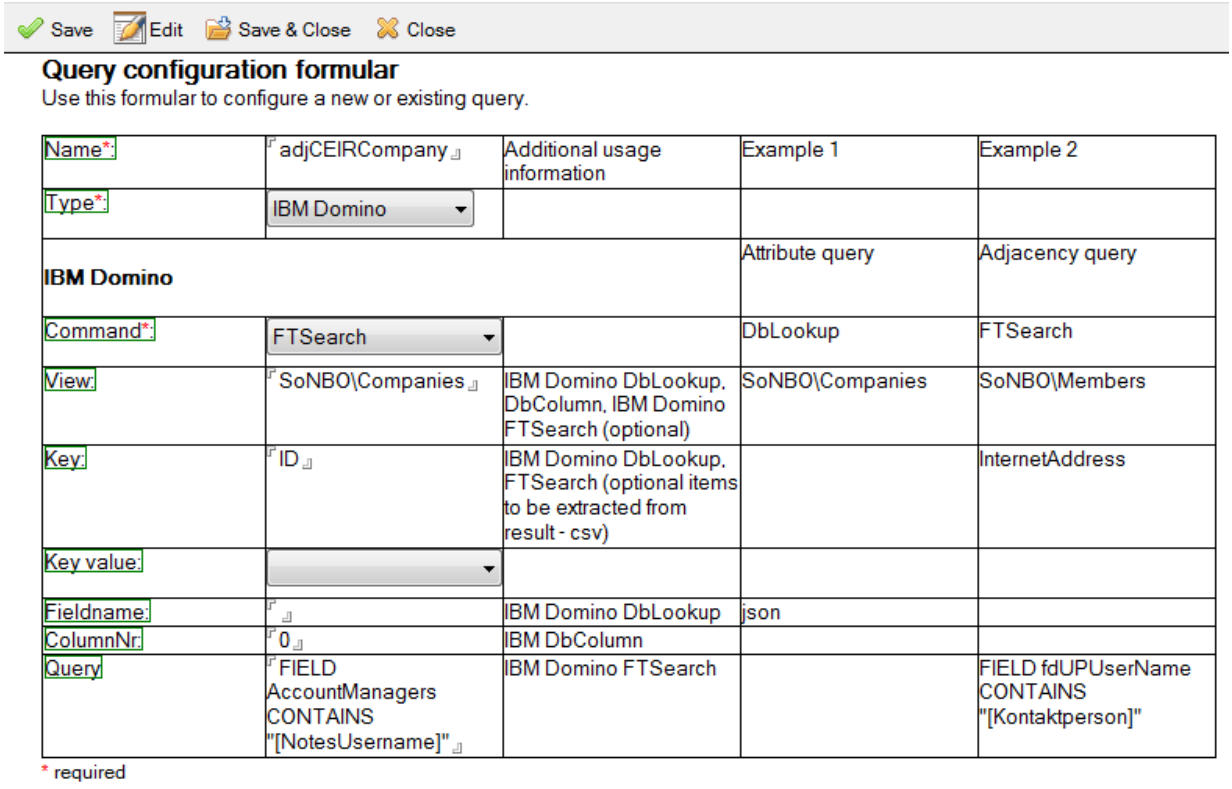


Abbildung 10: Neues Attribut im SoNBO-Explorer (eigene Darstellung)

Die Datenquellen und die Abfragen, die den Attributen zugeordnet werden, werden in einem separaten Formular angelegt (vgl. Abbildung 11). Für eine Datenquelle gibt der Anwender einen Namen und einen Pfad zur Datenbank an. Bei einer Abfrage muss unterschieden werden, ob sie für ein Attribut eines Social Business Objects wie in Abbildung 11 (Example 1) benötigt wird, oder für die Bedingung einer Kante zwischen zwei Objects, wie in Abbildung 11 (Example 2). Im ersten Fall wird der JSON-String

aus der entsprechenden View abgefragt. Hierfür gibt der Anwender der Abfrage einen Namen, wählt „DbLookup“ als Befehl aus, trägt die entsprechende View und den Feldnamen „json“ ein. Im zweiten Fall wird auf den Eintrag in der View eine Volltextsuche mit einer Bedingung durchgeführt. Dazu muss nach der Namensgebung „FTSearch“ als Befehl ausgewählt werden. Mit der Auswahl der View muss die ID-Spalte der View angegeben werden. Im unteren Eingabefeld trägt der Anwender die Bedingung ein, welche zutreffen muss, damit zwei Social Business Objects miteinander verbunden werden und als benachbart angezeigt werden.



Save Edit Save & Close Close

Query configuration formular

Use this formular to configure a new or existing query.

Name*:	adjCEIRCompany	Additional usage information	Example 1	Example 2
Type*:	IBM Domino			
IBM Domino			Attribute query	Adjacency query
Command*:	FTSearch		DbLookup	FTSearch
View:	SoNBO\Companies	IBM Domino DbLookup, DbColumn, IBM Domino FTSearch (optional)	SoNBO\Companies	SoNBO\Members
Key:	ID	IBM Domino DbLookup, FTSearch (optional items to be extracted from result - csv)		InternetAddress
Key value:				
Fieldname:		IBM Domino DbLookup	json	
ColumnNr:	0	IBM DbColumn		
Query	FIELD AccountManagers CONTAINS "[NotesUsername]"	IBM Domino FTSearch		FIELD fdUPUserName CONTAINS "[Kontaktperson]"

* required

Abbildung 11: Neue Abfrage im SoNBO-Explorer (eigene Darstellung)

Die erstellten Abfragen für die Kanten werden in einem weiteren Formular den Kanten zugeordnet (vgl Abbildung 12). Neben einer ID für die Verbindung werden Quelle in Form eines Social Subconcepts, ein Name für die Verbindung und ein weiteres Social Subconcept als Ziel festgelegt. Dann gibt der Anwender die View an, in welcher das Zielobjekt gespeichert ist. Im letzten Feld wird schließlich eine wie zuvor beschriebene Bedingung in Form einer definierten Abfrage eingetragen.

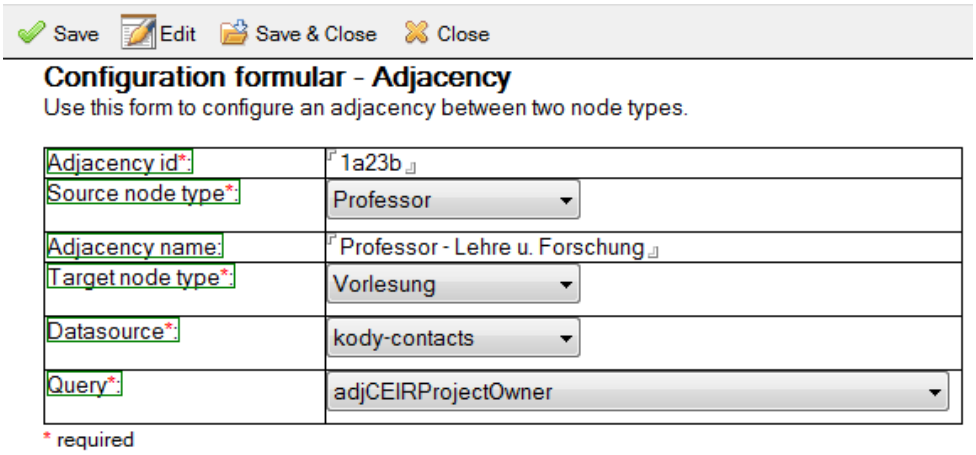


Abbildung 12: Neue Adjazenz im SoNBO-Explorer (eigene Darstellung)

In weiteren Views und Formularen, die der SoNBO-Explorer mit sich bringt, werden noch ID-Zuordnungen eingetragen, durch die der Explorer beschleunigt wird. Außerdem gibt es eine Zuordnung für Übersetzungen von Deutsch nach Englisch. Zusätzlich gibt es eine Übersicht, in der alle Konfigurationsdokumente aufgelistet sind, egal in welchem Formular sie erstellt wurden. Für die SoNBO-Graph-App werden diese Formulare nicht benötigt.

4 Anforderungen

Nach der Einarbeitung in den SoNBO-Explorer fanden Überlegungen hinsichtlich einer Umsetzung des SoNBO-Konzeptes, welches in Kapitel 3.4 erklärt ist, in einer SoNBO-Graph-App statt. Deren Ziele sind in Kapitel 1.1. erläutert. Hierfür sind Mockups entstanden, mit deren Hilfe die Problemstellung verdeutlicht wurde (vgl. Anhang: Mock-ups SoNBO-Graph-App, Mock-up: Bedienung der SoNBO-Graph-App). Die nachfolgenden Anforderungen sind die, die vor der Entwicklung erhoben wurden (Mit einem (*) markierte Anforderungen sind bis zum Ende der Entwicklung nicht umgesetzt worden).

Diese Anforderungen sind auf die Benutzeroberfläche, die dortigen Funktionen und die Darstellung der Graphen bezogen:

- Ein Social Network of Concepts muss als Graph dargestellt werden.
- Ein Social Network of Subconcepts muss als Graph dargestellt werden.
- Ein Social Network of Business Objects muss als Graph dargestellt werden.
- Social Network of Concepts, Social Network of Subconcepts und Social Network of Business Onjects müssen getrennt voneinander dargestellt werden.
- Social Networks of Concepts und Social Networks of Subconcepts müssen in ihren einzelnen Komponenten editierbar sein.
- Die Inhalte einer Kante müssen dargestellt werden.
- Die Inhalte eines Knotens müssen dargestellt werden.
- Zur Visualisierung soll die Library visjs (visjs.org) verwendet werden.
- Die Website kann das Responsive Design-Paradigma erfüllen. (*)

Da die Funktionalität der Anwendung im Vordergrund steht, ist der letzte Punkt nicht realisiert worden. Dessen Umsetzung kann bei einer Weiterentwicklung des Prototyps in Betracht gezogen werden.

Die nächsten Anforderungen beziehen sich auf den Zusammenhang der einzelnen Netzwerke. Dabei geht es um die Einhaltung der Regeln, die die Abhängigkeiten der Netzwerke untereinander betreffen. Diese Anforderungen sind grob formuliert, da die Regeln von Gewehr u. a. (2017) beschrieben werden und diese daher nicht explizit in die Anforderungen aufgenommen worden sind:

- Die Regeln eines Social Network of Concepts müssen auf ein Social Network of Subconcepts angewendet werden.
- Die Regeln eines Social Network of Concepts und eines Social Network of Subconcepts müssen auf ein Social Network of Business Objects angewendet werden.

Die Anbindung der SoNBO-Graph-App an die Systemlandschaft (vgl. Abbildung 14) wird in den nächsten Punkten beschrieben. Auch hierbei gelten die SoNBO-Regeln, die nicht explizit aufgelistet sind:

- Die Anwendung muss relevante Daten aus GEDYS IntraWare 8 auslesen.
- Die Anwendung kann Konfigurationsdokumente im SoNBO-Explorer (CRM) bearbeiten. (*)
- Die Anwendung muss via Webbrowser aufzurufen sein.

Die Möglichkeit, Konfigurationsdokumente in den SoNBO-Explorer zu exportieren, war zu Beginn der Arbeit geplant, ist aber im Laufe der Entwicklung verworfen worden, als der benötigte Aufwand für die Realisierung deutlich wurde. Allerdings ist es denkbar, dass ein weites Projekt mit dem Ziel, diesen Export der Konfiguration aus der SoNBO-Graph-App umzusetzen, gestartet wird.

Nach der Erhebung der zuvor genannten Anforderungen hat die eigentliche Entwicklung der Graph-App begonnen. Diese wird im folgenden Kapitel 5 beschrieben. Während der Entwicklung sind durch die Evaluationen mit den Betreuern der Arbeit weitere Anforderungen hinzugekommen:

- Ein Zurück-Button soll die letzte Änderung in der Graph-App rückgängig machen. (*)
- Datenbankeinträge mit gleicher ID (replication- oder saveconflict) sollen nicht zum Abstürzen der Anwendung führen.
- Die Generierung eines Social Network of Business Objects mit nicht fertig definierten Knoten muss möglich sein.
- Die Generierung eines Social Network of Business Objects mit nicht fertig definierten Kanten muss möglich sein.
- Undefinierte Social Subconcepts und undefinierte Kanten sollen grau eingefärbt sein.
- Definierte Social Subconcepts und Kanten sollen eingefärbt werden.
- Social Business Objects sollen farblich entsprechend der zugehörigen Social Subconcepts eingefärbt werden.
- Kanten im Social Network of Subconcepts sollen eine „Immer verbinden“-Konfigurationsmöglichkeit erhalten.

Der Zurück-Button sollte für die Verbesserung der Nutzerfreundlichkeit eingebaut werden. Allerdings steht der Implementierungsaufwand in keinem Verhältnis zu der möglichen Verbesserung, daher ist die Umsetzung dieser Anforderung verworfen worden.

5 SoNBO-Graph-App

In diesem Kapitel wird beschrieben, wie die Anforderungen aus Kapitel 4 konkret umgesetzt worden sind. Dabei geht es um die Anbindung an das CRM-System GEDYS IntraWare 8, die Visualisierung der Graphen, den Datenaustausch auf den verschiedenen Anwendungsschichten und die Benutzeroberfläche, auf der der Anwender arbeitet.

5.1 Technische Anbindung

Es gibt zwei Möglichkeiten, die SoNBO-Graph-App in die bestehende Systemlandschaft einzubetten: Der erste Ansatz hierfür bestand darin, den SoNBO-Explorer zu modifizieren. Das heißt, dass keine eigenständige Anwendung entwickelt worden wäre und die Systemlandschaft unverändert geblieben wäre. Die Vorteil bei diesem Ansatz ist die Möglichkeit, mit den Java-Methoden des Explorers arbeiten zu können. Diese Methoden sind darauf ausgelegt, als Rückgabewert ein Business Object und dessen Nachbarn auszugeben. Es ist jedoch eine Programmierschnittstelle entwickelt worden, welche auf die (hierfür leicht modifizierten) Java-Klassen im Explorer zurückgreift und wie von Tarjan (1972) beschrieben mittels Tiefensuche ein Social Network of Business Objects generiert (vgl. Abbildung 13).

```
function iterateAllNodes (startingNode) {
	// initialisiere Ausgangsknoten und Nachbarn
	soNBOManager.initGraph(startingNode.getId(),
		startingNode.getNodeTypeCategory(), startingNode.getNodeType(),
		null);
	var adjacentNodes = soNBOManager.getAdjacentNodeList();

	// erstelle JSON des Ausgangsknotens
	var myNodeJSON = createSingleNodeJSON(startingNode);

	// Ausgangsknoten wird in Liste geschrieben, damit er nicht nochmal
		besucht wird
	doneIds.push(startingNode.getId());

	// Ausgangsknoten der ErgebnisJSON hinzufügen
	allNodesJSON = allNodesJSON + myNodeJSON + ",\n";

	// alle Nachbarn herausfinden
	for (var j = 0; j < adjacentNodes.length; j++){
		var tmpEdgeJSON = createSingleEdgeJSON(startingNode,
			adjacentNodes[j]);
		allEdgesJSON = allEdgesJSON + tmpEdgeJSON + ",\n";
	}
	allEdgesJSON = allEdgesJSON + "\n\n";

	// Tiefensuche: Rekursiver Funktionsaufruf
	for (var i = 0; i < adjacentNodes.length; i++){
		if (!arrayContains(doneIds, adjacentNodes[i].getId()))
			iterateAllNodes(adjacentNodes[i]);
	}
}
```

Abbildung 13: Tiefensuche im SoNBO-Explorer (eigene Darstellung)

Der Anwender hätte ein Social Network of Concepts und ein Social Network of Subconcepts erstellt, diese Konfiguration in den SoNBO-Explorer übertragen und daraus dann das SoNBO generiert. Nach Gesprächen mit den Betreuern ist dieser Ansatz verworfen worden, da eine vom Explorer unabhängige Anwendung gewünscht ist. Außerdem hätte eine Funktion implementiert werden müssen, welche die Konfiguration des Netzwerks in die Konfigurationsdokumente des SoNBO-Explorers übernimmt. Diese Funktion war ursprünglich und unabhängig von der Art der Anbindung geplant (vgl. Kapitel 4), deren Entwicklung hätte jedoch den Rahmen dieser Thesis gesprengt. Dennoch kann die entwickelte Schnittstelle verwendet werden, wenn in einer zukünftigen Thesis oder einer ähnlichen Arbeit der SoNBO-Explorer weiterentwickelt wird, um Netzwerke visualisieren zu können.

Im zweiten Ansatz ist eine eigenständige Anwendung entwickelt worden, welche in die Systemlandschaft eingebunden wurde (vgl. Abbildung 14). Diese Anwendung ist unabhängig vom SoNBO-Explorer und via Webbrowser zu bedienen. Die Konfiguration der Graph-App findet nicht über Konfigurationsdokumente wie im Explorer statt, sondern über die Erstellung von Graphen auf der Weboberfläche. Dieser Vorgang wird in Kapitel 5.5 beschrieben. Der Prototyp greift auf die gleichen Datenbanken von GEDYS IntraWare 8 zu, sodass die Umsetzung der Integration dieser Datenbanken aus dem SoNBO-Explorer übernommen worden ist. Die Java-Klassen zur Erzeugung des Netzwerks aus dem Explorer sind nicht übernommen worden, da sie für die Erzeugung des Social Network of Business Objects – wie im letzten Abschnitt beschrieben - nicht gedacht sind. Eine Integration in IBM Connections ist für die Graph-App nicht geplant, da die Zielgruppe die Anwender sind, die das Netzwerk konfigurieren und nicht ausschließlich nutzen.

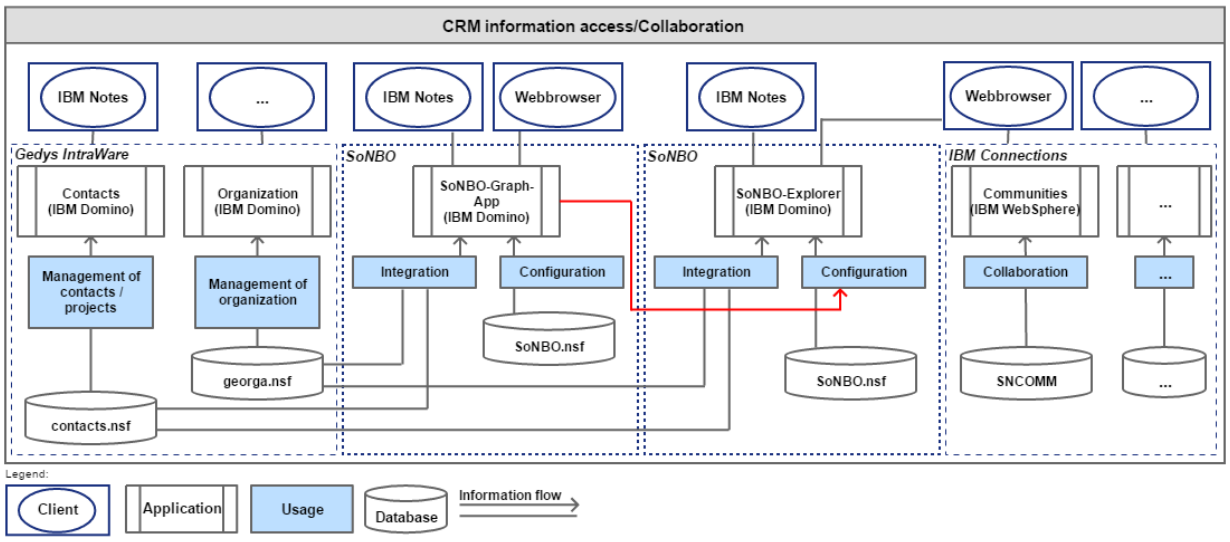


Abbildung 14: Systemlandschaft SoNBO Graph-App in der Forschungsgruppe betriebliche Anwendungssysteme, angelehnt an Götz & Gebel-Sauer (2018, S. 2001)

Der SoNBO-Explorer und die SoNBO-Graph-App sind über keine Programmierschnittstelle miteinander verbunden. Die Informationen aus der Graph-App fließen über den Anwender mit in die Konfiguration des Explorers ein. Die Implementierung einer solchen Schnittstelle wird in Kapitel 4 angesprochen.

5.2 Systemanalyse CRM-System GEDYS IntraWare 8

Die technische Anbindung an das CRM-System GEDYS IntraWare 8 ist im Kapitel 3.5.2 erklärt. Im Folgenden geht es darum, wie die Inhalte und deren Zusammenhänge gespeichert werden, sodass der Prototyp diese visualisieren kann.

GEDYS wird vom Anwender über ein Frontend, welches über IBM Notes und einen Webbrowser zu erreichen ist, bedient (vgl. Abbildung 14). Dabei werden die Daten der Business Objects in bestimmte Felder eingegeben und in einer der beiden Datenbanken (*contacts.nsf* oder *georga.nsf*) gespeichert. Die Software GEDYS IntraWare 8 IntraWare 8 ist auf den Betrieb in einem Unternehmen zugeschnitten. GEDYS ist daher von der Forschungsgruppe Betriebliche Anwendungssysteme auf den Betrieb an der Universität angepasst. Das heißt, dass zum einen festgelegt worden ist, in welchem Feld welche Inhalte stehen sollen. Beispielsweise werden „Professoren“ in *georga.nsf* gespeichert und deren zugeordnete Lehrveranstaltungen und Projekte im Reiter „Marketing“ im Feld „Schlagwörter“. Zum anderen sind auswählbare Schlüsselwörter ersetzt worden, sodass zum Beispiel im Feld „Rolle“ nicht mehr „Management“ oder „Marketing“ ausgewählt werden kann, sondern unter anderem „Professor“ oder „HiWi“. Weiterhin ist definiert worden, über welche Felder Business Objects miteinander verbunden sind, beispielsweise wenn im Feld „Schlagwörter“ einer Professorin „Anna Schäfer“ die Vorlesung „ERP-Systeme“ eingetragen ist (vgl. Kapitel 3.4, vgl. Abbildung 5).

Diese Metainformationen sind essenziell, wenn es um die Konfiguration der Graph-App geht, weil damit das Social Network of Business Objects erstellt werden kann. Im Fall des CRM-Systems GEDYS IntraWare 8 waren diese Metadaten bereits vorhanden, da sie im Rahmen der Entwicklung des SoNBO-Explorers erhoben worden sind (Götz, 2018) (vgl. Anhang: Ausschnitt Excel-Dokument: GEDYS IntraWare 8 Customising**Fehler! Verweisquelle konnte nicht gefunden werden.**).

5.3 Visualisierungsmöglichkeiten und Aufbau der Graphen

Es gibt verschiedene Möglichkeiten, SoNCs, SoNSCs und SoNBOs zu visualisieren. Auf IBM XPages kann JavaScript verwendet werden, daher wird eine Lösung auf Basis einer solchen Bibliothek verwendet. Nach der Recherche sind zwei Bibliotheken in die engere Auswahl gekommen: *Graph-Visualization*[1] und *vis.js*[2] (vgl. Abbildung 15). Graph-Visualization ist ein Git-Projekt (davidpiegza), welches einen dreidimensionalen Graphen mit WebGL[3] darstellt. Die Knoten- und Kantenelemente stammen aus der

[1] https://github.com/davidpiegza/Graph-Visualization

[2] http://visjs.org/docs/network/

[3] https://www.khronos.org/webgl/

Bibliothek three.js[4]. vis.js ist ebenfalls ein Open-Source Projekt, welches unter anderem eine Komponente beinhaltet, die zweidimensionale Netzwerke darstellen kann.

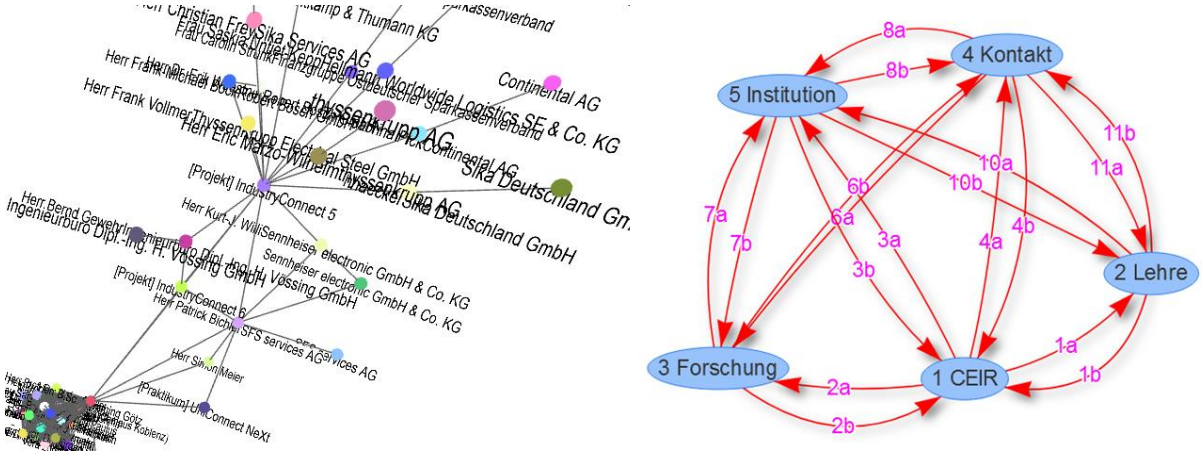


Abbildung 15: Beispiele für Visualisierungsmöglichkeiten, links: Graph-Visualization, rechts: vis.js (eigene Darstellung)

Das dreidimensionale Netzwerk bietet den Vorteil, dass mehr Raum zur Verfügung steht, Daten darzustellen. Außerdem gibt es weniger Überschneidungen von Kanten, sodass Beziehungen zwischen Objects besser nachzuvollziehen sind. Die Knoten sind anklickbar, sodass Funktionen wie zum Beispiel das Anzeigen von Informationen eines Objects leicht hinzuzufügen sind. Allerdings werden auch die Beschriftungen als Grafiken behandelt, sodass diese die Knoten überdecken können. Zudem werden die Schriftzüge je nach Perspektive in schlecht lesbaren Winkeln angezeigt. Die Bibliothek beinhaltet die Möglichkeit, die Gravitation der Objects untereinander und die Federkonstanten der Kanten anzupassen. Die Größe der angezeigten Elemente ist ebenfalls anpassbar.

Das zweidimensionale Netzwerk bietet bei einer großen Kantenmenge weniger Übersicht als das dreidimensionale Netzwerk, dafür aber deutlich mehr Konfigurationsmöglichkeiten. Ein Teil einer solchen Konfiguration ist in Abbildung 16 zu sehen. Die Knoten und Kanten können in ihrem Aussehen beliebig angepasst werden. Die Elemente können in ihren drei Zuständen *nicht ausgewählt*, *Maus befindet sich über dem Knoten* und *ausgewählt* unterschiedlich dargestellt werden. Beispielsweise können beim Bewegen der Maus über einen Knotens dessen Farbe geändert und alle verbundenen Kanten hervorgehoben werden. Es gibt mehrere vorimplementierte Gravitationsalgorithmen, die angepasst und zur Berechnung verwendet werden können. Die Bibliothek vis.js bringt eine Benutzerschnittstelle mit sich, mit der ein Netzwerk auf graphischer Ebene bearbeitet werden kann. Außerdem sind diverse Schnittstellen vorhanden, über die mit dem Netzwerk interagiert werden kann, wie zum Beispiel eine „on se-

[4] https://threejs.org/

lect“-Funktion, welche beim Klicken eines Elements aufgerufen wird, oder eine „add“-Funktion, die dem Netzwerk dynamisch Elemente hinzufügt.

```
var options = {
      edges:{
            color:{
                  color: 'grey'
            },
            font:{
                  color: '#ff00ff',
                  align: 'horizontal',
                  multi: 'true'
            },
            scaling:{
                  label: true
            },
            shadow: true,
            smooth: false
      },
      physics:{
            enabled: true,
            minVelocity: 0.75,

            barnesHut: {
                  gravitationalConstant: -10000,
                  centralGravity: 0.5,
                  springLength: 95,
                  springConstant: 0.04,
                  damping: 0.9,
                  avoidOverlap: 1
            },

            stabilization: {
                  enabled:true,
                  iterations:2000,
                  updateInterval:25,
                  fit: true
            }
      },
      interaction: {
            hideEdgesOnDrag: true,
            tooltipDelay: 200,
            selectConnectedEdges: false,
            hover: true,
            hoverConnectedEdges: true
      }
};
```

Abbildung 16: vis.js Konfiguration im Social Network of Business Objects (eigene Darstellung)

Eine Option war die Verwendung beider Bibliotheken. Der zweidimensionale Graph von vis.js sollte für das Social Network of Concepts und für das Social Network of Subconcepts verwendet werden, da diese nicht so groß werden und die Kantenzahl für die Übersicht nicht problematisch wird. Das Social Network of Business Objects sollte als dreidimensionales Netzwerk dargestellt werden, weil dieser Graph aus deutlich mehr Elementen besteht, als die anderen beiden Netzwerke.

Diese Option hat jedoch Nachteile, sodass die Visualisierung ausschließlich mit vis.js umgesetzt ist. Zum einen ist es wesentlich weniger programmiertechnischer Aufwand, wenn für alle drei Graphen die gleichen Schnittstellen verwendet werden. Es wäre für die Graph-Visualization-Bibliothek notwendig gewesen, einige Schnittstellen selber zu programmieren oder auf Funktionen, die aktuell implementiert sind, zu verzichten. Auch die Struktur der Daten, mit denen die Graphen arbeiten, ist leicht verschieden, sodass die Verwendung einer einheitlichen Lösung dieses Problem umgeht. Zum anderen sind die Optionen, mit der die Graphen von vis.js angepasst und konfiguriert werden, für die Qualität der Darstellung relevant. Beispielsweise trägt die Möglichkeit, mit der Maus Elemente hervorzuheben zu können, sehr zur Übersicht des Graphen bei. Auch ist die Physik-Engine besser anzupassen. Durch diese Konfigurationsmöglichkeiten ist vis.js insgesamt besser für die Visualisierung eines SoNBO geeignet. Daher sind alle drei Netzwerke grafisch mit vis.js umgesetzt.

Im Social Network of Concepts werden die Social Concepts als Knoten dargestellt, deren Label den Namen der Social Concepts entsprechen. Die Verbindungen werden durch ungerichtete und unbeschriftete Kanten dargestellt, da eine solche Verbindung immer bilateral ist und keinen Bezeichner benötigt. Es ist möglich, dass eine Kante ein Social Concept mit sich selbst verbindet. Im Social Network of Subconcepts entsprechen die Knoten den Social Subconcepts. Deren Zuordnung zu einem Social Concept wird zusätzlich zu dem Namen als Beschriftung des Knotens angezeigt. Die Social Subconcepts werden eingefärbt, sobald sie konfiguriert sind. Die zugehörigen Elemente im SoNBO erhalten die zugeordnete Farbe. Die verbindenden Kanten sind ebenfalls bilateral und unbeschriftet. Konfigurierte Verbindungen werden farblich markiert. Im Social Network of Business Objects ist jeder Knoten ein Business Object, welches durch die farbliche Kennzeichnung einem Social Subconcept zugeordnet ist. Wie der Anwender die Graphen erzeugt wird ab Kapitel 5.5 erklärt.

Anzumerken ist, dass die Anwendung bei der Darstellung von sehr vielen Elementen nicht mehr flüssig läuft – mit beiden Bibliotheken. Das Netzwerk stabilisiert sich nur langsam von alleine, da viele Gravitationskräfte wirken und der benötigte Rechenaufwand im Vergleich zu wenigen Elementen sehr viel größer ist. Da dieser Rechenaufwand clientseitig betrieben wird, ist die Performance der SoNBO-Graph-App auch vom verwendeten Endgerät abhängig.

5.4 Anwendungsarchitektur

Die Graph-App ist der von Haft & Olleck (2007) beschriebenen Drei-Schichten-Architektur zuzuordnen. Sie verwendet in ihren Architekturschichten unterschiedliche Programmiersprachen. Auf der Datenhaltungsschicht, die für den Datenbankzugriff zuständig ist, werden Java-Klassen benutzt, die die Inhalte der Datenbanken abholen. Auf der Applikationsschicht, die die abgeholten beziehungsweise auf der Präsentationsschicht vom Anwender eingegebenen Daten verarbeitet, kommt server-side JavaScript (SSJS) zum Einsatz. Auf der Präsentationsschicht und somit der Benutzerschnittstelle wird client-side JavaScript (CSJS) benutzt, um mit dem Graphen zu interagieren.

Vor und bei der Entwicklung der SoNBO-Graph-App wurde überlegt, welche Daten wie gespeichert und verarbeitet werden können. Wie bereits in Kapitel 3.4 erklärt, sollen nach dem SoNBO-Konzept keine Daten redundant abgelegt werden. Daher werden notwendige Daten mit Ausnahme der Datenbankanbindungen lediglich als (temporäre) Sitzungsvariablen gespeichert. Die Inhalte dieser Variablen sind JSON-Strings. Diese müssen von allen Anwendungsschichten verarbeitet werden können. Das Datenformat ist hierbei kein Problem, da JSON in allen verwendeten Programmiersprachen verwendet werden kann. Die zentrale Herausforderung hierbei ist der Datenaustausch zwischen den Schichten. Im Folgenden wird die Umsetzung dieses Datenaustauschs erklärt.

Im IBM Domino Designer besteht die Möglichkeit, mit *Managed Beans* (Teil des Framework-Standards JavaServer Faces, auf dem XPages basieren, vgl. Kapitel 2.2) zu arbeiten. Eine Managed Bean erhält als Attribute einen Namen, eine Java-Klasse und eine Gültigkeitsdauer (vgl. Abbildung 17). Damit ist es möglich, im SSJS Code auf eine Java-Klasse zuzugreifen und deren Methoden aufzurufen. Im folgenden Beispiel hat die Bean den Namen „soNBOManager“. Es wird die Funktion „doQuery“ der Java-Klasse „uniko.iwvi.fgbas.bameier.services.SoNBOManager“ ausgeführt. Gültig ist die Managed Bean bis zum Ablauf der http-Sitzung, danach werden sie und damit ihre Inhalte/Objekte gelöscht.

```
<!--sonbo-graph-app.nsf/WebContent/WEB-INF/faces-config.xml-->
<?xml version="1.0" encoding="UTF-8"?>
<faces-config>
  <managed-bean>
    <managed-bean-name>soNBOManager</managed-bean-name>
    <managed-bean-class>
      uniko.iwvi.fgbas.bameier.services.SoNBOManager
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>

//Aufruf einer Java-Funktion in SSJS
soNBOManager.doQuery(Database, View, Feldname, Key);
```

Abbildung 17: Managed Bean für den Datenbankzugriff (eigene Darstellung)

Eine Möglichkeit, die Kommunikation zwischen Applikationsschicht und Präsentationsschicht umzusetzen, ist das Verwenden des Kontrollelements *Remote Service* (jsonRpcService) aus der *XPages Extension Library*[5]. In diesem Element kann man SSJS-Code ausführen, wobei der Aufruf dieses Codes aus einer SSJS-Bibliothek erfolgt. In Abbildung 18 ist der jsonRpcService abgebildet, der sich auf der XPage befindet, auf der das Social Network of Subconcepts konfiguriert und angezeigt wird. In diesem Fall beinhaltet der Service drei verschiedene Funktionen. Im Attribut „skript“ steht jeweils der Code, welcher ausgeführt werden soll. In „name“ wird der Funktionskopf definiert, mit dem das Skript im CSJS ausgeführt wird. Wenn im aufzurufenden Skript Parameter übergeben werden sollen, müssen diese zusätzlich angegeben werden, wie im Beispiel bei der Remotefunktion „setGraphData“. Ein Funktions-

[5] https://extlib.openntf.org/

aufruf durch ein jsonRpcService wird parallel zum CSJS durchgeführt. Das heißt, dass die Anwendung nicht auf den Rückgabewert des Kontrollelements wartet und sofort den nachfolgenden Code ausführt. Diese asynchrone Ausführung ist das Standardverhalten, da bei einem synchronen Funktionsaufruf alle anderen Browseraktivitäten eingestellt werden und den Eindruck erwecken können, dass dieser sich aufgehängt hat. Wenn die Rückgabe jedoch im folgenden Code benötigt wird, muss die Ausführung der Remotefunktionen synchronisiert werden.

```
<!—Remote Service Control des Social Network of Subconcepts
auf der XPage-->
<xe:jsonRpcService id="strukturgraphDataRpc"
serviceName="strukturgraphDataRpc">
      <xe:this.methods>
            <xe:remoteMethod name="setStrukturgraphData"
                  script="return setGraphData(nodes, edges);">
                  <xe:this.arguments>
                        <xe:remoteMethodArg name="nodes"
                              type="string">
                        </xe:remoteMethodArg>
                        <xe:remoteMethodArg name="edges"
                              type="string">
                        </xe:remoteMethodArg>
                  </xe:this.arguments>
            </xe:remoteMethod>
            <xe:remoteMethod
                  name="initStrukturgraphData"
                  script="return initGraphData();">
            </xe:remoteMethod>
            <xe:remoteMethod
                  script="return getAdjacentNodesListRPC();"
                  name="getAdjacentNodesList">
            </xe:remoteMethod>
      </xe:this.methods>
</xe:jsonRpcService>

//Dojo code zum synchronen Ausführen des CSJS und SSJS
if( ! dojo._xhr )
    dojo._xhr = dojo.xhr;
dojo.xhr = function ( args, ioArgs, addArgs ){
    ioArgs["sync"] = true;
    return dojo._xhr( args, ioArgs, addArgs );
}

strukturgraphDataRpc.initStrukturgraphData()
      .addCallback(function (response){
      // -=> RESET DOJO XHR CALL BACK FIRST!
      dojo.xhr = dojo._xhr;
      delete( dojo._xhr );
      // -=> DO THE OTHER STUFF
      var noVisData = JSON.parse(response);
      nodes = new vis.DataSet(noVisData['nodes']);
      edges = new vis.DataSet(noVisData['edges']);
});
```

Abbildung 18: Remote Service für die Kommunikation zwischen Client und Anwendung (eigene Darstellung)

In der SoNBO-Graph-App ist das für jeden Remote Service der Fall, da die Rückgabe der Anwendungsschicht in der Präsentationsschicht von vis.js weiterverarbeitet wird. In dem Beispiel wird nach der Synchronisierung die Funktion zur Erzeugung des Social Network of Subconcepts aufgerufen. Sobald der Rückgabewert erhalten ist, wird die Synchronisierung wieder aufgehoben und der nachfolgende Code kann mit den erhaltenen Daten arbeiten.

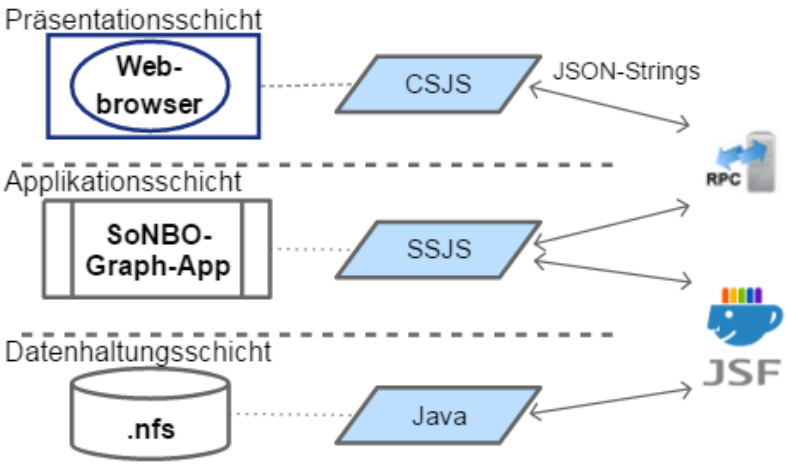


Abbildung 19: Kommunikation der Anwendungsebenen (eigene Darstellung)

Somit ist eine Kommunikation aller drei Anwendungsebenen untereinander möglich. Eine vom Anwender vorgenommene Konfiguration auf der Weboberfläche wird über den Remote Service als JSON-String an die Anwendung weitergegeben. Die Graph-App führt über die Managed Bean entsprechende Datenbankabfragen aus und erhält auf gleichem Weg die Daten. Die von der Datenhaltungsschicht erhaltenen JSON-Strings werden nach der Bearbeitung dem Webbrowser über den Remote Service zur Verfügung gestellt. Abbildung 19 visualisiert die Umsetzung dieses Datenaustausches.

5.5 Benutzeroberfläche

Im Folgenden wird die Benutzerschnittstelle vorgestellt und die Funktionsweise der SoNBO-Graph-App erklärt. Dabei ist die Anwendung auf drei verschiedene XPages verteilt, auf denen jeweils ein Netzwerk konfiguriert und dargestellt wird. Während der Bedienung kann frei zwischen den Netzwerken gewechselt werden; mit zwei Ausnahmen sind jederzeit alle Netzwerke einsehbar und Konfigurationen möglich. Die erste Ausnahme tritt auf, wenn noch kein Social Concept erstellt ist. Dann ist es nicht möglich, ein Social Subconcept korrekt zu erstellen, da dazu eine Zuweisung zu einem Social Concept notwendig ist. Die zweite Ausnahme ist, dass kein Social Network of Business Objects ohne mindestens ein fertig konfiguriertes Social Subconcept dargestellt wird.

Auf den drei XPages wird im unteren linken Bereich der jeweilige Graph angezeigt (A). Wenn die Maus über diesen Bereich bewegt wird, kann mit dem Mausrad gezoomt werden. Einzelne Objects sowie der gesamte Graph sind per Drag & Drop verschiebbar. Beim Verschieben von Elementen werden die

Kanten ausgeblendet, um die Performance der Anwendung zu verbessern. Oberhalb der Graphen gibt es Bedienelemente, mit denen der Anwender die Visualisierung anpassen kann (B). Der Graph kann manuell bis zur nächsten Änderung stabilisiert werden, damit sich die Elemente nicht mehr bewegen, außerdem kann die initiale Stabilisierung, während der der Graph nicht sichtbar ist, angehalten werden. Alternativ ist die Physik-Engine bis zum nächsten Neuladen der Seite pausierbar. Diese Funktionen sind bei Graphen mit vielen Kanten erforderlich, da diese sich unter Umständen sehr langsam oder sogar nie selbstständig stabilisieren können. Weiterhin kann der Anwender einen angeklickten Knoten fokussieren oder das gesamte Netzwerk anzeigen lassen. Eine weitere Gemeinsamkeit der drei XPages ist ein Feld, in dem alle Informationen zu einem ausgewählten Element angezeigt werden (C). In den übrigen Bereichen unterscheiden sich die Seiten.

5.5.1 Social Network of Concepts

Auf der ersten XPage wird ein Social Network of Concepts erstellt (vgl. Abbildung 20). Um ein SoNC zu erstellen stehen dem Nutzer zwei Eingabemasken zur Verfügung: Eine für die Social Concepts (1) und eine für die Kanten (2). Die erste Eingabemaske dient dazu, dem Graphen neue Social Concepts hinzuzufügen und bestehende Social Concepts zu bearbeiten oder zu entfernen. Beim Hinzufügen muss dem Knoten ein Bezeichner gegeben werden und auf den „Hinzufügen“- Button geklickt werden. Um ein Social Concept zu bearbeiten muss dieses angeklickt werden. Dann kann der Bezeichner angepasst werden. Per Klick auf den „Update“- Button wird die Änderung übernommen. Klickt der Anwender auf den „Entfernen“- Button, wird das aktuell ausgewählte Social Concept aus dem Social Network of Concepts entfernt. Die Verbindungen zwischen den Knoten werden analog bearbeitet. Einer Kante werden per Dropdown-Liste die beiden Social Concepts zugewiesen, welche sie verbinden soll. Aus Gründen der Übersicht wird in allen Netzwerken auf eine Beschriftung der Kanten verzichtet. Außerdem kann der Anwender erstellte Social Networks of Concepts als JSON-Strings exportieren und importieren (3), sodass über diese Schnittstelle erstellte Graphen gesichert und wiederhergestellt werden können. Zu erwähnen ist, dass vis.js eine Benutzerschnittstelle mit sich bringt, die die oben beschriebenen Funktionen zur Manipulation des Graphen implementiert. Allerdings wären einige Anpassungen notwendig gewesen (vor allem für das SoNSC); von der Verwendung dieser Benutzerschnittstelle wurde daher abgesehen.

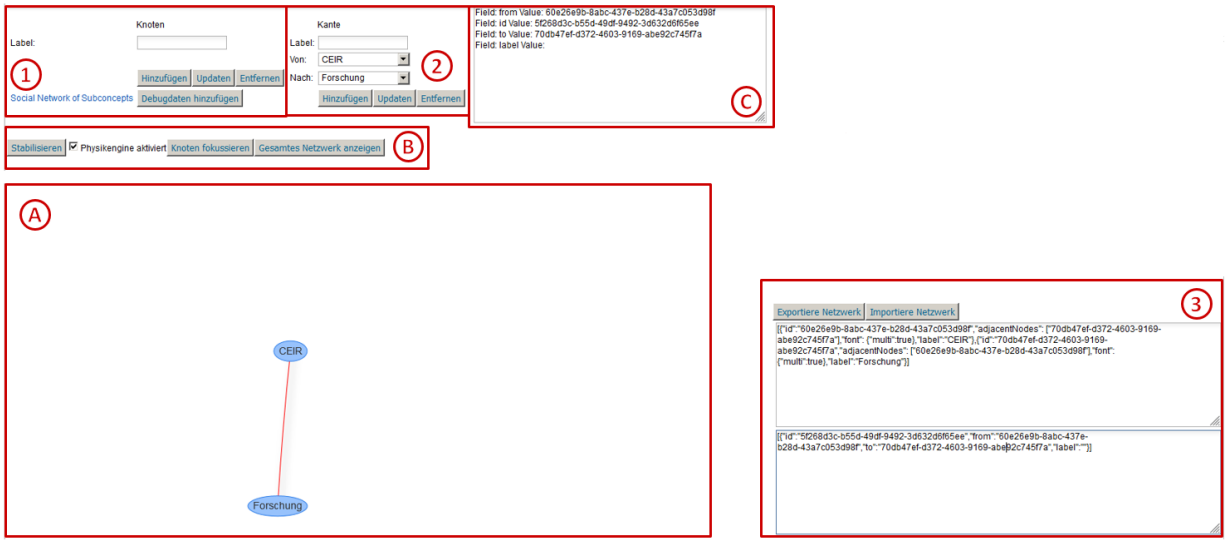


Abbildung 20: Benutzeroberfläche Social Network of Concepts (eigene Darstellung)

Vergleicht man den hier beschriebenen Konfigurationsvorgang mit dem im SoNBO-Explorer (vgl. Abbildung 8), fällt auf, dass im Explorer auf dieser Ebene keine Kanten definiert werden. Dies liegt daran, dass zum Zeitpunkt der Entwicklung des Explorers noch nicht zwischen Social Network of Concepts und Social Network of Subconcepts unterschieden wurde. Im Gegenzug fallen die Auswahl eines Bildes und die Auswahl der Anzeige in der Filterleiste in der SoNBO-Graph-App weg, da beide Konfigurationsmöglichkeiten nur für die Anzeige im Explorer benötigt werden, nicht aber für die Graph-App und damit ebenfalls nicht für die Erstellung des Netzwerks. Dies trifft auf alle Konfigurationen zu, die die Darstellung im SoNBO-Explorer betreffen.

5.5.2 Social Network of Subconcepts

Auf der zweiten XPage erstellt der Anwender ein Social Network of Subconcepts (vgl. Abbildung 21). Ein SoNSC wird auf der Basis eines Social Network of Concepts erstellt. Es gibt auch bei diesem Netzwerk zwei Eingabemasken (4), (5) für das Editieren von Elementen des Graphen. Die Grundfunktionen des Hinzufügens, Bearbeitens und Löschens verhalten sich analog zu den zuvor beschriebenen Funktionen. Beim Hinzufügen und Editieren von Social Subconcepts muss neben dem Bezeichner auch das zugehörige Social Concept angegeben werden. Falls ein Social Concept, auf dem ein Social Subconcept basiert, geändert oder gelöscht wird, wird dies in der Beschriftung des Social Subconcepts vermerkt. Per Klick auf den Button „Entferne vorgemerkte Knoten“ beziehungsweise „Aktualisiere vorgemerkte Knoten“ werden die entsprechenden Social Subconcepts angepasst. Dadurch, dass keine Social Supconcepts automatisch gelöscht werden, wenn der Anwender das zugehörige Social Concept löscht, sieht er, was er gerade verändert hat. Für die Kanten gibt es eine ähnliche Funktion, welche über den Button „Kanten erben“ durchgeführt wird. Dabei werden durch das SoNC definierte, aber im SoNSC nicht vorhandene Kanten eingezeichnet. Wird im SoNC eine Kante gelöscht, muss diese Änderung im

SoNSC entweder manuell durchgeführt werden, indem entsprechende Kanten gelöscht werden, oder ein Häckchen bei „Kanten überschreiben“ gesetzt werden. Dadurch werden alle Verbindungen gelöscht und nach den Regeln des Social Network of Concepts neu eingezeichnet. Allerdings gehen dabei die Konfigurationen aller Kanten verloren. Das manuelle Hinzufügen und Updaten von Kanten ist möglich, dadurch können die vom SoNC vorgegebenen Regeln jedoch verletzt werden.

Bis zu dieser Stelle funktioniert die SoNBO-Graph-App auch ohne Datenbankanbindung. Um ein Social Network of Business Objects zu erzeugen, müssen in einer weiteren Eingabemaske die Social Subconcepts (6) konfiguriert werden. Einem Social Subconcept wird zunächst eine Datenquelle zugeordnet, was in diesem Fall eine der beiden nsf-Datenbanken (*contacts.nsf* oder *georga.nsf*) ist. Danach wählt der Anwender die View aus, aus der die Daten ausgelesen werden sollen. Mit einem Klick auf „Abfrage durchführen“ werden die Namen aller Felder angezeigt, welche in der ausgewählten View gespeichert sind. Das sind die Felder, deren Werte den Social Business Objects zugeordnet werden, die zu dem ausgewählten Social Subconcept gehören. Die nächste Eingabe, die vorgenommen werden muss, ist die Definition, wann ein Business Object dem Subconcept zugehörig ist. Diese Zugehörigkeit ist als Schlüssel-Wertepaar (key value pair) definiert. Der Anwender trägt ein, in welchem der ausgegebenen Felder welcher Wert stehen muss. Beispielsweise ist ein Social Business Object genau dann ein „Professor“, wenn im Feld „Rolle“ der Wert „Professor“ eingetragen ist. Mit der Farbauswahl wird das Social Concept im Social Network of Subconcepts entsprechen eingefärbt, ebenso die Objects im Social Network of Business Objects. Dadurch sind diese Objects in einem großen Netzwerk leichter zu erkennen. Die letzte Angabe bei der Konfiguration ist das Feld, dessen Inhalt als Bezeichner des Business Objects angezeigt wird, zum Beispiel „fullName“. Mit einem Klick auf „Übernehmen“ werden die Angaben gespeichert. Falls die Konfiguration valide ist, werden jetzt bereits Social Business Objects im Social Network of Business Objects angezeigt.

Damit aus der Menge der Social Business Objects ein Netzwerk wird, werden die verbindenden Kanten in der nächsten Eingabemaske (7) konfiguriert. Zunächst wählt der Anwender die Verbindung aus, die er konfigurieren möchte. Jetzt gibt es zwei Möglichkeiten, die Verbindung zu definieren. Eine Möglichkeit ist das Anhaken des Kästchens „Immer verbinden“. Dadurch werden Social Business Objects, welche den immer verbundenen Social Subconcepts angehören, ebenfalls verbunden. Das kann der Fall sein, wenn beispielsweise jeder „Professor“ mit jedem anderen „Professor“ im Netzwerk verbunden ist. Hier gibt es also keine Bedingung, die zutreffen muss, damit zwei Business Objects in Beziehung zueinander stehen. Die zweite Möglickeit ist die Angabe einer Bedingung, wann Objects miteinander verbunden sein sollen. Dazu wählt der Benutzer die Richtung aus, in der er die Verbindung einrichten möchte. Da diese Verbindung, wie in Kapitel 5.3 beschrieben, bilateral ist, hat die Auswahl keinen Einfluss auf das Ergebnis. Danach gibt der Anwender die Bedingung an, indem er auswählt, welcher Inhalt eines Feldes des einen Business Objects (Quelle) in welchem Feld des anderen Business Objects (Ziel) stehen muss. Ein Beispiel: „Professoren“ sollen mit „Vorlesungen“ verbunden werden (vgl. Abbildung 5). Betrachtet wird nun die Richtung von „Professor“ nach „Vorlesung“. Die beiden Social Subconcepts sind dann verbunden, wenn der Professor als Dozent eingetragen ist. Daher lautet die Konfiguration:

Feldinhalt des Feldes „Name“ (des Professors) steht im Feld „Dozent“. Wenn die Definitionsrichtung andersherum ist, also von „Vorlesung“ nach „Professor“, lautet die Konfiguration: Feldinhalt des Feldes „Name“ (der Lehrveranstaltung) steht in dem Feld „Lehrveranstaltung“ – denn die Lehrveranstaltung ist auch dem Professor zugeordnet und dort eingetragen. Beide Konfigurationen führen zum gleichen Ergebnis. Erheblich hierbei ist, dass der Wert im Quellfeld, welches angegeben wird, kein Array ist, also nur einen einzelnen Wert enthält.

Der Anwender kann einer Kante beliebig viele Verbindungsbedingungen zuweisen. Hat das Social Subconcept „Lehrveranstaltung“ ein weiteres Attribut „Verantwortlicher“, so kann eine weitere Bedingung lauten: Feldinhalt des Feldes „Name“ (des Professors) steht im Feld „Verantwortlicher“. Falls mit dieser zusätzlichen Bedingung ein „Professor“ als „Dozent“ und ein anderer „Professor“ als „Verantwortlicher“ eingetragen ist, werden beide Social Business Objects im Social Network of Business Objects mit der Lehrveranstaltung verbunden. Auf zwei BOs können mehrere Verbindungen gleichzeitig zutreffen, wenn zum Beispiel ein „Professor“ als „Dozent“ und zusätzlich als „Verantwortlicher“ einer „Lehrveranstaltung“ zugewiesen ist. In dem Fall wird trotzdem nur eine Kante zwischen den beiden Business Objects eingezeichnet, da eine Kante lediglich wiedergibt, ob zwei BOs verbunden sind, nicht wie. Diese Information steht in den BOs selbst. Möchte der Anwender eine oder mehrere Bedingungen löschen, kann er dies per Auswahlfeld und Klick auf den „Entfernen“-Button tun. Auch auf dieser XPage findet sich ein Bereich, in dem der gezeichnete Graph exportiert bzw. ein zuvor gesicherter Graph importiert werden kann (8). Nach jedem Konfigurationsschritt kann das Social Network of Business Objects geöffnet werden, sodass das Zwischenergebnis sichtbar ist.

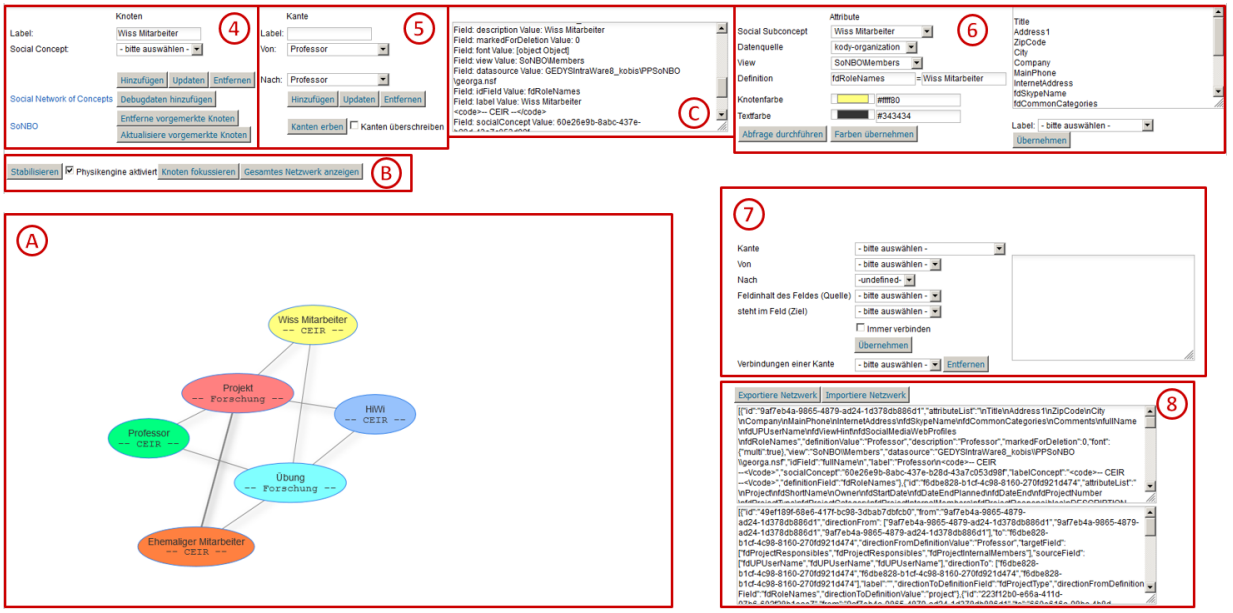


Abbildung 21: Benutzeroberfläche Social Network of Subconcepts (eigene Darstellung)

Betrachtet man die hier beschriebene Konfiguration des SoNSC und die im SoNBO-Explorer, so findet man einige Elemente wieder. Die Konfiguration der Knotentypen im Explorer (vgl. Abbildung 9) entspricht inhaltlich der Erstellung von Social Subconcepts (4). Die Konfiguration der Attribute im SoNBO-Explorer (vgl. Abbildung 10) findet man ebenfalls in der Graph-App wieder (6). Das Formular für die Datenquellen und die Abfragen (vgl. Abbildung 11) sowie das für die Zuweisung (vgl. Abbildung 12) sind in (6) und (7) aufgeteilt.

Ein Unterschied zwischen den beiden Anwendungen ist, dass im SoNBO-Explorer alle Konfigurationsmöglichkeiten voneinander getrennt sind. Beispielsweise werden die Abfragen separat erstellt und später zugewiesen – im Gegensatz zur Graph-App, in der die Abfragen direkt in den Social Subconcepts und den Kanten eingetragen werden. Gleiches gilt für die Attribute, die den SSCs hinzugefügt werden. Der Anwender hat in der SoNBO-Graph-App auf einer Seite auf alle Konfigurationen Zugriff, die er im entsprechenden Netzwerk vornehmen kann. Außerdem ist das Netzwerk immer im aktuellen Zustand abgebildet. Durch diese Übersicht soll dem Anwender die Interaktion mit den Daten erheblich erleichtert werden.

5.5.3 Social Network of Business Objects

Auf der dritten XPage befindet sich das Social Network of Business Objects (vgl. Abbildung 22). Das SoNBO wird auf Basis der zuvor vorgenommenen Konfiguration aus den angebundenen Datenbanken erzeugt. Daher sind auf dieser Ebene keine weiteren Konfigurationen am Netzwerk möglich. Ein SoNBO kann aus sehr vielen Elementen bestehen, daher gibt es zwei zusätzliche Funktionen, um die Arbeit mit den Business Objects zu erleichtern (9). Zum einen gibt es die Möglichkeit, aus einer Liste ein Social Business Object auszuwählen und dieses zu fokussieren. Somit muss das BO nicht im Graph gesucht werden. Zum anderen können die benachbarten Objects als Liste ausgegeben werden. Die weiteren Interaktionsmöglichkeiten mit dem Graph verhalten sich analog zu denen der beiden anderen Graphen.

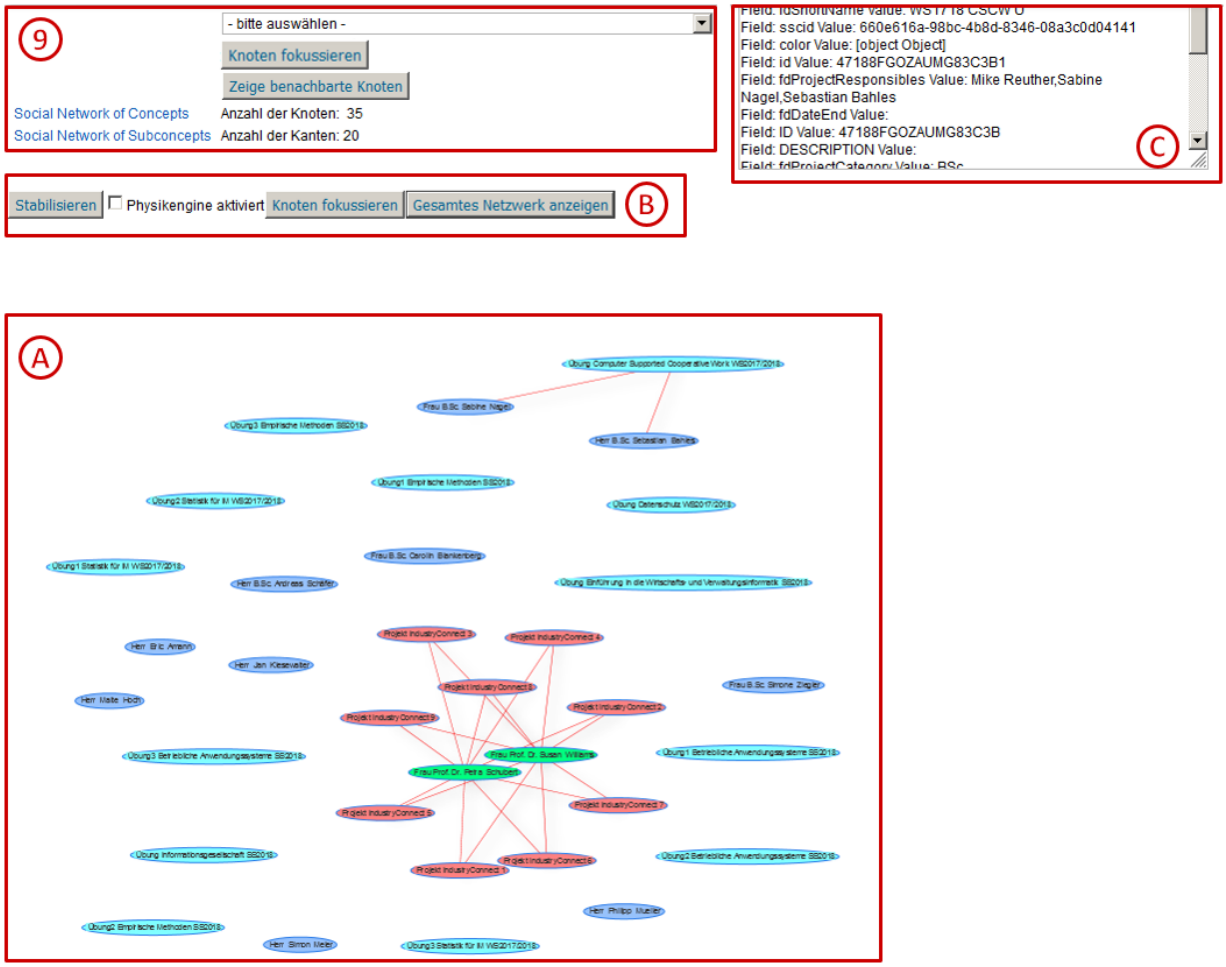


Abbildung 22: Benutzeroberfläche Social Network of Business Objects (eigene Darstellung)

Da dieses Netzwerk vollständig aus der zuvor eingegebenen Konfiguration erzeugt wird, folgt eine Erklärung aus technischer Sicht. Der entsprechende Code befindet sich im Anhang (vgl. Anhang: SoNBO: buildInstancegraph-Funktion). Die Funktion zur Generierung des Graphen ist in drei Schritte aufgeteilt:

1. Generierung der Business Objects
2. Generierung der Verbindungen zwischen den Business Objects
3. Löschen von überflüssigen Kanten

Im ersten Schritt wird für ein Social Subconcept eine Datenbankabfrage mit den hinterlegten Parametern *Datenquelle* und *View* durchgeführt. Nicht vollständig konfigurierte SSCs werden übersprungen. Nach einer erfolgreichen Abfrage erhält man eine Liste aller IDs, welche zu Objects gehören, die die angegebenen Parameter erfüllen. Über diese ID-Liste wird im nächsten Schritt iteriert, sodass für jede ID alle Daten eines Business Objects als JSON-Strings ausgelesen werden. Für den Fall, dass aufgrund von fehlerhaften Einträgen im CRM-System Dubletten gespeichert sind, bekommen Datensätze mit einer bereits existierenden ID eine neue ID zugewiesen. Danach werden die JSON-Strings mit Daten ergänzt, welche für die Erzeugung des Graphen notwendig sind, nämlich die eventuell neu generierte ID,

ein Label, das zugehörige Social Subconcept und die Farben des SSC, in welchen auch das Business Object angezeigt wird. Nun wird aus dem String ein Java Objekt generiert. An dieser Stelle werden für spezielle JSON-Strings Fehler ausgegeben, da die Generierung fehlschlägt. Der Grund hierfür ist nicht bekannt. Da noch keine Fehlerbehandlung implementiert ist, werden diese Strings übersprungen und nicht weiter beachtet. Schließlich wird noch die Definition/Bedingung überprüft, ob das aktuelle Business Object zum ausgewählten Social Subconcept gehört. Wenn in einem Feld eines BOs mehrere Werte stehen, zum Beispiel mehrere zugeordnete Lehrveranstaltungen, werden in der View mehrere Einträge mit jeweils einem dieser Werte gespeichert. Dieser Fall wird bei der Überprüfung gesondert behandelt, indem die Einträge vor der Überprüfung aggregiert werden. Wenn das Ergebnis der Überprüfung positiv ist, wird das Business Object in die Liste der zu erstellenden Knoten aufgenommen. Dieser gesamte Prozess wird für jedes Social Subconcept durchgeführt. Dann ist der erste Schritt abgeschlossen.

Im zweiten Schritt wird über alle Kanten iteriert, welche im Social Network of Subconceps vorhanden sind, wobei nicht definierte Kanten übersprungen werden. Bei definierten Kanten wird unterschieden, ob „Immer verbinden“ ausgewählt wurde oder nicht. Falls dies der Fall ist, muss keine Definition geprüft werden. Nun wird über alle im Schritt zuvor gelistete Business Objects iteriert. Wenn ein BO zu einem Social Subconcept gehört, welches verbunden werden soll, werden nochmals alle BOs durchsucht, um alle Verbindungspartner zu finden. Für alle BO-Paare wird eine Kante in die Liste aller im Social Network of Business Objects zu erstellenden Kanten eingetragen. Ist „Immer verbinden“ nicht ausgewählt wird auch über die BO-Liste iteriert, jedoch wird in diesem Fall überprüft, ob in einem Business Object das Key-Value Paar auftaucht, welches in der Kante definiert ist, oder anders gesagt, ob ein bestimmter Wert an einer bestimmten Stelle steht. Für alle BOs, bei denen das der Fall ist, wird die Liste aller BOs erneut analog nach dem zweiten Key-Value Paar durchsucht, welches in der Kante steht. Für alle auf diese Weise gefundenen Paare wird ebenfalls eine neue Kante in die Liste aller zu erstellenden Kanten eingetragen. Damit ist der zweite Schritt abgeschlossen.

Im dritten Schritt wird die Liste aller zu erstellenden Kanten durchlaufen. Dabei ist eine Kante überflüssig und wird aus der Liste gelöscht, falls

- Der Startpunkt der Kante gleich dem Endpunkt der Kante ist,
- eine Kante doppelt vorkommt, oder
- es eine andere Kante gibt, bei der Start- und Zielpunkt gleich, aber vertauscht sind.

Ein Business Object kann nicht mit sich selbst verbunden sein, da es dann sein eigener Nachbar wäre. Somit würde man fälschlicherweise das Vorkommen eines zweiten, gleichen Business Objects implizieren. Doppelte Kanten können entstehen, wenn einer Verbindung im Social Network of Subconcepts mehrere Bedingungen zugewiesen sind und auf zwei Business Objects im SoNBO gleichzeitig mehrere Bedingungen zutreffen. Da der Anwender hierdurch keine zusätzlichen Informationen erhält, werden doppelte Kanten gelöscht (vgl. Kapitel 5.5.2). Da Verbindungen zwischen BOs bilateral sind, ist eine

Kante mit vertauschten Start- und Zielpunkten wie eine doppelte Kante zu betrachten. Somit wird auch diese Kante aus der Liste entfernt.

Mit Abschluss des dritten Schrittes ist das Social Netwok of Business Objects erstellt und liegt als Java Objekt vor. Dieses Objekt wird an die Programmierschnittstelle von vis.js weitergegeben, sodass der Graph entsprechend generiert wird. Der Prototyp gibt nach der Eingabe des Netzwerks der Forschungsgruppe einen entsprechenden Graphen aus und funktioniert damit.

6 Zusammenfassung und Ausblick

Im letzten Kapitel geht es um die Zusammenfassung der Ergebnisse der Thesis, und Möglichkeiten zur Weiterentwicklung des Prototyps bzw. auf dem Gebiet Social Network of Business Objects.

6.1 Ergebnisse

Ziel dieser Bachelorthesis war die Entwicklung eines lauffähigen Prototyps, mit dem Daten aus zwei verschiedenen CRM-Datenbanken aggregiert werden können. Diese Daten sollten zusammenhängend als Netzwerk dargestellt werden. Die Konfiguration dieser Zusammenhänge sollte ebenfalls auf graphischer Ebene geschehen. Der Anwender, welcher für das Aggregieren der Daten zuständig ist, sollte mit Hilfe dieses Prototyps unterstützt werden, indem er eine bessere Übersicht und einen anderen Blickwinkel auf die Daten erhält. Es erfolgte eine Einarbeitung in den bestehenden Prototyp SoNBO-Explorer (vgl. Kapitel 3.5), um die Umsetzung des SoNBO-Konzepts zu verstehen. Nach der Einarbeitung wurden die Anforderungen an die SoNBO-Graph-App erhoben (vgl. Kapitel 4). Nach der Konzeptionierung der visuellen Darstellung (vgl. Kapitel 5.3) hat die Entwicklung des Prototyps stattgefunden. Das Ergebnis der Entwicklung ist die SoNBO-Graph-App, mit der nach dem SoNBO-Konzept Daten je nach Konfiguration aggregiert dargestellt werden.

Tabelle 2: Ergebnisse der Forschungsfragen

Forschungsfrage	**Ergebnis**
1. Wie ist der aktuelle Stand, auf dem der Prototyp aufbaut?	Einarbeitung in den SoNBO-Explorer (Kapitel 3.5)
2. Inwiefern erweitert die Anwendung den aktuellen Stand?	Umgesetzte Anforderungen (Kapitel 4), Einordnung in die Systemlandschaft (Kapitel 5.1)
3. Wie kann die Visualisierung sinnvoll umgesetzt werden?	Visualisierung mit vis.js (Kapitel 5.3)
4. Wie kann ein Anwender die Anwendung konfigurieren?	Benutzeroberfläche (Kapitel 5.5)
5. Wie ist der Prototyp bezüglich der Ziele zu bewerten?	Ergebnisse (Kapitel 6.1)
6. Welche Möglichkeiten zur Weiterentwicklung gibt es?	Mögliche Weiterentwicklungen (Kapitel 6.2)

Das Ergebnis der Evaluation mit den Betreuern anhand der erfüllten Anforderungen und der Eingabe des Social Network of Concepts, respektive Social Network of Subconcepts der Forschungsgruppe Betriebliche Anwendungssysteme, lässt sich folgendermaßen zusammenfassen:

Das Social Network of Business Objects, welches der Anwender in der SoNBO-Graph-App anhand seiner Konfiguration erstellt hat, ist dasselbe, durch das er im SoNBO-Explorer navigiert, wenn er dort dieselbe Konfiguration vornimmt. So unterstützt die Graph-App den Konfigurationsprozess: Der Anwender erstellt ein Social Network of Concepts und ein Social Network of Subconcepts. Damit erhält er ein entspechendes Social Network of Business Objects. Ändert er die Konfiguration im SoNC und/oder im SoNSC, sieht er sofort die Änderung im SoNBO. Dabei geht die Konfiguration schneller, ist übersichtlicher und flexibler als im SoNBO-Explorer. Außerdem entsteht durch die Graph-App ein Überblick über das gesamte Netzwerk. Damit erfüllt die SoNBO-Graph-App das anfangs erwähnte Ziel (vgl. Kapitel 1.1) als Tool zur visuellen Unterstützung des SoNBO-Explorers.

Mit dem Erfüllen der Anforderungen, der Beantwortung der Forschungsfragen und somit dem Erreichen des Ziels ist die Arbeit erfolgreich abgeschlossen. Zuletzt folgen Vorschläge für mögliche Weiterentwicklungen auf dem Gebiet SoNBO.

6.2 Mögliche Weiterentwicklung

Die SoNBO-Graph-App ist funktionsfähig, dennoch ist die Anwendung ein Prototyp und eine potentielle Weiterentwicklung wird zusätzlichen Mehrwert auf dem Forschungsgebiet SoNBO bieten. Nachfolgend werden daher einige solcher Möglichkeiten aufgeführt.

Der Prototyp ist aktuell an zwei IBM Notes CRM-Datenbanken angeschlossen. Das Anschließen der Anwendung an andere IBM Notes Datenbanken ist daher möglich, sofern entsprechende Programmierschnittstellen in den Datenbanken geschaffen werden. Die SoNBO-Graph-App verwendet die gleichen Java-Methoden für den Datenbankzugriff wie der SoNBO-Explorer, und auf deren Basis ist bereits eine SQL-Datenbank eines ERP-Systems angebunden worden (Riedle, 2018). Damit ist die Integration von Informationen aus anderen Enterprise-Systemen auch für die Graph-App möglich. Im Fall einer Weiterentwicklung des Prototyps kann eine solche Integration umgesetzt werden.

In dieser Thesis wird zusätzlich zur Konfiguration der SoNBO-Graph-App die des SoNBO-Explorers beschrieben. Auf dieser Basis kann die Graph-App weiterentwickelt werden, indem die ursprünglich gestellte Anforderung des Exports der Konfiguration in den SoNBO-Explorer (vgl. Kapitel 4) in einem separaten Projekt umgesetzt wird.

Literaturverzeichnis

Amberg, M., & Schumacher, J. (2002). CRM-Systeme und Basistechnologien. In M. Meyer (Hrsg.), *CRM-Systeme mit EAI: Konzeption, Implementierung und Evaluation* (S. 21–59). https://doi.org/10.1007/978-3-663-05775-8_2

Auer, S., Pietzsch, R., & Unbehauen, J. (2014). Datenintegration im Unternehmen mit Linked Enterprise Data. In T. Pellegrini, H. Sack, & S. Auer (Hrsg.), *Linked Enterprise Data.* (S. 85–101). Springer, Berlin, Heidelberg.

Bondy, J. A., & Murty, U. S. R. (1976). *Graph theory with applications* (5. Aufl.). New York, USA: North Holland.

Bray, T. (2017). *The javascript object notation (json) data interchange format* (Nr. RFC 8259).

Chen, I. J., & Popovich, K. (2003). Understanding customer relationship management (CRM) People, process and technology. *Business Process Management Journal*, *9*(5), 672–688. https://doi.org/10.1108/14637150310496758

Donnelly, M., Wallace, M., & McGuckin, T. (2017). *Mastering XPages: A Step-by-Step Guide to XPages Application Development and the XSP Language* (2. Aufl.). IBM Press.

Easley, D., & Kleinberg, J. (2010). *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge: Cambridge University Press.

Galkin, M., Auer, S., & Scerri, S. (2016). Enterprise Knowledge Graphs: A Backbone of Linked Enterprise Data. *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 497–502. https://doi.org/10.1109/WI.2016.0083

Gebel-Sauer, B., & Schubert, P. (2019, Februar). *Entwicklung einer Definition für Social Business Objects (SBO) zur Modellierung von Unternehmensinformationen*. Gehalten auf der Multikonferenz Wirtschaftsinformatik, Siegen.

Gewehr, B., Gebel-Sauer, B., & Schubert, P. (2017). Social Network of Business Objects (SoNBO): An Innovative Concept for Information Integration in Enterprise Systems. *Procedia Computer Science*, *121*, 904–912. https://doi.org/10.1016/j.procs.2017.11.117

Götz, F. (2018). *Development of a prototype for integrating multiple software systems with a user interface for graph-based navigation* (Masterthesis). Universität Koblenz-Landau, Koblenz.

Götz, F., & Gebel-Sauer, B. (2018). *Vom CRM-System ins Social Network of Business Objects (SoNBO): Entwicklung eines Prototyps für eine innovative Informationsintegration*. 1995–2006. Lüneburg.

Haft, M., & Olleck, B. (2007). Komponentenbasierte Client-Architektur. *Informatik-Spektrum*, *30*(3), 143–158. https://doi.org/10.1007/s00287-007-0153-9

Håkansson, H., & Snehota, I. (Hrsg.). (1995). *Developing relationships in business networks*. London, New York: Routledge.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105.

Hippner, H., & Wilde, K. D. (2001). CRM — Ein Überblick. In S. Helmke & W. Dangelmaier (Hrsg.), *Effektives Customer Relationship Management: Instrumente — Einführungskonzepte — Organisation* (S. 3–37). https://doi.org/10.1007/978-3-322-82348-9_1

Kharlamov, E., Jiménez-Ruiz, E., Zheleznyakov, D., Bilidas, D., Giese, M., Haase, P., … Waaler, A. (2013). Optique: Towards OBDA Systems for Industry. In P. Cimiano, M. Fernández, V. Lopez, S. Schlobach, & J. Völker (Hrsg.), *The Semantic Web: ESWC 2013 Satellite Events* (S. 125–140). Berlin, Heidelberg: Springer.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251–266. https://doi.org/10.1016/0167-9236(94)00041-2

Markus, M. L., & Tanis, C. (2000). The Enterprise System Experience - From Adoption to Success. In *Framing the Domains of IT Management: Projecting the Future Through the Past* (Bd. 173, S. 173–207). Pinnaflex Educational Ressources Inc.

Martin, R. C. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ: Prentice Hall PTR.

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2006). The design science research process: a model for producing and presenting information systems research. *Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006)*, *24*, 83–106.

Probst, G., Raub, S., & Romhardt, K. (2010). *Wissen managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen* (6. Aufl.). Wiesbaden: Gabler.

Riedle, M. (2018). *Entwicklung einer Applikation auf Basis eines Knowledge Graphen für Objekte in Navision und Realisierung eines Integrationsinterface mit dem SoNBO-Ansatz* (Masterthesis). Universität Koblenz-Landau, Koblenz.

Tarjan, R. (1972). Depth-First Search and Linear Graph Algorithms. *SIAM journal on computing*, *1*(2), 146–160.

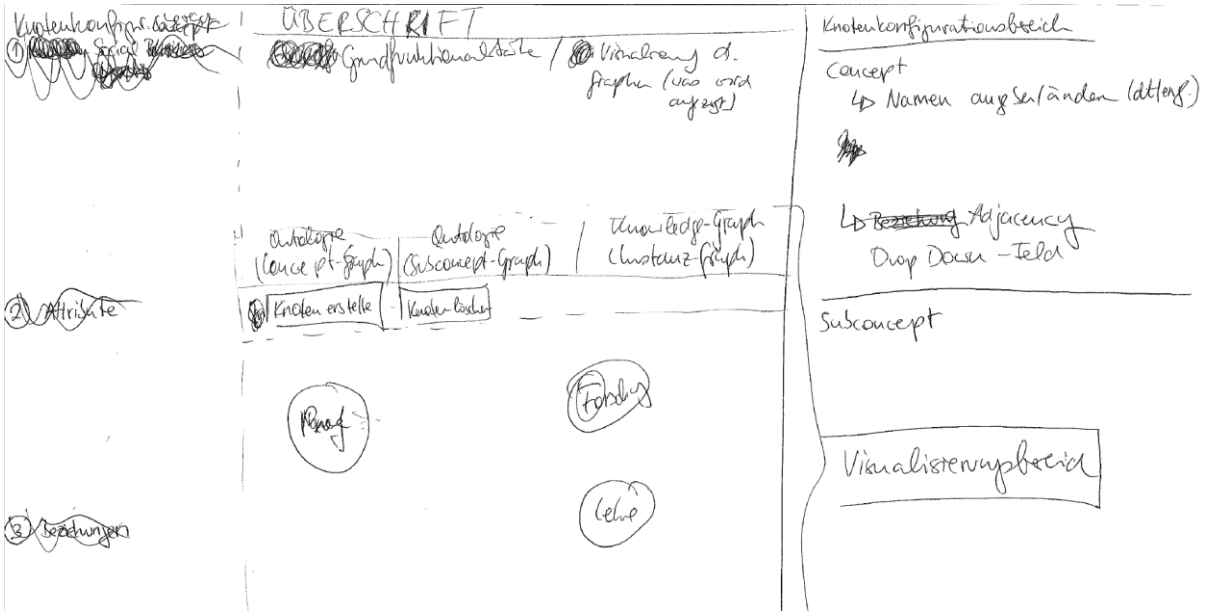
Torggler, M. (2008). The Functionality and Usage of CRM Systems. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, *2*(5), 771–779.

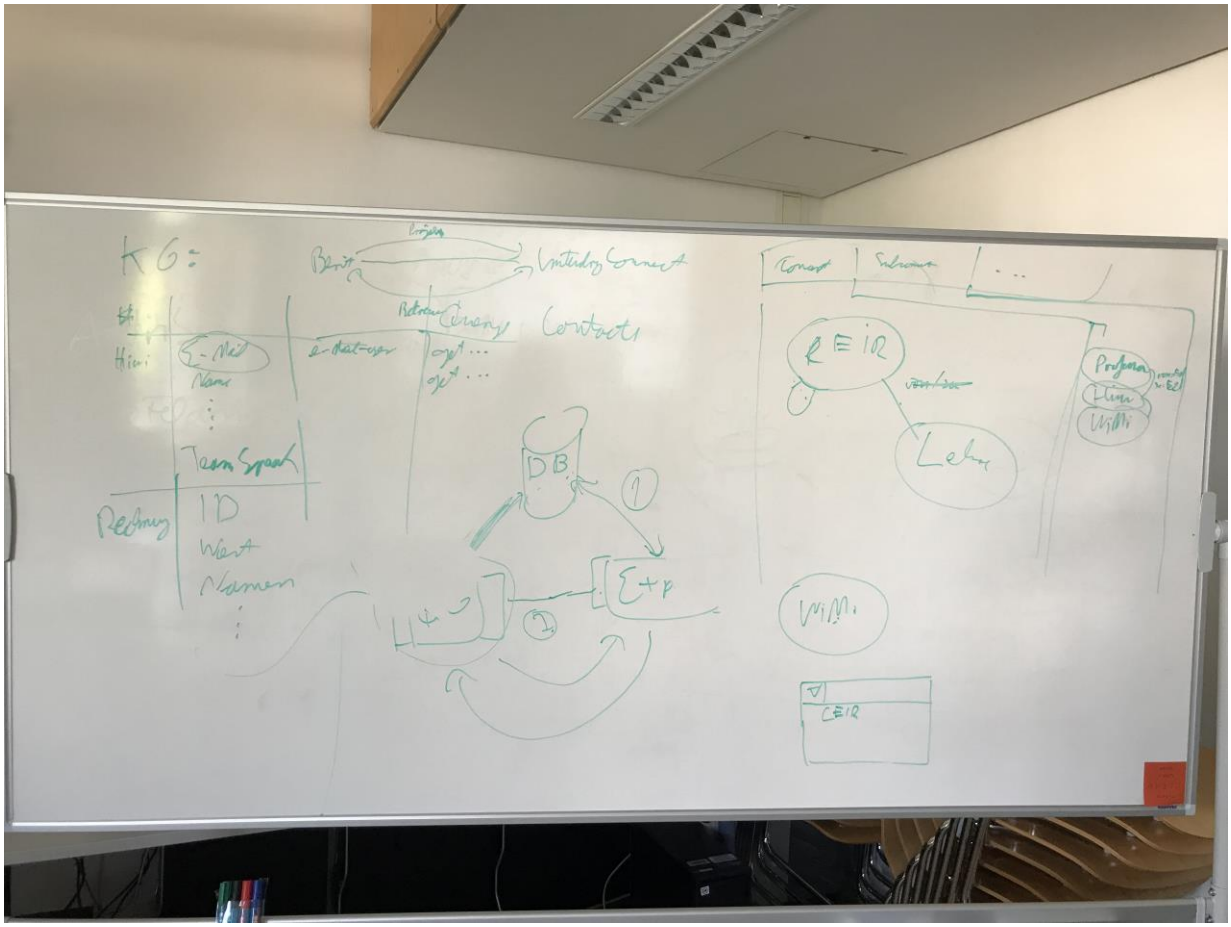
Urbach, N., & Ahlemann, F. (2016). Der Wissensarbeitsplatz der Zukunft: Trends, Herausforderungen und Implikationen für das strategische IT-Management. *HMD Praxis der Wirtschaftsinformatik*, *53*(1), 16–28. https://doi.org/10.1365/s40702-015-0192-7

Vaishnavi, V. K., & Kuechler, W., Jr. (2007). *Design Science Research Methods and Patterns: Innovating Information and Communication Technology* (1st Aufl.). Boston, MA, USA: Auerbach Publications.

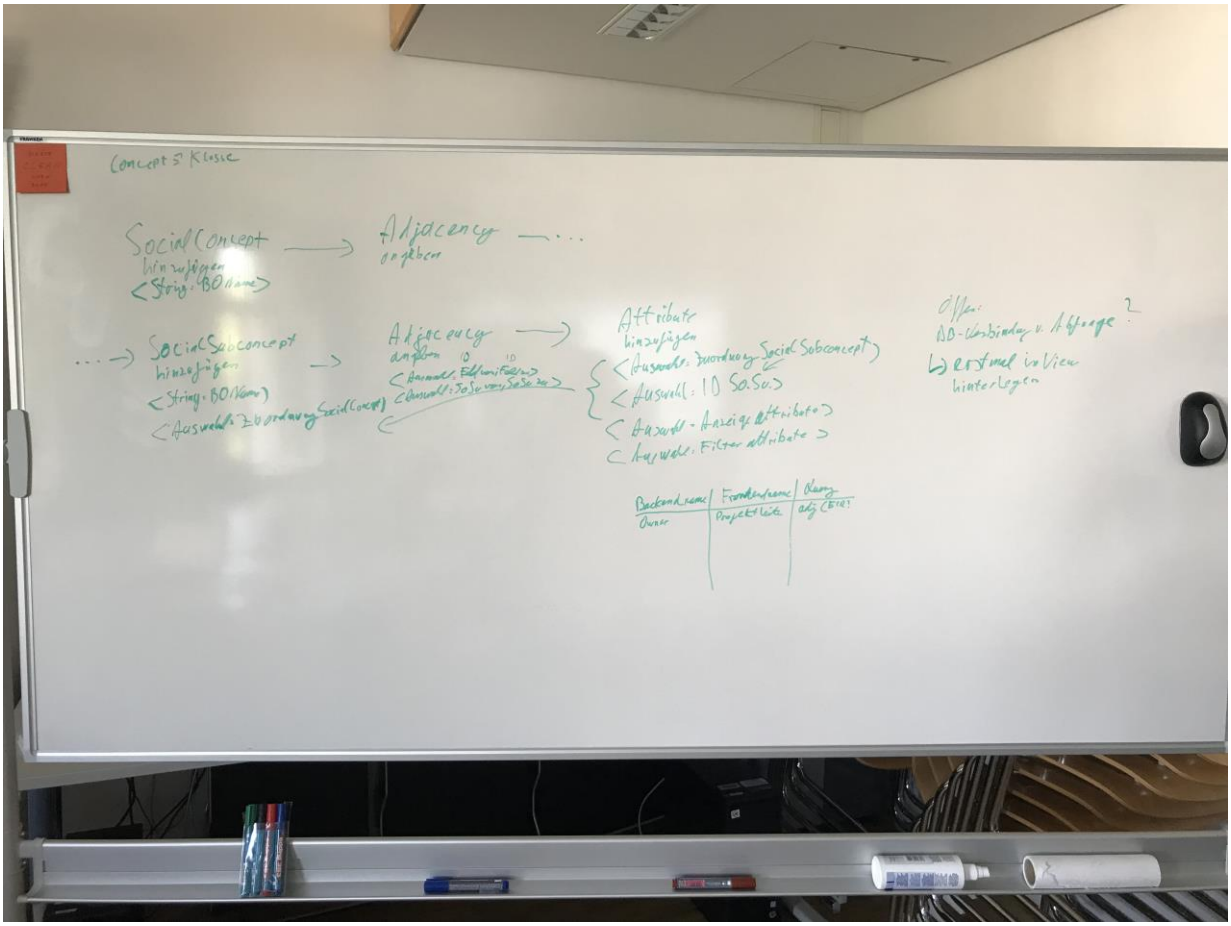
Anhang

Mock-ups SoNBO-Graph-App





Mock-up: Bedienung der SoNBO-Graph-App



Ausschnitt Excel-Dokument: GEDYS IntraWare 8 Customising

		BO-Kategorie | category of BO:	CEIR | CEIR												
		BO-Typ | Type of BO:	Professor | professor	Wiss. Mitarbeiter | res. Assistant	Nicht-wiss. Mitarbeiter | non-scien. Staff	Techn. Mitarbeiter | techn. Assistant	HiWi | student assistant								
Zielgrup				(1) Datenpflege		(2) JSON-String					(3) Konfigurationsdokumente				
Übersicht	Feldname (Kody)	Auswahlmöglichkeit (Kody)	Feldwert (Beispiel)	Pflichtfeld in Kody (Ja; Optional;)	ID (Ja;)	Key (für SoNBO)	Value (aus Kody)	Zusammengesetzt (Ja;) Falls Ja: Zugehörigkeit angeben	Zuordnung zu BO-Typ	Beziehung (1, 2, 3, 4, 6, 7, 8, 10, 11)	Filterattribut (Ja;)	Platzierung Header (Titel; 0; 1; 2; 3; 4)	Feldname im Frontend	Anzeige in Vorschau (Ja;)	Kommentar
Bereich Frontend										(2, 3) Relationenleiste	(6) Filterbereich	(1) Header	(1) Header	(4) Result Area	
	Anrede | Saluration	Herr; Frau; ...; Senora	Frau	Ja		/	fdSaluratoryAddress	Ja: fullName				0			
	Präfix | Prefix	Professor; Dr.; Dr. rer. Nat; ...M.Eng.	Prof. Dr.	Ja		/	fdContactTitle	Ja: fullName				0			
	Vorname | First name		Petra	Ja		/	Firstname	Ja: fullName				0			
	2. Vorname | Middle name		/	Ja		/	MiddleName	Ja: fullName				0			
	Nachname* | last name*		Schubert	Ja		/	LastName	Ja: fullName				0			
	Titel | Title	Prof.; Dr.; Dr. rer. Nat; ...; M.Eng.	/												
	Suffix | Suffix	I; II; III; ...; PHD.	/												
	Berufsbezeichnung | Job title		/	Optional		Title	Title					1	Berufsbezeichnung | Job title	Ja	falls leer, nicht anzeigen
	Abteilung | Department	CEIR; FG BAS; FG EIM	/	Optional		Department	Department								
	Land | Country	Liste mit Ländern	Deutschland			Country	Country								
	Adresse | Address		Universitätsstr. 1	Ja		fdCore_Address1 / fdCore_All	fdCore_Address1 / fdCore_All					2	Straße | Street		
	PLZ | Post code		56070	Ja		ZipCode	ZipCode					2	PLZ | Post code		
	Ort | City		Koblenz	Ja		City	City					2	Ort | city		
	Telefon (Zeichen)		/	Ja		MainPhone	MainPhone					2	Telefon | Phone		
	Mobil (Zeichen)		/	Ja		MobileNumber	MobileNumber					4	Mobil | Mobile phone		
	Fax (Zeichen)		/												
	E-Mail Adresse (Zeichen)		schubert@uni-koblenz.de	Ja	Ja	ID	InternetAddress					4	E-Mail | E-Mail	Ja	
	Web-Links		/	Optional		0						4			
	Rollen | Roles	Professor | professor; Nicht-wiss. Mitarbeiter | non-scientific staff; wiss. Mitarbeiter | research assistant; Techn. Mitarbeiter | technical assistant; HiWi | student	Management	Ja					X				Titel (Untertitel)	/	Ja	
	Skype | Skype		/	Optional		fdSkypeName	fdSkypeName					4	Skype | skype		falls möglich: Icon
	ICQ | ICQ		/												
	MSN | MSN		/												
	Bezeichnete Verteiler | Computed mailing list		/												
	Zusätzlicher Verteiler | additional mailing list	Anbieter\ERP; ...	/												
	Ansichtenweise | Designation	BSc | BSc; MSc | MSc	/	Ja (falls HiWi)		fdViewHint	fdViewHint				Ja	2	Angestrebter Abschluss | targeted degree	Ja	
	Schlagwörter |	WS16-CSCW; SS17-BAS; WS17-CSCW; WS17-PREngineering; SS18-		Ja		fdCommonCategories	fdCommonCategories			1a, 2b	Ja	0	Projekte | projects		

SoNBO: buildInstancegraph-Funktion

```
function buildInstancegraph(){
	// Es gibt eventuell doppelte IDs bei save oder replication conflicts.
Daher werden doppelte IDs ersetzt.
	var usedIds = new Array();

	var sscNodes = sessionScope.SSCdata['nodes'];
	var sscEdges = sessionScope.SSCdata['edges'];
	var nodesArray = new Array();
	var edgesArray = new Array();

	sessionScope.debugERRORjson = "";

	for (var i = 0; i < sscNodes.length; i++){
		var node = sscNodes[i];

		var Database = node['datasource'];
		var View = node['view'];
		var AttributeList = node['attributeList'];
		var IdField = node['idField'];
		var DefinitionField = node['definitionField'];
		var DefinitionValue = node['definitionValue'];
		var Feldname = "json";
		var value;

		// Id des SSC; wird mit an die Instanz übergeben:
		var sscId = node['id'];

		// Farbzuweisung der Instanzen entsprechend der SSCs
		var nodeColor = node['color'];
		var textColor = node['font'];
		// Quelle (View) des SSC wird verwendet für die Verbindung von
CEIR-Objekten untereinander.

		if (Database == undefined || View == undefined || AttributeList
== undefined || IdField == undefined || DefinitionField == undefined ||
DefinitionValue == undefined){
			//TODO: Information ausgeben, dass Knoten unvollständig
definiert ist.
			continue;
		}
		soNBOManager.doLookup(Database, View);
		var idList = soNBOManager.getColumnResults();

		// DbLookup: Führe für jede ID ein Lookup durch
		for (var j = 0; j < idList.length; j++){
			if ((idList[j] != "") && (idList[j] != undefined)){ //
WICHTIG: Inkonsistenz der Daten abfangen
				value = myQuery(Database, View, Feldname, idList[j],
"", "").toString();
				var myId = idList[j];

				// Überprüfen der doppelten IDs
				var idCounter = 1;
				while (usedIds.toString().indexOf(myId) != -1){
					myId = myId + idCounter.toString();
					idCounter++;
				}
				usedIds.push(myId);

				myLabel = getFieldValue(value, IdField);
```

```
                        if ((nodeColor != undefined) && (textColor !=
undefined)){
                              myNodeColor = toJson(nodeColor);
                              myTextColor = toJson(textColor);
                              var replaceString = ",\"id\":\"" + myId +
"\",\"label\":\"" + myLabel + "\",\"color\":" + myNodeColor + ",\"font\":" +
myNodeColor + ",\"sscid\":\"" + sscId + "\"}";
                        }
                        else {
                              var replaceString = ",\"id\":\"" + myId +
"\",\"label\":\"" + myLabel + "\",\"sscid\":\"" + sscId + "\"}";
                        }
                        value = value.replace("}", replaceString);

                        try {
                              var obj = fromJson(value);
                        }
                        catch (err){
                              var tmp = value + err;
                              sessionScope.debugERRORjson =
sessionScope.debugERRORjson + tmp + "\n";
                              var obj = null; // Das fängt den Fehler eines
ungültigen JSON-Strings ab.
                              continue;
                              // TODO: Herausfinden, wann und warum JSON-
Strings ungültig sind
                        }

                        // Achtung: Folgender Code funktioniert nur für
String-Felder!
                        if (obj.length == 1){
                              // Abfrage, ob Bedingung für das Objekt
zutrifft
                              if (obj[0][DefinitionField] ==
DefinitionValue){
                                    nodesArray.push(obj[0]);//fromJson:
Interpretiert String als Objekt
                              }

                        }
                        else { // Pro Zeile sind mehrere JSON-Strings
                              var mergedObj = obj[0];

                              for (var k = 1; k < obj.length; k++){
                                    inputObj = obj[k];

                                    for (var l = 0; l < mergedObj.length;
l++){

                                          attributeTo = mergedObj[l];
                                          attributeFrom = inputObj[l];

                                          if (attributeTo.toString() !=
attributeFrom.toString()){
                                                mergedObj[l] = mergedObj[l] +
", " + attributeFrom;
                                          }
                                    }
                              }
                              // Abfrage, ob Bedingung für das Objekt
zutrifft
                              if (mergedObj[DefinitionField] ==
```

```
DefinitionValue){
                                        nodesArray.push(mergedObj);
                                  }
                            }
                      }
                }
          }
//
################################################################################
################################

      for (var i = 0; i < sscEdges.length; i++){
            var edge = sscEdges[i];
            // Alle Arrays sind gleich groß.
            if (edge['directionFrom'] == undefined){
                  //TODO: Information ausgeben, dass Kante unvollständig
definiert ist.
                  continue;
            }
            else {
                  for (var l = 0; l < edge['directionFrom'].length; l++){
                        var fromNode = edge['directionFrom'][l]; //
Unterschied zu from: directionFrom beinhaltet die Richtung von sourceField
und targetField
                        var toNode = edge['directionTo'][l];
                        var sourceField = edge['sourceField'][l];
                        var targetField = edge['targetField'][l];

                        // 14.01.2019: Bedingung, falls Ziel und Quelle ==
"view":
                        if (edge['linkAlways'] == true){
                              var from = edge['from'];
                              var to = edge['to'];

                              for (var j = 0; j < nodesArray.length; j++){
                                    var sourceNode = nodesArray[j];

                                    if (sourceNode['sscid'] == from){

                                          for (var k = 0; k <
nodesArray.length; k++){
                                                var targetNode =
nodesArray[k];

                                                if (targetNode['sscid'] ==
to){

                                                      var newEdge = (

      {"from":sourceNode['id'],"to":targetNode['id'],"linkAlways":true}

                                                      );

      edgesArray.push(newEdge);
                                                }
                                          }
                                    }
                              }
                              continue;
                        }

                        for (var j = 0; j < nodesArray.length; j++){
```



```
                                var sourceNode = nodesArray[j];
                                var sourceValue = sourceNode[sourceField];
                                var definitionFieldFrom =
edge['directionFromDefinitionField'];
                                var definitionValueFrom =
edge['directionFromDefinitionValue'];
                                var definitionFieldTo =
edge['directionToDefinitionField'];
                                var definitionValueTo =
edge['directionToDefinitionValue'];

                                var definition =
sourceNode[definitionFieldFrom];

                                if (definition == definitionValueFrom){

                                      for (var k = 0; k < nodesArray.length;
k++){
                                            var targetNode = nodesArray[k];
                                            var targetValue =
targetNode[targetField];

                                            definition =
targetNode[definitionFieldTo];

                                            if (definition ==
definitionValueTo){

                                                  // Bedingung
                                                  if ((sourceValue !=
undefined) && (targetValue != undefined)){
                                                        sourceValue =
sourceValue.toString();
                                                        targetValue =
targetValue.toString();

                                                        if
(targetValue.contains(sourceValue)){
                                                              var newEdge = (

      {"from":sourceNode['id'],"to":targetNode['id'],"linkAlways":false}

                                                              );

      edgesArray.push(newEdge);
                                                        }
                                                  }
                                            }
                                      }
                                }
                          }
                    }
              }
      }

      //Doppelte Kanten und Kanten, bei denen Ziel- und Quellknoten gleich
sind, werden entfernt.

      for (var i = 0; i < edgesArray.length; i++){
            var edge = edgesArray[i];

            if ((edge['from'] == edge['to'])){
```

```
                  edgesArray = edgesArray.splice(i, 1); // Lösche Kante aus
dem Array, SSJS: Funktioniert nicht in place.
                  i--; //An gleicher Position ist nun ein neues Objekt,
welches ebenfalls geprüft werden muss.
            }
            else {
                  for (var j = i+1; j < edgesArray.length; j++){ // var j =
i+1: Wenn j = i wäre die Kante identisch.
                        var compareEdge = edgesArray[j];

                        if ((edge['from'] == compareEdge['from']) &&
(edge['to'] == compareEdge['to'])){
                              edgesArray = edgesArray.splice(j, 1);
                              j--;
                        }
                        else if ((edge['from'] == compareEdge['to']) &&
(edge['to'] == compareEdge['from'])){
                              edgesArray = edgesArray.splice(j, 1);
                              j--;
                        }
                  }
            }
      }
      sessionScope.instanzgraphData = {
            nodes: nodesArray,
            edges: edgesArray
      };
      return;
}
```