# Universität Heidelberg

Institut für Informatik
Lehrstuhl für Software Engineering


**Masterarbeit**

# Ontologie-basierte Navigation für CION


Anatoli Zeiser

Matrikel-Nr.: 3243599


9. Januar 2017


Erstbetreuerin: Prof. Dr. Barbara Paech

Zweitbetreuer: Christian Kücherer

# Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die mich bei der Erstellung dieser Masterarbeit tatkräftig unterstützt haben.

Ganz besonderen Dank gilt meinem Betreuer Christian Kücherer für sein aufschlussreiches und zahlreiches Feedback zur Erstellung meiner Arbeit. Ebenfalls danke ich dem SNIK-Team: Prof. Dr. Barbara Paech, Prof. Dr. Alfred Winter, Fraziska Jahn, Michael Schaaf und Konrad Höffner. Ohne die Vorschläge und Hilfe des SNIK-Teams wäre die Erstellung dieser Arbeit in dem Umfang nicht möglich gewesen.

Abschließend möchte ich meinen Eltern, Rosa und Vladimir Zeiser, danken, die mir mein Studium durch ihre umfassende Unterstützung erst ermöglicht haben.

Vielen Dank!

# Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die Masterarbeit mit dem Titel „Ontologie-basierte Navigation für CION“ selbstständig verfasst, noch nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel genutzt, wörtlich und sinngemäße Zitate als solche gekennzeichnet, sowie die „Sicherung guter wissenschaftlicher Praxis “ der Universität Heidelberg beachtet habe.

Heidelberg, 9. Januar 2017 _____

**Abstract**

The aim of this work is to develop a software that allows the navigation and pathfinding in an ontology. The classes and relations, which were founded must be linked with real data. Thus, the found paths also contain information about actual data. The aim of the software is to answer possible questions by using pathfinding. This software is intended to support a CIO (a person, who is responsible for the information management) by making decisions and explanations of projects.

For this purpose, a structured requirement collection is carried out according to the method of task-oriented requirements engineering (TORE). Based on the defined requirements, a snowballing literature research is used to find approaches and solutions for navigation in ontologies and criteria for the pathfinding. Based on this, a conceptual design of a solution for the implementation of the prototype CIONw is carried out. The prototype was developed as a plug-in for the application Cytoscape. The navigation in the ontology is realized by the ontology browser of Cytoscape. The implementation of the pathfinding is done by using the Dijkstra and Yen algorithms. The found paths can be linked to the real data by connecting instance data. In the end, the developed prototype is validated by a system and acceptance test.

**Zusammenfassung**

Das Ziel dieser Arbeit ist es, eine Software zu entwickeln, die das Navigieren und Auffinden von Pfaden in einer Ontologie ermöglicht. Die Klassen und Relationen der gefundenen Pfade müssen mit Realdaten verknüpft werden. Dadurch beinhalten die gefundenen Pfade auch Informationen zu tatsächlichen Daten. Das Ziel der Software ist es, mit dem Auffinden von Pfaden mögliche Fragestellungen zu beantworten. Diese Software soll einen CIO (ein Verantwortlicher des Informationsmanagements) beim Entscheiden und Begründen von Projekten unterstützen.
Dazu wird eine Strukturierte Anforderungserhebung nach der Methode des Aufgabenorientierten Requirements Engineering (TORE) durchgeführt. Auf den definierten Anforderungen wird mittels einer Snowballing Literaturrechere nach Ansätzen und Lösungsideen für das Navigieren in Ontologien und für Kriterien zur Pfadbestimmung gesucht. Darauf aufbauend erfolgt die Konzeption eines Lösungsentwurfes für die Implementierung des Prototypen CIONw. Der Prototyp wurde als eine App für die Anwendung Cytoscape entwickelt. Dabei wird die Navigation in der Ontologie durch einen Ontologiebrowser in Cytoscape umgesetzt. Die Umsetzung der der Pfadfindung erfolgt durch die Anwendung vom Dijkstra und Yen Algorithmus. Die gefundenen Pfade können durch die Anbindung von Instanzdaten mit Realdaten verknüpft werden. Am Ende wird der Entwickelte Prototyp durch einen System- und Akzeptanztest validiert.

# Inhaltsverzeichnis

# 1 Einleitung

Im ersten Kapitel wird eine Einleitung zum Thema und Rahmen der Arbeit gegeben. Anschließend folgt eine Darstellung der Ziele und ein Überblick über den Aufbau der Arbeit.

## 1.1 Motivation

Im Rahmen des DFG-Projektes „Semantisches Netz des Informationsmanagements im Krankenhaus“ (SNIK[1]) entsteht eine Ontologie, die das Wissen aus ausgewählter Literatur des Informationsmanagements abbildet. Bei Ontologien handelt es sich um die Darstellung von Wissen in konzeptueller Form [1]. Das bedeutet, dass Wissen in einer einfachen und formalen Form dargestellt wird. In der Ontologie werden dadurch Zusammenhänge zwischen Klassen (engl. Concepts) mithilfe von Verbindungen (Relationen) dargestellt. Weil Ontologien schnell unübersichtlich werden und es insbesondere sehr viele Verbindungen zwischen einzelnen Klassen der Ontologie geben kann, wird es mit zunehmender Größe schwierig, Zusammenhänge zu finden. Insbesondere ist für den Anwender schwierig, neue Zusammenhänge, basierend auf den Verbindungen zwischen den Einträgen, zu erschließen. Die SNIK Ontologie lässt sich auch als Graph interpretieren und ist als Visualisierung auf einer Webseite verfügbar[2]. Diese Darstellung gibt einen Einblick in die Komplexität, welche aufgrund der vielen Klassen und deren Verbindung entsteht. Abbildung 1.1 zeigt die im Dezember 2016 vorhandene SNIK-Onotlogie. Aufgrund der schwierigen Übersicht durch die Vielzahl an Klassen und Verbindungen in der Ontologie entstand die Idee, der Navigation innerhalb der Ontologie. Zusätzlich zur Navigation sollen zu den Klassen gehörenden Daten angezeigt und miteinander verknüpft werden. Die Idee basiert auf einem Anwendungsszenario des Informationsmanagements (IM) im Krankenhaus: Der Chief Information Officer (CIO), im deutschen vergleichbar mit IT-Leiter bzw. IT-Manager, eines Krankenhauses, ist für das IM verantwortlich. Dabei hat der CIO zahlreiche Verantwortungsbereiche, welche strategische, taktische und operative Aufgaben umfassen. All diese Aufgaben werden durch Daten und letztendlich durch Systeme unterstützt, die Informationen zur Aufgabenbearbeitung bereitstellen.

[1] www.snik.eu
[2] www.snik.eu/graph

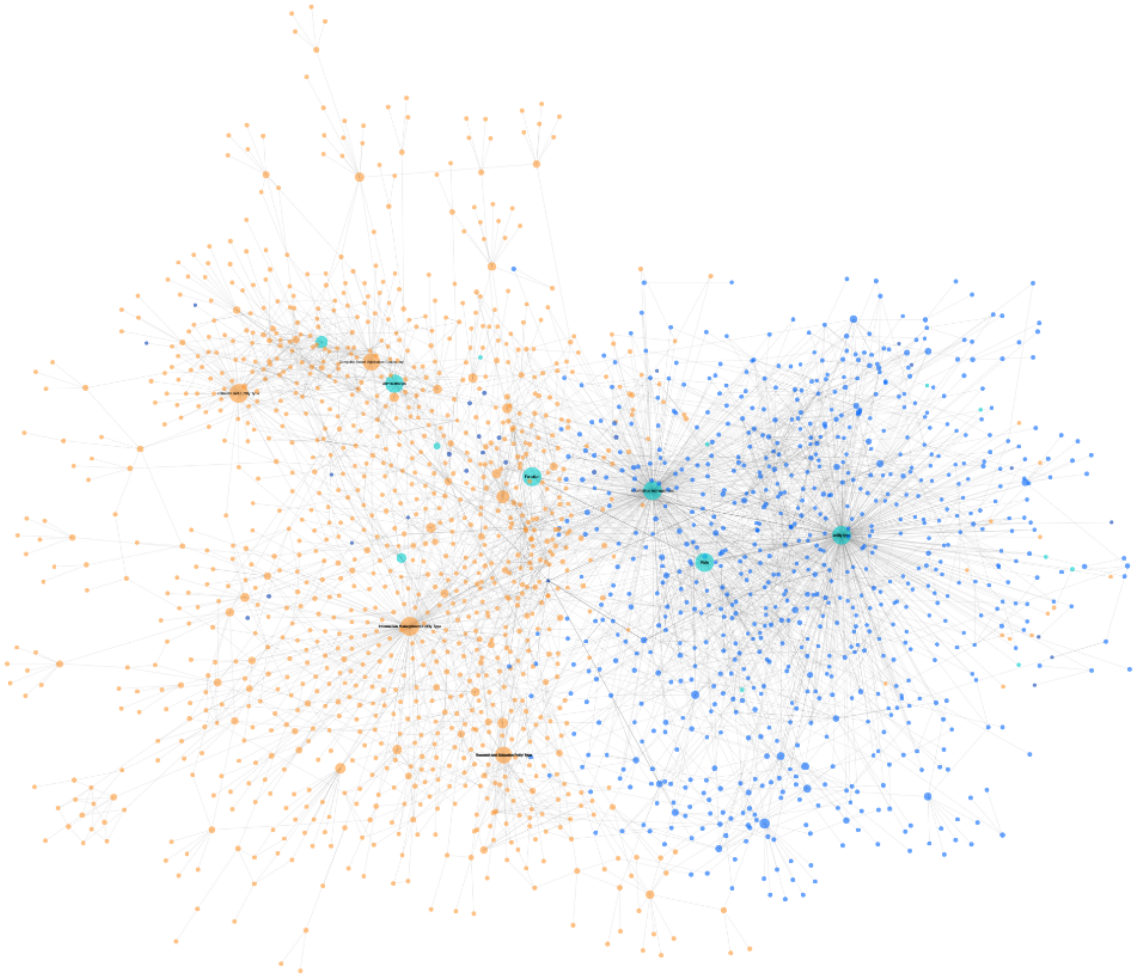

Abbildung 1.1: Gesamte SNIK-Ontologie dargestellt in www.snik.eu/graph

Bei den im IM laufenden Projekten, z.B. Systemmigration oder Neueinführung von Systemen, geht häufig der Zusammenhang zu den strategischen Zielen des IMs verloren. Um diesen Zusammenhang explizit deutlich zu machen, kann auf einer existierenden Ontologie die Verbindungen von Projekten und den Zielen des Informationsmanagements des Krankenhauses gesucht werden. Diese Verbindungen bestehen aus einen oder mehreren Pfaden in der Ontologie. Dabei besteht ein Pfad aus einer Verkettung von Klassen und den Beziehungen zwischen den Klassen. Ein weiteres Anwendungsszenario ist es, bisher unbekannte Verbindungen zwischen bestimmten Klassen aufzufinden, indem auf der Ontologie entlang von Klassen navigiert wird. Dabei können dem CIO bisher unbekannte Zusammenhänge klar werden. Für einige Zusammenhänge kann es Daten geben, für andere nicht. Dennoch bieten diese Zusammenhänge wichtiges Wissen an, welches beispielsweise für die Rechtfertigung von Projekten im operativen Bereich geeignet ist. Des Weiteren können die fehlenden Daten dem CIO auf eine bisher mangelnde Dokumentation bzw.

Datenzusammenhänge hinweisen. Die möglichen Erkenntnisse können durch exploratives Navigieren entstehen, oder durch eine gezielte Fragestellung, wie z.B. „Haben Projekte eine weitere Verbindung zu einem Strategischen Ziel?“

Im Rahmen des SNIK-Projekts wird der CIO-Navigator (CION) entwickelt, welcher den CIO bei seinen Entscheidungen im IM unterstützt. CION ist ein Werkzeug, welches bedarfsgerechte Informationen für das Treffen von Entscheidungen bereitstellt. CION stellt nur vorher festgelegte Datenklassen dar. In dieser Masterarbeit wird ein Prototyp entwickelt, mit dem Anwender neue Zusammenhänge innerhalb von Ontologien aufdecken können und auf deren Basis neue unbekannte Daten für weitere Entscheidungen (z.B. für CION) gefunden werden können. Aufgrund der Verbindung zu CION wird dieser Prototyp im folgenden CIONw genannt, dabei steht das „w“ für Ontologie*w*eit.

## 1.2 Zielsetzung

Im Nachfolgenden werden die einzelnen Ziele für diese Arbeit beschrieben.

**Z1 - Anforderungsspezifikation** Für die Erstellung von CIONw muss eine Anforderungserhebung erfolgen, mit dem Ziel eine Spezifikation für den zu erstellenden Prototyp zu erhalten. Dabei erfolgt die Anforderungserhebung nach der Methode des Aufgabenorientierten Requirements Engineering TORE [2] (vgl. Kapitel 2.3).

**Z2 - Navigation in Ontologien** Es müssen die Anforderungen für das Navigieren in Ontologien definiert werden. Dabei müssen die Anforderungen das Auffinden von Klassen in der Ontologie und das Finden von Pfaden zwischen mindestens einer Start- und einer Zielklasse unterstützen. Es ist wichtig, dass nicht nur ein einziger Pfad gefunden wird, sondern mehrere alternative Pfade aufgedeckt werden.

Für das Navigieren müssen die Kriterien, nach denen ein kürzester Weg gefunden werden kann, festgelegt werden. Um das Navigieren auf verschiedene Kriterien anpassen zu können, müssen zunächst, die dafür notwendigen Parameter, definiert werden.

**Z3 - Prototyp Entwicklung** Der aus den Anforderungen hergeleitete Ansatz wird als Prototyp implementiert. Dabei wird ein definierter Ausschnitt aus der SNIK Ontologie verwendet und damit die Nutzbarkeit demonstriert.

## 1.3 Aufbau der Arbeit

Zu Beginn der Arbeit werden in Kapitel 2 relevante Grundlagen aus dem Bereich Informationsmanagement, Anforderungserhebung und Ontologien erläutert, welche für die Thematik der Arbeit notwendig sind. Dabei wird ein kurzer Einblick in die Domäne und die Ontologie von SNIK und der daraus ausgewählte Ausschnitt für CIONw gegeben. In Kapitel 3 wird die durchgeführte Literaturrecherche und deren Ergebnisse vorgestellt. Im Anschluss wird in Kapitel 4 die durchgeführte Anforderungserhebung und die daraus resultierenden Anforderungen nach TORE beschrieben. Basierend auf den Anforderungen wird in Kapitel 5 die Problemstellungen für die Implementierung von CIONw erläutert und Lösungen für diese erstellt. In Kapitel 6 wird die Implementierung und die dazugehörenden Tests samt der Auswertung des durchgeführten Akzeptanztestes beschrieben. Im Anschluss zu der Implementierung erfolgt in Kapitel 7 eine Betrachtung für die weitere mögliche zukünftige Verwendung von CIONw und eine abschließende Zusammenfassung der Ergebnisse dieser Arbeit.

# 2 Grundlagen

## 2.1 Informationsmanagement im Krankenhaus

In diesem Kapitel wird die Domäne der Ontologie des SNIK Projektes kurz erläutert. Die Beschreibungen basieren dabei auf dem Buch „*Health Information Systems*“ von Winter et al. [3].

**Informationssystem** Ein Informationssystem ist ein soziotechnisches System, welches sowohl aus Menschen (z.B. Mitarbeiter im Krankenhaus) als auch aus technischen Komponenten (z.B. Computer) besteht, welche in der Informationsverarbeitung involviert sind. Sobald Daten, Informationen und Wissen generiert, verwaltet oder gespeichert werden, spricht man von einem Informationssystem. Es beschränkt sich dabei nicht nur auf die technischen Komponenten, sondern beinhaltet auch das Verwalten der sozialen Komponenten.

**Informationsmanagement** Im Allgemeinen bezeichnet der Begriff Informationsmanagement alle verwaltenden Tätigkeiten, die für die Bewerkstelligen eines Vorhabens oder eines gesamten Prozesses mit Bezug auf das Informationsmanagement notwendig sind. Dabei ist das Informationsmanagement immer auf eine definierte Rahmenumgebung festgelegt. Solche Festlegungen können alle informationsverarbeitenden Prozesse eines Krankenhauses sein. Das Ziel des Informationsmanagements ist die Erfüllung von definierten strategischen Zielen. Nach Winter et al. unterteilen sich die Aufgaben des Informationsmanagements in drei Teile: Planung, Steuerung und Überwachung.

Des Weiteren kann das Management in drei Arten unterteilt werden: strategisches, taktisches und operatives Management. Das strategische Management umfasst alle Maßnahmen, die zur strategischen Ausrichtung der IM-Abteilung notwendig sind. Das taktische Management setzt die strategischen Vorhaben um. So ist beispielsweise das Projektmanagement eine zentrale Aufgabe des taktischen Management. Das Operative Management ist zuständig für den operativen Betrieb des Informationsmanagements. Hierzu zählen z.B. alle administrativen Tätigkeiten.

**Informationssysteme im Krankenhaus** Das Informationssystem eines Krankenhauses (engl. Hospital Information System, kurz HIS) unterstützt im Idealfall alle Informationsprozesse des Krankenhauses. Dabei beinhaltet es die sozialen und technischen Komponenten des Informationsprozesses. Des Weiteren sind auch Unternehmensfunktionen, Geschäftsprozesse und Anwendungen Bestandteil eines Informationssystems im Krankenhaus.

Einige weitere detaillierte Bestandteile des IM werden im Zusammenhang mit dem Ausschnitt aus der SNIK-Ontologie im Anhang A.1.1 beschrieben.

## 2.2 Ontologie

Der Begriff Ontologie hat seinen Ursprung aus der Philosophie und beschreibt die Darstellung der Natur in Kategorien zu einer bestimmten Sichtweise auf die Welt [4]. Erst später kam der Begriff Ontologie auch in der Informatik zu tragen. Dabei beschreibt Gruber [1] den Begriff Ontologie, als eine explizite Spezifikation der Konzeptualisierung. Diese Definition beschreibt eine strukturierte Abbildung von Konzepten der realen Welt und deren Beziehungen untereinander. Im genaueren findet eine solche Abbildung in einem für Menschen und für Maschinen (Computer) lesbaren Form statt. Dabei können Konzepte der realen Welt in Klassen und Verbindungen zwischen den Konzepten in Relationen dargestellt werden. Das besondere an Ontologien ist, dass diese an die gewünschte Domäne vollständig angepasst werden können. Darüber hinaus unterliegen Ontologien keiner vordefinierten Darstellungsart. Ontologien können durch einfache Tupel (meist 3-Tupel) oder durch ganze Sätze dargestellt werden. Zusätzlich können Ontologien grafisch dargestellt werden wie z.B. die SNIK Ontologie in Abbildung 1.1 [5].

Nach Guarino [6] können Ontologien in die folgenden vier Arten eingeteilt werden:

- *Top-level* Ontologien: stellen allgemeine Beschreibungen dar, ohne einen konkreten Bezug zu einer Domäne oder Problemstellung zu haben
- *Domain* Ontologien: stellen Beschreibungen zu bestimmten Domänen bereit, wie z.B. Branchen oder Wissenschaftszweige
- *Task* Ontologien: stellen Beschreibungen zu bestimmten Aufgaben bereit
- *Application* Ontologien: kombinieren die *Domain* und *Task* Ontologien zu einem bestimmten Anwendungsgebiet

### 2.2.1 SNIK Ontologie

Im Rahmen des Forschungsprojekts SNIK wurde eine Ontologie erstellt, die das strategische, taktische und operative Informationsmanagement im Krankenhaus abbildet. Die Modellierung der Ontologie basiert auf einer händischen Extraktion aus den beiden Fachbüchern „IT-Projektmanagement im Gesundheitswesen“ [7], „Health Information Systems: Architectures and Strategies“ [3] und den Ergebnissen von mehreren Interviews mit einem CIO eines Krankenhauses. Die Extraktionen wurden in jeweils getrennte Ontologien abgebildet, abhängig von den verwendeten Quellen. Die Ergebnisse der Extraktion wurden in einer Exceltabelle aufgenommen und nach mehreren Reviews mithilfe eines Tools in ein OWL Format transferiert [8].

OWL wird nach W3C als „Web Ontologie Language“ bezeichnet und wird als Beschreibungsmittel für Ontologien genutzt[3]. OWL ist wiederum in drei Sprachen eingeteilt: OWL Lite, OWL DL und OWL Full. Jede der Sprachen ist für bestimmte Beschreibungsmöglichkeiten optimiert. Wobei gilt, dass eine Ontologie in OWL Lite auch eine gültige Ontologie in OWL DL ist und jede OWL DL Ontologie auch eine gültige OWL Full Ontologie ist.

**Metamodell SNIK-Ontologie** Für die SNIK-Ontologien wurde ein Metamodell entwickelt, welches alle Klassen der Ontologie in folgende Typen eingeteilt: `Function`, `Role` und `EntityType`. Klassen vom Typ `Function` stellen Aufgaben dar, welche auf ein `EntityType` angewendet werden bzw. davon unterstützt werden. Die Aufgaben werden von einer `Role` durchgeführt. `Role` stellt eine Rolle des Informationssystems dar (im Allgemeinen eine Person mit einer bestimmten Position, z.B. der CIO oder ein Projektleiter). `EntityType` sind Entitäten des Informationsmanagements. Entitäten sind Objekte des Informationsmanagements, die weder einer `Role` noch eine `Function` zugeordnet sind. Alle Typen stehen durch unterschiedliche Beziehungen in Verbindung zueinander. In Abbildung 2.1 sind die Typen und deren Beziehungen dargestellt (Stand: 14.12.2016). Da die Ontologie und das dazugehörende Metamodell ständigen Verbesserungen unterliegen, können die Beziehungsarten im Laufe der Zeit angepasst werden.

[3] https://www.w3.org/TR/2004/REC-owl-features-20040210/

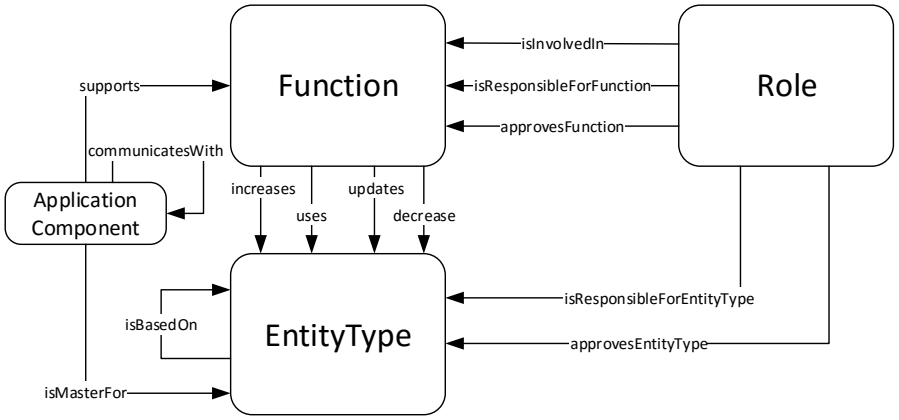


Abbildung 2.1: Vereinfachter Ausschnitt aus dem Metamodell der SNIK-Ontologie

**RDF** Das SNIK Projekt stellt die Ontologie mithilfe vom Cytoscape JavaScript[4] Framework auf einer Webseite[5] zur Visualisierung zur Verfügung. Zur Darstellung der Ontologie wurde diese im JSON-Format benötigt. Um diese zu erhalten und um aus mehreren Ontologien eine gemeinsame Ontologie darzustellen wurden die einzelnen Ontologien im RDF Modell auf Basis einer XML Syntax für RDF abgebildet.

RDF steht für „Resource Description Framework“ und ist eine Sprache zur Darstellung von Informationen aus dem Web [9]. Dazu wird RDF in der Syntax von XML geschrieben. Das RDF Model basiert auf der 3-Tupel Darstellung von *Subjekt*, *Prädikat* und *Objekt*. Der Beispielausschnitt aus der Onto-

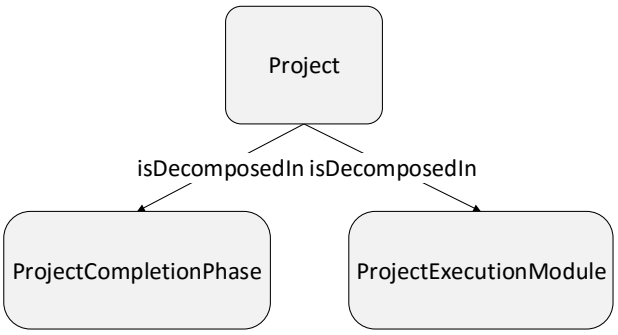


Abbildung 2.2: Beispiel eines minimalen Ausschnitts aus der Ontologie

logie in Abbildung 2.2 drückt aus, dass ein Projekt unterteilt wird in eine Projekt-Abschlussphase (engl. Project Completion Phase) und eine Projekt-

[4] http://js.cytoscape.org/
[5] www.snik.eu/graph

Durchführungseinheit (engl. Project Execution Modul). DIes wird im RDF Model mit der Notation von OWL wie folgt dargestellt:

```
<owl:Class rdf:about="Project">
    <rdfs:label xml:lang="en">Project</rdfs:label>
    ...
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&meta;isDecomposedIn"/>
            <owl:someValuesFrom rdf:resource="
                ProjectCompletionPhase"/>
        </owl:Restriction>
    </rdfs:subClassOf>
            <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&meta;isDecomposedIn"/>
            <owl:someValuesFrom rdf:resource="
                ProjectExecutionModule"/>
        </owl:Restriction>
    </rdfs:subClassOf>
            ...
</owl:Class>
```

**Instanziierung von Klassen in Ontologien** Ontologien beschreiben Sachverhalte durch die Definition von Klassen und Beziehungen. Durch die Beschreibung lassen sich aber keine Beziehungen zwischen konkreten Individuen der Klassen darstellen. So stellt beispielsweise die Aussage `ProjectManager isInvolvedIn Project` dar, dass ein Projektleiter (Person A oder Person B) in ein Projekt (Projekt X oder Projekt Y) involviert ist, aber es kann keine Information gewonnen werden, welcher Projektleiter in welchem Projekt involviert ist. Dieses Problem kann dadurch gelöst werden, dass Individuen der Klassen als Instanzen in der Ontologie aufgenommen werden. Dadurch kann beispielsweise eine Instanz `M. Musterman` von der Klasse `ProjectManager` und eine Instanz `Musterprojekt` von der Klasse `Project` in der Ontologie modelliert werden. Anschließend ist es möglich zwischen den beiden Instanzen durch die `isInvolvedIn` Beziehung eine klare Zuordnung festzulegen. In der SNIK-Ontlogie hingegen findet nur eine Abbildung auf Klassenebene statt, so dass keine konkrete Instanzen innerhalb der Ontologie existieren. Da Teile von

CIONw auf Instanzen arbeiten müssen, wird eine Instanziierung von Klassen außerhalb der Ontologie benötigt. Dabei handelt es sich bei der Instanziierung um Realdaten, welche den Klassen der Ontologie zugeordnet werden können. Im weiteren Verlauf der Arbeit wird der Begriff *Instanz* als Synonym für Realdaten genutzt.

### 2.2.2 Ontologien als Graph

In der Informatik sind Graphen elementare Datenstrukturen. Vereinfacht gesagt ist ein Graph eine Menge von Knoten (engl. node) $N$ und Kanten (engl. edges) $E$. Knoten in einem Graphen werden durch Kanten miteinander verbunden und beschreiben dadurch eine Beziehung zwischen den verbundenen Knoten.

$$e_k = \{n_i, n_j\} \quad i \neq j,\ n_i, n_j \in N,\ e_k \in E$$

Graphen können anhand der verwendetet Eigenschaften von Kanten nachfolgend klassifiziert werden:

**einfacher Graph** Zwischen zwei Knoten ist nur eine Kante erlaubt. Die Kante hat keine Richtung.

**gerichteter Graph** Kanten zwischen Knoten sind immer gerichtet. D.h. jede Kante hat einen Quellknoten und einen Zielknoten.

**Multigraph** Zwischen zwei Knoten können mehrere Kanten existieren. Ein Mutligraph kann entweder aus gerichteten Kanten oder aus ungerichtete Kanten bestehen.

Da in Ontologien zwischen Klassen Beziehungen existieren, können diese Beziehungen durch Graphen dargestellt werden. Klassen werden als Knoten und Beziehungen zwischen Klassen als Kanten zwischen Knoten dargestellt. Durch diese Interpretation ist eine Analyse der Ontologie mithilfe von Graphenalgorithmen möglich. So stellt das minimale Beispiel aus Abbildung 2.2 bereits einen Graphenrepräsentation dar. Dabei sind die Knoten $N$ jeweils $n_1 = Project, n_2 = ProjectCompletionPhase, n_3 = ProjectExecutionModule$ und die Kanten $E$ jeweils $e_1 = \{n_1, n_2\}, e_2 = \{n_1, n_3\}$. Beide Kanten $e_1$ und $e_2$ haben einen Namen als Eigenschaft (`isDecomposed`).

In den Abbildungen 2.2 und 2.3 handelt es sich jeweils um gerichtete Graphen, weil die Kanten mit einer Pfeilrichtung versehen sind. Der in Abbildung 2.3 dargestellte Graph, ist ein Multigraph, weil zwischen einigen Knoten mehrere

Kanten existieren. Somit erfolgt die Graphendarstellung der SNIK-Ontologie in einem gerichteten Mutligraphen.

Ein Pfad in einem Graph ist eine geordnete Angabe von Kanten oder Knoten, welche von einem Startknoten zu einem Zielknoten führt. Pfade können entweder aus Kanten, Knoten oder Knoten-Kanten Paaren bestehen, je nachdem welche Informationen in den Kanten und den Knoten abgelegt werden. So kann der Weg von `Application_Component` nach `Project` aus Abbildung 2.3 als folgender Pfad dargestellt werden:

$$P = \{(n_0, \emptyset), (n_1, e_0), (n_2, e_1)\}$$
$$n_0 = Application\_Component, n_1 = Portfolio\_Management, n_2 = Project$$
$$e_0 = \{n_0, n_1\}(uses), e_1 = \{n_1, n_2\}(updates)$$

Bei dieser Art der Darstellung beschreibt das Paar $(Knoten, Kante)$ jeweils einen Knoten und die Kante von welcher der Knoten erreicht wurde. Deswegen besitzt das erste Paar keine Kante, da es sich hierbei um den Startknoten handelt.

Ein Pfad kann einen bestimmten Wert haben, der den Aufwand oder die Kosten beschreibt, die Notwendig sind um von einem Knoten zu einem anderen Knoten zu gelangen. Dieser Aufwand kann beispielsweise dadurch entstehen, wenn eine Kante ein Gewicht bzw. Kostenfaktor bekommt. Die Kosten können dann als die Summe aller Kanten in einem Pfad interpretiert werden. Die Kantengewichte können auch negative Werte aufweisen.

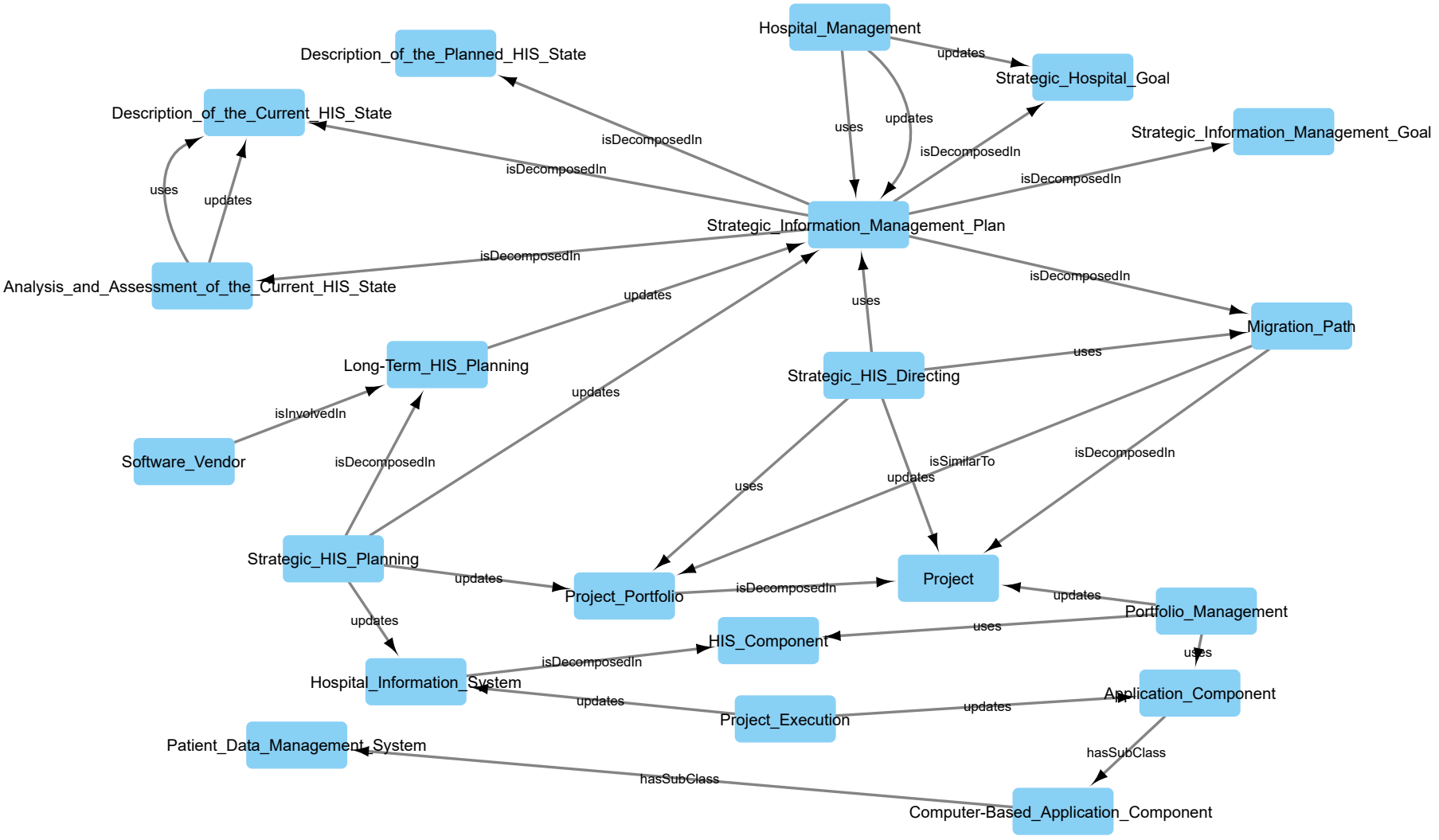


Abbildung 2.3: Ausgewählter Ausschnitt der Ontologie für die Instanziierung

### 2.2.3 Ausgewählter Ausschnitt der Ontologie

Für das Testen von CIONw wird eine Ontologie benötigt, welche übersichtlich und leicht verständlich ist, aber gleichzeitig genügend Kanten und Knoten enthält, dass es mehrere Pfade zwischen zwei beliebigen Knoten geben kann. Dafür wurde ein kleiner Teilausschnitt der SNIK-Ontologie als Grundlage zur Verfügung gestellt. Die für das Testen notwendigen Instanzen werden durch realistische Beispieldaten per Hand erstellt. Im Nachfolgenden werden die Klassen und Verbindungen aus dem Beispielausschnitt genauer erklärt. Die Beschreibung der Knoten ist aus den Definitionen der SNIK-Ontologie und den dazugehörenden Fachbücher entnommen. In der Abbildung 2.3 ist der ausgewählte Ausschnitt in Cytoscape als Export dargestellt. Der Ausschnitt basiert auf der Ontologie vom 16.04.2016 und kann durchaus zu der aktuellsten Version Unterschiede aufweisen, weil fortlaufend an der Ontologie weitergearbeitet wird.

**Knoten der Ontologie** Die in Abbildung 2.3 dargestellten Klassen werden im Anhang A.1.1 gelistet und deren inhaltliche Bedeutung dargestellt. Die jeweiligen Instanztabellen zu den Klassen (vgl. 5.2.1 und 5.2.2) mit den Beispiel-Instanziierungen befinden sich im Anhang A.1.2.

**Relationen** Die sechs Relationsarten, die die Beziehung zwischen den Klassen des Ausschnittes der Ontologie darstellen, werden im nachfolgenden kurz erläutert. Analog zu den Klassen sind die Relationstabellen im Anhang A.1.3 dargestellt. In der gesamten SNIK-Ontologie können auch weitere Relationen existieren, die in diesem Abschnitt nicht genannt werden. Die SNIK-Ontologie ist durch kontinuierliche Weiterentwicklung und Verbesserung Änderungen unterworfen, so sind bereits Anpassungen an den Bezeichnungen der Relationen im Zeitraum dieser Arbeit entstanden.

**uses** wird zwischen einer Aufgabe und einem Objekttypen verwendet und beschreibt, dass die Aufgabe auf Daten eines Objekttyps zugreift.

**updates** wird zwischen Aufgaben und Objekttypen verwendet und beschreibt, dass die Aufgabe Daten des Objekttyps erstellt oder verändert.

**isInvolvedIn** wird zwischen Rollen und Aufgaben verwendet und beschreibt, dass die Rolle in die Aufgabe involviert ist.

**isDecomposedIn** wird zwischen Objekttypen, Aufgaben oder Rollen zu dem selben Typ verwendet um damit den Begriff in einzelne Bestandteile zu zerlegen. Damit wird eine hierarchische Beziehung beschrieben.

**isSimilarTo** wird zwischen Objekttypen, Aufgaben oder Rollen zu dem selben Typ verwendet um damit eine Ähnlichkeit zwischen den Klassen zu beschreiben.

**hasSubClass** beschreibt eine vererbende Beziehung zwischen zwei Klassen der Ontologie.

## 2.3 TORE

Für die Anforderungserhebung bei den Stakeholdern wurde die Methode des Aufgabenorientierten Requirements Engineering (engl. Task Oriented Requirements Engineering, kurz TORE) gewählt, die in diesem Kapitel kurz erläutert wird.

TORE wurde von Paech und Kohler in [2] vorgestellt und anschließend von Adam et al. [10] um zwei Punkte erweitert. Der TORE Ansatz orientiert sich bei der Anforderungserhebung an den tatsächliche Aufgaben (Tasks) des Nutzers. Dabei werden die Nutzeraufgaben unabhängig von der Anwendung analysiert und darauf aufbauend die Anforderungen für die Anwendung erstellt.

Der TORE Ansatz betrachtet die einzelnen Entscheidungen (engl. Decisions), die im Verlauf des Requirements Engineering für die Systementwicklung getroffen werden müssen. Dabei werden insgesamt 18 Entscheidungspunkte unterschieden, die in vier Ebenen eingeteilt werden. In Abbildung 2.4 wird

diese Einteilung dargestellt. TORE gibt keine feste Reihenfolge für die Bearbei-

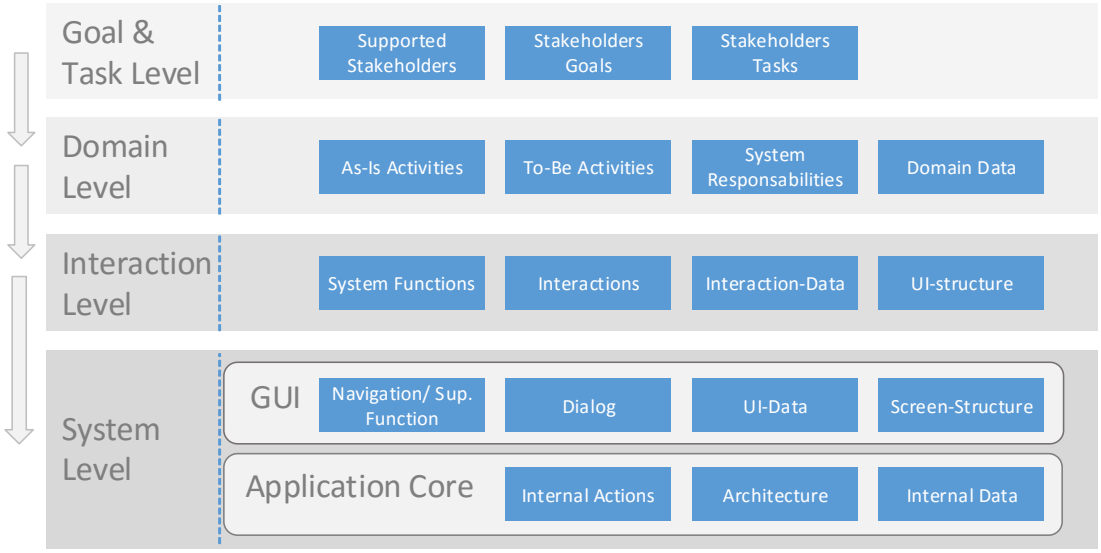


Abbildung 2.4: Entscheidungspunkte und Spezifikation von TORE [10]

tung der Entscheidungspunkte vor, jedoch kann es durchaus einfacher sein, sich von der ersten Ebene hin zur letzten Ebene durchzuarbeiten. So ist es beispielsweise schwierig eine Entscheidung zu `As-is Activities` zu treffen, wenn die eigentliche `Task` nicht bekannt ist.

Die Task und Goal Ebene untersucht die Anforderungen anhand der Aufgaben und Ziele des Nutzers. In dieser Ebene werden die Stakeholder (`Supported Stakeholders`) festgelegt und deren Aufgaben auf oberster Ebene ermittelt und dokumentiert. Dies ermöglicht die explizite Entscheidung, welche Stakeholder und welche Aufgaben im Rahmen der Anwendungsentwicklung berücksichtigt werden müssen.

In der Domäne Ebene werden die Aufgaben weiter detailliert untersucht. So wird bestimmt wie die Aufgabe derzeit durchgeführt wird und bei welchen Tätigkeiten die Anwendung den Nutzer wie genau unterstützen soll.

In der Interaktionsebene werden die Interaktionen zwischen dem Nutzer und dem System festgelegt. Dabei sind hier bereits Nutzeroberflächenstrukturen zu entwerfen. Die Entwürfe müssen dabei nicht unbedingt der tatsächlichen Oberfläche der Anwendung entsprechen, sondern müssen den wesentlichen Inhalt bzw. eine Gruppierung von Informationen festlegen.

Zur Systemebene gehören alle Entscheidungen, die die Anwendung selbst betreffen. Darunter sind auch die tatsächliche Nutzeroberfläche, die Architektur, unterstützende Systemfunktionen und weitere Anwendungsfunktionen.

## 2.4 Dokumentation von Anforderungen

Jede Art von Anforderung muss durch ein geeignetes Format dokumentiert werden. In diesem Kapitel werden die in der Arbeit verwendeten Dokumentationsarten eingeführt. TORE unterstützt dabei für die 18 Entscheidungspunkte unterschiedliche Dokumentationsformen.

### 2.4.1 Task und Subtasks

Für die Dokumentation der Aufgaben nach TORE ist der Ansatz „Task & Support“ von Lauesen geeignet [11]. Lauesen definiert einzelne Aufgaben (Tasks) als eine Tätigkeit, die von Nutzer und einem Informationssystem gemeinsam durchgeführt werden. Eine Unteraufgabe (Subtask) dient der feineren Beschreibung der übergeordneten Aufgabe. Da in TORE die User Task auf der Task Ebene befinden, beschreiben diese nur die reine Nutzeraufgabe ohne direkte Betrachtung des vorhandenen (oder geplanten) Systems. Die Subtasks werden in der Form in TORE nicht direkt abgebildet, stattdessen sind sie in den Entscheidungspunkten `As-is Activities` und `To-be Activities` vorhanden. Durch die Untersuchung der aktuellen Lösungsumsetzung einer Aufgabe (`As-is Activities`) und der möglichen bzw. gewünschten verbesserten Lösungsumsetzung (`To-be Activities`) und einer Kombination beider Entscheidungspunkte können Subtasks dargestellt werden.

Lauesen wählt für die Dokumentation der Task und Subtasks eine tabellarische Form. Dabei werden zu jedem Task vorhandene Subtasks zugeordnet und zu jedem Subtask können Beispiellösungen angegeben werden, sofern die Lösungen zum Zeitpunkt der Anforderungserhebung bekannt sind. Zusätzlich kann zu jeder Aufgabe ein Start- und Zielzustand, die Häufigkeit und die Schwierigkeit der Aufgabe angegeben werden.

### 2.4.2 User Stories

User Stories sind Beschreibungsmittel aus dem Bereich der Agilen Softwareentwicklung. Sie dienen dem einfachen Beschreiben von Anforderungen und Funktionalitäten an eine Anwendung oder an ein System [12]. User Stories können auch als Dokumentationsart für Anforderungen genutzt werden, sollten aber möglichst kurz und einfach formuliert sein. User Stories können detaillierte Informationen beinhalten, jedoch sollten die User Stories immer aus einer

Sicht formuliert sein, die für die Stakeholder nachvollziehbar ist. Technische Details (wenn sie nicht essenziell sind) sollten in User Stories nicht geklärt werden. Die User Stories können nach folgenden Satzschema modelliert werden:

„Als <Rolle> möchte ich <Ziel>, um <(optional) Begründung>“

Ein Beispiel für eine User Story ist:

„Als Nutzer möchte ich Klassen in der Ontologie suchen, um deren Vorhandensein zu überprüfen. “

Der nachfolgende Satz ist beispielsweise keine User Story:

„Die Anwendung wird in Java entwickelt.“

User Stories können bei der Implementierung Zusatzinformationen für eine bestimmte Entscheidung liefern und damit dem Entwickler einen besseres Verständnis für die Funktion bzw. Anforderung bieten.

# 3 Literaturrecherche

In diesem Kapitel wird die Methodik der Literaturrecherche und die daraus gewonnenen Ergebnisse beschrieben. Die Recherche dient der Ermittlung des aktuellen Stands der Forschung im Bereich der Navigation in Ontologien. Dadurch können bereits vorhandener Ansätze in den Lösungsansatz der Arbeit einfließen.

## 3.1 Forschungsfragen

Zu Beginn einer Literaturrecherche steht eine Fragestellung, die sich aus den Zielen der Arbeit ergibt. Die Fragestellungen sollen mithilfe der Literaturrecherche beantwortet werden. Für diese Arbeit wurden die nachfolgenden Forschungsfragen formuliert:

**RQ1** Welche Lösungen gibt es für die Navigation in Ontologien?

**RQ2** Welche Kriterien haben Einfluss auf das Navigieren in Ontologien bzw. Graphen?

Die Forschungsfrage RQ1 bestrebt für die Ziele Z2 - Navigation in Ontologien und *Z3 - Prototyp Entwicklung* bereits bekannte Vorgehen oder Lösungen zu finden und um darauf die Entwicklung zu stützen. Die Forschungsfrage RQ2 ist für das Ziel *Z2 - Navigation in Ontologien* relevant. Durch das Extrahieren von Einflussfaktoren für die Pfadbestimmung können bei der Lösungsfindung diese Faktoren beachtet werden und Einfluss auf die verwendeten Algorithmen nehmen.

## 3.2 Snowballing Methode

Literaturrecherchen können auf den beiden Methoden Systematische Literaturrecherche [13] oder Snowballing [14] basieren. Jalali und Wohlin [15] haben in einer Studie beide Methoden gegenüber gestellt und sind zu dem Ergebnis gekommen, dass die Systematische Literaturrecherche initial mehr Zeit benötigt, weil insbesondere die Suchterme für jede Suchmaschine neu angepasst werden müssen. Weiterhin hat sich gezeigt, dass bei der Systematischen Literaturrecherche im Allgemeinen eine sehr große Anzahl von Artikeln analysiert

werden müssen. Dies führt zu einem größeren Aufwand beim Reduzieren auf die relevanten Artikel. Beim Snowballing hingegen ist die Anzahl der zu bearbeitenden Artikel von Beginn an geringer. Sobald bereits einschlägige Quellen für relevante Artikel vorliegen, kann dies beim Snowballing einen erheblichen zeitlichen Vorteil gegenüber der Systematischen Literaturrecherche bedeuten. Jedoch kann beim Snowballing eine sehr einseitige Recherche stattfinden, da alle referenzierten Artikel im gleichen Themengebiet bleiben. Ist durch die Wahl der Startartikel das Themengebiet nicht ausreichend gestreut, kann es zum genannten Nachteil führen.

Da im Rahmen des SNIK Projektes eine Expertenempfehlung für Ontologien im Kontext von Business Intelligence vorhanden ist, wird für diese Literaturrecherche das Snowballing verwendet. Jalali und Wohlin unterteilen die vollständige Vorgehensweise beim Snowballing in drei Schritte, die in der Abbildung 3.1 schematisch dargestellt und in den nachfolgenden Kapiteln genauer beschrieben werden.

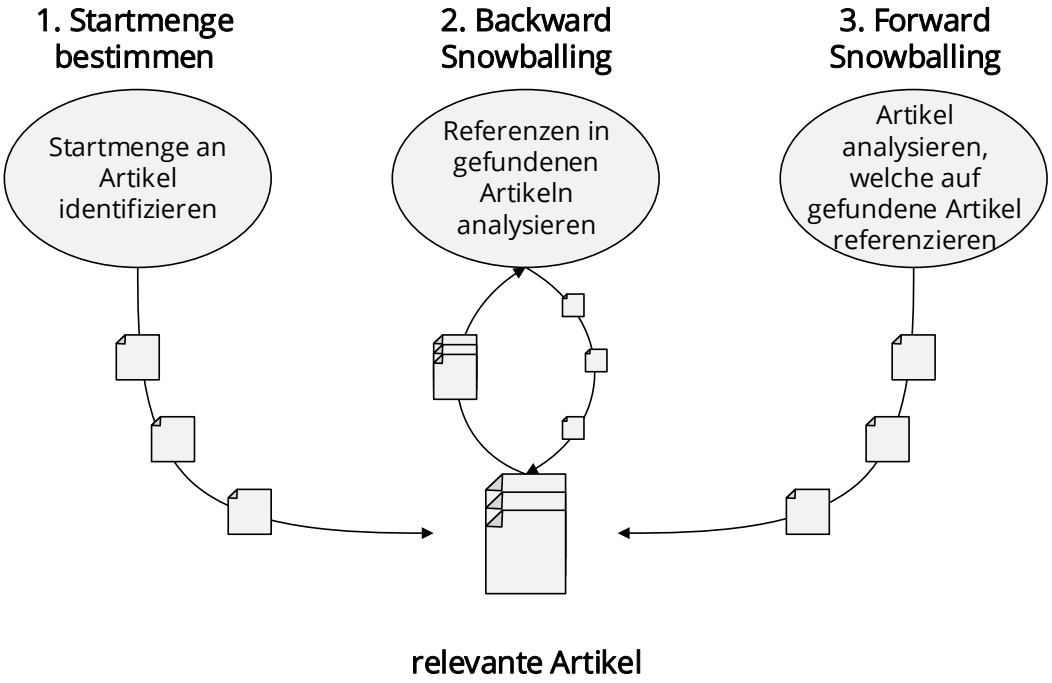


Abbildung 3.1: Ablauf einer Snowballing Literaturrecherche

### 3.2.1 Startmenge bestimmen

Zu Beginn des Snowballings muss eine Startmenge von zu analysierenden Artikeln bestimmt werden. Dazu sollten aktuelle und zum Thema passende Konferenzen oder Journals als Startpunkt für die Identifizierung der relevanten Artikeln genutzt werden. Ausgehend von den Artikeln werden diese für das Thema, anhand der festgelegten Kriterien (vgl. Kapitel 3.3), untersucht und deren Eignung für die weitere Untersuchung durch das Snowballing eingeschätzt. Am Ende werden nur die relevanten Artikel für das weitere Vorgehen verwendet. Dabei ist ein Artikel als relevant zu betrachten, sobald alle Kriterien erfüllt werden.

### 3.2.2 Backward Snowballing

Nachdem eine Startmenge an relevanten Artikeln im ersten Schritt identifiziert wurde, erfolgt für jeden Artikel die Überprüfung der referenzierten Artikel. Sobald bei der Überprüfung wiederum neue relevante Artikel nach den Kriterien identifiziert wurden, werden diese zu der Menge der relevanten Artikel gezählt. Anschließend beginnt eine neue Iteration des Backward Snowballings auf den neuen identifizierten Artikeln. Das Backward Snowballing endet, sobald alle Referenzen der relevanten Artikel überprüft wurden und keine neuen relevanten Artikel identifiziert wurden.

### 3.2.3 Forward Snowballing

Sobald durch das Backward Snowballing keine weiteren relevanten Artikel identifiziert werden, beginnt das Forward Snowballing. Dabei werden alle anderen Arbeiten, die einen relevanten Artikel referenzieren, untersucht. Dies können folglich nur neuere Artikel sein, die nach der Veröffentlichung des Artikels enstanden sind. Ist ein solcher Artikel gefunden, wird auch dieser nach den Kriterien auf neue relevante Artikel untersucht (sowohl durch das Backward als auch Forward Snowballing). Falls ein neuer relevanter Artikel gefunden wird, so wird dieser zu den relevanten Artikeln aufgenommen. Nachdem alle Artikel durch das Forward Snowballing untersucht wurden, müssen die dabei gefundenen relevanten Artikel wiederum durch das Backward Snowballing untersucht werden. Das gesamte Snowballing endet, sobald weder durch das Backward noch durch das Forward Snowballing neue relevante Artikel identifiziert werden können.

| ID | Thema | Beschreibung | Ziel |
|---|---|---|---|
| T1 | Navigation in Onotologien | Alle Ansätze zum Navigieren (Suchen und Finden von Klassen, finden von Pfaden) in Ontologien müssen analysiert werden. | Z1, Z2 |
| T2 | Navigation in Graphen | Da Ontologien als Graphen verstanden werden können, müssen alle Ansätze zur Navigation in Graphen ebenfalls analysiert werden. | Z1, Z2 |
| T3 | Pfadfindung in Ontologien | Alle Ansätze zum Auffinden von Pfaden in einer Ontologie müssen betrachtet werden. | Z2 |
| T4 | Pfadfindung in Graphen | Wie beim Navigieren, ist auch die Pfadfindung in Graphen relevant. | Z2 |
| T5 | Kriterien für Pfadfindung | Alle Informationen zu Kriterien für die Pfadfindung müssen aufgenommen werden. | Z2 |
| T6 | Verknüpfung mit Realdaten | Die Verknüpfung von Realdaten mit Ontologien stellt zwar kein direktes Ziel der Arbeit dar, jedoch können mögliche Ansätze wichtige Informationen, für die Verbindung des Ausschnittes der Ontologie (vgl. Kapitel 2.2.3) mit Beispieldaten, liefern. | - |
| T7 | Repräsentation von Ontologien | Zwar muss keine (aufwendige) grafische Darstellung der Ontologie und der gefunden Pfade stattfinden, aber auch nicht-grafische Darstellungsmethoden sollten näher betrachtet werden. | Z3 |

Tabelle 3.1: Themen für die Einschlusskriterien

## 3.3 Kriterien für die Relevanz von Artikeln

Durch die Literaturrecherche werden häufig sehr viele Artikel gefunden. Um aus dieser Vielzahl an Artikeln nur die relevanten Artikel zu identifizieren, müssen Kriterien definiert werden. Die Kriterien beinhalten sowohl allgemeine Kriterien wie z.B. die Sprache und Verfügbarkeit, als auch die aus den Zielen abgeleiteten Kriterien. Für die Kriterien ist eine Definition von Themengebieten notwendig. Dazu werden zunächst in Tabelle 3.1 alle für diese Arbeit relevanten Themen aus den Zielen abgeleitet. Im nächsten Schritt werden diese Themen bei den Kriterien wiederverwendet. Dabei wird sowohl die Betrachtung für Ontologien als auch für Graphen durchgeführt, da Ontologien auch als Graphen darge-

stellt werden können (vgl. Kapitel 2.2.2). In Tabelle 3.2 werden die Kriterien für relevante Artikel dargestellt. Die Kriterien K1 und K2 stellen den Zusammenhang der Artikel zu den definierten Themen T1-T7 dar. Beide Kriterien müssen mindestens ein Thema abdecken. Die Kriterien K3-K5 sollen eine grundlegende Mindestqualität und Verfügbarkeit der Artikel gewährleisten. Dabei wird nach dem Lesen des Titels und der Zusammenfassung (engl. Abstract) geprüft, ob ein Beitrag zu den Themen aus der Tabelle 3.1 existiert. Sollte nach dem Lesen vom Titel und Abstract dennoch Unklarheit bestehen, wird die Zusammenfassung (engl. Conclusion) als Bewertungsgrundlage herangezogen. Damit ein Artikel als relevant eingestuft wird, müssen alle Kriterien K1-K5 gemeinsam erfüllt sein. Es ist auch möglich, dass ein Artikel zwar die Kriterien K1-K5 erfüllt, aber nach dem Lesen des gesamten Artikels dennoch kein Mehrwert für die Forschungsfragen bietet. Diese Situation kann entstehen, wenn sowohl Titel (K1), Abstract (K1) und die Zusammenfassung (K2) zwar einen Bezug zu den Themen T1-T7 aufweisen und somit erfüllt sind, aber im Artikel selbst dazu keine neueren Informationen entnommen werden können. In diesem Fall wird dieser Artikel als nicht relevant eingestuft. Sollten aber im Artikel die definierten Themen über Referenzen angedeutet sein, so kann dennoch von dem Artikel ein weiteres Snowballing durchgeführt werden.

| ID | Beschreibung |
|---|---|
| K1 | Der Artikel muss einen Beitrag zu mindestens einem Thema T1-T7 aus Tabelle 3.1 im Titel und/oder Abstract aufweisen. |
| K2 | Der Artikel muss in der Zusammenfassung Bezug zu mindestens einem Thema T1-T7 aus Tabelle 3.1 aufweisen. |
| K3 | Der Artikel muss in englischer oder deutscher Sprache verfasst sein. |
| K4 | Der Artikel muss Peer Reviewed sein. |
| K5 | Der Artikel muss verfügbar und zugänglich sein. |

Tabelle 3.2: Kriterien zur Beurteilung der Relevanz von Artikeln

## 3.4 Ergebnisse der Literaturrecherche

In diesem Abschnitt werden die quantitativen und qualitativen Ergebnisse der Literaturrecherche beschrieben.

| **Quelle** | **# BS** | **# FS** | **untersuchte Artikel (BS)** | **untersuchte Artikel (FS)** |
|---|---|---|---|---|
| First international workshop on Ontology-supported business intelligence [16] | 11 | | [17], **[18]** | |
| An ontology-based cluster analysis framework [17] | 11 | 16 | **[19]**, [20] | - |
| **Supporting business intelligence by providing ontology-based end-user information self-service** [18] | 19 | 20 | [21], [22] | **[5]**, [23] |
| **A comparative study of ontology based term similarity measures on PubMed document clustering** [19] | 19 | 64 | - | **[24]** |
| OSS: A Semantic Similarity Function based on Hierarchical Ontologies [20] | 17 | 106 | [25], [26], [27] | - |
| Data integration using semantic technology: a use case [21] | 6 | 18 | - | - |
| Using Semantic Web Technologies to Build Adaptable Enterprise Information Systems [22] | 41 | 5 | - | - |
| QuizRDF: Search technology for the semantic web [25] | Abbruch. Kein weiteres Snwoballing. | | | |
| Concept expansion using semantic fisheye views [26] | Abbruch. Kein weiteres Snwoballing. | | | |
| Algorithmic detection of semantic similarity [27] | Abbruch. Kein weiteres Snwoballing. | | | |
| **Ontology-enhanced user interfaces: A survey** [5] | Abbruch. Kein weiteres Snwoballing. | | | |
| Wissensbasiertes Business Intelligence für die Informations-Selbstversorgung von Entscheidungsträgern [23] | Abbruch. Kein weiteres Snwoballing. | | | |
| **Efficient Concept-based Document Ranking [24]** | Abbruch. Kein weiteres Snwoballing. | | | |

Tabelle 3.3: Verlauf des Snowballing mit den untersuchten Artikeln. #BS - Anzahl gefundener Artikel durch Backward Snowballing. #FS - Anzahl gefundener Artikel durch Farward Snowballing

### 3.4.1 Verlauf des Snowballings

**Startmenge bestimmen** Da im SNIK-Projekt bereits eine Expertenempfehlung für den Workshop „*The First International Workshop on Ontology-supported Business Intelligence*“ vorliegt, wurde dieser Workshop für die Bestimmung der Startmenge gewählt. Es wurden insgesamt 11 Artikel gelistet. Aus dieser Startmenge ergeben sich die beiden Artikel „*An ontology-based cluster analysis framework*“ von Lula und Paliwoda-Pękosz [17] und „*Supporting business intelligence by providing ontology-based end-user information self-service* “ von Spahn et al. [18] als relevante Artikel. Wobei der Artikel von Lula und Paliwoda-Pękosz nach dem Lesen des gesamten Artikels als nicht relevant eingestuft wurde. Trotzdem weist der Artikel einige Referenzen zu den ausgewählten Themen auf. Aus dem Grund wurde das Snowballing ausgehend von [17] durchgeführt. Diese beiden Artikel bilden die Startmenge für das Backward Snowballing.

**Backward Snowballing** Ausgehend von [17] und [18] wurden insgesamt drei Iterationen des Backward Snowballings durchgeführt. Dabei wurde in der ersten Iteration insgesamt vier neue relevante Artikel identifiziert. Wobei auch diese, nach dem vollständigen Lesen des Artikels, als irrelevant klassifiziert wurden, aber trotzdem in den Referenzen Bezug zu den definierten Themen aufweisen. Ausgehend von den vier Artikeln wurde die zweite Iteration durchgeführt. Dabei wurden drei neue relevante Artikel identifiziert, die sich in der nachfolgenden dritten Iteration als nicht relevant herausgestellt haben. Aufgrund dessen, dass in der letzten Iteration die untersuchten Artikel keine Relevanz aufgezeigt haben, wurde an diesem Punkt das Backward Snowballing beendet.

**Forward Snowballing** Ausgehend aus den sechs (zwei aus der Startmenge plus vier aus dem Backward Snowballing) identifizierten Artikeln wurde das Forward Snowballing durchgeführt. Dabei wurde die Suche von Google Scholar[6] verwendetet. Insgesamt wurden drei neue Artikel als relevante Artikel identifiziert, wobei wiederum nach dem Lesen der gesamten Artikeln ein Artikel keinen Bezug zu den Themen aufwies. An dieser Stelle wurde das Forward Snowballing beendet. Es wurde auf die neu identifizierten Artikel keine neue Iteration des gesamten Snowballings durchgeführt. Stattdessen wurden die Referenzen der jeweiligen Artikel untersucht.

Insgesamt wurden durch das gesamte Snowballing 353 Artikel nach den Kriterien K1-K5 untersucht. Aus den 353 Artikel wurden 12 Artikel nach dem Lesen von Titel, Abstract und ggf. Conclusion als relevant eingestuft. Erst nach dem Lesen der vollständigen Artikeln wurden aus den 12 Artikeln nur noch vier als relevant beibehalten. Die Erkenntnisse aus den Artikeln werden im folgenden Abschnitt 3.4.2 beschrieben. In Tabelle 3.3 ist der Verlauf des Snowballings samt den untersuchten Artikeln dargestellt. Dabei ist zu jedem untersuchtem Artikel angegeben, wie viele Artikel durch das Backward Snowballing (Spalte # BS) und das Farward Snowballing (Spalte # FS) insgesamt referenziert sind. Alle referenzierten Artikel wurden untersucht und daraus jeweils weitere Artikel für das Snowballing entnommen (Spalte Untersuchte Artikel). Wurden keine Artikel für das weitere Untersuchen gefunden, ist dies mit einem „-“ Symbol markiert. Die hervorgehobenen Artikel sind Artikel, die zur Beantwortung der Forschungsfragen beitragen.

### 3.4.2 Ergebnisse des Snowballings

**Ontologie Darstellung** Paulheim und Probst [5] untersuchen Ontologien und deren Auswirkung auf das Design der Nutzeroberfläche. Dabei zeigen sie, dass Ontologien nicht nur zur Gestaltung der Nutzeroberflächen herangezogen werden, sondern auch abhängig vom Einsatzzwecke direkt dem Nutzer präsentiert werden. Dabei gibt es vier Arten für die Darstellung von Ontologien:

- Listendarstellung
- Grafische Darstellung, z.B. als Graph
- verbale Darstellung (bei der Beziehungen im Wortlaut dem Nutzer dargestellt werden)

[6] https://scholar.google.de/

- Quellcode Darstellung, z.B. als RDF/OWL

Die Darstellung als Quellcode ist in den meisten Fällen unpraktisch, weil hierfür technisches Hintergrundwissen des Nutzers vorausgesetzt wird. In der Untersuchung zeigt sich, dass die Listendarstellung am häufigsten verwendet wird, weil diese einen geringeren Entwicklungsaufwand benötigt. Die grafische Darstellung benötigt zwar einen höheren Entwicklungsaufwand, bietet aber gerade bei größeren Ontologien eine bessere Übersicht für den Nutzer und sollte somit nach Möglichkeit präferiert werden. Des weiteren definieren Paulheim und Probst drei wesentliche Vorteile von Ontologien im Kontext von Nutzeroberflächen:

- Verbesserung der Visualisierungsmöglichkeiten
- Verbesserung der Interatktionsmöglichkeiten
- Verbesserung des Entwicklungsprozesses

**Beispiel für Ontologien mit Nutzerinteraktion** Spahn et al. [18] präsentieren eine Anwendung für die Interaktion von Nutzern mit Ontologien. Das Ziel der Anwendung ist, dass der Nutzer ohne technisches Vorwissen bezüglich Anfragen auf Datenquellen an die gewünschten Daten gelangt. Dabei dienen mehrere Ontologien auf unterschiedlichen Ebenen der Erstellung von Anfragen. Der Nutzer bekommt die Möglichkeit in einer Ontologie in Graph-Form zu Navigieren und aus der Ontologie die gewünschten Daten zu erhalten. Im Hintergrund wird die Auswahl des Nutzer auf der Ontologie in eine Business Ontologie (BO) transformiert. Die BO verbindet mehrere technische Ontologien (TO) zu einer Ontologie, auf der die Anfragen des Nutzers ausgewertet werden. Die technischen Ontologien stellen echte Datenquellen aus unterschiedlichen Systemen (z.B. Datenbanken, Anwendungen) jeweils als Ontologien dar. Durch diese Verbindung der einzelnen Ontologien zu einer einzigen Ontologie ist es möglich, dass der Nutzer Zugriff auf die Daten bekommt ohne, dass dieser Hintergrundwissen im Bezug auf die tatsächliche Datenquelle und deren technischen Besonderheiten haben muss. In Abbildung 3.2 wird der Aufbau der Anwendung und die Einordnung der einzelnen Ontologien (BO, TO, Ontologie Graph) dargestellt.

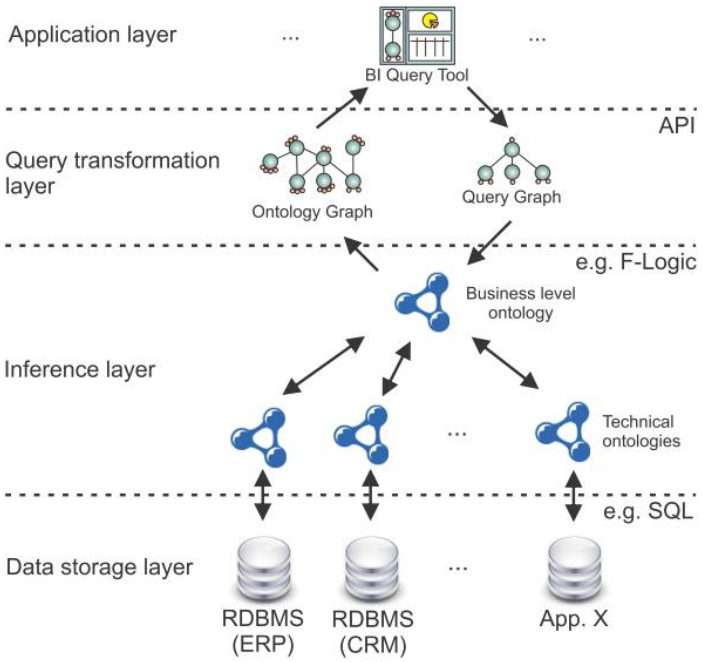


Abbildung 3.2: Architekturschichten von mit der Einbindung der unterschiedlichen Ontologien [18]

Spahn et al. bieten dem Nutzer in ihrer Anwendung die Möglichkeit direkt auf der graphenbasierten Darstellung der Ontologie zu navigieren. Sie stellen dem Nutzer unter anderem eine Vorschau der Daten von der ausgewählten Klasse dar. Ebenso ermöglichen sie es auch Pfade zwischen zwei ausgewählten Klassen zu finden, was jedoch in dem Artikel nicht näher erläutert wird. Dennoch hat die Arbeit von Spahn et al. eine starke Ähnlichkeit mit CIONw. Der größte Unterschied besteht hingegen aus der Vorgehensweise zur Erstellung der Ontologien. Die im SNIK Projekt entstandene Ontologie repräsentiert Wissen ohne die Betrachtung von möglichen konkreten Datenquellen. Die Datenquellen müssen nachträglich auf die Ontologie abgebildet werden. Spahn et al. hingegen verwenden einen Bottom-Up Ansatz: Sie bilden die Datenquellen schrittweise in Ontologien ab, so dass am Ende eine Ontologie entsteht, die für die konkrete Datenquellen angepasst ist. Trotzdem kann ihre Arbeit als ein gutes Vorbild für den Entwurf der Nutzeroberfläche verwendet werden.

**Änlichkeitsbestimmung in Ontologien** Bei der Snowballing Literaturrecherche ist ein weiterer Schwerpunkt auf die Bestimmung von ähnlichen Begriffen bzw. Klassen in einer Ontologie entstanden. Dabei werden Begriffe in Ontlogien verglichen und ihre Ähnlichkeit zueinander bestimmt. Die Ähnlichkeit kann aufgrund der Syntax oder der semantischen Bedeutung erfolgen. Die Relevanz für diesen Fokus entstand auf der Basis, dass einige Verfahren den Pfad zwischen den zu vergleichenden Klassen in die Berechnung der Ähnlichkeit einbeziehen. Zhang et al. [19] haben die unterschiedlichen Vergleichsverfahren untersucht und dabei gezeigt, dass die meisten Verfahren die kürzeste Distanz zwischen den zu vergleichenden Klassen berechnen. Dabei spielt sowohl die Distanz der Klassen zueinander eine Rolle, als auch die Distanz von den beiden Klassen zur Wurzelklasse in einer hierachischen Ontologie. Ebenso gibt es Ähnlichkeitsbestimmungen, die die Distanz zweier Klassen anhand eines kleinsten gemeinsamen Vorfahren in der Ontologie betrachten (welches wiederum nur auf hierarchischen Ontologien möglich ist). In allen Verfahren wird gezeigt, dass der Vergleich von Klassen anhand von Pfaden immer auf dem kürzesten Pfad basiert.

Mit der Thematik der Ähnlichkeitsbestimmung wurde auch der Artikel von Arvanitis et al. [24] durch das Snowballing gefunden. Arvanitis et al. beschreiben u.a. ein Verfahren, um $k$-nächste Klassen zu einer Anfrage zu finden. Dies bedeutet, dass nicht nur die ähnlichste Klasse gefunden wird, sondern eine festgelegte Anzahl $k$ Klassen mit absteigender Ähnlichkeit gefunden werden. Aus der Kombination von der Ähnlichkeitsbestimmung, anhand der kürzesten Distanz zweier Klassen, mit der Bestimmung von $k$-nächsten Klassen, entstehen $k$-kürzeste Wege. Arvantis et al. suchen ausgehend von einer Frage mögliche Antworten in den Klassen der Ontologie. Dabei wird die am besten zutreffende Klasse als Antwort ausgegeben. Jedoch ist diese Situation für CIONw nicht anwendbar, weil in CIONw die Antwort auf eine Fragestellung nicht durch eine einzige Klasse gefunden wird, sondern durch einen oder mehrere Pfade. Die oben genannte Kombination von Ähnlichkeitsbestimmung und $k$-nächste Klassen kann als Idee genutzt werden um zwischen zwei Klassen einer Ontologie $k$-kürzeste Pfade zu finden. Sobald ein ausreichend großes $k$ gewählt wird, können alle Pfade gefunden werden.

### 3.4.3 Beantwortung der Forschungsfragen

**RQ1 wird wie folgt beantwortet:** Das Snowballing hat mit dem Artikel von Spahn et al. ein Beispiel für eine ähnliche Problematik wie CIONw aufgedeckt und hat somit eine Antwort auf die Forschungsfrage RQ1 geliefert. Durch die Kombination von mehreren Ontologien auf unterschiedlichen Ebenen, zu einer, für den Nutzer verständlichen, Ontologie, kann dem Nutzer die Navigation erleichtert werden. Der Nutzer muss keine komplexe Logik zur Datenabfrage und Datenintegration kennen. Insbesondere ist es ausstreichend dem Nutzer die Ontologie in grafischer Form darzulegen und nur bei Bedarf detalierte Informationen darzustellen.

**RQ2 wird wie folgt beantwortet:** Mit der Möglichkeit der Kombination aus Ähnlichkeitsbestimmung anhand von Distanzen und der Auffindung von $k$-nächsten Klassen können $k$-kürzeste Pfade innerhalb einer Ontologie gefunden werden. In den betrachteten Artikeln wird nicht direkt auf Kriterien für die Navigation oder das Bestimmen von Pfaden zwischen Klassen eingegangen, jedoch ist erkennbar, dass im allgemeinen nur die kürzesten Pfade gefunden werden. Dabei ist das wichtigste Kriterium für die Bestimmung eines kürzesten Pfades die Berechnung durch eine Distanzfunktion. Diese wird im allgemeinen durch die Anzahl an Klassen zwischen den zu vergleichenden Klassen bestimmt. Somit ergibt sich eine Teilantwort auf RQ2. Insbesondere kann aufbauend auf der Idee der $k$-kürzeste Pfade in den Grundlagen der Graphentheorie nach Algorithmen für die Bestimmung jener Pfade gesucht werden.

**Fazit Snowballing** Die Snowballing Methode zeigt, dass die Ergebnisse stark von der Ausgangsmenge abhängig sind und insbesondere das Risiko besteht, in bestimmte Themengebiet zu vertiefen, die nur indirekt auf die eigentlichen Forschungsfragen eingehen. Dennoch liefert das Snowballing Antworten für die definierten Forschungsfragen RQ1 und RQ2.

# 4 Anforderungserhebung

In diesem Kapitel werden die Zielsetzungen *Z1 - Anforderungsspezifikation* und *Z2 - Navigation in Ontologien* (vgl. Kapitel 1.2) bearbeitet. Dabei wird im ersten Teil das Vorgehen zur Anforderungserhebung beschrieben und im zweiten Teil die Ergebnisse der Anforderungserhebung detailliert beschrieben.

## 4.1 Vorgehen zur Anforderungserhebung

Bei der Anforderungserhebung erfolgte die Planung und Durchführung basierend auf der Methodik nach TORE (vgl. Kapitel 2.3). Dazu wurde als Erhebungsmethode die Interviewbefragung gewählt.

### 4.1.1 Dokumenten-basiertes Requirements Engineering

Im Rahmen des SNIK Projektes entstand die Idee zu CIONw. Dabei wurde die Idee in einem Telefonkonferenz-Protokoll ausformuliert. Aus dem Protokoll und einer weiteren schematischen Zeichnung der Ideen, wurden die ersten Schritte der Anforderungserhebung durchgeführt. Diese Dokumente dienten dem grundlegenden Verständnis, was CIONw ist bzw. sein soll. Darüber hinaus dienen diese Dokumente der Bildung einer ersten Lösungsvorstellung und der Identifizierung von damit verbundenen Unklarheiten. Die Unklarheiten stellten auch die ersten Grundlagen für offene Fragestellungen für die darauffolgenden Interviews dar. Da bereits eine dokumentierte Beschreibung vorliegt, müssen die Fragen für die Interviews bereits bekanntes Wissen nicht nochmals abfragen, sondern bestätigt oder angepasst werden.

### 4.1.2 Interview-Vorbereitung

Ausgehend von den vorhandenen Dokumenten wurden die Stakeholder klar festgelegt und in zwei Gruppen eingeteilt: primäre Stakeholder und sekundäre Stakeholder. Ausgehend von dieser Aufteilung wurde die Interviewplanung wie folgt aufgebaut:

1. Interview: jeweils getrennte Befragung von primären und sekundären Stakeholdern

2. Interview: mit Teilnehmern aus beiden Stakeholdergruppen
3. Interview: mit primären Stakeholdern

In den nachfolgenden Abschnitten werden die Inhalte der drei Interviews festgelegt und beschrieben.

**1. Interview** In den ersten Interviews wurden die beiden Stakeholdergruppen getrennt befragt. Damit werden erste Ideen der Stakeholder gesammelt, ohne dass eine gegenseitige Beeinflussung stattfindet. Zusätzlich wurde mit dieser Trennung versucht gezielt mögliche Konflikte oder gar konkurrierende Ideen für CIONw zu identifizieren. Die beiden Interviews der beiden Stakeholdergruppen umfassen die gleichen Fragen.

Der Fragekatalog für das Interview wurde in drei Schritten erstellt:

1. via Brainstorming Fragen erstellen
2. Einordnung der Fragen nach TORE Entscheidungspunkten
3. Ergänzung von Fragen für die einzelnen Entscheidungspunkte, welche nicht abgedeckt wurden (begrenzt auf Task u. Goal Ebene, Domain Ebene und Interaction Ebene)

TORE fokussiert zunächst eine Betrachtung der Nutzeraufgaben und Zielen, die unabhängig von einer konkreten Lösung des Systems verstanden werden müssen. Daher wird die System Ebene in den ersten Interviews nicht forciert.

Ziel des ersten Interviews ist, ein Verständnis über die Nutzeraufgaben, den benötigten Daten und deren Zusammenhänge zueinander im Rahmen von CIONw zu erhalten. Insbesondere sollen die Ziele von CIONw und die Aufgaben, die ein Nutzer damit erfüllt, identifiziert werden. Im ersten Interview werden keine bekannten Umsetzungsprobleme (Implementierungsdetails) besprochen.

**2. Interview** Die Ergebnisse der beiden ersten Interviews wurden aufgearbeitet und dabei die Unterschiede in den Ergebnissen herausgearbeitet. Vorhandene Unklarheiten oder Unstimmigkeiten dienten als Basis für den neuen Fragekatalog. Wobei die Fragen nur geringfügig angepasst sind und durch das zweite Interview vertiefend geklärt werden sollen. Neben der Aufarbeitung wurden nun auch detailliertere Fragen zur Interaction Ebene und System Ebenen gestellt. In diesem Interview wurden auch mögliche Umsetzungsideen betrachtet.

| Interview | Datum | Stakeholder | Dauer |
|---|---|---|---|
| Interview 1 | 22.06.16 | primäre Stakeholder | 100 Min. |
| Interview 1 | 24.06.16 | sekundäre Stakeholder | 85 Min. |
| Interview 2 | 04.07.16 | primäre u. sekundäre Stakeholder | 90 Min. |
| Interview 3 | 07.07.16 | primäre Stakeholder | 60 Min. |

Tabelle 4.1: Übersicht zu den durchgeführten Interviews

**3. Interview** Das letzte Interview diente der vertiefenden Betrachtung von Fragen zu Interactions und System Ebene. Dabei sollen am Ende alle offenen Fragen ausreichend beantwortet sein, so dass die Dokumentation der Anforderungen abgeschlossen werden und auf dieser Basis eine Implementierung erfolgen kann.

### 4.1.3 Durchführung der Interviews

Für die einzelnen Interviews wurden jeweils 90 bis 120 Minuten Termine vereinbart. Das erste Interview mit den primären Stakeholdern dauerte ca. 100 Minuten, dabei wurden fast alle vorbereiteten Fragen initial durchgesprochen. Die gestellten Fragen sind im Anhang A.2.1 dargestellt. Das gleiche Interview mit den sekundären Stakeholdern wurde in ca. 85 Minuten durchgeführt.

Das zweite Interview mit beiden Stakeholderngruppen dauerte ca. 90 Minuten. Es wurden ebenfalls fast alle vorbereiteten Fragen durchgesprochen, zusätzlich wurden einige Rückfragen zu bereits gestellten Fragen gestellt. Im Anhang A.2.2 befinden sich die gestellten Fragen.

Das letzte Interview mit den primären Stakeholdern war aufgrund der geringen Anzahl an Fragen innerhalb einer Stunde durchgeführt. Im Anhang A.2.3 sind die besprochenen Fragen aufgelistet. In der Tabelle 4.1 ist eine Übersicht über die durchgeführten Interviews dargestellt.

## 4.2 Ergebnisse der Anforderungserhebung

Auf Basis der Antworten aus allen drei Interviews erfolgte eine Definition und Dokumentation der Anforderungen. Für die Dokumentation der Nutzeraufgaben wird der Ansatz „Task & Support“ von Lauesen verwendet [11] (vgl. Kapitel 2.4.1).

Alle nachfolgenden definierten Artefakte sind zusätzlich in Jira dokumentiert[7].

[7] http://jira-se.ifi.uni-heidelberg.de/projects/MAZEISER16

| **T1** | **Navigation in einer Ontologie** |
|---|---|
| Start | Ontologie liegt in Graphen-Form vor |
| Ende | Der Nutzer kann sich innerhalb der Ontologie orientieren bzw. navigieren |
| Frequenz | regelmäßig |
| Schwierigkeit | In großen Ontologien kann die Übersicht verloren gehen |

| **Subtask** | **Beispiellösung** |
|---|---|
| 1. Klassen in der Ontologie finden | Das System bietet eine Suchmaske an und zeigt gefundene Klassen an |
| 2. Nachbarn zu einer ausgewählten Klasse finden | Das System zeigt direkten Nachbarn (Nachbarschaftsgrad: 1) an, um den Nutzer bei der Orientierung in der Ontologie zu unterstützen |
| 3. Instanzen zu konkreten Klassen finden | Das System zeigt zu einer ausgewählten Klasse (falls vorhanden) die hinterlegten Instanzdaten an |

Abbildung 4.1: Aufgabe T1 - Navigation in einer Ontologie

### 4.2.1 Task und Subtasks

Nachfolgend sind die drei identifizierten Aufgaben (T1, T2 und T3) nach der Vorlage von Lauesen dargestellt.

**T1 - Navigation in einer Ontologie** Die Aufgabe T1 in Abbildung 4.1 beschreibt die Möglichkeit, dass der Nutzer sich innerhalb der Ontologie zurechtfindet. Dafür soll es möglich sein, Klassen zu suchen, zu ausgewählten Klassen zusätzliche Instanzen angezeigt zu bekommen und ebenfalls die direkten Nachbarn einer ausgewählten Klasse dargestellt zu bekommen. Diese Möglichkeiten sollen vor allem den Überblick in größeren Ontologien ermöglichen.

**T2 - Pfade in Ontologien untersuchen** In Abbildung 4.2 wird die Aufgabe beschrieben, dass der Nutzer gewünschte Pfade innerhalb Ontologie auffinden kann. Dabei soll der Nutzer durch eine unterschiedliche Gewichtung von Kanten die Variation der Pfade ermöglichen. Ebenso sollen hier weitere Einstellungen ermöglicht werden: Maximale Pfadlänge, maximale Anzahl an Pfaden, Start-, Zwischen- und Zielknoten auswählen. Dadurch soll die Pfadsuche der gewünschte Fragestellung des Nutzers angepasst werden. Für einen gefundenen Pfad sollen die hinterlegten Instanzen entlang des Pfades als Datensatz angezeigt werden. Dadurch kann der gefundene Pfad mitsamt den dabei vorhandenen Daten vom Nutzer detaillierter Untersucht werden.

| **T2** | **Pfade in Ontologien untersuchen** |
|---|---|
| Start | Ontologie liegt in Graphform vor |
| Ende | Pfade zwischen ausgewählten Klassen werden gefunden und dargestellt |
| Frequenz | regelmäßig |
| Schwierigkeit | große Vielzahl an möglichen Pfaden |

| **Subtask** | **Beispiellösung** |
|---|---|
| 1. Einstellungen für die Pfadsuche machen | Einstellungsmöglichkeiten für Kantengewichte, maximale Pfadlänge, Auswahl von Start- und Endeklasse, Auswahl von Zwischenklassen |
| 2. Pfade finden | Durch Standard-Pfadalgorithmen der Graphentheorie Pfade finden |
| 3. Auswerten der Instanzen entlang des Pfades | Instanzen entlang des Pfades auslesen und verknüpfen. Verknüpften Instanzpfad in CSV exportieren |

Abbildung 4.2: Aufgabe T2 - Pfade in Ontologien untersuchen

**T3 - Fehlende Instanzen auffinden** Die Aufgabe in Abbildung 4.3 beschreibt die Möglichkeit fehlende Instanz zu einer Ontologie aufzudecken. Dabei muss für die SNIK-Ontologie darauf geachtet werden, dass Klassen existieren, welche nicht instanziiert werden können. Dies sind vom Typ `Function` und stellen damit Aufgaben dar. Aufgaben können keine Instanzen besitzen und müssen beim verknüpfen der Instanzen entlang eines Pfades gesondert bearbeitet werden.

### 4.2.2 User Stories

Auf Grundlage der identifizierten Aufgaben und den Ergebnissen der Interviews, sind User Stories definiert worden. Die Interviewergebnisse sind so durch die User Stories greifbarer und begründen ggf. bestimmte Entscheidungen. Insbesondere stellen User Stories kleinere Anforderungen oder Wünsche der Stakeholder durch einen geringen Aufwand dar. In TORE werden User Stories als Beschreibungsmittel nicht erwähnt, aber dennoch können diese als Grundlage für die weiteren Entscheidungspunkte genutzt werden. Denn durch die Angabe von einer Begründung innerhalb der User Story kann die Motivation für eine Anforderung besser nachvollzogen werden. In der Tabelle 4.2 sind die erstellten User Stories und deren Verbindung zu den Tasks dargestellt.

| ID | User Story | unterstützte Task |
|---|---|---|
| US01 | Als Nutzer möchte ich in der Ontologie Klassen suchen, um die Klasse eines Start- oder Endknoten zu finden oder Klassen auf ihr Vorhandensein zu prüfen. | T1 |
| US02 | Als Nutzer möchte ich zu einer ausgewählten Klasse der Ontologie die Nachbarklassen sehen, um die Zusammenhänge der Klasse zu anderen Klassen erkennen zu können. | T1 |
| US03 | Als Nutzer möchte ich zu einer ausgewählten Klasse der Ontologie alle Instanzen sehen. | T1 |
| US04 | Als Nutzer möchte ich Pfade zwischen zwei Klassen der Ontologie finden, um deren Zusammenhang zu verstehen. | T2 |
| US05 | Als Nutzer möchte ich Pfade zwischen zwei Klassen der Ontologie über eine oder mehrere ausgewählten Klassen dazwischen finden. | T2 |
| US06 | Als Nutzer möchte ich die Instanzen aller auf dem gefundenen Pfad liegenden Klassen angezeigt bekommen. | T2 |
| US07 | Als Nutzer möchte ich zwischen mehreren Pfaden auswählen können, um den ausgewählten Pfad und die dazugehörenden Instanzen für die weitere Verarbeitung, exportieren zu können. | T2 |
| US08 | Als Nutzer möchte ich die zusammenhängenden Instanzen mit den Relationen eines Pfades als CSV-Datei ausgegeben bekommen. | T2 |
| US09 | Als Nutzer möchte ich eine maximale Länge der Pfade angeben können, um die Anzahl der gültigen Pfaden zu reduzieren. | T2 |
| US10 | Als Nutzer möchte ich die gefundenen Pfade nach unterschiedlichen Kriterien sortieren können. | T2 |
| US11 | Als Nutzer möchte ich die Gewichtungen der Kanten von der Ontologie anpassen können, um Einfluss auf die Pfadfindung zu haben. | T2 |
| US12 | Als Nutzer möchte ich über fehlende Instanzdaten auf dem Pfad informiert werden. | T2 |

Tabelle 4.2: User Stories

| **T3** | **Fehlende Instanzen auffinden** |
|---|---|
| Start | Vorhandener Pfad liegt vor |
| Ende | Fehlende Instanzen auf einem Pfad werden gemeldet |
| Frequenz | unregelmäßig |
| Schwierigkeit | Erkennen von nicht instanziierbaren Klassen |

| **Subtask** | **Beispiellösung** |
|---|---|
| 1. Nicht instanziierbare Klassen erkennen | Klassen vom Typ Aufgaben (`Function` vlg. 2.2.1) können nicht instanziiert werden |
| 2. Klassen ohne hinterlegte Instanzen erkennen | Keine Beispiellösung vorhanden |
| 3. Fehlende Relationsinstanzen, für die Verknüpfung eines Pfades auf Instanzebene, erkennen | Keine Beispiellösung vorhanden |

Abbildung 4.3: Aufgabe T3 - Fehlende Instanzen auffinden

### 4.2.3 Systemfunktionen

Aus den Task und User Stories resultieren die nachfolgenden Systemfunktionen. Die Systemfunktionen sind dem TORE Entscheidungspunkt `System Function` auf der Interaktionsebene zugeordnet und beschreiben die Funktionen des Systems, die die einzelnen Aufgaben unterstützen. In der Tabelle 4.3 sind die einzelnen Systemfunktionen und deren Zuordnungen zu den Tasks dargestellt. Dabei sind die Systemfunktionen SF11 und SF12 im Rahmen einer Zwischenpräsentation des Entwicklungsstandes entstanden.

| ID | Systemfunktion | unterstützte Task |
|---|---|---|
| SF01 | Grafische Darstellung der Ontologie bzw. einen Teilausschnitt der Ontologie | T1 |
| SF02 | Suchen einer Klasse in der Ontologie anhand des Klassennamens | T1 |
| SF03 | Klasse mit der Nachbarschaftsbeziehung 1 Grades zu einer ausgewählten Klasse finden und in der Ontologie grafisch darstellen | T1 |
| SF04 | Instanzen zu einer ausgewählten Klasse finden bzw. auslesen | T1 |
| SF05 | Suchen von möglichen Pfaden zwischen markierten Klassen unter Berücksichtigung der max. Pfadlänge und Gewichtung der Kanten. Die Anzahl der gefundenen Pfade kann begrenzt werden | T1 |
| SF06 | Festlegung der Kriterien für die Pfadsuche: Gewichtung der Kantentypen und maximale Pfadlänge | T1 |
| SF07 | Sortieren der Pfadliste nach Länge, Gewichtung des Pfades und Vollständigkeit der Instanzdaten | T2 |
| SF08 | Ausgewählte Pfade mit Instanzdaten per natural Join verknüpfen | T2 |
| SF09 | Export eines ausgewählten Pfades als CSV-Datei | T2 |
| SF10 | Klassen vom Typ `Function` erkennen und anzeigen | T3 |
| SF11 | Gefundenen Pfad in Ontologie grafisch hervorheben | T2 |
| SF12 | Einstellungen von Kantengewichtungen speichern | T2 |

Tabelle 4.3: Systemfunktionen

# 5 Lösungsentwurf

In diesem Kapitel werden aus den in Kapitel 4 dargestellten Anforderungen, die wesentlichen Problemstellungen extrahiert, welche für eine Implementierung zunächst gelöst werden müssen. Darauf aufbauend werden dazu Lösungen präsentiert. Die daran anknüpfende Entscheidung zwischen einer vollständigen Eigenentwicklung der Software oder einer Erweiterung einer bereits existierenden Software wird ebenfalls in diesem Kapitel diskutiert und getroffen.

## 5.1 Problemstellung: Pfadsuche

Für die Umsetzung von Ziel *Z2 - Navigation in Ontologien* und insbesondere der Problematik vom Auffinden von Pfaden zwischen ausgewählten Klassen muss ein Verfahren entwickelt werden. Dabei sollen keine Algorithmen neu entwickelt werden, sondern bereits existierende Algorithmen an die Anforderungen angepasst werden. Wie sich in der Literaturrecherche gezeigt hat, werden in Ontologien überwiegend nur die kürzesten Pfade zwischen zwei Klassen gesucht. Für das Auffinden von Pfaden in der Ontologie wird auf dieses Vorgehen aufgebaut. Dabei wird vorerst die Betrachtung nur für eine Start- und eine Zielklasse durchgeführt und erst im Anschluss eine Erweiterungsmöglichkeit für Zwischenklassen untersucht. Da Ontologien als Graphen interpretiert werden können, eignen sich für die Pfadsuche die Standardalgorithmen der Graphentheorie.

### 5.1.1 Dijkstra Algorithmus

Das Auffinden des kürzesten Pfades von einem Knoten zu einem anderen Knoten in einem Graphen ist eine grundlegende Aufgabe in der Graphentheorie. Das bekannteste Verfahren hierfür ist der Algorithmus von Dijkstra [28]. Der Dijkstra Algorithmus kann nur auf nicht-negative Kantengewichte innerhalb eines Graphens erfolgreich einen kürzesten Weg finden. Das Verfahren baut iterativ eine Menge von günstigen Kanten auf und bestimmt anhand dieser Menge die nächste günstige Kante. Bei einem solchen Vorgehen spricht man von dem Greedy-Prinzip [28], da immer nur die aktuell bekannteste „billigste“ Kante für den Pfad gewählt wird. Der Dijkstra Algorithmus findet durch das

iterative Vorgehen zu einem Knoten die Distanz zu allen anderen Knoten des Graphen.

In Algorithmus 1 ist der Dijkstra Algorithmus in Pseudocode beschrieben.

**Algorithmus 1** Dijsktra Algorithmus

```
1: function DIJKSTRA(G, S)
2:     initialise(Q, P, D);
3:     while Q.notEmpty do
4:         u ← Q.getMinimum(); Q.remove(u);
5:         for each v from u.getOutgoingEdges() do
6:             if D[u] + dist(u, v) < D[v] then
7:                 D[v] ← D[u] + dist(u, v);
8:                 P[v] ← u;
9:     return D, P
```

Beim dargestellten Pseudocode wird dem Algorithmus der Graph $G$ und der Startknoten $S$ als Eingabe übergeben. Zu Beginn müssen drei Datenstrukturen initialisiert werden (Zeile 2). Dabei ist $Q$, eine Prioritätenwarteschlange, welche den nächsten am günstigsten erreichbaren Knoten beinhaltet. In $P$ wird zu jedem Knoten ein Vorgängerknoten gespeichert. $D$ beinhaltet die aktuelle bekannte Distanz vom Startknoten zu jeden anderen Knoten im Graphen. Zu Beginn ist $Q = G$, $P[i] = \infty$ für alle Kanten $e_i, i = 0..||E||$ und $P[j] = null$ für all Knoten $n_j, j = 0..||N||$. Im Anschluss wird, solange $Q$ nicht leer ist (Zeile 3), jeweils der aktuell günstigste Knoten aus $Q$ entnommen und für jede seiner ausgehenden Kanten die Distanz berechnet (Zeile 5-6). Ist die errechnete Distanz kürzer als die derzeitig gespeichert (Zeile 6) so wird die Distanz in $D$ gespeichert und der Vorgänger aktualisiert (Zeile 7-8). Am Ende wird $D$ und $P$ ausgegeben.

Der Kürzeste Pfad kann aus $P$ rekonstruiert werden indem vom ausgehenden Zielknoten $Z$ der Vorgänger in $P$ betrachtet wird, bis der Startknoten $S$ erreicht ist. Das Ergebnis ist dann der Weg vom Ziel zum Start und muss ggf. in umgekehrter Reihenfolge gelesen werden.

Der dargestellte Algorithmus berechnet alle Abstände vom Startknoten $S$ zu allen anderen Knoten. Da aber nur der Weg von einem Startknoten zu einem Zielknoten gesucht ist, kann der Algorithmus vorzeitig beenden, sobald der aktuell betrachtete Knoten $u$ gleich dem Zielknoten ist. Dazu muss dem Algorithmus der Zielknoten als Parameter $Z$ übergeben werden. Ebenso muss $D$ für den kürzesten Pfad nicht mit zurückgegeben werden.

### 5.1.2 Yen Algorithmus

Die Anforderungen T2, US04, US05 und SF05 besagen, dass nicht nur ein Weg, sondern mehrere (ggf. sogar alle) gefunden werden müssen. Daher reicht hierfür der Dijkstra Algorithmus nicht aus. Der Dijkstra Algorithmus findet nur einen kürzesten Weg vom Start- zum Zielknoten. Aus dem Grund wird die Idee der $k$-nächsten Klassen aus der Literaturrecherche übernommen und auf das Problem der Graphen angepasst. Dadurch müssen nun $k$-kürzeste Pfade von Start- zum Zielknoten gefunden werden.

Ein möglicher Algorithmus dafür ist der Algorithmus von Yen [29]. Der Algorithmus ist analog zum Dijkstra Algorithmus nur auf nicht-negative gewichteten Graphen anwendbar. Er findet dabei eine festgelegte Anzahl ($k$) von Pfaden zwischen zwei Knoten. Angefangen bei dem kürzesten Pfade, wird der nächst kürzere Pfad iterativ gefunden. Dies geschieht solange bis die festgelegt Anzahl an Pfaden gefunden wurde oder keine weiteren Pfade mehr existieren. Dabei gilt für jeden Pfad

$$Cost(P_i) \leq Cost(P_j), \forall i, j : i < j, i \neq j$$

wobei $Cost$ hier die Kosten für einen Pfad darstellt. Ein Pfad zählt dann als kürzester wenn er die geringsten Kosten aufweist. Wenn die Kosten auf Basis von Kantengewichten berechnet werden, sind die Gesamtkosten gleich der Gesamtgewichtung von einem Pfad.

Der Algorithmus benötigt dabei intern einen Algorithmus zur Berechnung von einem kürzesten Pfad. Dafür kann beispielsweise der oben vorgestellte Dijkstra Algorithmus verwendet werden.

Der Pseudocode Algorithmus 2 iteriert über das gegebene $K$. Für den letzten bestimmten Pfade $P_{k-1}$ wird wiederum über jeden Knoten $node$ des Pfades iteriert. Dabei wird geprüft ob der Teilpfad von $0..node$ aus $P_{k-1}$ ebenfalls in allen bisher gefundenen kürzesten Pfaden $P_j, j = 0..||A||$ ($A$ ist die Menge aller bisher gefundenen kürzeste Pfade) als Teilpfad existiert (Zeile 8), falls ja so wird die Kante zwischen den Knoten $node$ und $node + 1$ aus $P_j$ im gesamten Graphen $G$ auf $\infty$ gesetzt. Anschließend wird ein neuer kürzester Pfad berechnet. Da in $G$ nun Kanten auf $\infty$ gesetzt sind, wird der Dijkstra Algorithmus den nächsten kürzeren Pfad finden. Der gefundene Pfad wird zusammengesetzt und falls er nicht bereits in $B$ abgelegt ist, so wird er dort gespeichert (Zeile 11-14). Anschließend wird aus $B$ der nächst kürzere Pfad in $A$ übertragen (Zeile 15-19). Bevor die nächste Iteration über $K$ startet müssen alle veränderten Kanten in

**Algorithmus 2** Yen Algorithmus [29]

```
1:  function YEN(G, S, Z, K)
2:      initialise(A, B);
3:      A[0] ← Dijkstra(G, S, Z);
4:      for k = 1..K do
5:          P_{k-1} ← A[k − 1];
6:          for each node from P_{k-1} do
7:              for each P_j from A do
8:                  if P_k.subPath(0..node) = P_j.subPath(0..node) then
9:                      edge ← G.getEdgeBetween(node, P_j.Node(node + 1));
10:                     edge.setWeight(∞);
11:             root ← Path(1, node);
12:             spur ← Path(Dijkstra(G, S, Z));
13:             tPath ← join(root, spur);
14:             B.add(tPath);
15:         shortestPath ← B.getShortestPath();
16:         if A.containsNot(shortestPath) then
17:             A.add(shortestPath); B.remove(shortestPath);
18:         else
19:             B.remove(shortestPath);
20:         G.restoreAllEdges();
21:     return A
```

$G$ wiederhergestellt werden (Zeile 20). Am Ende stehen in $A$ die $k$-kürzesten Pfade jeweils nach der Länge geordnet.

### 5.1.3 Anpassung des kürzesten Pfades

Die beiden vorgestellten Algorithmen bieten zusammen die Möglichkeit die $k$-kürzesten Pfade zu finden. Aber in einer Ontologie ist nicht immer der kürzeste Pfad auch der gesuchte Pfad. Der Pfad ist abhängig von der zugrundeliegenden Fragestellung des Nutzers. So kann die Klassifizierung von bestimmten Klassen eine Fragestellung (und der damit verbundenen Auswahl von Knoten) sein, bei welcher insbesondere hierarchische Beziehungen in der Ontologie wichtiger sind. Ist stattdessen die Auswirkung von einer Klasse auf eine andere gesucht, so stellen nicht-hierarchische Beziehungen dies besser dar. Um mit dem Auffinden der $k$-kürzesten Pfade unterschiedliche Fragestellungen abzudecken, müssen für jeden Beziehungstyp in der Ontologie die Gewichtungen variierbar sein. Somit können zu jeder Fragestellung die Gewichte der Kanten im Graphen angepasst werden. Dadurch variiert auch der kürzeste Pfad in einer Ontologie, vorausgesetzt die Berechnung der Distanz zwischen zwei Klassen (Dijkstra

Zeile 6) basiert auf dem Gewicht der Kante dazwischen. Somit muss die $dist()$ Prozedur wie folgt aussehen:

```
1: procedure DIST(S, Z)
2:     edges ← G.allEdgesBetween(S, Z);
3:     distance ← edges[0].getWeight();
4:     for eachedgeinedges do
5:         if edge.getWeigth() < distance then
6:             distance ← edge.getWeigth();
7:     return dist
```

### 5.1.4 Erweiterung für Zwischenknoten

Die Anforderungen (insbesondere T2) beschreiben eine Pfadsuche auch mit Zwischenknoten. Dabei besagt dies, dass zwischen einem Start- und dem Zielknoten jeweils mehrere Zwischenknoten besucht werden müssen (falls ein solcher Pfad existiert). Somit entspricht ein Zwischenknoten einem Teil des Pfades auf dem Weg zum Zielknoten.

Die Umsetzung dieser Anforderung stellt sich als ein größeres Problem dar.Es konnten keine einfachen Graphenalgorithmen gefunden werden, welche Pfade über Zwischenknoten auffinden können. Dabei gibt es mehrere Probleme die einen solchen Algorithmus erschweren.

**Sackgassen** Ein solcher Algorithmus muss in der Lage sein, für jeden Zwischenknoten zu erkennen, ob es sich hierbei um einen Sackgasse handelt. Dies bedeutet, dass der Knoten nur durch eine Kante erreicht werden kann und keine weiteren Kanten besitzt. So ist in dem Ausschnitt aus der SNIK-Ontologie (Abbildung 2.3) der Knoten `Patienten_Data_Management_System` eine Sackgasse, da er nur über eine Kante erreicht werden kann. Das Problem hierbei ist, sobald ein Zwischenknoten in einer Sackgasse ist, so kann kein sinnvoller Pfad gefunden werden. Die einzige Möglichkeit von Start- zum Zielknoten über einen Zwischenknoten in einer Sackgasse zu gelangen ist es, entlang des Pfades zurück zu gehen. Solche Pfade sind jedoch bei der Interpretierung für den Anwender fraglich.

**Blockierende Verbindungen** Ein weiteres Problem entsteht, sobald ein Zwischenknoten nur durch einen Pfad erreicht werden kann, welcher zwischen dem Start- und Zielknoten liegt. Solang nur ein Zwischenknoten angegeben

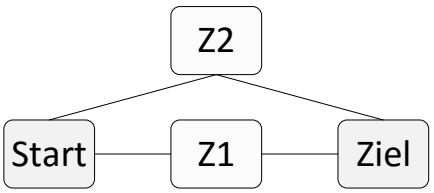


Abbildung 5.1: Beispiel für einen Zwischenknoten zwischen Start- und Ziel

wird, ist dies kein Problem, wird hingegen ein weitere Zwischenknoten angegeben der nicht auf dem Pfad liegt so kann wiederum kein Pfad gebildet werden der ohne zurückgehen möglich ist. Ein solches Beispiel ist in der Abbildung 5.1 dargestellt. Wenn ein Pfad von `Start` nach `Ziel` mit den beiden Zwischenknoten `Z1` und `Z2` gesucht wird, so kann `Z1` zwar erreicht werden, aber anschließend nicht `Z2`.

**Handlungsreisender** Unabhängig von den dargelegten Problemen, kann das Auffinden von Pfaden mit Zwischenknoten als ein vereinfachtes Problem des Handlungsreisenden-Problems interpretiert werden. Das Handlungsreisender-Problem ist ein grundlegendes Problem der Graphentheorie. Das Ziel beim Handlungsreisenden ist es $n$ (erreichbare) Knoten eines Graphen von einem Startknoten zu besuchen und am Ende wieder zum Startknoten zu gelangen. Dabei sollte der möglichst kürzeste Pfad gefunden werden und es dürfen keine Knoten mehrmals besucht werden. Dieses Problem hat eine exponentielle Laufzeit, aus diesem Grund werden hierfür Optimierungsverfahren zur Lösung des Problems verwendet [28]. Diese gewährleisten aber nicht immer eine optimale Lösung.

**Dijkstra-Erweiterung** Aufgrund der Komplexität der Pfadbestimmung mit Zwischenknoten, wird im Rahmen dieser Masterarbeit hierfür keine Lösung angeboten. Das Problem kann jedoch für den einfachen Fall von maximal einem Zwischenknoten und dem Auffinden vom kürzesten Pfad gelöst werden. Dazu wird die Suche des kürzesten Pfades durch den Dijkstra Algorithmus zweimal verwendet, mit jeweils unterschiedlicher Eingabe und am Ende werden die Teilergebnisse zu einem Pfad verknüpft. In Algorithmus 3 ist diese Erweiterung als Pseudocode dargestellt.

**Algorithmus 3** Dijkstra für Zwischenknoten

```
1: function DijkstraErweitert(G, S, Z, Zw)
2:     rootPath ← Path(Dijkstra(G, S, Zw));
3:     spurPath ← Path(Dijkstra(G, Zw, Z));
4:     if rootPath.notEmtpy and spurPath.notEmtpy then
5:         return rootPath + spurPath
```

## 5.2 Problemstellung: Instanzen

Neben dem Auffinden von Pfaden, ist eine weitere Anforderung (T1, T2, US03, US06, US08, US12, SF04, SF08, SF09, SF10), die Pfade mit Instanzinformationen anzureichern. Da die SNIK-Ontologie keine Instanziierungen besitzt, muss eine für CIONw anwendabare Lösung entwickelt werden. Die Instanzdaten zu den unterschiedlichen Klassen können aus heterogenen Datenquellen stammen, beispielsweise Excel-Tabellen oder Webseiten, deshalb muss für die Instanzdaten ein Quellen unabhängiges Format gewählt werden. Dazu eigenen sich möglichst einfache Tabellenstrukturen. Diese können im CSV-Format einfach und systemunabhängig dargestellt werden.

Um zwischen zwei Knoten eine Verbindung anhand der Instanzen zu erstellen, muss über die Verbindung zwischen den Klassen (die Kanten in dem Graphen) die Beziehung zwischen den einzelnen Instanzen abgebildet werden. Somit müssen zwei unterschiedliche Arten von Instanzentabellen existieren: Instanztabellen zu Klassen und Instanztabellen zu Relationen. In der Abbildung 5.2 ist ein Beispiel für die Verbindung von Instanztabellen (Klassen-Instanzen und Relationen-Instanzen) und der Klassen der Ontologie dargestellt. Die oberen Tabellen `Migration_Path.csv` und `Project.csv` stellen jeweils Instanztabellen zu den Klassen `Migration_Path` und `Project` dar. Die unter Tabelle beschreibt die Relation `isDecomposedIn` zwischen den beiden Klassen, die eine Verknüpfung der Instanzdaten beider Klassen ermöglicht. Die einzelnen Tabellenarten werden in den nachfolgenden Abschnitten näher erläutert.

### 5.2.1 Instanztabellen zu Klassen

Für die Darstellung von Klasseninstanzen müssen die einzelnen Instanztabellen zur jeder Klasse der Ontologie angelegt werden. Dazu kann der Einfachheit halber für jede Klasse eine eigenständige CSV-Datei erstellt werden. Die Verbindung zwischen der Klasse in der Ontologie und der dazugehörigen Instanztabelle ist über den Namen gelöst. Dabei ist der Dateiname der Instanztabelle `<Klassenname>.csv`, d.h. für die Klasse `Project` heißt die CSV-Datei `Project.csv`.

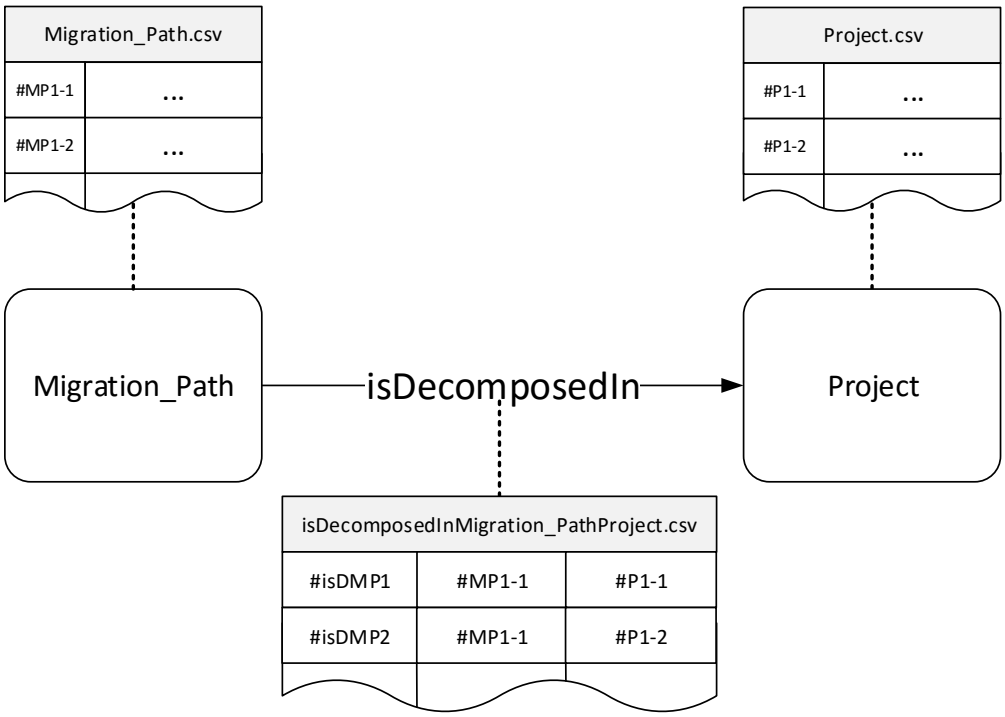


Abbildung 5.2: Verbindung zwischen Instanztabellen und der Ontologie

Der Aufbau der Instanztabellen ist so einfach wie möglich gehalten, um damit spätere Erweiterungen bezüglich der Datenbeschaffung zu ermöglichen. Eine CSV-Struktur kann im allgemeinen mit wenig Aufwand aus vielen Datenquellen exportiert werden. Dadurch lassen sich für zukünftige Erweiterungen die Instanztabellen beispielsweise aus einer Datenbank auslesen. Eine kleines Beispiel für eine Instanztabelle zur Klasse `Project` ist in Abbildung 5.3 dargestellt. Da die Instanztabellen alle den selben Aufbau haben, können in der CSV-Datei die Spaltenbezeichnung entfallen.

| **ID** | **Instanz** |
|---|---|
| #P1-1 | Austausch Router Bereich A |
| #P1-2 | Austausch Router Bereich B |
| #P2-1 | Schulung WIN 10 |

| **Project.csv** |
|---|
| "#P1-1","Austausch Router Bereich A" |
| "#P1-2","Austausch Router Bereich B" |
| "#P2-1","Schulung WIN 10" |

Abbildung 5.3: Beispieltabelle für die Klasse Projekt (links) und die entsprechende CSV-Datei (rechts)

### 5.2.2 Instanztabellen zu Relationen

Neben den Instanztabellen zu den Klassen muss für jede existierende Relation in der Ontologie jeweils eine weitere Tabelle erstellt werden. Diese setzt Instanzen aus den verbundenen Klassen zueinander in Verbindung. Die Relationstabellen werden ebenfalls als CSV-Datei dargestellt. Da die Instanznamen keine eindeutige Identifizierung gewährleisten, werden die IDs der jeweiligen Instanzen (aus den Instanztabellen zu jeder Klasse) zueinander in Relation gestellt. In Abbildung 5.4 wird eine beispielhafte Relationstabelle für die `isDecomposedIn` Relation zwischen `Migration_Path` und `Project` dargestellt. Dazu wird zu jeder Relation jeweils die Beziehung zwischen zwei Instanzen ebenfalls mit einer ID versehen. Die CSV-Datei wird analog zu den Instanztabellen durch ihren Namen zu der jeweiligen Relation zugeordnet. Dabei ist der Namensaufbau der Datei wie folgt (ohne die dargestellten Klammerungen):

`<Kantentyp><Name Ursprungsknoten><Name Zielknoten>`

| **isDecomposedInMigration_PathProject** |
|---|
| "#isDMP1","#MP1-1","#P1-1" |
| "#isDMP2","#MP1-1","#P1-2" |
| "#isDMP3","#MP2-1","#P2-1" |

Abbildung 5.4: CSV Darstellung der `isDecomposedIn` Beziehung

**Nicht instanziierbare Klassen** Die in Abbildung 5.4 und 5.2 dargestellte Beispiel-Relation ist in der SNIK-Ontologie eine Ausnahme. Normalerweise ist in der SNIK-Ontologie immer eine Rolle oder Entität durch eine Funktion mit einer anderen Rolle oder Entität verbunden (vgl. Meta-Modell der SNIK-Ontologie in Abbildung 2.1). Dies wird in der Ontologie dadurch gekennzeichnet, dass jeder Klasse eine hierarchische Zuordnung zu einem der jeweiligen Typen `Role`, `Function` und `EntityType` erfolgt. Dies hat zufolge, dass eine einfache Beziehung zwischen einer Rolle oder Entität mit einer Funktion, wie zum Beispiel `Portfolio_Management updates Project`, nicht in einer Relationstabelle, wie sie oben Beschrieben ist, dargestellt werden kann (vgl. Kapitel 2.2.1). Da `Portfolio_Management` eine Funktion ist und keine Instanzen besitzen kann, kann in der `updatesPortfolio_ManagementProject` Tabelle auch keine Verbindung der Instanzen stattfinden. Dieser Sachverhalt muss bei der Verbindung der Pfade auf Instanzebene berücksichtigt werden.

### 5.2.3 Natural Join

Damit ein gefundener Pfad, mit konkreten Instanzen, dem Nutzer präsentiert werden kann, muss entlang eines Pfades zu jedem Knoten und jeder Kante die jeweilige Instanztabelle existieren. Auf Basis der Relationen in den Relationstabellen kann der Pfad mit Instanzdaten verbunden werden. Dabei wird Iterativ ein natural Join zwischen zwei Klassen entlang des Pfades durchgeführt. Der natural Join ist ein Begriff aus dem Bereich der Datenbanken und beschreibt eine Verknüpfung zweier Tabellen durch eine Spalte, die in beiden Tabellen auftaucht [30]. Dabei werden nur die Einträge in einen neue Tabelle übernommen, die in beiden Tabellen existieren.

Mithilfe des natural Joins können die Relationstabellen nach und nach miteinander verknüpft werden, solange es Einträge gibt die immer wieder in der neuen Tabelle und der zu verknüpfenden Tabelle existieren.


Abbildung 5.5: Ursprünglicher Pfad vom Knoten `Application_Component` zum Knoten `Project`

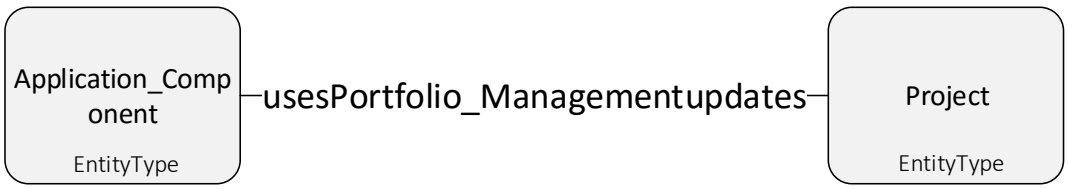


Abbildung 5.6: Transformierter Pfad zwischen `Application_Component` und `Project`

Da aber in der SNIK-Ontologie Funktionen nicht instanziiert werden können, würde der natural Join bei der ersten Relationstabelle, die eine Aufgabe (`function`) beinhaltet, abbrechen und eine leeres Ergebnis ausgeben. Um dies zu verhindern müssen die Pfade auf Funktionen untersucht werden und falls welche vorhanden sind, so müssen diese mit den umgebenden (eingehende und ausgehende) Kanten zu einer neuen Relation zusammengesetzt werden.

So kann Beispielsweise aus dem Pfad in Abbildung 5.5 der angepasste Pfad in Abbildung 5.6 werden.

Dadurch wird die nicht instantiierbare Klasse zu einer Relation transformiert. Jetzt kann in der neuen Relation `usesPortfolio_Managementupdates` eine genaue Verknüpfung von `Application_Component` mit `Project` durchgeführt werden. Zwar spielt für einen Pfad die Richtung der Kanten keine Rolle, dennoch muss die Richtung bei dem natural Join beachtet werden. Denn die Relationen sind nicht immer kommutativ.

## 5.3 Eigententwicklung vs. Erweiterung bestehender Anwendung

Für die Entwicklung des Prototyps muss die grundlegende Entscheidung getroffen werden, ob der Prototyp als eine alleinstehende Anwendung entwickelt wird oder eine existierenden Anwendung erweitert werden soll.

Der Vorteil einer eigenständigen Anwendung ist, dass die Darstellungsart der Ontologie frei gewählt werden kann. Es ist möglich die Datenstruktur für die Darstellung von Knoten, Kanten und dem Graphen selbst zu wählen und diese an die Implementierung für die verwendeten Algorithmen anzupassen. Jedoch muss bei einer solchen Eigenentwicklung ggf. sehr viel Aufwand für die selbst gewählten Datenstrukturen betrieben werden.

Bei der Erweiterung einer existierenden Anwendung besteht dieser Mehraufwand nicht unbedingt. Jedoch sind die zu verwendenden Datenstrukturen von der zu erweiternden Anwendung vorgegeben und können nicht unbedingt beliebig erweitert werden. Ebenso muss eine Einarbeitung in die API der Anwendung erfolgen. Dafür können aber ggf. bereits geforderte Funktionalitäten durch die Anwendung selbst abgedeckt sein. Die Entwicklung von CIONw als eine Erweiterung einer existierenden Anwendung wurde aus mehreren Gründen gewählt. Zum einen sind Anwendungen i.d.R. hinsichtlich der verwendeten Datenstrukturen stabil und verringern dadurch möglich Fehler, zum anderen kann eine Anwendung bereits Funktionalitäten für die grafische Darstellung anbieten. Für die Arbeit mit Ontologien gibt es unterschiedliche Anwendungen. Dazu wurden Untersuchungen von Dudáš et al. durchgeführt [31]. Diese zeigen dass je nach Anwendungsgebiet unterschiedliche Arten der Darstellung präferiert werden. So ist für das Erstellen und Bearbeiten von Ontologie eher eine nicht grafische Darstellung besser geeignet. Für das Suchen und Orientieren in Ontologien sind stattdessen Anwendungen mit einer Graphen-Darstellung bes-

ser geeignet. Eine Standardanwendung für eine grafische Ontologdarstellung ist Protégé[8] mit dem Ontograf Plugin[9].

Im SNIK Projekt ist eine Visualisierung der Ontologie, durch die Verwendung des JavaScript Framework von Cytoscape.js[10] innerhalb eines Browsers umgesetzt worden. Dafür wird die Ontologie in Cytoscape (Desktop-Anwendung) geladen, um sie in ein passendes Format für das JavaScript Framework zu transferieren. Aus diesem Grund wurde Cytoscape und seine Web-Variante Cytoscape.js als nähere Kandidat für CIONw betrachtet.

### 5.3.1 Cytoscape Desktop vs. Cytoscape webbasiert

Cytoscape existiert als Java Anwendung für den Desktop und bietet die Möglichkeiten Anwendungen als App zu entwickeln. Daneben existiert auch eine JavaScript Implementierung von Cytoscape, welche es ermöglicht Ontologien im Borwser per JavaScript darzustellen und somit die Ontologien auch über das Internet zugänglich zu machen. Für CIONw wurde die Desktop Variante gewählt, weil diese bereits Funktionalitäten anbietet, welche für CIONw notwendig sind:

- Darstellen der Ontologie insbesondere auch als Graph (Task T1)
- Suchen von Klassen (und Kanten) (Task T1)
- Nachbarschaft 1 Grades anzeigen (Taks T1)

Des Weiteren muss im Gegensatz zur JavaScript Version keine Nutzeroberflächen für die gelisteten Funktionalitäten implementiert werden.

Cytoscape (Desktop) ermöglicht es einfach eigene Nutzeroberflächen einzubinden. Da Cytoscape auf OSGi basiert, sind alle Funktionalitäten über Bundles realisiert. Bundles stellen Services in OSGi dar und werden von Cytoscape verwaltet, wodurch der Lebenszyklus der Erweiterung von Cyotscape selbst überwacht wird. Durch die Bundles können die Auswirkungen von Fehlern lokal begrenzt werden und wirken sich meistens nicht auf andere Teile von Cytoscape aus.

### 5.3.2 Grobentwurf CIONw als Cytoscape App

In der Abbildung 5.7 ist der grobe Entwurf für CIONw, als eine App für Cytoscape, als UML-Komponentendiagramm dargestellt. CIONw wird als OSGi-Service

[8] http://protege.stanford.edu/
[9] https://github.com/protegeproject/ontograf
[10] http://js.cytoscape.org

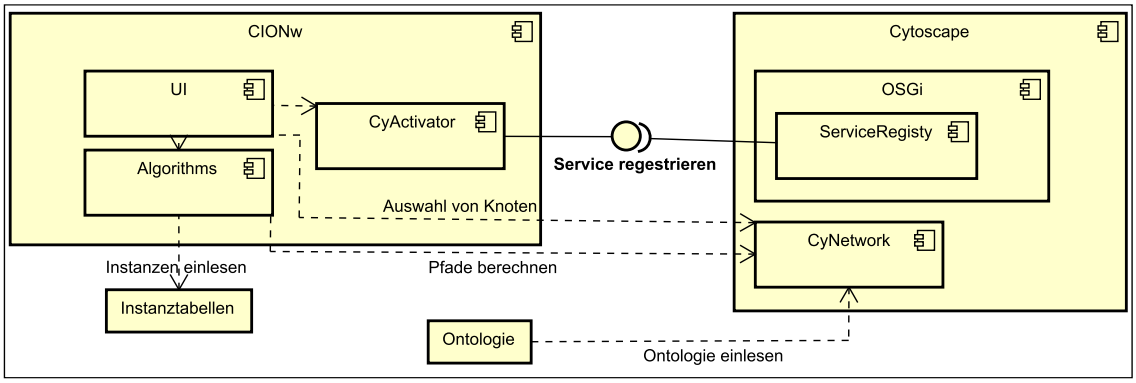


Abbildung 5.7: UML-Komponentendiagramm des CIONw Entwurfs

in Cytoscape durch den `CyActivator` registriert. Die Ontologie wird von Cytoscape eingelesen und als `CyNetwork` für alle Services zur Verfügung gestellt. Daran knüpft sowohl die Auswahl der Knoten durch die `Ui`, als auch die Berechnung der Pfad an. Die Pfade werden durch die Komponenten `Algorithms` auf der Ontologie in `CyNetwork` berechnet. Zusätzlich werden durch die Instanztabellen die Instanzinformationen für die Knoten und Kanten eines Pfades geladen. In der Komponente `Algorithms` werden durch den Dijkstra und Yen Algorithmus die Pfade berechnet. Die Verbindung der Instanzen entlang des Pfades werden durch eine eigenständige `Join` Komponenten innerhalb von `Algorithms` durchgeführt.

Cytoscape bietet drei Bereiche für die Einbindung von GUIs an, die jeweils mit den Himmelsrichtung positioniert werden. In dem `WEST` Bereich werden die Control Panels, für die Einstellungen und Aktionen abgelegt. Im `SOUTH` Bereich werden alle Table Panels dargestellt. Diese stellen tabellarisch Informationen zu Knoten, Kanten und weiteren Element dar. Im `EAST` Panel werden alle Result Panels abgelegt. In den Result Panels sollten alle Ergebnisse von möglichen Berechnungen auftauchen. Diese drei Bereiche werden für die Darstellung von CIONw wie folgt genutzt:

**`WEST`** hier werden alle Einstellungen und Aktivitäten für CIONw dargestellt

**`SOUTH`** hier werden die gefundenen Pfade tabellarisch aufgelistet

**`EAST`** entgegen der Vorgaben von Cytoscape, werden hier die Instanzinformationen für Klassen dargestellt

In Abbildung 5.8 ist die Oberfläche von Cytoscape (Desktop) mit den jeweils markierten Panels dargestellt.

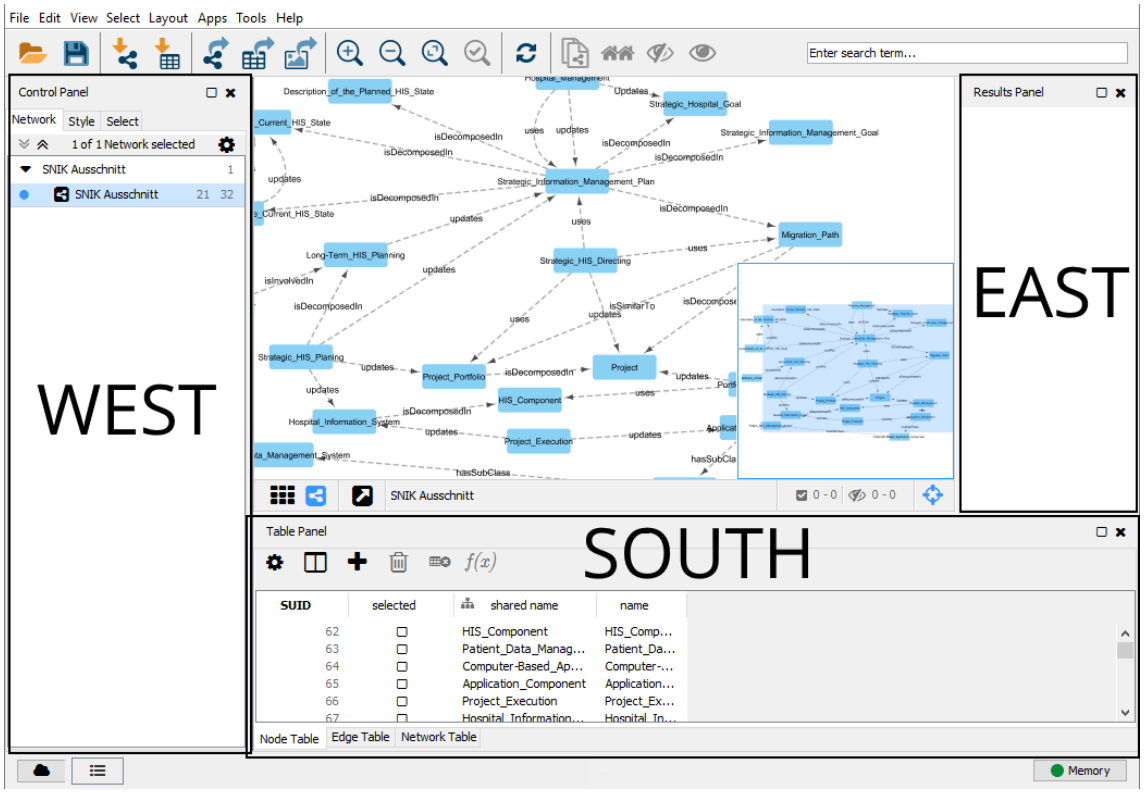


Abbildung 5.8: Cytoscape mit markierten Panels

# 6 Lösungsumsetzung

In diesem Kapitel wird die Umsetzung des Lösungsentwurfes im Detail beschrieben. Die Softwarearchitektur wird dargestellt und die für die Lösung wesentlichen Klassen näher beschrieben. Zur Qualitätssicherung werden verschiedene Tests durchgeführt, die anschließend beschrieben sind.

## 6.1 OSGi Bundle

Da CIONw als eine App für Cytoscape entwickelt wird, wird die von Cytoscape vorgegebene Grundstruktur verwendet. Diese Grundstruktur basiert auf der Implementierung von OSGi Bundles. Eine Bundle App für Cytoscape wird in Java mithilfe von dem Apache Maven Build-Management Tool entwickelt. Dadurch werden alle Abhängigkeiten zu externen Bibliotheken durch die `POM.xml` Datei von Maven verwaltet. Darüber hinaus muss die Klasse `CyActivator` in der Anwendung implementiert werden. In der `CyActivator` Klasse müssen alle eigenständigen Bestandteile der App als Service an das OSGi Framework angemeldet werden (vlg. 5.3.2). Im Konkreten handelt es sich dabei bei CIONw um:

**Menu Einträge** Alle Menüeinträge, sowohl in der Menüleiste als auch Kontextmenü Einträge (vgl. Kapitel 6.4.4)

**Listeners** Alle Event Listener von CIONw

**Readers** ein Property Reader für CIONw

Somit stellt die `CyActivator` Klasse den Einstiegspunkt für alle weiteren Funktionen und Bestandteile von CIONw dar.

## 6.2 Architektur von CIONw

CIONw ist in drei Pakete und zwei unkategorisierten Klassen unterteilt. Die beiden unkategorisierten Klassen sind zum einen die oben erwähnte `CyActivator` Klasse und die `MenuAction` Klasse. Die drei Pakete teilen die weiteren Klassen nach ihrer Verwendung ein. Das Paket `Algorithms` beinhaltet die Implementierung vom Dijkstra und Yen Algorithmus. Zusätzlich beinhaltet es Datenstrukturen für die beiden Algorithmen und ein Paket für die Bearbeitung der Instanzen.

Im Paket `Ui` sind alle Klassen die im Zusammenhang mit der Oberfläche von Cytoscape stehen enthalten. In dem dritten Paket `Utils` wird die statische Klasse `GlobalSettings` und weitere nicht direkt zugeordneten Klassen abgelegt.

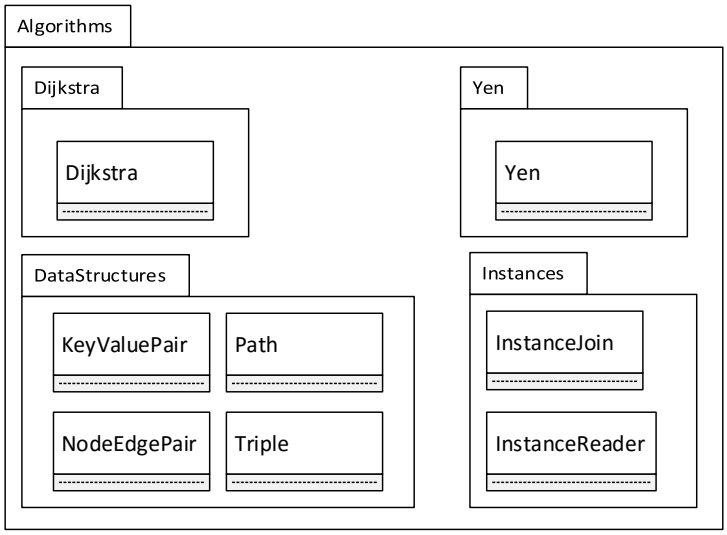


Abbildung 6.1: `Algorithms` Paket mit den Unterpakete und Klassen

In Abbildung 6.1 ist das Paketdiagramm des `Algorithms` Paket abgebildet. Das Paket ist in vier Teilpakete unterteilt. Die Pakete `Dijkstra` und `Yen` beinhalten jeweils eine Klasse die den zugehörigen Algorithmus implementiert (vgl. Kapitel 6.2.1). Für die beiden Algorithmen werden die Datenstrukturklassen aus dem Paket `DataStructures` benötigt. Dieses Paket stellt zusätzlich Datenstrukturen für das `Instances` Paket bereit. In dem `Instances` Paket befindet sich die Klasse `InstanceReader`, welche für das Einlesen von Instanztabellen zuständig ist. In der Klasse `InstanceJoin` werden die eingelesenen Instanzen per natural Join miteinander verbunden.

In Abbildung 6.2 ist das `Ui` Paketdiagramm dargestellt. Dieses Paket beinhaltet alle Klassen, welche die grafische Elemente von CIONw in Cytoscape implementieren. Das sind zum einen drei Kontexmenü Einträge zur Auswahl des Start-/End-/Zwischenknoten und zum anderen die drei Anzeigebereiche in Cytoscape (vgl. 5.3.1). Die Klasse `ControlPanel` stellt die gesamte UI für das `WEST` Panel in Cytoscape bereit. Die `InstancePanel` Klasse stellt die UI für die Darstellung der Instanzen im `EAST` Panel bereit. Die gefundenen Pfade werden tabellarisch durch die `PathTable` Klasse im `SOUTH` Panel dargestellt. Neben den

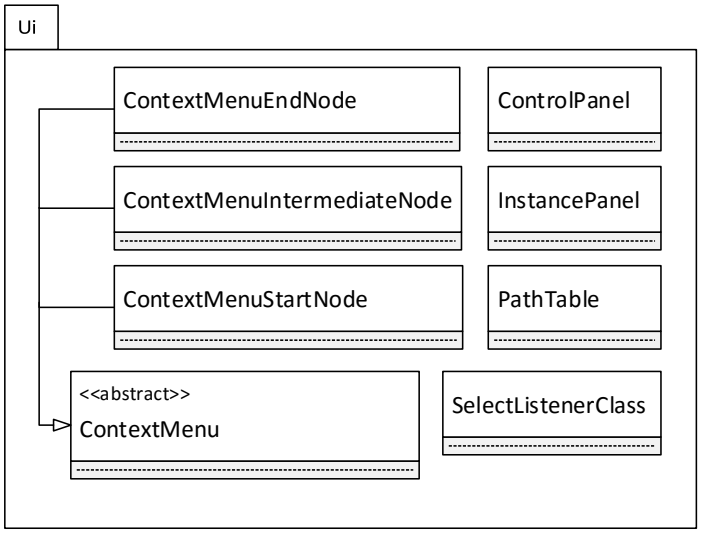


Abbildung 6.2: Ui Paket mit dazugehörenden Klassen

6 UI Klassen ist die Klasse `SelectListenerClass` im Paket enthalten. Diese Klasse stellt keine direkten UI Elemente dar, stattdessen ist es ein Event Listener, der auf das Markieren von Knoten in der Ontologie reagiert. Aus diesem Grund befindet sich der Listener im Paket `Ui`.

Im Paket `Utils` befindet sich die statische Klasse `GlobalSettings`, welche Einstellungen, wie z.B. die Kantengewichte und die ausgewählten Knoten, beinhaltet. Des Weiteren sind die Klassen `PropertyReader` und `SessionListener` in dem Paket `Utils`. Die Klasse `PropertyReader` ist eine triviale Implementierung, die dazu verwendet wird OSGI bzw. Cytoscape Einstellungen (engl. Properties) zu laden. Der `SeassionListener` ist für das Laden und Speichern der Einstellungen beim Einlesen einer Cytoscape Sitzung zuständig.

In Abbildung 6.3 ist das Klassendiagramm der gesamten Anwendung CIONw dargestellt. Zur besseren Übersicht wurden die Klassen aus dem Paket `Ui` nicht einzeln dargestellt. Sie werden durch ihr Paket repräsentiert. Es ist erkennbar, dass alles aus der `CyActivator` Klasse hervorgeht. Die `CyActivator` und die `MenuAction` Klassen erstellen zusammen alle Elemente der UI, welche wiederum die Klassen für die Berechnung von Pfaden instanziiert. Bei den Klassen `GlobalSettings` und `InstanceReader` handelt es sich um statische Klassen.

In der Abbildung 6.4 ist dargestellt welche Klasse welche anderen Klassen erstellt. Dabei werden die Kontextmenüs, der `PropertyReader`, die beiden Lis-

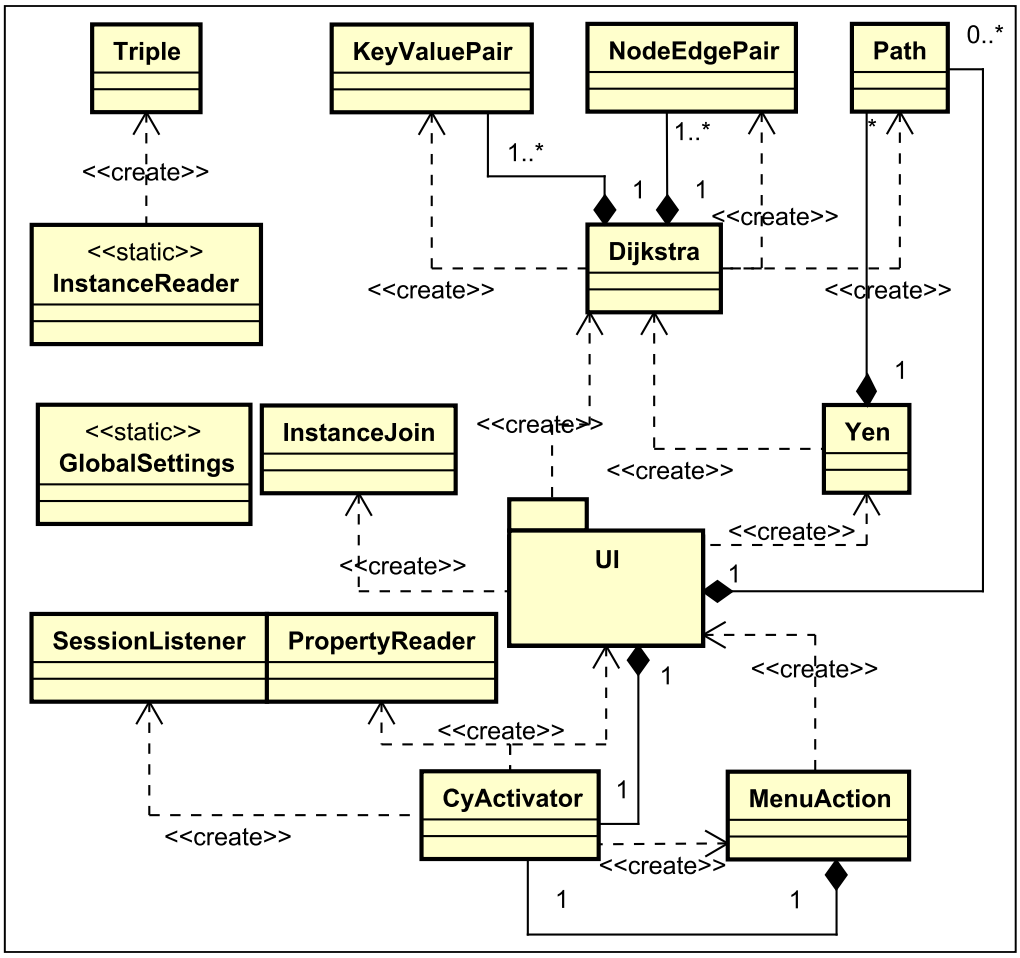


Abbildung 6.3: UML-Klassendiagramm von CIONw

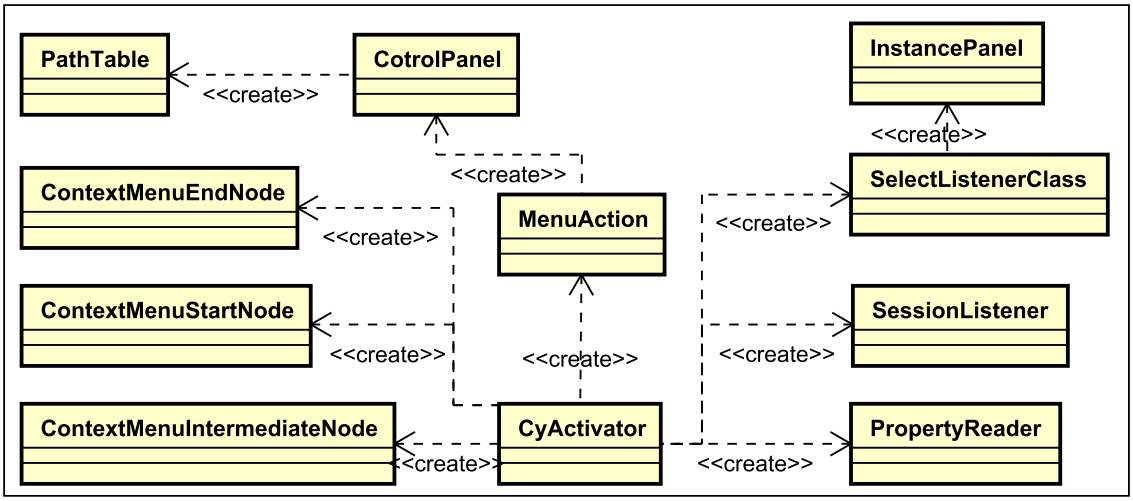


Abbildung 6.4: Klasseninstanziierung der UI

tener `SelectListenerClass` und `SessionListener` von `CyActivator` erstellt. Zusätzlich wird auch die `MenuAction` Klasse erstellt. Erst diese Klasse erstellt das `ControlPanel` und die `PathTable`. Während das `InstancPanel` erst durch einen Listener erstellt wird.

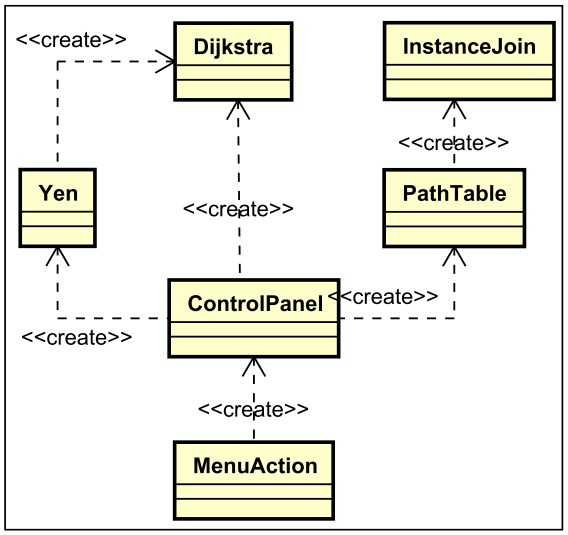


Abbildung 6.5: Zusammenhang des `ControlPanels` mit den Algorithmen

In Abbildung 6.5 ist der Zusammenhang von den beiden Pfadfindungsalgorithmen, den gefundenen Pfaden in der Klasse `PathTable` Klasse und der Klasse `InstanceJoin` dargestellt. Das `ControlPanel` erstellt sowohl Instanzen von `Dijkstra` als auch von `Yen` um damit Pfade zu finden. Anschließend werden die gefundenen Pfade in dem `TablePanel` angezeigt. Das `TablePanel` wiederum nutzt den `InstanceJoin` um damit Pfade mit Instanzen zu verbinden. In dem Sequenzdiagramm 6.6 ist der Ablauf zur Bestimmung von $k$-Pfaden mithilfe von Dijkstra und Yen Algorithmus dargestellt. Zusätzlich wird auch die Erstellung des Pfades mit Instanzinformationen durch `PathTable` mithilfe von `InstanceJoin` dargestellt. Dabei wird im ersten Schritt das `ControlPanel` durch das Aktivieren von CIONw im Menü (`MenuAction`) erstellt. Nachdem der Nutzer Start- und Endklasse ausgewählt hat, wird mit dem Button `Yen` ein Objekt von `Yen` erstellt. Das Objekt berechnet anschließend die $k$-kürzesten Pfade. Dafür wird in einer Schleife jeweils ein Objekt von `Dijkstra` erstellt und darin ein kürzester Pfad berechnet. Nachdem der Yen Algorithmus alle Pfade gefunden hat, werden diese in einer `PathTable` Instanz tabellarisch dargestellt.

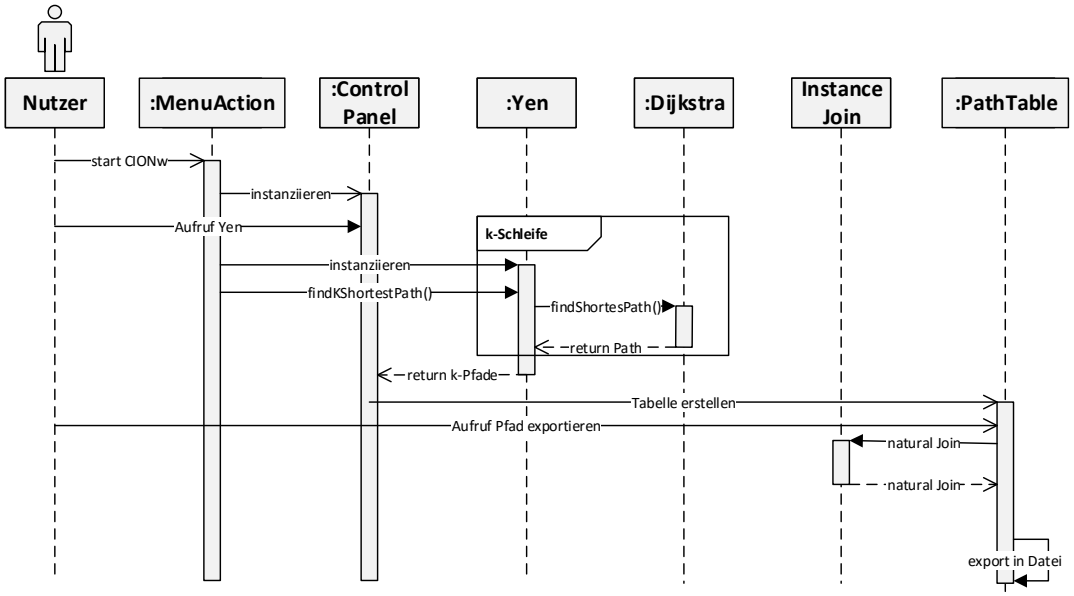


Abbildung 6.6: Sequenzdiagramm für eine Nutzerintaktion

### 6.2.1 Implementierung von Dijkstra und Yen Algorithmus

Die grundlegenden Algorithmen für die Bestimmung von Pfaden wurden in `Algorithms` Paket weiter in eigenständige Pakete unterteilt. Beide Algorithmen sind an die Cytoscape Datenstrukturen vollständig angepasst. Graphen werden als `CyNetwork`, Kanten als `CyEdge` und Knoten als `CyNode` dargestellt.

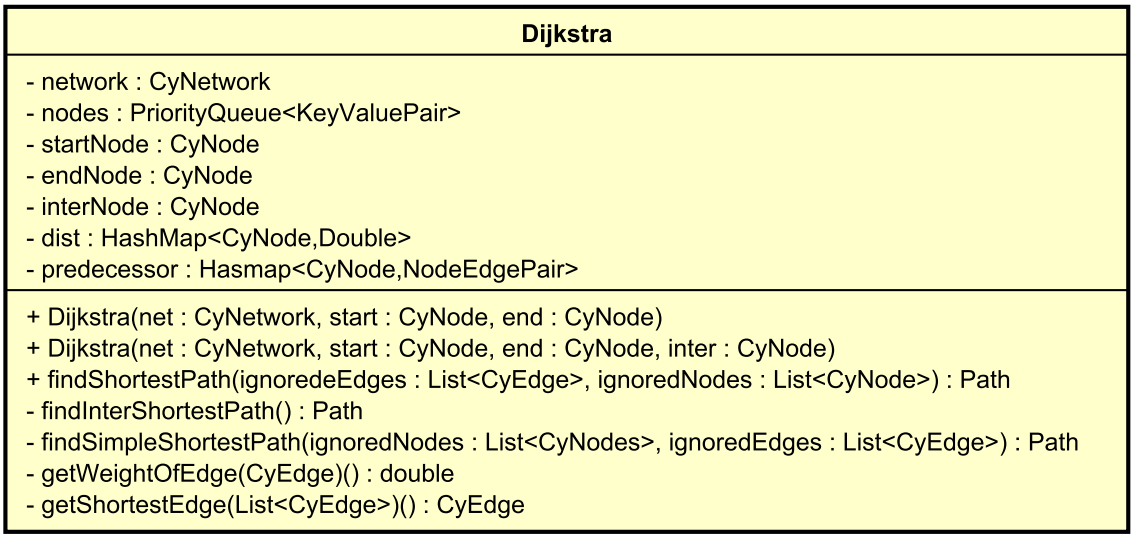


Abbildung 6.7: Klassendiagramm von `Dijkstra` Klasse

**Dijkstra** In Abbildung 6.7 ist das Klassendiagramm der `Dijkstra` Klasse abgebildet. Um mit Hilfe der Klassen einen kürzesten Pfad zu berechnen, muss zuerst ein Objekt der `Dijkstra` Klasse erstellt werden, bei welchem der aktuelle Graph, der Startknoten und der Zielknoten übergeben werden. Anschließend kann mithilfe der `findShortestPath` Methode der kürzeste Pfad ermittelt werden. Im Gegensatz zu dem Algorithmus 1 in Kapitel 5.1.2 handelt es sich hierbei bereits um die angepasste Version des Dijkstra Algorithmus, welche nach dem Auffinden des Zielknotens vorzeitig abbricht. Die Warteschlange `nodes` entspricht der Menge $Q$ aus dem Pseudocode-Algorithmus.

Die Methode `findShortestPath` nimmt eine Liste von Kanten und eine Liste von Knoten entgegen. Diese Listen stellen Elemente des Graphens dar, welche vom Dijkstra Algorithmus ignoriert werden. Dies entspricht dem Löschen von Kanten bzw. dem erhöhen der Kantengewichte auf $\infty$. Dadurch muss der Graph selbst nicht verändert und wiederhergestellt werden. Damit der Algorithmus für zukünftige Verbesserungen bzw. Anpassungen bezüglich der Distanzberechnung ausgelegt ist, wurde die Methoden `getShortestEdge` und `getWeightOfEdge` implementiert. Dadurch kann sowohl die Bestimmung der Gewichtung, als auch die Berechnung der nächst kürzesten bzw. günstigsten Kante angepasst werden.

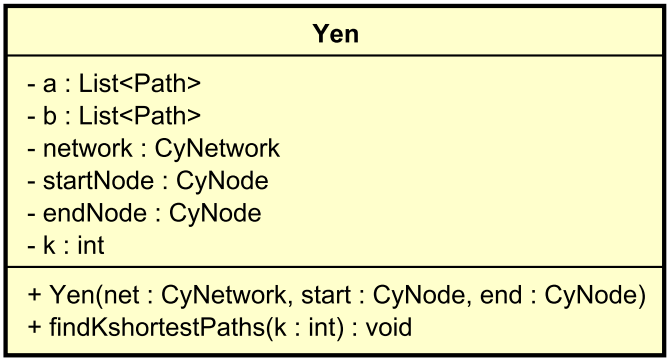


Abbildung 6.8: Klassendiagramm von `Yen` Klasse

**Yen** In Abbildung 6.8 ist das Klassendiagramm der `Yen` Klasse dargestellt. Die Klasse implementiert den Algorithmus 2. So stellt die Liste `b` den möglichen Kandidaten für den nächsten kürzeren Pfad dar, welcher in die Liste `a` ggf. übertragen wird. Die Liste `a` beinhaltet, nach dem Durchlauf der `findKshortestPaths`, geordnet die $k$-kürzesten Pfade (oder weniger, falls nicht so viele existieren).

Die Methode `findKshortestPaths` erstellt in einer Schleife immer wieder eine neue Instanz von `Dijkstra` und sucht nach dem kürzesten Pfad. Dabei werden

hier die Kanten, welche eine $\infty$-Gewichtung vom Yen Algorithmus erhalten, als Liste an die `findShortestPath` von `Dijkstra` übergeben. Zusätzlich werden bereits besuchte Knoten in einer Liste übergeben, um vom Dijkstra Algorithmus ignoriert zu werden. Dadurch werden Pfade mit Zyklen (mehrfaches auftauchen eines Knoten) verhindert.

### 6.2.2 Implementierung des natural Joins

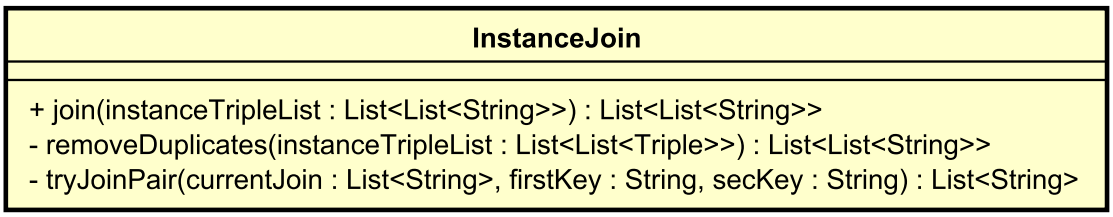


Abbildung 6.9: UML-Klassendiagramm der `InstanceJoin` Klasse

Die Bildung des natural Joins erfolgt in der Klasse `InstanceJoin`. In Abbildung 6.9 ist das dazugehörende Klassendiagramm abgebildet. Die Klasse bietet nur die `public` Methode `join` an. Die Methode bekommt eine Liste von 3-Tupel, welche die Relationstabelle aus Kapitel 5.2.2 repräsentieren. Die 3-Tupel werden durch die Klasse `Triple` implementiert. Die Liste ist in einer weiteren Liste gepackt. Dadurch werden die einzelnen Relationstabellen eines Pfades dargestellt. In Abbildung 6.10 ist der Zusammenhang zwischen mehreren Relationstabellen entlang eines Pfades mit der Darstellung als `List<List<Triple»` in der `join` Methode abgebildet.

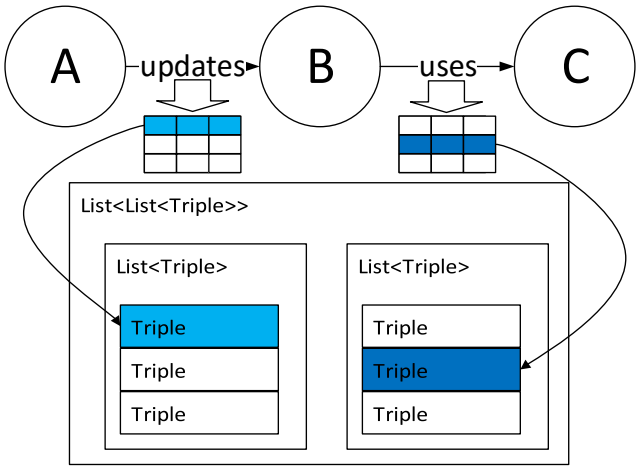


Abbildung 6.10: Schematischer Zusammenhang zwischen Relationstabellen und `List<List<Triple»`

| | Gesamt LOC | Code LOC | | Kommentar LOC | |
|---|---|---|---|---|---|
| **Klasse** | **#** | **#** | **%** | **#** | **%** |
| ContextMenu | 45 | 31 | 69 | 11 | 24 |
| ContextMenuEndNode | 50 | 39 | 78 | 6 | 12 |
| ContextMenu-IntermediateNode | 37 | 27 | 73 | 5 | 14 |
| ContextMenuStartNode | 50 | 39 | 78 | 6 | 12 |
| ControlPanel | 410 | 331 | 81 | 42 | 10 |
| CyActivator | 76 | 57 | 75 | 6 | 8 |
| Dijkstra | 190 | 118 | 62 | 58 | 31 |
| GlobalSettings | 53 | 34 | 64 | 13 | 25 |
| InstanceJoin | 115 | 78 | 68 | 32 | 28 |
| InstancePanel | 115 | 71 | 62 | 29 | 25 |
| InstanceReader | 80 | 50 | 62 | 24 | 30 |
| KeyValuePair | 47 | 20 | 43 | 21 | 45 |
| MenuAction | 75 | 52 | 69 | 14 | 19 |
| NodeEdgePair | 66 | 39 | 59 | 21 | 32 |
| Path | 100 | 46 | 46 | 43 | 43 |
| PathTable | 390 | 320 | 82 | 49 | 13 |
| PropertyReader | 20 | 7 | 35 | 11 | 55 |
| SelectListener | 88 | 65 | 74 | 14 | 16 |
| SessionListener | 51 | 32 | 63 | 13 | 25 |
| Triple | 50 | 20 | 40 | 25 | 50 |
| Yen | 115 | 76 | 66 | 34 | 30 |
| **Gesamt** | **2223** | **1552** | **70%** | **477** | **21%** |

Tabelle 6.1: Statische Analyse der Codezeilen durch Statistic-Plugin (ohne Testklassen)

## 6.3 Statische Codeanalyse

Eine zentrale Code-Metrik zur Analyse von Software ist die Anzahl der Quellcodezeilen (engl. Lines of Code, kurz LOC). In diesem Kapitel wird ebendiese Metrik zur Analyse des Codes verwendet. Die gesamte Implementierung von CIONw ist in 2595 Codezeilen geschrieben. Dabei entfallen 372 Zeilen auf Testfälle. In Tabelle 6.1 ist die Analyse der geschrieben LOC durch das Statistic-Plugin[11] dargestellt. Für jede Klasse wird die Gesamtanzahl an Codezeilen (samt Leerzeilen) und deren Aufteilung auf tatsächliche Codezeilen und Kommentarzeilen gelistet. In der Tabelle wurden die Testklassen nicht aufgezählt, da für die meisten

[11] https://plugins.jetbrains.com/idea/plugin/4509-statistic

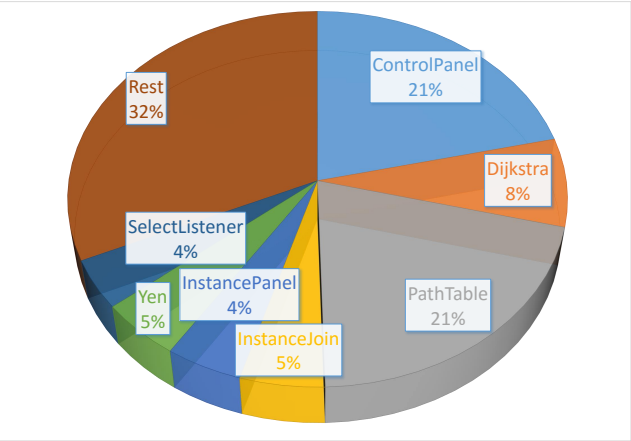


Abbildung 6.11: Codeverteilung an der Gesamtanzahl an Codezeilen

Tests viel Codezeilen geschrieben werden, nur um einen Graphen zu erstellen. Die Tabelle zeigt, dass 21% der Codezeilen für Kommentare verwendet werden.

In Abbildung 6.11 ist der prozentuale Anteil von Klassen zu der Gesamtanzahl an Codezeilen dargestellt. Dabei sind alle Klassen mit weniger als 60 Codezeilen zusammengefasst (Rest). In der Abbildung sind insbesondere die Klassen `ControlPanel`, `PathTable` und `Dijkstra` auffällig. Die hohe Anzahl an Codezeilen in den beiden Klassen `ControlPanel` und `PathTable` ist zum einen dadurch begründet, dass es sich hierbei um UI Elemente handelt und zum anderen, dass es Klassen sind, die den meisten Funktionsumfang implementieren. Die Klasse `Dijkstra` ist mit 8% die drittgrößte Klasse, weil sie den Kern der Pfadfindung darstellt.

**Code Dokumentation** Die gesamte Implementierung ist ausführlich im Code Dokumentiert. Dazu wurden JavaDoc Kommentare zu allen Klassen und Methoden erstellt. Darüber hinaus wurden an nicht trivialen Codestellen direkte Kommentare im Code erstellt.

Neben den Dokumentation im Code findet auch ein Logging der Programmausführung zur Laufzeit von CIONw statt. Das Logging wird durch das Packet `org.slf4j` ermöglicht. Die erstellten Logeinträge werden in den standardmäßigen Log Dateien von Cytoscape eingetragen. Die Dateien befinden sich unter `.../CytoscapeConfiguration/3/`.

## 6.4 UI

In diesem Abschnitt werden die UI Elemente beschrieben und die dazugehörenden Klassen näher betrachtet.

Die gesamte UI wurde, wie in Kapitel 5.3.1 beschrieben, durch die Panels in Cytoscape umgesetzt. Die drei erstellten Panels stellen auch gleichzeitig eine Trennung der Aufgaben von CIONw sowohl auf Implementierungsebene als auch teilweise auf der Anforderungsebene dar.

### 6.4.1 ControlPanel

In dem `ControlPanel` im `WEST` Bereich von Cytoscape werden alle Einstellungen zu CIONw verwaltet, die Auswahl des Nutzers dargestellt und die beiden Pfad-Algorithmen Dijkstra und Yen gestartet. Das `ControlPanel` wird als ein Tab im `WEST` Bereich eingeblendet, sobald CIONw gestartet wird. Die möglichen Einstellungen in dem Panel sind:

- maximale Anzahl an Pfaden ($k$-Parameter vom Yen Algorithmus)
- maximale Länge eines Pfades
- Gewichtung zu jeder Kante im aktuell geladenen Graph (Gewichtung muss $> 0$ sein)
- Verzeichnis mit Instanztabellen
- Ausgabeverzeichnis für Pfad-Exporte

Alle Einstellungen werden beim Starten und Beenden von einer Sitzung in Cytoscape gespeichert und wieder geladen. Dies wird durch den `SessionListener` durchgeführt. Die gesamte Nutzeroberfläche und die Funktionalitäten zu dieser UI sind in der Klasse `ControlPanel` implementiert. In Abbildung 6.12 ist das Control Panel dargestellt.

### 6.4.2 PathTable

Sobald durch den Yen oder Dijkstra Algorithmus Pfade gefunden wurden, werden diese im `SOUTH` Bereich von Cytoscape als Tabelle in einem neuen Tab dargestellt. Die Pfade können anhand ihrer Länge oder ihrer Gesamtgewichtung (Summe aller Kantengewichte) sortiert werden. Durch das Auswählen eines Pfades in der Tabelle wird dieser in der Ontologie farbig hervorgehoben.

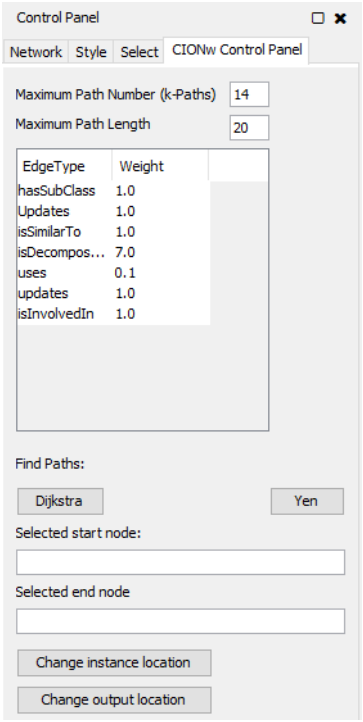


Abbildung 6.12: Screenshot vom Control Panel

Zusätzlich kann durch den `export selected Path` Button der markierte Pfad als CSV-Datei exportiert werden. Dies wird nur durchgeführt, wenn entlang des Pfades mithilfe des natural Joins mindestens ein durchgehender Join erstellt werden kann. Sollten Instanzen auf dem Pfad fehlen, wird dies durch eine Meldung dem Nutzer mitgeteilt (samt der fehlenden Instanz). Sollten mehrere Instanzen fehlen, wird immer nur die erste fehlende Instanztabelle dem Nutzer gemeldet. Die weiteren fehlenden Instanztabellen werden durch ein erneutes betätigen des Export-Buttons iterativ gemeldet. Eine sofortige Meldung ist nur bedingt möglich, da der natural Join iterativ stattfindet und dadurch nicht sofort alle fehlenden Instanztabellen erkennbar sind. Die Export-Funktionalität wird durch die `PathTable` Klasse realisiert. In Abbildung 6.13 ist die `PathTable` mit Beispielhaften Pfaden dargestellt.

### 6.4.3 InstancePanel

Für die Darstellung von Instanzinformationen zu einer ausgewählten Klasse in der Ontologie wurde das `InstancePanel` im `EAST` Bereich von Cytoscape entwickelt. Das `InstancePanel` wird von der gleichnamigen Klasse Implementiert und reagiert auf Auswahlereignisse (sog. Events) innerhalb der Ontologie. Sobald ein Event eintrifft, wird im (durch das ControlPanel eingestellte) Verzeichnis

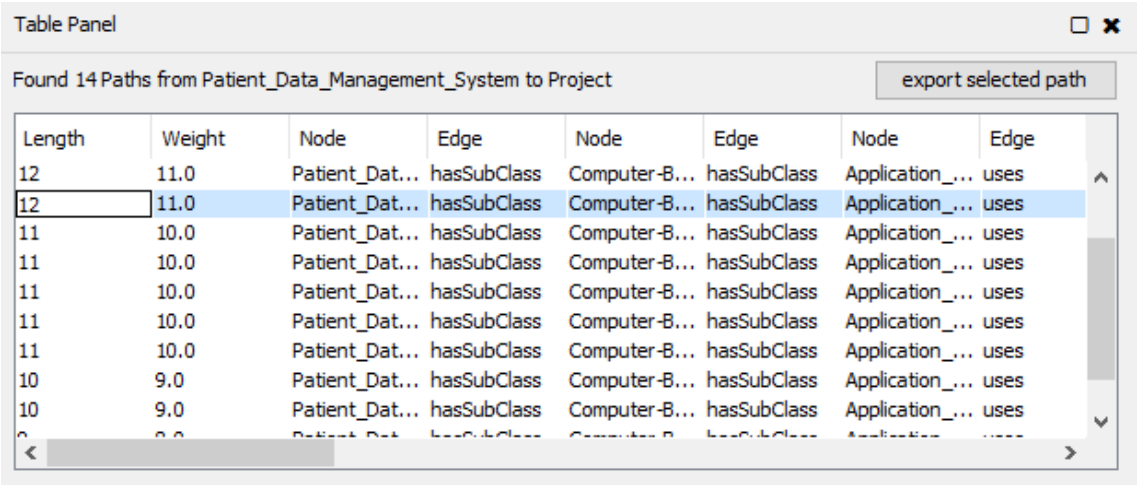


Abbildung 6.13: Screenshot von `PathTable`

nach der Instanztabelle zu der ausgewählten Klasse gesucht. Falls eine Instanztabelle vorhanden ist, werden dessen Einträge angezeigt, ansonsten wird dem Nutzer mitgeteilt, dass keine Instanzen gefunden wurden. In Abbildung 6.14 ist das `IUnstancePanel` mit Beispielinstanzen zu dem Knoten `Migration_Path` dargestellt.

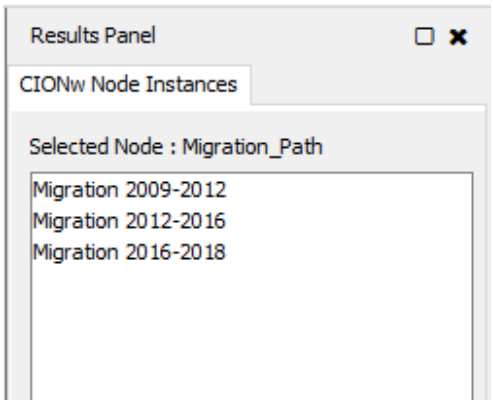


Abbildung 6.14: Screenshot von `InstancePanel`

### 6.4.4 Menüs

Da CIONw eine App für Cytoscape ist, muss es nach der Installation vom Nutzer zuerst aktiviert werden. Dazu wird durch die Klasse `MenuAction` ein neuer Eintrag in der Menüleiste von Cytoscape unter `Apps` erstellt. Erst dieser Eintrag startet CIONw vollständig.

In CIONw werden die Knoten für Start-, Zwischen- und Zielknoten über den Cytoscape Ontologiebrowser direkt in der Ontologie ausgewählt. Dazu sind

drei Kontextmenüeinträge implementiert, welche beim Rechtsklick auf einen Knoten in der Ontologie unter `Apps` gelistet sind. Wird ein Start- oder Zielknoten durch das Kontextmenü ausgewählt, so wird dieser in der Ontologie markiert und zusätzlich im ControlPanel der Name des Knotens dargestellt.

## 6.5 Tests

Um die Qualität der Implementierung zu sichern wurden automatische und manuelle Test durchgeführt. Diese Tests werden in den nachfolgenden Abschnitten beschrieben.

### 6.5.1 Komponententests

Für das Testen der entwickelten Komponenten, existieren zu Klassen JUnit[12] Tests, die im Rahmen einer Testautomatisierung für eine kontinuierliche Qualitätssicherung ausgeführt werden. Der Vorteil JUnit Tests ist, dass bei jedem Kompilieren durch Maven die Tests automatisch durchgeführt werden und das Ergebnis ausgegeben wird. Dadurch werden sofort mögliche Fehler oder negative Auswirkungen von Codeänderungen erkannt. Da die Funktionalitäten von CIONw eine starke Kopplung mit der Benutzeroberfläche aufweisen, wurden in den JUnit Tests nur die von der Benutzeroberfläche unabhängigen Klassen getestet. In der Tabelle 6.2 sind die jeweiligen JUnit Tests für die Klassen und das getestete Verhalten gelistet.

Da die Klassen `Dijkstra` und `Yen` einen Graphen in den Cytoscape Datenstrukturen benötigen, musste hierfür über Maven die Cytoscape Bibliotheken `event-api` und `model-imp` geladen werden. Anschließend konnte damit ein `CyNetwork` instanziiert werden ohne Cytoscape starten zu müssen. Für alle anderen Klassen wurden keine JUnit Tests erstellt, da die Klassen entweder UI Elemente Implementieren oder Listener sind. Ein automatischer Test der UI würde den Rahmen der Implementierung überschreiten und wurde deswegen nicht umgesetzt. Stattdessen wird die UI durch die Systemtests getestet.

### 6.5.2 Systemtests

Für den Systemtest wurde der ausgewählte Ausschnitt der Ontologie in Cytoscape erstellt. Mithilfe der Ontologie und den Anforderungen aus Kapitel 4.2 wurden anschließend der Systemtest durchgeführt. In der Tabelle 6.3 sind die

[12] http://junit.org/junit4/

| Test Methode | Zu Testendes Verhalten |
|---|---|
| **Klasse** `Path` | |
| `createPath()` | Test, ob ein Objekt von `Path` korrekt erstellt werden kann |
| `getLength()` | Test, ob die Länge eine Pfades korrekt ist |
| `getWeight` | Test, ob das Gesamtgewicht eine Pfades korrekt ist |
| `joinPaths()` | Test, ob zwei Pfade richtig miteinander verknüpft werden durch die Methode `addAll()` |
| **Klasse** `InstanceJoin` | |
| `join()` | Tes,t ob die Methode `join()` den natural Join richtig berechnet |
| **Klasse** `Yen` | |
| `findShortestPath()` | Test, ob der Yen Algorithmus korrekt implementiert ist. |
| **Klasse** `Dijkstra` | |
| `join()` | Test, ob die Methode `findShortestPath()` den Dijkstra Algorithmus korrekt umsetzt und den kürzesten Pfad bestimmt |

Tabelle 6.2: Komponententest

Tests zu den User Stories und den System Funktionen aus Kapitel 4.2 beschrieben. Dabei wird zu jeder Anforderung eine Eingabe und das erwartete Ergebnis angegeben. In der letzten Spalte wird dargestellt ob der Test erfolgreich war oder nicht.

Test 7 zeigt, dass die Anforderung US05 nicht erfüllt wird. Mit diesem Test wird das Problem des Auffinden von Pfaden mit mehreren Zwischenknoten aus Kapitel 5.1.4 aufgedeckt. Wie bereits beschrieben ist eine Implementierung für mehrere Zwischenknoten in Rahmen der Arbeit nicht möglich und somit kann die Anforderung US05 nicht erfüllt werden. Der Test 8 Zeigt, dass die Anforderung US06 nur teilweise erfüllt ist. CIONw zeigt zwar zu einem Pfad Instanzdaten in einer exportierten CSV-Datei an, jedoch werden dabei nur Instanzen angezeigt, die über einen natural Join entlang des gesamten Pfades existieren. Aus dem Grund ist die Anforderung US06 nur teilweise erfüllt.

Ebenso zeigt der Test 16, dass Klassen vom Typ `function` nicht direkt von CIONw angezeigt werden, aber dennoch vom System erkannt werden. Die Information über den Typ der Klasse ist als Knoteneigenschaft in der Knotentabelle von Cytoscape abespeichert und wird dadurch dem Nutzer ersichtlich. Jedoch muss hierfür die aus der Pfadtabellen-Ansicht auf die Knotentabellen-Ansicht gewechselt werden. Dieser Wechsel ist von der Nutzbarkeit unpraktisch und aus dem Grund wird diese Funktionalität nur teilweise erfüllt.

| Nr. | Anf. | Eingabe | erwartete Ergebnis | erfüllt? |
|---|---|---|---|---|
| 1 | US01, SF02 | `Project` in Suchmaske | `Project`, `Project_Portfolio`, `Project_Execution` werden hervorgehoben | ja |
| 2 | US02, SF03 | `Project` ausgewählt | `Migration_Path`, `Strategic_HIS_Directing`, `Project_Portfolio`, `Portfolio_Management` | ja |
| 3 | US03, SF04 | `Migration_Path` ausgewählt | `Migration 2009-20012`, `Migration 2012-2016`, `Migration 2016-2018` | ja |
| 4 | US04, SF05 | Startknoten: `Patienten_Data_Management_System`, Endknoten: `Project` | Kürzester Pfad | ja |
| 5 | US04, SF05 | Startknoten: `Patienten_Data_Management_System`, Endknoten: `Project` | weitere kürzere Pfade | ja |
| 6 | US05, SF05 | Startknoten: `Patienten_Data_Management_System`, Endknoten: `Project`, Zwischenknoten: `Strategic_HIS_Planing` | Kürzesten Pfad über Zwischenknoten `Strategic_HIS_Planing` | ja |
| 7 | US05 | Startknoten: `Patienten_Data_Management_System`, Endknoten: `Project`, Zwischenknoten: `Strategic_HIS_Planing` und `Long-Term_HIS_Planning` | Kürzesten Pfad über Zwischenknoten `Strategic_HIS_Planning` und `Long-Term_HIS_Planning` | nein |

| Nr. | Anf. | Eingabe | erwartete Ergebnis | erfüllt? |
|---|---|---|---|---|
| 8 | US06 | Gefundenen Pfad auswählen | Zu jedem Element des Pfades alle verfügbaren Instanzen anzeigen | teilweise |
| 9 | US07 | Auswahl gefundener Pfade | Alle gefundenen Pfade werde aufgelistet und können ausgewählt werden | ja |
| 10 | US08, SF08, SF09 | Gefundenen Pfad auswählen und exportieren | Alle zusammenhängenden Instanzen entlang des Pfades in Exportierter Datei als Tabelle gelistet | ja |
| 11 | US09, SF05, SF06 | Startknoten: `Application_Component`, Endknoten: `Hospital_Management`, Maximale Pfadlänge: 7 | Anzeigen von Pfaden mit Länge <= 7 | ja |
| 12 | US10, SF07 | Sortieren nach Länge und Gewicht | List der Pfade ist nach Länge und Gewicht sortiert | ja |
| 13 | US11, SF06 | Gewichtung von Kante `isDecomposed` auf 100 ändern | Gewichtung wird gespeichert und Pfade mit ohne `isDecomposed` werden bevorzugt | ja |
| 14 | US12 | Startknoten: `Portfolio_Management`, Endknoten: `Project_Portfolio` gefundenen Pfad exporieren | Meldung, dass `updateProjectPorfolio_ManagementProject` fehlt | ja |
| 15 | SF01 | Laden von `Ontologie.cys` in Cytoscape | Ontologie wird als Graph dargestellt | ja |
| 16 | SF10 | Auswahl `Project_Execution` | Anzeigen, dass `Project_Execution` eine `function` ist | teilweise |
| 17 | SF11 | Gefundenen Pfad auswählen | ausgewählter Pfad wird in Ontologie markiert | ja |
| 18 | SF12 | Gewicht von `uses` auf 0.1 setzten | nach Neustart von Cytoscape ist `uses` auf 0.1 | ja |

Tabelle 6.3: Durchgeführte Systemtests

| Task | Ergebnis |
|---|---|
| **T1: Navigation in einer Ontologie** | |
| Klassen in der Ontologie finden | Vollständig abgedeckt |
| Nachbarn zu einer ausgewählten Klasse finden | Vollständig abgedeckt |
| Instanzen zu konkreten Klassen finden | Vollständig abgedeckt |
| **T2: Pfade in Ontologie untersuchen** | |
| Einstellungen für die Pfadsuche machen (Gewichtung, Pfadlänge, ...) | Vollständig abgedeckt |
| Pfade finden | Vollständig abgedeckt |
| Auswerten der Instanzen entlang des Pfades | Teilweise abgedeckt |
| **T3: Fehlende Instanzen auffinden** | |
| Nicht instanziierbare Klassen erkennen | Vollständig abgedeckt |
| Klassen ohne hinterlegten Instanzen erkennen | Vollständig abgedeckt |
| Fehlende Relationen Instanzen, für Join der Pfade, erkennen | - |

Tabelle 6.4: Ergebnis der Akzeptanztest zur Aufgabenunterstützung

### 6.5.3 Akzeptanztests

Für den Akzeptanztest wurde ein Fragebogen zu den jeweiligen Tasks und Subtasks aus Kapitel 4.2 erstellt. Das Ziel des Fragebogen ist es zu überprüfen inwieweit die Stakeholder mithilfe von CIONw die definierten Aufgaben lösen können. Dabei kann jede Aufgabe entweder vollständig, teilweise, geringfügig oder gar nicht abgedeckt sein. Neben der Abdeckung der Aufgaben wird mit dem Fragebogen auch die Bedienbarkeit für die einzelnen Aufgaben nach dem selben Bewertungsmöglichkeiten untersucht. Im Anhang A.3 befindet sich der Fragebogen für den Akzeptanztest. In Tabelle 6.4 ist das Ergebnis zu der Unterstützung von den definierten Aufgaben durch CIONw in Verbindung mit Cytoscape dargestellt. In der Tabelle 6.5 ist die Auswertung der Bedienbarkeit der einzelnen Aufgaben mit CIONw und Cytoscape dargestellt.

Die Aufgabe T1 mitsamt den drei Subtasks wird vollständig durch Cytoscape unterstützt. Dadurch erfolgt hier durchweg eine positive Bewertung.

Die Subtask „Auswerten der Instanzen entlang des Pfades“ wurde nur teilweise abgedeckt und ist auch von der Bedienbarkeit als teilweise kompliziert eingestuft worden. Zum einen liegt die Ursache dafür in einer fehlenden Meldung bei erfolgreicher Exportierung eines Pfades und zum anderen daran, dass zu wenige Relationstabellen hinterlegt sind und dadurch nur geringfügig diese

| Task | Ergebnis |
|---|---|
| **T1: Navigation in einer Ontologie** | |
| Klassen in der Ontologie finden | Sehr einfach / intuitiv |
| Nachbarn zu einer ausgewählten Klasse finden | Sehr einfach / intuitiv |
| Instanzen zu konkreten Klassen finden | Sehr einfach / intuitiv |
| **T2: Pfade in Ontologie untersuchen** | |
| Einstellungen für die Pfadsuche machen (Gewichtung, Pfadlänge, ...) | Sehr einfach / intuitiv |
| Pfade finden | Sehr einfach / intuitiv |
| Auswerten der Instanzen entlang des Pfades | Teilweise aufwendig / kompliziert |
| **T3: Fehlende Instanzen auffinden** | |
| Nicht instanziierbare Klassen erkennen | Teilweise aufwendig / kompliziert |
| Klassen ohne hinterlegten Instanzen erkennen | Sehr einfach / intuitiv |
| Fehlende Relationen Instanzen, für Join der Pfade, erkennen | sehr aufwendig / kompliziert |

Tabelle 6.5: Ergebnis der Akzeptanztest zur Bedienbarkeit

Subtask getestet werden kann.

Die Subtask „Nicht instanziierbare Klassen erkennen“ ist von der Bedienbarkeit als teilweise kompliziert eingestuft worden. Die Ursache dafür ist, dass im `InstancePanel` keine separate Meldung für Klassen vom Typ `Function` erfolgt. Aber dennoch kann diese Information aus der Knotentabelle von Cytoscape abgelesen werden. Jedoch ist es unpraktisch, ständig in die Knoten Tabelle zu wechseln.

Die Subtask „Fehlende Relationen Instanzen, für Join der Pfade, erkennen“ wurde teilweise von den Stakeholdern nicht beantwortet. Deshalb kann dafür keine Auswertung erfolgen. Jedoch ist in den Kommentaren zu dieser Subtask deutlich geworden, dass die Meldung bei fehlenden Instanzen schwer zu interpretieren sind. Daher kann diese Subtask als erfüllt aber mit einer komplizierten Bedienbarkeit eingestuft werden.

Insgesamt wurden alle Tasks und Subtasks entweder als vollständig oder als teilweise erfüllt durch die Stakeholder eingestuft.

# 7 Ausblick und Zusammenfassung

In diesem Kapitel werden zunächst weitere mögliche Verbesserungen von CIONw beschrieben und im letzten Abschnitt eine abschließende Zusammenfassung der Arbeit dargestellt.

## 7.1 Ausblick

Der im Rahme dieser Arbeit entwickelte Prototype CIONw ist mit Abschluss der Arbeit nicht endgültig fertiggestellt. Da es ein Prototyp ist, können noch einige Fehler entdeckt und ausgebessert werden. Desweiteren wurden im Verlauf des Entwicklungsprozesses bereits weitere Verbesserungen und Funktionalitäten von den Stakeholdern angedeutet. Einige davon werden im folgenden kurz beschrieben.

**SNIK-Ontologie Migration** Zu Beginn muss eine Migration von dem Ontologieausschnitt auf die aktuelle SNIK-Ontologie erfolgen, die eine deutlich höhere Komplexität hat. Der hier verwendetet Ontologieausschnitt ist nahe an der SNIK-Ontologie angelehnt, aber dennoch können kleinere Probleme bei der Verwendung auf der SNIK-Ontologie entstehen. Beispielsweise wird in der SNIK-Ontologie der vollständige URI als Name für Knoten und Kanten verwendet. Das hat zufolge, dass die Lesbarkeit der Pfade sehr eingeschränkt ist. Dieses und weiter mögliche Probleme müssen bei der Migration analysiert und angepasst werden.

**Nutzbarkeit verbessern** Die Anforderungen, die im Rahmen des Systemtests nur teilweise erfüllt werden, können weiter ausgebaut und verbessert werden. Beispielsweise kann das `InstancePanel` für eine bessere Darstellung der Instanzen verbessert oder neu gestaltet werden. Weiterhin muss eine klare Meldung für Klassen vom Typ `Function` erfolgen.

**Zwischenknoten** Ein schwerwiegendes Problem stellt die Bestimmung von Pfaden mit Zwischenknoten dar. Wie im Entwurf dargelegt, ist dieses Problem relativ aufwendig und benötigt bei größeren Datenmengen eine Performanz-orientierte Implementierung. Zusätzlich muss für diese Problemstellung eine

weitere Literaturrecherche durchgeführt werden, um geeignete Algorithmen zu finden, die das Konzept der Zwischenknoten in der geforderten Form unterstützen.

**Pfadsuche in Web-Visualisierung** Die Integration der Pfadsuche und weiterer Funktionalitäten von CIONw in die Webdarstellung der SNIK-Ontologie, kann als ein weiteres Arbeitsfeld betrachtet werden. Es können die hier vorgestellten Algorithmen in JavaScript nachgebaut und damit die Web-Visualisierung erweitert werden. Auch eine Möglichkeit für einen Serverbetrieb von Cytoscape kann entwickelt werden und damit die Berechnung aus dem Browser des Nutzer zu einem Server zu verlagern.

**Instanzbeschaffung** CIONw bezieht die Instanzen zu Klassen und Kanten des Graphens bzw. der Ontologie aus den in Kapitel 5.2 dargestellten CSV-Dateien. Diese Datenquellen wurde gewählt, weil zum aktuellen Zeitpunkt nicht bekannt ist, aus welchen Quellen die Instanzinformationen kommen. Für die zukünftige Nutzung von CIONw in Verbindung mit der SNIK-Ontologie ist es von großem Interesse neue und bestehende Datenquellen für die Instanzdaten anzubinden. Dafür muss die Klasse `InstancReader` für die neuen Datenquellen erweitert werden. Beispielsweise können die Instanzen aus einer Datenbank per SQL Anfrage geladen werden. Die Integration von heterogenen Daten ist bereits in der Bachelorarbeit von Tröster untersucht worden [32]. Dabei wird der Ontologie-basierter Datenzugriff (engl. Ontology Based Data Access) genauer untersucht. Die dabei gewonnen Erkenntnisse können bei der Instanzbeschaffung von CIONw behilflich sein.

Zusätzlich zu den genannten Problemstellungen, können weitere Anpassungen und Funktionalitäten für CIONw entworfen und umgesetzt werden. Beispielsweise kann eine statistische Betrachtung der SNIK-Ontologie erfolgen, um damit die Gewichtung und somit die Pfadfindung zu verbessern. Dadurch kann auch eine initiale Empfehlung für die Gewichtung der Kanten angeboten werden.

## 7.2 Zusammenfassung

Das Ziel dieser Masterarbeit ist es, durch ein strukturiertes Requirements Engineering die Anforderungen an ein System zur Pfadfindung in Ontologien und Verknüpfung beteiligter Daten, zu ermitteln und deren Umsetzung mit einem Prototypen (CIONw) zu zeigen. Dafür wurden die Anforderungen der Stakeholder nach der TORE Methode erhoben und analysiert. Zusätzlich zu der Anforderungserhebung wurde eine Literaturrecherche durchgeführt. Mit der Recherche wurde eine ähnliche Arbeit von Spahn et al. untersucht und daraus Anreize für das Design von CIONw übernommen. Zusätzlich wurde durch die Literaturrecherche die Bestimmung von $k$-kürzesten Pfaden in einer Ontologie als Lösung für das Auffinden von mehreren Pfaden identifiziert.

Durch die Erkenntnisse aus der Literaturrecherche wurden die beiden Algorithmen von Dijkstra und Yen als Grundlage für die Pfadbestimmung gewählt. Die Implementierung von CIONw ist als eine App für das Programm Cytoscape, umgesetzt worden. Diese Entscheidung wurde maßgeblich durch das SNIK-Projekt geprägt, weil Cytoscape (sowohl Desktop als auch JavaScript Version) aktiv im Projekt genutzt wird. CIONw deckt die wesentlichen Anforderungen erfolgreich ab. Die Abdeckung wurde durch Systemtests zu jeder Anforderung und zusätzlich durch einen Akzeptanztest durch die Stakeholder untersucht. Dabei beschränkte sich der Akzeptanztest auf die definierten Tasks und Subtasks.

Die drei Ziele der Arbeit (Z1 - Anforderungsspezifikation, Z2 - Navigation in Ontologien und Z3 - Prototyp Entwicklung) wurden umgesetzt. Dabei unterstützte die TORE Methode in wichtigen Bereichen die Anforderungserhebung. Insbesondere ermöglichte die TORE Methode ein frühzeitiges erkennen von offenen bzw. unklaren Probleme, welche in den durchgeführten Interviews gelöst wurden. Dennoch konnte die Anforderung der Pfadsuche mit mehreren Zwischenklassen nicht erfüllt werden. Diese Anforderung beschreibt eine Problemstellung der Pfadfindung, welche weder durch den Dijkstra noch den Yen Algorithmus erfüllt werden kann. Insbesondere zeigte sich, dass diese Anforderung ein komplexes eigenständiges Problem darstellt, welches in einer gesonderten Arbeit untersucht werden muss.

Aufgrund dessen, dass CIONw eine Prototyp-Entwicklung darstellt, können noch viele weitere Verbesserungen und Erweiterungen von Funktionalitäten entwickelt werden. Einige möglichen Verbesserungen und Erweiterungen wurden exemplarisch dargelegt (z.B. Erweiterung der Pfadsuche für Zwischenknoten und Instanzen aus heterogenen Quellen beschaffen).

# Literaturverzeichnis


[1] Gruber T. R, et al.: A translation approach to portable ontology specifications. Knowledge acquisition 5(2) 1993 199–220

[2] Paech B, Kohler K. In: Task-Driven Requirements in Object-Oriented Development. Springer US, Boston, MA 2004 45–67

[3] Winter A, Haux R, Ammenwerth E, Brigl B, Hellrung N, Jahn F: Health Information Systems: Architectures and Strategies. Health Informatics. Springer London 2011

[4] Guarino N, Oberle D, Staab S: What is an ontology? In: Handbook on ontologies. Springer 2009 1–17

[5] Paulheim H, Probst F: Ontology-enhanced user interfaces: A survey. Int. J. Semant. Web Inf. Syst. 6(2) April 2010 36–59

[6] Guarino N: Formal ontology and information systems. In: Proceedings of FOIS. Volume 98. 1998 81–97

[7] Ammenwerth E, Haux R, Knaup-Gregori P, Winter A: IT-Projektmanagement im Gesundheitswesen: Lehrbuch und Projektleitfaden Taktisches Management von Informationssystemen. Schattauer Verlag 2014

[8] Schaaf M, Jahn F, Tahar K, Kücherer C, Winter A, Paech B: Entwicklung und Einsatz einer Domänenontologie des Informationsmanagements im Krankenhaus

[9] Roussey C, Pinet F, Kang M. A, Corcho O: An introduction to ontologies and ontology engineering. In: Ontologies in Urban Development Projects. Springer 2011 9–38

[10] Adam S, Doerr J, Eisenbarth M, Gross A: Using task-oriented requirements engineering in different domains–experiences with application in research and industry. In: 2009 17th IEEE International Requirements Engineering Conference, IEEE 2009 267–272

[11] Lauesen S: Task & support-task descriptions as functional requirements. In: Proceedings of AWRE, Citeseer 2001

[12] Cohn M: User Stories Applied: For Agile Software Development. Addison-Wesley signature series. Addison-Wesley 2004

[13] Kitchenham B, Charters S: Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, EBSE Technical Report EBSE-2007-01 2007

[14] Webster J, Watson R. T: Analyzing the past to prepare for the future: Writing a literature review 2002

[15] Jalali S, Wohlin C: Systematic literature studies: Database searches vs. backward snowballing. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '12, New York, NY, USA, ACM 2012 29–38

[16] OBI '08: Proceedings of the First International Workshop on Ontology-supported Business Intelligence, New York, NY, USA, ACM 2008

[17] Lula P, Paliwoda-Pękosz G: An ontology-based cluster analysis framework. In: Proceedings of the First International Workshop on Ontology-supported Business Intelligence. OBI '08, New York, NY, USA, ACM 2008 7:1–7:6

[18] Spahn M, Kleb J, Grimm S, Scheidl S: Supporting business intelligence by providing ontology-based end-user information self-service. In: Proceedings of the First International Workshop on Ontology-supported Business Intelligence. OBI '08, New York, NY, USA, ACM 2008 10:1–10:12

[19] Zhang X, Jing L, Hu X, Ng M, Zhou X: A comparative study of ontology based term similarity measures on pubmed document clustering. In: International Conference on Database Systems for Advanced Applications, Springer 2007 115–126

[20] Schickel-Zuber V, Faltings B, et al.: Oss: A semantic similarity function based on hierarchical ontologies. In: IJCAI. Volume 7. 2007 551–556

[21] Angele J, Gesmann M: Data integration using semantic technology: a use case. In: 2006 Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML'06), IEEE 2006 58–66

[22] Caires B, Cardoso J: Using semantic web technologies to build adaptable enterprise information systems. IBIS 3 2006 41–58

[23] Mertens M: Wissensbasiertes Business Intelligence für die Informations-Selbstversorgung von Entscheidungsträgern. In: Grundlagen von Datenbanken. 2011 43–48

[24] Arvanitis A, Wiley M. T, Hristidis V: Efficient concept-based document ranking. In: EDBT. 2014 403–414

[25] Davies J, Weeks R: Quizrdf: Search technology for the semantic web. In: System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on, IEEE 2004 8–pp

[26] Janecek P, Schickel V, Pu P: Concept expansion using semantic fisheye views. In: International Conference on Asian Digital Libraries, Springer 2005 273–282

[27] Maguitman A. G, Menczer F, Roinestad H, Vespignani A: Algorithmic detection of semantic similarity. In: Proceedings of the 14th international conference on World Wide Web, ACM 2005 107–116

[28] Saake G, Sattler K.-U: Algorithmen und Datenstrukturen: Eine Einführung mit Java. dpunkt. verlag 2014

[29] Yen J. Y: Finding the k shortest loopless paths in a network. management Science 17(11) 1971 712–716

[30] Kahate A: Introduction to Database Management Systems. Pearson Education (Singapore) 2004

[31] Dudáš M, Zamazal O, Svátek V. In: Roadmapping and Navigating in the Ontology Visualization Landscape. Springer International Publishing, Cham 2014 137–152

[32] Tröster B: Einsatz einer Ontologie als Datenmodell zur Datenaggregation. Bachelorarbeit, Universität Heidelberg 2016

[33] Sommerville I: Software Engineering. International computer science series. Addison-Wesley 2007

[34] Pohl K, Rupp C: Basiswissen Requirements Engineering: Aus- und Weiterbildung zum "Certified Professional for Requirements Engineering"; Foundation Level nach IREB-Standard. ISQL-Reihe. dpunkt-Verlag 2011

# Glossar

**Apache Maven** ist ein Build-Management Tool, welches alle möglichen Java Projekte verwalten und compilieren kann[13]. Maven ermöglicht es den einfachen Zugriff auf viele Java Programmen durch ein Repository.

**API** application programming interface. Ist eine Programmierschnittstelle einer Anwendung, welcher es ermöglicht die Anwendung in anderen Anwendungen einzubinden oder zu erweitern.

**Bottom-Up** ist eine Betrachtungsweise bzw. Bearbeitungsweise von der konkreten Sichtweise hin zu einer abstrakten Sichtweise. Die umgekehrte Richtung ist Top-Down.

**Brainstorming** ist ein unstrukturiertes Vorgehen zum Sammeln von Gedanken und Ideen zu einem Thema. Brainstorming wird zumeist am Anfang einer Aufgaben- oder Problemstellung durchgeführt.

**Build-Management Tool** ist eine Anwendung, die den gesamten Erstellungsprozess einer Software gebündelt koordiniert. Es schließt das automatische Kompilieren und Testen des Softwarecodes ein.

**CSV** Comma separated values. Ist ein strukturierte Textdatei, die zur (systemunabhängigen) Speicherung von Werten dient. Die Werte werden jeweils mit Kommas getrennt. CSV Dateien können ganze Tabellen darstellen. Dabei kann die Spaltenüberschrift in der ersten Zeile angegeben werden.

**Framework** ist eine Sammlung von Softwareartefakten, welche eine Wiederverwendbarkeit für ähnliche Problemstellungen ermögliche [33] .

**GUI** grafical user interface. Steht für eine grafische Nutzeroberfläche in einer Software.

**IT** Informationstechnik.

**JavaDoc** ist ein Dokumentationswerkzeug für Java Code. Aus den JavDoc Kommentaren können automatisch HTML Seiten erstellt werden, die die Dokumentation beinhalten.

[13] https://maven.apache.org/what-is-maven.html

**JSON** JavaScript Object Notation. Stellt eine Beschreibung für Objekte dar um diese zwischen Programmen austauschen zu können.

**Nachbarschaftsgrad** beschreibt die Tiefe der Nachbarschaftsbeziehung. In einem Graphen gibt der Nachbarschaftsgrad an, wie viele Kanten zwischen dem Knoten und seinem Nachbarn liegen dürfen. Nachbarschaftsgrad = 1 entspricht den direkten Nachbarn, 2 entspricht alle Nachbarn der Nachbarn mit Grad 1 usw.

**OSGi** ist ein dynamisches Kompomentensystem für Java. Es bietet die Möglichkeit Komponenten zu Kapseln und für die Wiederverwendung über Services zur Verfügung zustellen[14].

**Requirements Engineering** bezeichnet das gesamte Vorgehen zum systematischen erheben von Anforderungen im Bereich der Softwareentwicklung. Es hat zum Ziel, alle Anforderungen von den Stakeholdern zu identifizieren und diese strukturiert zu dokumentieren [34].

**XML** Extensible Markup Language. XML ist eine Auszeichnungssprache mit welcher Datenstrukturen beschrieben werden können .

[14] https://www.osgi.org/developer/architecture/

# A Anhang

## A.1 SNIK-Ontologie Ausschnitt

### A.1.1 Klassen des SNIK-Ausschnittes

**Strategic_Information_Management_Plan** Der Strategische Information Management Plan entsteht als Ergebnis aus der strategischen HIS Planung und bildet eine wichtige Grundlage für das Steuern und Überwachen vom HIS. Der Plan besteht im Allgemeinen aus 5 Teilen: Ziele des Informationsmanagements, Beschreibung des aktuellen Zustandes, Analyse des aktuellen Zustandes, Beschreibung des geplanten Zustandes und der Weg vom aktuellen zum geplanten Zustand.

**Strategic_Information_Management_Goal** Die Ziele des strategischen Informationsmanagements basieren auf der Beschreibung der Ziele des Krankenhauses.

**Discription_of_the_Current_HIS_State** Die Beschreibung des aktuellen Standes des HIS dient unter anderem als Grundlage für die Bestimmung der Umgesetzten und nicht umgesetzten Funktionalitäten und deren Unterstützung des Informationsmanagements.

**Analysis_and_Assessment_of_the_Current_HIS_State** Bei der Analyse des aktuellen Standes des HIS wird die Erfüllung der Informationsmanagement Strategie betrachtet und somit der aktuelle Stand des HIS bewertet. Da es sich bei der Analyse um eine Funktion des Informationsmanagements handelt, gibt es zu dieser Klasse keine Instanzen(vgl. Kapitel 5.2.2).

**Description_of_the_Planned_HIS_State** In der Beschreibung des geplanten Standes des HIS werden, aufbauend auf der Analyse des aktuellen Standes, die geplanten Änderungen am HIS beschrieben. Diese Beschreibung dient der Verdeutlichung der zu verrichtenden Arbeit, um ein strategisches Ziel zu erreichen.

**Migration_Path** Der Migrationspfad beschreibt detailliert den Prozess zur Migration eines Systems vom aktuellen Zustande, bis hin zum geplanten Zustand. Dabei kann jeder Teilschritt ein eigenständiges Projekt darstellen.

**Strategic_Hospital_Goal** Diese Klasse beschreibt die einzelnen strategischen Ziele eines Krankenhauses und ist Teil des strategischen Informationsmanagement Planes und der Krankenhausziele.

**Hospital_Management** Das Krankenhausmanagement ist für die grundlegende Ausrichtung und Entwicklungen des Krankenhauses verantwortlich. Insbesondere hat das Krankenhausmanagement Auswirkungen auf die Ziele, die Strategie, die Angestellten, das Budget und Investitionen des Krankenhauses. Da das Krankenhausmanagement eine Funktion ist, besitzt diese Klasse keine Instanzen.

**Long-Term_HIS_Planning** Die langfristige HIS Planung beschreibt die Planung des HIS für einen längeren Zeitraum (z.B. 3-5 Jahre). Die Ergebnisse der langfristigen HIS Planung fließen in den strategischen Informationsmanagement Plan ein. Da die langfristige HIS Planung eine Funktion des Informationsmanagements ist, besitzt diese Klasse keine Instanzen.

**Strategic_HIS_Planing** Die strategische HIS Planung ist eine Aufgabe des Informationsmanagements und ist für den Aufbau des HIS und der Organisierung des Informationsmanagements verantwortlich. Es ist ebenfalls für die kurzfristige und langfristige HIS Planung zuständig. Da die Strategische HIS Planung eine Funktion des Informationsmanagements ist, besitzt diese Klasse keine Instanzen.

**Software_Vendor** Der Begriff Software Vendor bezeichnet einen Softwarehersteller.

**Strategic_HIS_Directing** Die Strategische HIS Steuerung ist eine Teilaufgabe des Strategischen Informationsmanagements und besteht darin, den strategischen Informationsmanagement Plan in viele kleine Teilaufgaben zu zerlegen. Die Teilaufgaben wirken auf das HIS, so dass dieses den strategischen Plan unterstützt. Da die Strategische HIS Leitung eine Funktion des Informationsmanagements ist, besitzt diese Klasse keine Instanzen.

**Project_Portfolio** Das Projekt Portfolio ist eine Auflistung von Projekten, welche von der kurzfristigem HIS Planung erstellt wird und stellt die Gesamtheit aller laufenden Projekte dar.

**Project** Ein Projekt ist ein Vorhaben, das im Wesentlichen durch die Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z. B.

klare Zielvorgabe, zeitliche Begrenzung, personelle und finanzielle Begrenzungen, Abgrenzung gegenüber anderen Vorhaben, und projektspezifische Organisation[15].

**Portfolio_Management** Das Portfolio Management kategorisiert und bewertet ausgewählte Komponenten des Informationssystems, auf ihre Bereicherung zum Unternehmen. Ebenso werden die Risiken dieser Komponenten für das Unternehmen bewertet. Die Komponenten können Anwendungen, physische Datenverarbeitungssysteme und auch Projekte sein. Da das Portfolio Management eine Funktion des Informationsmanagements ist, besitzt diese Klasse keine Instanzen.

**Hospital_Information_System** siehe Informationssysteme im Krankenhaus in Abschnitt 2.1.

**Project_Execution** Die Projekt Ausführung fasst die Durchführung aller Projektbestandteile zusammen. Da es sich hierbei um eine Funktion handelt, besitzt diese Klasse keine Instanzen.

**Application_Component** Eine Anwendungskomponente ist eine Ansammlung von Regeln für die Kontrolle von physischen Datenverarbeitungsprozessen. Die Regeln müssen so aufgestellt sein, dass sie auch tatsächlich ausgeführt werden können.

**Computer-Based_Application_Component** Eine computerbasierte Anwendungskomponente ist eine Anwendungskomponente, bei der die Regeln als eine (vom Computer) ausführbare Anwendung vorliegt (z.B. Software).

**Patient_Data_Management_System** Das Patientendaten Managementsystem ist eine Anwendungskomponente, welche für die Überwachung, Speicherung und Darstellung von Patientendaten zuständig ist.

**HIS_Component** beschreibt alle Arten von Bestandteilen des Krankenhaus Informationssystems.

[15] nach DIN 69901

### A.1.2 Instanztabellen für Klassen

| #SIMP | Strategic_Information_Management_Plan |
|---|---|
| #SIMP01 | SIM-Plan UML 2012 |
| #SIMP02 | SIM-Plan UML 2016 |
| #SIMP03 | SIM-Plan UML 2018 |

| #SIMG | Strategic_Information_Management_Goal |
|---|---|
| #SIMG0101 | Verkürzung der Kommunikationswege |
| #SIMG0102 | Patientenfreundliche Informationsdarstellung in den Wartebereichen |
| #SIMG0201 | Digitalisierung des Krankenhauses |
| #SIMG0202 | Datenaustausch verbessern |
| #SIMG0301 | Verbesserte Krankenhauslogistik |

| #SIMPDCS | Discription_of_the_Current_HIS_State |
|---|---|
| #SIMPDCS0101 | Aktueller HIS Stand von 2012 |
| #SIMPDCS0201 | Aktueller HIS Stand von 2016 |
| #SIMPDCS0301 | Aktueller HIS Stand von 2018 |

| #CBAC | Computer_Based_Application_Component |
|---|---|
| #CBAC1 | PDMS Copra 6 |
| #CBAC2 | PDMS LOWTeq |
| #CABC3 | Tisoware.PEP |
| #CABC4 | ORBIS AIMS |
| #CABC5 | ORBIS OP-Management |
| #CABC6 | PDMS CareVue 9000 |

| #PDMS | Patient_Data_Management_System |
|---|---|
| #PDMS1 | PDMS Copra 6 |
| #PDMS2 | PDMS LOWTeq |
| #PDMS3 | PDMS CareVue 9000 |

| #SIMPDP | Description_of_the_Planned_HIS_State |
|---|---|
| #SIMPDP0101 | *Vision*: Patienten bekommen im Wartebereich eine bessere Informationsvermittlung<br>*geplante Funktionen*: Darstellung von Informationen über ein Display im Wartebereich<br>*geplante Anwendungen*: Anwendung zur Bestimmung und Darstellung der Patienten-Reihenfolge<br>*Geplante Peripheriegeräte*: Display im Wartebereich, Eingabeterminal im Aufnahmebereich |
| #SIMPDP0102 | *Vision*: Mitarbeiter können über separate Kommunikationskanäle miteinander kommunizieren<br>*geplante Anwendungen*: Kombination von E-Mail + Datenaustausch + Voice/Video Messages in einer Anwendung<br>*geplante Peripheriegeräte*: Pro Terminal eine Webcam |
| #SIMPDP0201 | *Vision*: Alle Datenerfassungen erfolgen über Tablets<br>*geplante Anwendungen*: Mobile Patientenerfassung, Mobile Arztbrieferstellung<br>*geplante Peripheriegeräte*: Android Tablets |
| #SIMPDP0202 | *Vision*: Eine zentrale Datenablage mit ausgiebiger Rechteverwaltung und Versionierung<br>*geplante Anwendungen*: Serveranwendung für Datenablage<br>*geplante Peripheriegeräte*: - |
| #SIMPDP0301 | *Vision*: Jederzeit feststellbar, wo sich Gerätschaften im Haus befinden<br>*geplante Anwendungen*: Überwachung und Monitoring Anwendung über alle Geräte<br>*geplante Peripheriegeräte*: Pro elektronischem Gerät einen RFID Chip, RFID Lokalisierungssystem |

| #SIMPMP | Migration_Path |
|---|---|
| #SIMPMP0101 | Migration 2009-2012 |
| #SIMPMP0201 | Migration 2012-2016 |
| #SIMPMP0301 | Migration 2016-2018 |

| #SHG | Strategical_Hospital_Goal |
|---|---|
| #SHG0101 | Verringerung der losen Papierunterlagen um 90% |
| #SHG0102 | Verringerung der Suche- und Wartezeiten für Gerätschaften/Instrumente |
| #SHG0201 | Ausbau der Aufnahmekapazität der Notaufnahme um 50% |
| #SHG0301 | Vernetzung mit der Universität |

| #SV | Software_Vendor |
|---|---|
| #SV01 | Mustermann GmbH |
| #SV02 | Otto Freelancer |
| #SV03 | Medical Consulting |
| #SV04 | Dynamic Integration AG |
| #SV05 | Medical Techonologies and Systems AG |

| #PP | Project_Portfolio |
|---|---|
| #PP01 | Strategische Migration 2009-2012: Austausch Router Bereich A und B, Einrichtung Display Wartebereich, Entwicklung einer (Warte-) Reihenfolge Verwaltung, ... |
| #PP02 | Strategische Migration 2012-2016: Schulung WIN10, Schulung MS Excel 16, ... |
| #PP03 | Strategische Migration 2016-2018: RFID Einführung, RFID Einbindung Geräte Bereich D, Zentrale Geräte Lokalisierung, ... |

| #P | Project |
|---|---|
| #P001-1 | Austausch Router Bereich A |
| #P001-2 | Austausch Router Bereich B |
| #P002-1 | Schulung WIN 10 |
| #P002-2 | Schulung MS Excel 16 |
| #P002-3 | Schulung MS Project 16 |
| #P002-3 | Schulung MS Word 16 |
| #P003-1 | RFID Einführung |
| #P003-2 | RFID Einbindung Geräte Bereich D |
| #P003-3 | Zentrale Geräte Lokalisierung |
| #P004-1 | Prototyp Test: Elek. Aufnahmeprotokoll |
| #P005-1 | Einrichtung Display Wartebereich |
| #P005-2 | Entwicklung einer (Warte-)Reihenfolge Verwaltung |

| #HIS | Hospital_Information_System |
|---|---|
| #iHIS1 | i.s.h.med |

| #AC | Application_Component |
|---|---|
| #AC1 | PDMS Copra 6 |
| #AC2 | PDMS LOWTeq |
| #AC3 | Tisoware.PEP |
| #AC4 | ORBIS AIMS |
| #AC5 | ORBIS OP-Management |
| #AC6 | PDMS CareValue 9000 |
| #AC7 | Papierbasierte Patientenkarten Archive |
| #AC8 | Papierbasierte Pflegedokumentation |
| #AC9 | Ausgehängter Schichtplan Bereich A |

### A.1.3 Relationstabellen

| **hasSubClassComputer-Based_Application_ComponentPatient_Data_Management_System** | | |
|---|---|---|
| „#hasSubClass1“ | „#CBAC1“ | „#PDMS1“ |
| „#hasSubClass2“ | „#CBAC2“ | „#PDMS2“ |
| „#hasSubClass3“ | „#CBAC6“ | „#PDMS3“ |

| **hasSubClassApplication_ComponentComputer-Based_Application_Component** | | |
|---|---|---|
| „#hasSubClassACCBAC1“ | „#AC1“ | „#CBAC1“ |
| „#hasSubClassACCBAC2“ | „#AC2“ | „#CBAC2“ |
| „#hasSubClassACCBAC3“ | „#AC3“ | „#CBAC3“ |
| „#hasSubClassACCBAC4“ | „#AC4“ | „#CBAC4“ |
| „#hasSubClassACCBAC5“ | „#AC5“ | „#CBAC5“ |
| „#hasSubClassACCBAC6“ | „#AC6“ | „#CBAC6“ |

| **usesPortfolio_ManagementupdatesApplication_ComponentProject** | | |
|---|---|---|
| „#uPMu1“ | „#AC8“ | „#P004-1“ |
| „#uPMu2“ | „#AC8“ | „#P005-1“ |
| „#uPMu3“ | „#AC1“ | „#P006-1“ |

| **isDecomposedInMigration_PathProject** | | |
|---|---|---|
| „#isDMP1“ | „#SIMPMP0101“ | „#P001-1“ |
| „#isDMP2“ | „#SIMPMP0101“ | „#P001-2“ |
| „#isDMP3“ | „#SIMPMP0201“ | „#P002-1“ |

| **updatesProject_ExecutionupdatesApplication_Component Hospital_Information_System** | | |
|---|---|---|
| „#uPEuACHIS“ | „#AC2“ | „#iHIS1“ |

## A.2 Interviewfragen

### A.2.1 Erstes Interview

1. Wer soll die Anwendung Nutzen?
2. Wer könnte ein zusätzliches Interesse an der Anwendung haben?
3. Was ist das Ziel von CIONw?
4. Welche Ziele verfolgen die Anwender mit der Verwendung der Anwendung?
5. Bei welchen Aufgabenstellungen soll CIONw den Nutzer unterstützen?
6. Welche Vorteile sollen dem Anwender durch die Verwendung des Programmes entstehen?
7. Welche Aufgaben hat der Nutzer in seiner Rolle als CIO (Unabhängig von CIONw)?
8. Welche Aufgaben will der Anwender mit der Applikation lösen oder bearbeiten?
9. Welche bereits bestehenden Verfahren (unabhängig von den Verwendeten Tools) lösen das Problem auf ähnliche Weise und wie funktionieren diese Verfahren?
10. Wie soll der Nutzer das Problem mithilfe der Anwendung lösen?
11. Anhand welcher Kriterien sollen die gefunden Wege zwischen zwei Klassen bewertet werden?
12. Welche Daten werden benötigt, um die oben genannten Aufgaben zu erledigen?
13. Falls auf einem Ausschnitt der Daten gearbeitet wird; wie wird der Ausschnitt definiert/bestimmt? Wie werden Klassen (des Ausschnittes) mit Instanzen gefüllt?
14. Wie sollen die Beziehungen interpretiert werden, gerichtet oder angerichtet? Falls gerichtet: darf nur in die Pfeilrichtung geschaut werden? Dürfen trotzdem Wege „gegen die Pfeilrichtung“ betrachtet werden?
15. In welchem Format liegen die Instanz-Informationen vor?
16. Welche Schritte müssen vom Nutzer im Einzelnen durchgeführt werden um an die Lösung zu gelangen?
17. Welche Schritte sollen automatisch vom Programm durchgeführt werden?
18. Wie soll der Nutzer mit den Daten interagieren können?
19. Welcher Teil der Daten muss dem Nutzer angezeigt werden?
20. Was sind die Attribute der einzelnen Klassen? Wo kommen die Attribute

her? Inwieweit sind die Attribute bereits heute in der SNIK Ontologie enthalten?
21. Woher kommen die Informationen für die Relation zwischen zwei Klassen?
22. Wie müssen die Daten und Interaktionen mindestens dem Nutzer dargestellt werden?
23. Wie sollten die Daten und Interaktionen am besten dem Nutzer dargestellt werden?
24. Was soll geschehen, wenn mehrere Wege zwischen zwei Klassen gefunden werden?

### A.2.2 Zweites Interview

1. Was ist der Unterschied zwischen CIONw und CIONs?
2. Was sind die Rahmenbedingungen für das Explorieren?
3. Was ist hier als Tätigkeit gemeint? Sollte es nicht einfach nur eine Verbindung sein?
4. Ist hier Kausalität der richtige Term? Oder sollte es doch nur Nachverfolgbarkeit bzw. Traceability sein?
5. Welche Informationen müssen zu jeder Klasse auf der „Kette“ (dem Weg) angezeigt werden?
6. Wie wird die Zuordnung zwischen Ontologie und Daten hergestellt?
7. Wer legt diese Zuordnung fest und wann?
8. Wie wird bestimmt, was Relevante Informationen sind?
9. Was passiert mit Information die nicht aggregiert werden können?
10. Soll CIONw auf ein Ereignis (hier Ausfall) reagieren?
11. Wie können verbundene Prozesse (somit Klassen der Ontologie) identifiziert und gefunden werden?
12. Sollen dabei (nur die passenden) Instanzen oder nur die Klassen gefunden werden?
13. Woher kommen die Berichte? Sollen die Berichte als Schablonen vorher vom Nutzer erstellt werden, gespeichert und bei Bedarf wieder geladen werden?
14. Was passiert, wenn eine Klasse der Ontologie ausgewählt wird, die keine Instanz hat?
15. Was passiert bei „undefinierten“Auswahlpaaren: Klasse CIO + Klasse Mitarbeiter
16. Wie sollen neu Ideen (bzw. Verknüpfungen) aus dem Graphen hergeleitet werden?

17. Welche Daten werden benötigt, um die oben genannten Aufgaben zu erledigen?
18. Sind die Daten für die Anwendung vollständig? Falls nein, welche Auswirkungen haben diese unvollständigen Daten auf das Problem? Inwieweit können die Daten noch ermittelt werden?
19. Was sollte passieren, wenn die Anwendung auf nicht selbst erstellte Daten arbeitet und dabei auf Unvollständige Daten stößt?
20. Von wem wird der Ausschnitt bestimmt?
21. Wann wird der Ausschnitt bestimmt?
22. Für welche Aufgabe bzw. welchen Schritt werden die Relationstabellen benötigt

### A.2.3 Drittes Interview

1. Woher kommen die Verknüpfungen von den Relationen (in der Ontologie) zu den Relationstabellen ?
2. Wer legt diese Zuordnung fest?
3. Wenn die Tabellen in Excel ausgegeben werden sollen, wie sollen darin die Beziehungen (Relationen) zwischen den einzelnen Klassen dargestellt werden?
4. Sollen mehrere Instanzen einer Klasse auswählbar sein?
5. Falls ja, wie wirkt sich das auf die Ergebnisse (des Joins) aus?
6. Wirkt sich dir Relationsrichtung auf die Ausgabe in der Tabelle aus?
7. Attribute von Klassen werden wiederum als Klassen in der Ontologie dargestellt. Sollen dennoch die Einträge für Attribute in den Klassen Tabellen beibehalten werden?
8. Wie sollen die Attribute (der Tabelle) bei der Ausgabe angezeigt werden?
9. Soll es Beschränkungen für zu lange Wege geben?
10. Für was werden die IDs in den Relationstabellen (erste Spalte) benötigt?
11. Ist der natural Join die richtige Art der Verknüpfung?
12. Was passiert im natural Join, wenn eine Instanz nicht gefunden werden kann?
13. Wie soll mit dieser Vielzahl an Ergebnissen gehandhabt werden?

## A.3 Fragebogen Akzeptanztest

Wie beurteilen Sie die Unterstützung der **Aufgaben**, die Sie mit dem CIONw Prototyp in Verbindung mit Cytoscape lösen möchten?

| Task | Vollständig abgedeckt | Teilweise abgedeckt | Geringfügig abgedeckt | Nicht abgedeckt | Kommentar |
|---|---|---|---|---|---|
| **T1: Navigation in einer Ontologie**<br>Klassen in der Ontologie finden<br>Nachbarn zu einer ausgewählten Klasse finden<br>Instanzen zu konkreten Klassen finden | | | | | |
| **T2: Pfade in Ontologie untersuchen**<br>Einstellungen für die Pfadsuche machen (Gewichtung, Pfadlänge, ...)<br>Pfade finden<br>Auswerten der Instanzen entlang des Pfades | | | | | |
| **T3: Fehlende Instanzen auffinden**<br>Nicht instanziierbare Klassen erkennen<br>Klassen ohne hinterlegten Instanzen erkennen<br>Fehlende Relationsinstanzen, für Join der Pfade, erkennen | | | | | |

Wie beurteilen Sie die **Bedienbarkeit der einzelnen Aufgaben** die Sie mit CIONw und Cytoscape lösen wollen?

| Task | Sehr einfach / intuitiv | Teilweise aufwendig / kompliziert | sehr aufwendig / kompliziert | Nicht bedienbar | Kommentar |
|---|---|---|---|---|---|
| **T1: Navigation in einer Ontologie**<br>Klassen in der Ontologie finden<br>Nachbarn zu einer ausgewählten Klasse finden<br>Instanzen zu konkreten Klassen finden | | | | | |
| **T2: Pfade in Ontologie untersuchen**<br>Einstellungen für die Pfadsuche machen (Gewichtung, Pfadlänge, ...)<br>Pfade finden<br>Auswerten der Instanzen entlang des Pfades | | | | | |
| **T3: Fehlende Instanzen auffinden**<br>Nicht instanziierbare Klassen erkennen<br>Klassen ohne hinterlegten Instanzen erkennen<br>Fehlende Relationsinstanzen, für Join der Pfade, erkennen | | | | | |

# Abbildungsverzeichnis

# Tabellenverzeichnis