Technische Hochschule Wildau
Technical University of Applied Sciences

Master-Thesis

zur Erlangung des akademischen Grades
Master

Technische Hochschule Wildau

Fachbereich Wirtschaft, Informatik, Recht

Studiengang Bibliotheksinformatik (M.Sc.)

Thema (deutsch): (Teil-)Automatisierung eines Workflows zur Bewertung und Integration heterogener Datenbestände in eine Linked-Data-basierte Forschungsdateninfrastruktur

Thema (englisch): (Part) Automation of a data integration and valuation workflow for heterogeneous data in a linked data based research infrastructure

Autor: Tracy Hoffmann
Matrikelnr.: 050030158
Seminargruppe: BIM / 15

Betreuer: Peter Morcinek M.Sc.
Gutachter: Prof. Dr. Thomas Riechert

Reg.-Nr.: BIM 511/17

Eingereicht am: 11.05.2018

Zusammenfassung

Die Motivation für diese Arbeit entstand durch den ansteigenden Bedarf an Unterstützung bei der Datenverarbeitung innerhalb von Digital Humanities Projekten. Die vorliegende Masterarbeit hat das Ziel, den Workflow für die Datenintegration im Forschungsprojekt diggr in den Punkten Transparenz, Nachvollziehbarkeit, Wiederholbarkeit und Automatisierung zu verbessern. Diese Aspekte werden berücksichtigt, indem eine koordinierte Toolchain aufgebaut wird, die überwiegend mit dem Resource Description Framework (RDF) arbeitet. Innerhalb dieser Toolchain werden Datenqualitätsprüfungen durch ein formalisiertes Verfahren unterstützt. Zuvor werden Anwendungsfälle betrachtet, aus denen anschließend Anforderungen extrahiert und beschrieben werden. Danach folgt die Entwicklung und Implementierung der einzelnen Komponenten.

Schlagwörter: Linked Data, Digital Humanities, Datenintegration, Workflow, SHACL

Abstract

The motivation for this master thesis arose from the increasing need for support in data processing within digital humanities projects. The aim was to improve the data integration workflow of the research project diggr with regard to transparency, traceability, repeatability and automation. These aspects are taken into account by building a coordinated toolchain using the Resource Description Framework (RDF). Within this toolchain, data quality checks are supported by a formalized procedure. Use cases were considered from which requirements were subsequently be extracted and described. This was followed by the development and implementation of the individual components.

Keywords: Linked Data, Digital Humanities, Data Integration, Workflow, SHACL

Inhaltsverzeichnis

1 Einleitung

> *But the field of digital humanities is still maturing; it is embedded in and it supports a research domain that is based on heterogeneous data and divergent research questions.*
>
> van Zundert, 2012

Der Einsatz von Informationstechnologien und die damit verbundenen Änderungen haben in den vergangenen Jahren zu einer Transformation der Kultur- und Geisteswissenschaften hin zu den sogenannten Digital Humanities (DH) geführt. Die draus resultierenden neuen Arbeitsschwerpunkte und Aufgaben müssen innerhalb von Forschungsprojekten dieser Art berücksichtigt werden. Bisher lag der Fokus in vielen DH-Projekten darauf, Daten digital zu erfassen oder Daten, die bereits digital vorliegen, auszuwerten. Es gibt jedoch auch Projekte, die Daten aus Online-Quellen für die Forschung in den digitalen Geisteswissenschaften aufbereiten und nutzen. Als Online-Quellen können alle über das Internet zugänglichen Informationen dienen. Das können sowohl klassische Datenbanken als auch Weblogs, Foren, Webseiten oder Social Media Plattformen sein. Der Zugriff auf diese Informationen erfolgt zumeist über definierte Schnittstellen (*Application Programming Interface*, API) oder durch die Extraktion von Informationen aus dem HTML Quellcode einer Webseite (*webscraping*).

Die Nutzung von Online-Datenquellen bietet Forschern aus den Geisteswissenschaften neue Möglichkeiten und Blickwinkel für ihre Forschung. Historiker können beispielsweise laut [Mor10] mit der Verknüpfung von Online-Datenbanken deren Inhalte für Metaauswertungen nutzen, wie sie mit herkömmlichen Methoden nicht oder nur mit viel Aufwand möglich wären. Mit dem Projekt PCP-on-Web[1] wird in der geschichtswissenschaftlichen Forschung das Ziel verfolgt, verschiedene Datenbanken mit unterschiedlichen Datenmodellen und andere digitale Ressourcen miteinander zu verknüpfen, um bestimmte Forschungsfragen zu beantworten [RB16]. Weitere kultur- und geisteswissenschaftliche Disziplinen verfolgen ähnliche Bestrebungen. Bei dieser

[1] `https://pcp-on-web.htwk-leipzig.de/project/`

Art von Projekten steigen jedoch die Ansprüche und die Herausforderungen an das erforderliche Management der Forschungsdaten. Probleme treten besonders dann auf, sobald die Daten nicht in einer Datenquelle enthalten, sondern über mehrere Datenquellen verteilt sind. Grund dafür ist, dass diese Datenquellen jeweils über eigene Datenmodelle und -strukturen mit einer unterschiedlicheren Granularität oder Struktur verfügen. Das trifft auch auf Daten zu, welche einer gemeinsamen Domäne zugeordnet werden können. Ursachen hierfür sind unter anderem die unterschiedlichen Ziele und Interessen, die den jeweiligen Daten und ihren Datenmodellen zugrundeliegen. [ES13] Der professionelle Umgang mit heterogenen Datenmengen wird damit zu einem wesentlichen Bestandteil der Arbeit in DH-Projekten.

Für Forschende aus den digitalen Geisteswissenschaften wird es mit zunehmender Komplexität schwieriger die Auswirkungen von Datentransformationen und den Einfluss bestimmter Veränderungen der Daten einzuschätzen. Insbesondere dann, wenn nur geringe Kompetenzen im Bereich der elektronischen Datenverarbeitung seitens der Forschenden vorhanden sind. Das kann dazu führen, dass die Aussagen, die über die Daten getroffen werden falsch sind. Größere Datenmengen können nicht mehr händisch geprüft werden, somit steigt die Unsicherheit und das Risiko für falsche Interpretationen der Daten. Um Fehlinterpretation zu vermeiden, erhält die Transparenz bei den Schritten, die Daten zu Forschungsdaten machen, einen hohen Stellenwert.

1.1 Einordnung der Arbeit

Nach [KE17] vollziehen die Geisteswissenschaften „eine Entwicklung zu Arbeitsweisen wie sie bisher typisch für die Natur- und Lebenswissenschaften sind und wecken gleichzeitig vergleichbare Ansprüche bzgl. der Reproduzierbarkeit konkreter Experimente und Analysen (sogenannte ‚Reproducible Science') oder der effizienten Wiederverwendung bestehender Systeme für neue Datenbestände."

Im deutschsprachigen Raum wird versucht über komplexe Forschungsinfrastrukturen in den Geisteswissenschaften wie DARIAH[2] und CLARIN-D[3] Forschenden unter anderem die Möglichkeiten zu geben, Daten aus Online-Quellen einzusammeln, aufzubereiten und zu analysieren. Dafür werden sowohl Online-Services als auch Programme für den lokalen Einsatz zur Verfügung gestellt. Für einzelne geisteswissenschaftliche Disziplinen werden parallel dazu domänenspezifische Infrastrukturen und Methoden

[2] `https://de.dariah.eu`
[3] `https://www.clarin-d.net/de/`

aufgebaut. Als Beispiel hierfür kann das europäische Forschungsnetzwerk für Universitätsgeschichte - Heloise - dienen. Mit dem Heloise Common Research Model (HCRM) soll eine standardisierte Methode in dieser Domäne etabliert werden. Ziel, ist die Bereitstellung einer integrierten Datenbank, um eine förderierte Forschung auf den Datenbanken dieses Netzwerks zu ermöglichen. [RB16] Hierfür definiert das HCRM drei Schichten: Repository Layer, Application Layer und Research Interface Layer. [RB17] Für die Realisierungen werden Linked-Data-Technologien und domänenspezifische Vokabulare eingesetzt.

Die genannten Infrastrukturen zeigen, dass es einen Bedarf für die Unterstützung bei der Verlinkung, Verarbeitung und Bereitstellung von Daten aus dem geisteswissenschaftlichen Bereich gibt. Im Rahmen der Arbeit erfolgt die Anwendung, der aus dem Betrieb von Data-Warehouses bekannten Vorgehensweisen auf Bereiche der geisteswissenschaftlichen Forschung. Die notwendigen Verarbeitungsschritte, um Daten aus heterogenen Datenquellen in ein zentrales Speichersystem zu überführen, lassen sich in einen klassischen Extract, Transform, Load (ETL) Prozess einordnen. [Bau13] Diese Arbeit konzentriert sich auf die Transformationsphase mit Datenqualitätsprüfungen. Um den gesamten Prozess zu unterstützen wird der Einsatz von Workflow Management Systemen anhand einer prototypischen Umsetzung vorgestellt. Die Transformationsphase wird zusätzlich durch Linked-Data-Technologien unterstützt. Neben etablierten Vorgehensweisen und Programmen werden auch neue Ansätze implementiert. Der 2017 als Empfehlung des World Wide Web Consortiums (W3C) verabschiedete Standard zur Beschreibung von Anwendungsprofilen, SHACL, wird sowohl zur Analyse der eingehenden Schemata, als auch zur Prüfung der RDF Daten als Resultat des Transformationsprozesses eingesetzt. Zur Exploration der resultierenden Daten wird eine Webpräsentation mit Hilfe des Template-basierten RDF Visualisierungswerkzeugs Jekyll-RDF erstellt.

1.2 Motivation

Die Situation im Forschungsprojekt diggr (Databased Infrastructure on Global Game Culture Research; vgl. Abschn. 3.1 auf S. 20) ist ähnlich der in der Einführung skizzierten Problematik. Es müssen Daten aus verschiedenen Online-Datenquellen in einen zentralen Speicherort überführt werden, damit bestimmte Forschungsfragen beantwortet werden können. Als Datenquellen dienen überwiegend Plattformen und Datenbanken in denen Informationen über Videospiele gesammelt und bereitgestellt werden. Die Informationen werden dabei kollaborativ von Fans erstellt, die Webseitenbetreiber stellen dafür die Infrastruktur zur Verfügung. Unterschiedliche (kommerziell

und nicht-kommerziell) Interessen und inhaltliche Schwerpunkte der Webseiten führen dazu, dass heterogene Daten und Datenschemata im Projekt diggr genutzt werden. Die im Projekt bevorzugten Linked-Data-Technologien bieten das Potential diese heterogenen Datenquellen, unter Erhaltung der Struktur, in ein gemeinsames Datenformat zu integrieren. Jedoch steigen mit zunehmender Anzahl der Quellen und Daten, die Komplexität und die Herausforderungen an die Datenintegration. Hinzu kommt, dass die Daten ohne Informationsverlust und nur mit geringer Datenmanipulation integriert werden sollen, da sie sich auf die spätere Datenanalyse auswirken. Um diesen Ansprüchen an die Integration gerecht zu werden, müssen geeignete Lösungen entwickelt werden. Daher wird angestrebt, mit den bisher gesammelten Erfahrungen den Workflow zu optimieren. Dazu zählt unter anderem, die Möglichkeit den Forschenden stärker in den Prozess einbeziehen zu können. Dafür muss zuvor der gesamte Arbeitsprozess unter anderem in Hinblick auf Transparenz und Nachvollziehbarkeit ausgebaut werden. Darüber hinaus wird eine weitestgehende Automatisierung des Workflows angestrebt, um die Reproduzierbarkeit sicherzustellen.

1.3 Ziele

Das Ziel dieser Arbeit ist, den bereits bestehenden Workflow zur Datenintegration im Projekt diggr in Hinblick auf Transparenz, Nachvollziehbarkeit, Wiederholbarkeit und Automatisierung auszubauen. Damit die Prüfung von Daten transparenter wird, soll ein für Menschen leicht verständliches Format zur Beschreibung und Formulierung von Bedingungen, für Daten eingesetzt werden. Um die Nachvollziehbarkeit sicherzustellen, soll der Workflow in einzelne, logische Verarbeitungsschritte zerlegt werden, die sowohl separat, als auch als Verarbeitungskette durchgeführt werden können. Für die Automatisierung und Wiederholbarkeit soll ein Workflow Management System die Verarbeitungsschritte koordinieren und in Abhängigkeit zueinander setzen. Zusätzlich sollen bestimmte Prozesse, wie beispielsweise die Erzeugung der Datenbeschreibung, automatisiert stattfinden.

Um diese Ziele zu erreichen muss eine Weiterentwicklung des Workflows stattfinden, der die wichtigsten Verarbeitungsschritte enthält und durch zusätzliche Schritte erweitert wird. Das Projekt diggr bietet dabei den Rahmen für die prototypische Entwicklung. Durch ein generisches Konzept soll sich der Workflow auch auf ähnliche Szenarien anwenden lassen.

1.4 Aufbau der Arbeit

In Kapitel 2 werden einleitend Grundlagen und Fachbegriffe aus den Bereichen von Linked Data, deren Prüfung sowie verschiedene Datenaustauschformate erläutert, die zum Verständnis der Arbeit notwendig sind. Daran schließt sich Kapitel 3, mit der Vorstellung des Projekts diggr und dem Ist-Zustand des Datenintegrationsworkflows, an. Darauf folgt in Kapitel 4, die Beschreibung der Anwendungsfälle und die daraus resultierenden Anforderungen. Auf Basis der Anforderungen wird in Kapitel 5 ein Entwurf erarbeitet, um den Workflow prototypisch umzusetzen. Kapitel 6 erläutert die Implementierung, gefolgt von Tests bei denen der Workflow mit verschiedenen Datensätzen aus unterschiedlich strukturierten Datenquellen getestet wird. Die Beschreibung der Tests und die daraus resultierenden Ergebnisse werden in Kapitel 7 beschrieben und diskutiert. Abschließend werden in Kapitel 8 die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick auf mögliche Ergänzungen und Erweiterungen des Workflows gegeben.

2 Grundlagen

Die Arbeit ist Teil einer größeren Linked-Data-basierten Forschungsinfrastruktur. Daher ist es nötig verschiedene Linked-Data-Konzepte zu erläutern. Zu diesem Zweck werden im Folgenden die in der Arbeit genutzten Begriffe und Konzepte vorgestellt. Neben Datenformaten (Abschn. 2.2 auf S. 12 und Abschn. 2.4 auf S. 15), die im Workflow genutzt werden, werden Ansätze zur Validierung von Daten (Abschn. 2.5 auf S. 17), die als Linked Data vorliegen, vorgestellt.

2.1 Extensible Markup Language

Die Extensible Markup Language (XML) ist eine Auszeichnungssprache für text-basierte Daten. Sie dient dem Austausch semistrukturierter Informationen. Die Datenobjekte werden in der W3C Recommendation [MYB+08] als XML Dokumente bezeichnet, wenn sie wohlgeformt sind, d. h. wenn:

- am Beginn eine XML-Deklaration stattfindet,
- es mindestens ein Datenelement gibt und
- es ein Datenelement als äußerstes Element (Dokument-Element), das alle anderen Datenelemente enthält, gibt.

Mit XML gibt es verschiedene Wege Daten zu repräsentieren. XML bietet lediglich eine Syntax, das darunterliegende Datenmodell ist ein Baum. Beispielsweise können Informationen zu einer Person, wie in Listing 2.1 dargestellt, repräsentiert werden.

```
<?xml version="1.0"?>
<person>
     <name>Bob</name>
     <age>29</age>
     <knows>Alice</knows>
</person>
```

Listing 2.1: Beispiel für ein wohlgeformtes XML Dokument

Wenn XML Dokumente wohlgeformt sind bedeutet das nicht, dass sie auch gültig sind. Das bedeutet, dass ein XML Dokument bestimmten Regeln folgt, die durch eine Grammatik vorgegeben ist. Um zu verstehen, worum es sich bei einem XML Dokument handelt und wann es gültig ist, wird eine Document Type Definition (DTD) oder ein XML Schema benötigt. Sie definieren die Struktur und die Regeln für gültige XML Dokumente. Damit lassen sich XML Dokumente nach definierten Regeln prüfen. In Listing 2.2 wird ein XML Schema für Listing 2.1 auf S. 11 definiert.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com" elementFormDefault="qualified">

<xs:element name="person">
  <xs:complexType>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
    <xs:element name="knows" type="xs:string"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Listing 2.2: Beispiel für ein XML Schema

XML ist die Grundlage für weitere wichtige Standards im Internet, wie zum Beispiel Webservices oder die Sprachen des semantischen Webs.

2.2 Resource Description Framework

Das Resource Description Framework (RDF) ist ein Standard für den Datenaustausch und die Datenrepräsentation im Internet. Es bietet Funktionen, die das Zusammenführen von Daten erleichtern, auch wenn die zugrunde liegenden Schemata unterschiedlich sind. [RS14] RDF basiert auf der Verwendung von Uniform Resource Identifier (URI), um Ressourcen eindeutig zu identifizieren und Beziehungen zwischen ihnen herzustellen. Bei Ressourcen kann es sich um Webdokumente, Abschnitte von Dokumenten oder irgendein zu identifizierendes Datum handeln. Dabei entstehen sogenannte Triple, welche Aussagen (engl. *statements*) über bestimmte Ressourcen treffen und sie damit näher beschreiben. Diese Aussagen können als Sätze angesehen werden,

bei denen auf ein Subjekt, ein Prädikat und darauf ein Objekt folgt. In Abb. 2.1 auf S. 13 werden Aussagen zu einer Ressource exemplarisch dargestellt.

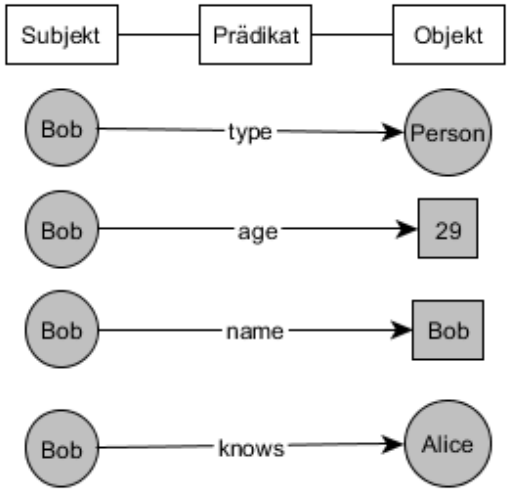

Abbildung 2.1: Die Ressource Bob ist vom Typ *Person* und besitzt die Eigenschaften *age*, *name* und eine Beziehung *knows* mit der Ressource Alice

Für die durch Triple dargestellten Datensätze existieren unterschiedliche Repräsentationsformen, die sich vorrangig in ihrer Lesbarkeit für den Menschen und in der Programmunterstützung unterscheiden. Weit verbreitet sind die Formate RDF/XML, Turtle und JSON-LD. (vgl. [Hit08, S. 40 ff.]) Werden, ausgehend von einem Objekt, weitere Aussagen in dieser Form getroffen, entstehen Netzwerke von mehreren Datensätzen, die als Graph bezeichnet werden. Diese Graphen repräsentieren eine Menge an Dingen und ihre Beziehungen untereinander.

Das RDF Modell macht keine Aussagen darüber, wofür die verwendeten URIs stehen, daher wird RDF in der Regel in Kombination mit Vokabularen verwendet. [RS14] Vokabulare, wie Taxonomien oder Ontologien, beschreiben die Bedeutung der URIs näher und stellen Informationen über die Beziehung zu anderen Ressourcen im Vokabular bereit. Dabei lassen sich zwei Typen unterscheiden: Klassen (engl. *classes*) und Eigenschaften (engl. *properties*). Klassen identifizieren Dinge. Dinge besitzen Eigenschaften. Mittels Vererbung lassen sich Hierarchien von Klassen zu anderen Klassen (engl. *subclasses*) ausdrücken. Das Gleiche gilt für Untereigenschaften (engl. *subproperties*), die eine Vererbung der Bedeutung von Eigenschaften ermöglichen.

Innerhalb einer Aussage können Objekte mit URIs auf andere Ressourcen verweisen, sie können jedoch auch ein Datenobjekt, wie eine Nummer, ein Datum, ein String oder einen benutzerdefinierten Datentyp beinhalten. Diese Datenobjekte werden als Literale bezeichnet und stellen Werte dar, über die keine weiteren Aussagen getroffen werden. Eine konkrete Ausprägung eines Objekts, über welches eine Aussage getroffen wird, ist eine Instanz. Sie stellt dabei das vorhandene Wissen dar.

In sogenannten Namensräumen (engl. *namespace*) wird das Vokabular der Klassen und Eigenschaften zusammengefasst. Dabei handelt es sich um den gleichbleibenden Teil einer URI. Dieser Teil wird zur besseren Lesbarkeit häufig mit einem Präfix (engl. *prefix*) abgekürzt. Der folgende Code repräsentiert die Darstellung aus Abb. 2.1 auf S. 13 in einen RDF Graphen in Turtle. Die ersten drei Zeilen sind Präfix Deklarationen, gefolgt von RDF Triplen.

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22—rdf—syntax—ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:bob rdf:type foaf:Person ;
       foaf:name "Bob" ;
       foaf:knows :alice ;
       foaf:age 29 .

:alice rdf:type foaf:Person ;
       foaf:name "Alice" ;
       foaf:age 31 .
```

Listing 2.3: Beispiel für einen RDF Graphen in Turtle Serialisierung

2.3 Linked Data und RDF

Linked Data bezeichnet Daten, die in strukturierter Form veröffentlicht werden und mit anderen Daten verlinkt sind bzw. verlinkt werden können. RDF bildet neben anderen Standards die Basis von Linked Data. Tim Berners-Lee schlägt Linked Data als best-practice für die Veröffentlichung von Daten im Internet vor und basiert auf vier Prinzipien. [Tim06]

- Verwende URIs um Dinge zu bezeichnen.
- Verwende HTTP-URIs, so dass sich die Bezeichnungen nachschlagen lassen.

- Stelle Informationen bereit, wenn jemand eine URI nachschlägt (mittels der Standards RDF und SPARQL).
- Nutze Links auf andere URIs, über die weitere Objekte entdeckt werden können.

Werden diese Regeln eingehalten so ergibt sich die Möglichkeit Daten miteinander in Beziehung zu setzen und diese auch miteinander zu verbinden (engl. *linking*). Über die Beschreibung der Beziehung mit Hilfe von Vokabularen lassen sich anschließend semantische Zusammenhänge durch Mensch und Maschine erkennen.

2.4 JSON und JSON-Schema

Bei der JavaScript Object Notation (JSON) handelt es sich um ein text-basiertes, semistrukturiertes Datenformat. Es verwendet für seine Struktur Schlüssel-Werte-Paare und geordnete Listen von Werten. Die Daten können in diesen Strukturen beliebig verschachtelt werden. Das dabei entstehende JSON Datenmodell kann als Baum dargestellt werden. Listing 2.4 zeigt exemplarisch Daten im JSON Format. Das Beispiel enthält ein JSON Objekt mit drei Schlüsseln: `name`, `age` und `knows`. Der Wert von `name` ist ein String, der von `age` ein Integer und der Wert von `knows` ist ein Array mit einem Objekt.

```
{
  "name": "Bob",
  "age": 29,
  "knows": [
    {
      "name": "Alice",
      "age": 31
    }
  ]
}
```

Listing 2.4: JSON-Beispiel

JSON Schema erlaubt die Annotation und Prüfung von JSON Dokumenten ähnlich wie XML Schema für XML. Mit Hilfe des Vokabulars von JSON Schema können Daten beschrieben und auf dieser Grundlage validiert werden. Es enthält die folgenden vordefinierten Datentypen: null, Boolean, object, array, number and string. Die Spezi-

fikation befindet sich zum Zeitpunkt dieser Arbeit in der draft Version 07[1]. Listing 2.5 auf S. 16 stellt die Definition eines JSON Schemas zum Objekt Person dar.

```
{
    "title": "Person",
    "type": "object",
    "properties": {
        "name": {
            "type": "string"
        }
        "age": {
            "description": "Age in years",
            "type": "integer",
            "minimum": 0
        }
    },
    "required": ["name"]
}
```

Listing 2.5: JSON-Schema

JSON enthält keine Informationen über die Semantik der enthaltenden Daten. Diese Informationen können nur außerhalb der Daten als Dokumentation hinterlegt werden. Mit JSON-LD (JSON for Linked Data) ist jedoch eine JSON-basierte Repräsentation von RDF möglich. Hierfür stellt JSON-LD Schlüsselwörter zur Verfügung, mit denen der Kontext (`@context`) und eindeutige Bezeichner (`@id`) ausgezeichnet werden können. JSON-LD ist vollständig kompatibel mit JSON.

Das Turtle Beispiel aus Listing 2.3 auf S. 14 kann in JSON-LD folgendermaßen repräsentiert und erweitert werden:

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "age": "http://xmlns.com/foaf/0.1/age",
    "knows": "http://xmlns.com/foaf/0.1/knows"
  },
   "@id": "http://example.org/bob",
   "@type": "http://xmlns.com/foaf/0.1/Person",
   "name": "Bob",
   "age": 29,
```

[1] `http://json-schema.org/specification-links.html#draft-7`

```
   "knows": [
    {
      "@id": "http://example.org/alice",
      "@type": "http://xmlns.com/foaf/0.1/Person",
      "name": "Alice",
      "age": 31
    }
  ]
}
```

Listing 2.6: JSON-LD

2.5 Validierung von RDF

Um eine automatisierte Prüfung von Daten stattfinden zu lassen, ist es notwendig die Bedingungen, unter denen Daten als valide definiert werden, in eine maschinenlesbare Form zu bringen. In welcher Art und Weise diese Bedingungen formuliert und angewendet werden, ist abhängig vom eingesetzten Speicher- und Datenformat. Die meisten, weit verbreiteten strukturierten Datenformate verfügen über zusätzliche strukturierte Sprachen, mit denen Daten beschrieben werden und eine Form der Datenkonsistenz erzwungen wird. Beispielsweise können in relationalen Datenbanken über DDL (Data Definition Language) Daten beschrieben werden. Für Daten, die in XML vorliegen, kann DTD, XML Schema oder RelaxNG[2] eingesetzt werden [LGPSB]. Für JSON hat sich das JSON-Schema für diese Zwecke etabliert.

Bei RDF kann die Art bzw. die Form eines Knotens, die Anzahl möglicher Verbindungen (eingehend/ausgehend) und die möglichen Werte, die mit den Verbindungen assoziiert sind, beschrieben werden. RDF Validation ist ein Zwischenschritt bei der Verarbeitung von Daten, der prüfen kann, ob Instanzen einem bestimmten Schema entsprechen.

Für RDF wurden verschiedene Ansätze für die Validation entwickelt, es hat sich aber bisher kein Standard etabliert. Die Query-based Validation verwendet Abfragesprachen um Einschränkungen (engl. *constraints*) auszudrücken. Insbesondere mit der RDF Abfragesprache SPARQL[3] lassen sich auf diese Weise viele Prüfbedingungen abde-

[2] http://www.relaxng.org

[3] http://www.w3.org/TR/sparql11-overview/

cken. SPARQL Queries können jedoch schnell sehr komplex werden und sind damit unbrauchbar für Laien. [LPB18, S. 46]

Mit Shape Expressions (ShEx) und Shapes Constraint Language (SHACL) stehen zwei Shape[4]-basierte Ansätze zur Verfügung mit denen RDF Graphen beschrieben und validiert werden können. Shapes können zur internen sowie zur externen Dokumentation verwendet werden, da nicht nur das Datenschema beschrieben wird, sondern auch die Bedingungen und Restriktionen nachvollzogen werden können.

2.5.1 ShEx

ShEx ist eine Schemasprache um Strukturen von RDF Graphen zu beschreiben. Sie wurde 2013 entwickelt, um eine maschinenlesbare Syntax für OSLC Resource Shapes[5], einem high-level Vokabular zur Beschreibung häufig vorkommender Einschränkungen für RDF Graphen bereitzustellen. Nach [LPB18] definiert ShEx ein Set von Graphen, um

- Datenstrukturen, Prozesse und Eingaben zu kommunizieren,
- Daten zu erzeugen oder zu validieren,
- die Erzeugung und Navigation von Benutzeroberflächen zu steuern.

Die ShEx Compact Syntax (ShExC) wurde so gestaltet, dass sie durch Menschen gelesen und bearbeitet werden kann. Sie folgt einigen Konventionen, die auch in Turtle oder SPARQL enthalten sind. Dazu zählen unter anderem die Art der Angabe von Präfixen und die Nutzung von `a` als Abkürzung für `rdf:type`. [LPB18, S. 59]

2.5.2 SHACL

SHACL wurde von der W3C RDF Data Shapes Working Group 2014 initiiert mit dem Ziel, eine Sprache zu entwickeln mit der strukturelle Bedingungen auf RDF Graphen

[4] „[..] a shape is a collection of constraints to be applied to some node in an RDF graph“[LPB18]
Der Begriff *Shape* lässt sich in diesem Kontext mit dem deutschen Begriff *Profil* übersetzen. Im deutschen wird auch von *Anwendungsprofil* gesprochen. [Eve15] Für diese Arbeit wird der englische Ausdruck Shape verwendet.

[5] `https://www.w3.org/Submission/shapes/`

angewendet werden können. Im Oktober 2015 veröffentlichte das W3C die First Public Working Draft Version von SHACL, welche im Juni 2017 den Status der *Proposed Recommendation* und am 20.07.2017 die Empfehlung durch das W3C erhalten hat.[6] Die Sprache wurde durch SPIN[7], eine auf SPARQL-basierte Regel- und Beschreibungssprache und einigen Teilen von OSLC Resource Shapes und ShEx beeinflusst. Ursprünglich sollte SHACL die bisherigen Ansätze und Bestrebungen vereinigen. Dieses Ziel wurde jedoch nicht erreicht, so dass beide Technologien ShEx und SHACL zur Verfügung stehen und derzeit getrennt weiterentwickelt werden.

SHACL verfolgt das Ziel, RDF Graphen gegen eine Sammlung von Bedingungen zu validieren. Hierfür stellt SHACL ein Vokabular zur Verfügung, mit dessen Hilfe Shapes definiert werden können. SHACL besteht aus zwei Teilen.

1. SHACL Core, ein RDF Vokabular zur allgemeinen Beschreibung von Shapes und Bedingungen.
2. Eine Erweiterung durch SPARQL.

Die Definitionen in SHACL stellen nicht nur in Aussicht, RDF Daten zu validieren, sondern auch, dass einmal erstellte Shapes für weitere Zwecke nachgenutzt werden können. Die enthaltenen Beschreibungen und Bedingungen könnten zur Erstellung von Formularen und strukturierten Ausgaben von Daten auf einer Webseite dienen. [KK17] Weitere Anwendungsszenarien und Anforderungen sind bei [SC17] beschrieben.

[6] `https://www.w3.org/blog/news/archives/6421`
[7] `https://www.w3.org/Submission/spin-overview/`

3 Ausgangssituation

Die Arbeit setzt sich mit einem bereits bestehenden Workflow zur Datenintegration innerhalb eines DH-Projektes auseinander. In den folgenden Abschnitten wird das Forschungsprojekt diggr und der bisher praktizierte Workflow zur Datenintegration beschrieben.

3.1 Forschungsprojekt diggr

Das DFG-geförderte Forschungsprojekt diggr[1] ist ein gemeinschaftliches Projekt der Universitätsbibliothek und dem Institut für Japanologie der Universität Leipzig. Den Forschungsschwerpunkt bildet die Erforschung japanischer Videospiele im Kontext globaler Videospielkultur. Im Zeitraum von 2017 bis Ende 2019 arbeitet ein interdisziplinäres, sechsköpfiges Team an Methoden und Prozesse zur datenbasierten Forschung an kulturwissenschaftlichen Fragestellungen. Das Management von Forschungsdaten an Bibliotheken sowie deren erfolgreiche Nutzung durch Forschende stehen im Mittelpunkt des Projekts. Hierfür wird auf den Aufbau einer datengetriebenen Forschungsinfrastruktur, die u.a. Linked-Open-Data-Technologien verwendet, gesetzt.

Ein Projektziel stellt die Bereitstellung semantischer Zusammenhänge innerhalb der Daten und zwischen den Datenquellen dar. Um dieses Ziel zu erreichen werden im diggr Projekt Linked-Data-Technologien eingesetzt. Das Wissen aus den heterogenen Modellen der Datenquellen soll in eine homogene Repräsentation überführt werden. Die daraus entstehende Wissensbasis soll den Forschenden zur Beantwortung ihrer Forschungsfragen dienen. [HFM17] Den Forschenden soll die Möglichkeit gegeben werden, die Datenquellen sowohl inhaltlich als auch strukturell miteinander zu vergleichen. Die Datenquellen sollen in Kontext zueinander gesetzt werden können, um anschließend eine gemeinsame Sicht auf die Daten bereitstellen zu können. Das soll die

[1] http://gepris.dfg.de/gepris/projekt/316697723
https://diggr.link/

Forschenden dazu befähigen, über die Gesamtheit der Daten Fragestellungen zu beantworten. Die Feinheiten und Widersprüche aus einzelnen Datenquellen sollen wo möglich erhalten bleiben, da diese ebenso Erkenntnisse zur Beantwortung bestimmter Forschungsfragen liefern können.

3.2 Workflow zur Datenintegration

Der Workflow zur Datenintegration stellt einen Teil innerhalb des gesamten Forschungsprozesses zur Beantwortung einer Forschungsfrage dar. Schwerpunkte im Projekt diggr bilden Fragen in Hinblick auf Globalisierungsprozesse in der Spieleentwicklung und der Verbreitung japanischer Videospiele. Dafür werden Metadaten wie Veröffentlichungstermine und -orte, an der Produktion beteiligte Firmen und Personen sowie Informationen zur jeweiligen Plattform, auf der das Spiel erschienen ist, benötigt. Es existiert keine Datenbank mit vollständigen und gesicherten Informationen zur Veröffentlichung aller Videospiele, die bisher weltweit erschienen sind. Daher ist es notwendig, über mehrere Datenquellen die Informationen zu beziehen und sie in Kontext zueinander zu setzen. Im folgenden Abschnitt wird der gesamte Workflow skizziert, in dem sich die Datenintegration einordnen lässt. Einen Überblick über die Ausgangssituation gibt Abbildung 3.1.

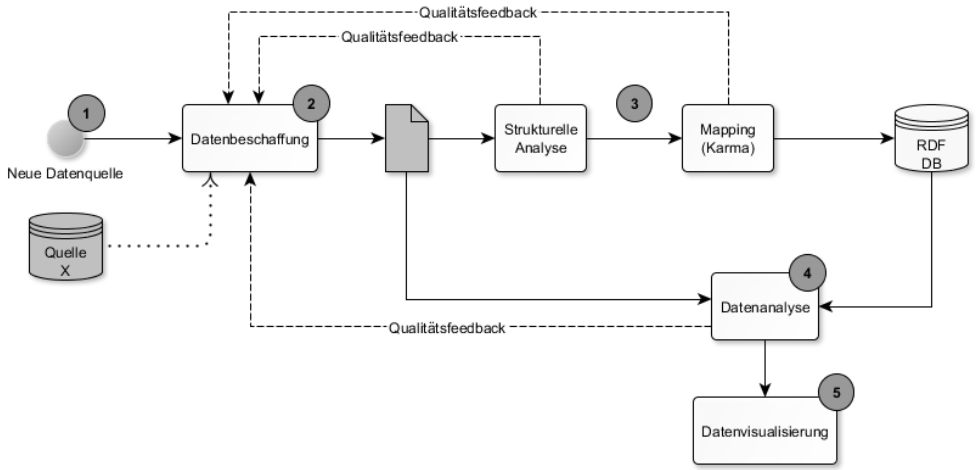

Abbildung 3.1: Datenintengrationsworkflow im Projekt diggr (Ist-Zustand)

Im ersten Schritt werden potentiell geeignete *Datenquellen identifiziert* (1). Anschließend werden über verschiedene Wege (webharvesting über bestimmte Schnittstellen

oder per webscraping) die *Daten beschafft* (2) und in die interne Infrastruktur geladen, wo sie als eine oder mehrere Dateien vorliegen. Die nachfolgende *Datenaufbereitung* (3) schließt den Datenintegrationsprozess mit ein. Daneben können, je nach Forschungsanliegen, auch Schritte durchgeführt werden, die der Filterung oder Ergänzung der Daten dienen. Nachdem die Daten für eine Forschungsfrage aufbereitet wurden, folgen abschließend die *Datenanalyse* (4) und *Datenvisualisierung* (5) durch den Forschenden. Die Datenaufbereitung stellt damit einen Schritt dar, von dem die weitere Auswertung und Interpretation der Daten durch den Forschenden abhängig ist.

Zusammenfassend lässt sich der Prozess der Datenintegration als klassischen ETL-Prozess betrachten, der zyklisch wiederholt wird. Der Ablauf des Workflows zur Datenintegration ist bisher durch manuelle und wiederkehrende Schritte gekennzeichnet, die sich hauptsächlich auf zwei Aspekte beziehen:

1. Der Überführung in ein übergreifendes Format und
2. der Einschätzung hinsichtlich der Datenqualität der genutzten Quellen.

Für jede Datenquelle wird ein Skript erstellt, welches die Menge und Verteilung der Datenfelder auswertet. Es folgen inhaltliche Stichproben für einzelne Datenfelder. In Rücksprache mit den Forschenden wird entschieden, ob die Datenquelle für weitere Auswertungen verwendet werden soll. Bestimmte Qualitätsansprüche, können von Quelle zu Quelle unterschiedlich sein. In Fällen, in denen die Daten einer Quelle nicht den Erwartungen des Forschenden, beispielsweise hinsichtlich Umfang oder Qualität, entsprechen, kann ein Abbruch des gesamten Prozesses stattfinden. Eine Rückmeldung zur Qualität (Qualitätsfeedback) geschieht an mehreren Prozessschritten durch verschiedene Akteure. Dieses Feedback führt zu einer erneuten Prozessierung der gesamten Daten. Insbesondere am Ende des Prozesses, an dem die Forschenden Abfragen zur Beantwortung ihrer Forschungsfragen stellen, sollten Einschränkungen oder Lücken innerhalb der Daten bereits bekannt sein.

Für die Auswertung stehen mehrere Optionen zur Verfügung. Zum einen können die Daten einer Quelle unabhängig von anderen Daten ausgewertet werden. Hierfür ist die Integration in eine zentrale Wissensbasis nicht zwingend notwendig. Zum anderen können die Daten im Kontext zu weiteren Daten betrachtet werden. Ist das der Fall, müssen die Daten für die Integration in den zentralen Speicherort vorbereitet werden. Dafür wird manuell mit Hilfe der Datenintegrationssoftware Karma[2], ein Mapping, des Quellschemas auf ein homogenes, projektspezifisches Schema durchgeführt. Dies geschieht zur Zeit für einzelne Forschungsfragen ad-hoc und umfasst nicht alle Daten

[2] `http://usc-isi-i2.github.io/karma/`

die zur Verfügung stehen. Diese Daten werden abschließend in die Wissensbasis (RDF Datenbank) importiert.

Die Abbildung der unterschiedlichen Quellschemata auf das projektspezifische Schema kann zu Schwierigkeiten bei der Datenanalyse führen. Bei der Abbildung eines Ausgangsschemas auf ein Zielschema findet immer eine Interpretation der vorliegenden Strukturelemente statt. Diese wird erschwert, wenn keine Dokumentation des Ausgangsschemas vorliegt. Betrachtet man das Schema als semantische Auszeichnung, so kann von einem Ontology Matching Problem (vgl. [ES13, S. 41 ff.]) gesprochen werden. Beim Ontology Matching wird versucht Relationen zwischen Entitäten zu identifizieren und zu beschreiben, die gleich, äquivalent oder auch unscharf sein können. Dabei können eine, mehrere oder auch keine Relation zustande kommen. Im Bereich des Data Warehouse wird dabei von Schemakonflikten gesprochen, die nach [Bau13] „jegliche Inkonsistenzen, die bei unterschiedlich modellierten Welten auf Schemaebene wie Datentyp-, Struktur-, Namenskonflikte oder fehlende Attribute auftreten können.“

4 Anforderungen

Das Ziel dieser Arbeit ist, einen bereits bestehenden Workflow zur Datenintegration in Hinblick auf Transparenz, Nachvollziehbarkeit, Wiederholbarkeit und Automatisierung zu optimieren. Während der Arbeit an dem Projekt diggr haben sich bestimmte Funktionalitäten herausgebildet, die auch weiterhin zur Verfügung stehen müssen. Sie bilden bereits die Grundlage für die Beantwortung bestimmter Forschungsfragen. Diese und weitere noch nicht berücksichtigte Funktionalitäten sollen durch eine Überarbeitung des Workflows zur Verfügung gestellt werden. Um die Anforderungen zu erfassen und zu formulieren wurden in Feedbackgesprächen Wünsche und Erwartungen der beteiligten Personen erfasst und zu Anwendungsfällen zusammengefasst. Nach der Beschreibung der beteiligten Rollen werden Anwendungsfälle (AF) beschrieben, aus denen die Benutzeranforderungen (BA; funktionale Anforderungen) extrahiert werden. Anschließend werden grundlegende Anforderungen (NF; nicht-funktionale Anforderungen) an das System definiert.

4.1 Rollen

Der Datenintegrationsworkflow im Projekt diggr ist ein Prozess, der in enger Abstimmung zwischen drei bestimmten Personen bzw. Rollen realisiert wird. Folgende Rollen sind beteiligt:

Forscherin und Forscher (F) formuliert Forschungsfragen, die durch eine oder mehrere Datenquellen beantwortet werden sollen. Sie/Er formuliert Anforderungen und Qualitätskriterien an die Daten. F formuliert seine/ihre Erwartungen und Wünsche gegenüber MA Curate und MA Load.

MitarbeiterIn Curate (MA Curate) bereitet die vorliegenden Daten für die weitere Integration vor. Er/Sie ist am vorgelagerten Prozess des Datenimports in die Projektinfrastruktur beteiligt.

MitarbeiterIn Load (MA Load) ist verantwortlich für die Integration der Daten in ein zentrales Speichersystem und das projektinterne Datenmodell.

4.2 Anwendungsfälle

Die folgenden Anwendungsfälle beschreiben die Wünsche und Erwartungen an den Workflow zur Datenintegration aus Sicht der einzelnen Rollen. Aus jedem Anwendungsfall werden die Arbeitsschritte extrahiert. Von den Arbeitsschritten und Funktionen wird auf die entsprechende Benutzeranforderung verwiesen. Die Schritte für den Datenimport werden hierbei nicht betrachtet, da sie nicht Bestandteil dieser Arbeit sind.

Das Anwendungsfalldiagramm (Abb. 4.1) gibt einen Überblick über das Zusammenspiel der Anwendungsfälle und der beteiligten Rollen bzw. Akteuren.

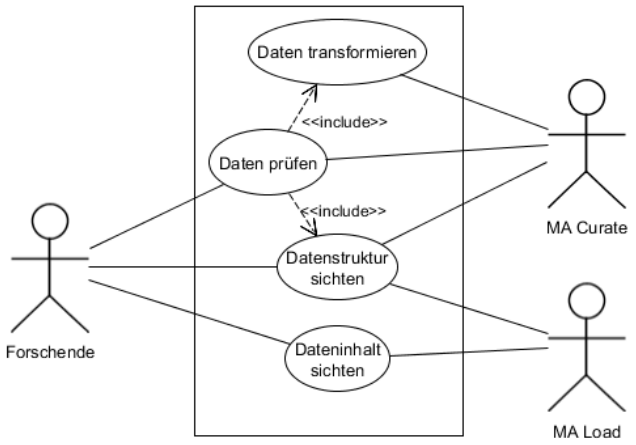

Abbildung 4.1: Darstellung der Anwendungsfälle und die beteiligten Akteure

4.2.1 Daten transformieren

/AF1/

Als Ergebnis des vorgelagerten Extraktionsprozess für die Datenbeschaffung (vgl. Abschn. 3.2 auf S. 21) stehen Daten aus unterschiedlichen Quellen in der Projektinfrastruktur zur Verfügung. Bevor die Daten in die zentrale Wissensbasis geladen werden können, müssen sie in ein einheitliches internes Format gebracht werden. Die Daten liegen an dieser Stelle bereits im einheitlichen JSON Format vor, aber noch nicht im RDF Format.

Um die Daten später als Forschungsdaten nutzen zu können, ist es wichtig, dass zu jedem Zeitpunkt bekannt ist woher die Daten stammen.

Ziel: Daten zur weiteren Verarbeitung vorbereiten

Vorbedingung: Quelldaten liegen vor

Erfolg: Daten sind für den zentralen Speicherort vorbereitet, Informationen über den Verarbeitungsschritt sind in den Daten enthalten

Fehlschlag : Daten liegen nicht im RDF Format vor

Akteur: MA Curate

Auslösendes Ereignis: Es wurden Daten auf Wunsch des Forschenden in die Projektinfrastruktur importiert

Beschreibung:

1. MA Curate gibt an aus welcher Quelle die Daten stammen (BA 6)
2. Transformation der Quelldaten in RDF Format (BA 1)

Erweiterung:

1a Datenquelle aktualisieren

Zugehörige Benutzeranforderungen

- Anforderung „Datentransformation“ in BA 1 auf S. 29
- Anforderung „Nachvollziehbarkeit“ in BA 6 auf S. 32

4.2.2 Datenstruktur sichten

/AF2/

Damit Daten aus externen Quellen in ein zentrales Speichersystem integriert werden können, müssen Kenntnisse darüber erworben werden, wie die Daten strukturiert sind; mit anderen Worten: eine syntaktische Analyse muss stattfinden. Diese soll erste Anhaltspunkte bieten, um die Quelle und die Schritte für die weitere Verarbeitung bzw. ihrer Eignung für die Fragestellung des Forschenden besser einschätzen zu können. Die Kenntnisse über das Vokabular, die verwendeten Datentypen und eventuell vorhandene Einschränkungen sind bei der Beurteilung und Einschätzung über die Eignung der Daten wichtig. Da die Daten über Dritte bezogen werden und es teilweise

keine Dokumentation über das verwendete Datenmodell gibt, muss die Struktur der Daten aus den Daten selbst abgeleitet werden.

Ziel: Überblick über das verwendete Vokabular und die Struktur der Datenquellen

Vorbedingung: Quelldaten liegen vor

Erfolg: Informationen über die Datenstruktur und des Vokabulars liegen vor

Fehlschlag : Es liegen keine Informationen über die Datenstruktur vor

Akteur: MA Curate, Forschender, MA Load

Auslösendes Ereignis: Wunsch über Informationen zur Datenstruktur liegt vor

Beschreibung:

1. MA Curate gibt an aus welcher Quelle die Daten stammen (BA 6)
2. Quelldatenstruktur wird extrahiert (BA 2)

Zugehörige Benutzeranforderungen

- Anforderung „Strukturelle Informationen speichern“ in BA 2 auf S. 30
- Anforderung „Nachvollziehbarkeit“ in BA 6 auf S. 32

4.2.3 Daten prüfen

/AF3/

Datenqualitätsaspekte stellen eine wichtigen Aspekt bei DH-Projekten dar. „Die Qualität der Daten bestimmt maßgeblich, inwiefern diese für die Forschung nutzbar und nachnutzbar sind.“[DAR15] Der Forschende soll in die Lage versetzt werden, seine Ansprüche an die Daten formulieren zu können. Gleichermaßen muss er die Ergebnisse einer Qualitätsprüfung interpretieren können. Was die Qualität der Daten bestimmt, ist abhängig vom Verwendungszweck und von der spezifischen Forschungsfrage. Beispiele für typische Qualitätsmängel bei heterogenen Quelldaten sind laut [Bau13, S. 47]:

- inkorrekte Daten, verursacht durch Eingabefehler
- logisch widersprüchliche Daten
- unvollständige, ungenaue bzw. zu grobe Daten
- uneinheitlich repräsentierte Daten

Ziel: Daten, die bestimmten Qualitätskriterien nicht entsprechen sind identifiziert

Vorbedingung: Daten liegen im RDF Format vor, Qualitätskriterien sind bekannt, Datenstruktur ist bekannt

Erfolg: Prüfbericht ist erstellt

Fehlschlag : Es liegt kein Prüfbericht vor

Akteur: MA Curate, Forschender

Auslösendes Ereignis: Prüfwunsch besteht

Beschreibung:

1. Forscher formuliert Qualitätskriterien für Datenquelle (BA 3)
2. Gesamtes Datenset prüfen (BA 3)
3. Prüfbericht erstellen (BA 4)

Erweiterung:

1a Forscher formuliert Qualitätskriterien, die informierenden Charakter haben sollen (BA 3)

2a Einzelne Datenobjekte werden geprüft (BA 3)

4a Prüfbericht informiert über Fehler, die informierenden Charakter haben (BA 4)

4b Prüfbericht informiert über schwerwiegende Fehler (BA 4)

Zugehörige Benutzeranforderungen

- Anforderung „Datenvalidierung“ in BA 3 auf S. 30
- Anforderung „Prüfberichte erzeugen“ in BA 4 auf S. 31

4.2.4 Dateninhalt sichten

/AF4/

Bei der Arbeit in einem mutli-disziplinär besetzten DH-Projekt, wie dem Projekt diggr sollte darauf geachtet werden, dass alle Beteiligten auf dem gleichen Informationsstand sind. Das erleichtert die Kommunikation und beugt Missverständnissen vor. Eine einfache Darstellung abseits technischer Datenaustauschformate ist daher wünschenswert.

Ziel: Daten, liegen in einer allgemeinverständlichen und -zugänglichen Darstellung vor

Vorbedingung: Daten liegen im RDF Format vor, Datenstruktur ist bekannt

Erfolg: Datenansicht ist einfach und selbsterklärend

Fehlschlag : Daten liegen in einer nicht verständlichen Form vor, Daten werden nicht angezeigt

Akteur: MA Load, Forschender

Auslösendes Ereignis: Ansicht der Daten wird gewünscht

Beschreibung:

1. Datenansicht wird erstellt (BA 5)

Zugehörige Benutzeranforderungen

- Anforderung „Dateninhalte anzeigen“ in BA 5 auf S. 31

4.3 Benutzeranforderungen

Aus den Anwendungsfällen ergeben sich Benutzeranforderungen an den Workflow und den einzelnen Bestandteilen. Der Workflow bildet dabei die logische Abfolge bestimmter Funktionen ab und bettet sich in ein technisches System ein. In den folgenden Abschnitten werden die Anforderungen näher beschrieben und Funktionalitäten des Systems abgeleitet.

4.3.1 Datentransformation

/BA1/

Das System muss dem Anwender die Möglichkeit bieten Quelldaten, die im JSON-Format vorliegen, in das RDF Format zu transformieren. Dieser Prozess soll durch den Anwender manuell gestartet werden können. Falls während der Verarbeitung ein Fehler auftritt muss das System eine Fehlermeldung auf dem Bildschirm anzeigen. Das System muss das Ergebnis der Transformation (die Daten) im Dateisystem speichern. Die Quelldaten dürfen vom System nicht verändert werden (engl. *immutable*).

Anwendungsfall 1: „Daten transformieren“ (S. 25)

4.3.2 Strukturelle Informationen speichern

/BA2/

Das System muss die gesamte Struktur (Datenobjekte und Eigenschaften) beliebiger, im JSON Format vorliegender Quelldaten extrahieren können. Dabei muss auch die Verarbeitung komplexer Strukturen, d. h. tiefe Verschachtelung möglich sein. Die extrahierte Struktur soll als Beschreibung der vorliegenden Daten dienen. Es soll möglich sein diese Beschreibungen manuell zu bearbeiten und zu erweitern. Um die Datenvalidierung (vgl. BA3) zu unterstützen, soll die Struktur im RDF Format vorliegen. Die Struktur muss in einer Datei gespeichert werden.

Anwendungsfall 2: „Datenstruktur sichten“ (S. 26)

4.3.3 Datenvalidierung

/BA3/

Die Validierung muss auf den Daten im RDF Format stattfinden. Die Bedingungen unter denen Daten zulässig sind, müssen sowohl maschinenlesbar als auch für Menschen verständlich sein. Dafür soll eine bereits bestehende Sprache zur Beschreibung von RDF Daten genutzt werden. Die Art und Weise wie Bedingungen formuliert werden können, sollte einfach erlernbar sein. Um das zu unterstützen, soll die Beschreibung in einem einheitlichen Format erfolgen. Um die Wartbarkeit und Lesbarkeit der Bedingungen zu unterstützen, sollen Anmerkungen möglich sein.

Es soll die Möglichkeit bestehen, einfache und komplexe Bedingungen zu beschreiben. Einfache Bedingungen zielen darauf ab einzelne Werte zu prüfen. Komplexe Bedingungen vergleichen mehrere Werte auch in Abhängigkeit zueinander.

Die Validierung muss über das gesamte Datenset stattfinden. Das System muss die Möglichkeit bieten, zu prüfen, wie oft bestimmte strukturelle Eigenschaften im gesamten Datenset vorhanden sind oder fehlen. Es muss möglich sein, einzelne Datenobjekte und ihre Eigenschaften zu prüfen. Die Datenobjekte und die Eigenschaften ergeben sich aus der Struktur der externen Datenquellen (siehe BA 2). Es muss die Möglichkeit bestehen, die Kardinalität von Eigenschaften sowie den Datentyp zu überprüfen. Die Kardinalitäten sollten gängigen Mustern entsprechen [0..1], [1..1], [0..*] und [1..*]. Das System soll die Möglichkeit bereitstellen, mit Regulären Ausdrücken die Daten auf bestimmte Muster in zu prüfen. Es solle möglich sein, die Länge von Zeichenketten

bestimmter Eigenschaften der Datenobjekte zu prüfen. Das System soll die Fehler in den Daten nicht korrigieren.

Anwendungsfall 3: „Daten prüfen“ (S. 27)

4.3.4 Prüfberichte erzeugen

/BA4/

Nach jeder Prüfung muss ein Prüfbericht gespeichert werden. Der Bericht sollte die Ergebnisse der Prüfung in einer übersichtlichen Darstellung auf einer HTML-Seite anzeigen. Das System soll auch die Ausgabe des Prüfberichts in ein kompakteres, maschinenlesbares Format unterstützen. Der Prüfbericht muss Informationen darüber enthalten, was geprüft wurde, welche Einschränkungen fehlgeschlagen sind und wie hoch die Anzahl der damit ermittelten Fehler ist. Der Prüfbericht muss verschiedene Level von Prüffehlern, die während der Prüfung aufgetreten sind darstellen können. Den Akteuren soll die Möglichkeit gegeben werden die Level bzw. den Schweregrad für Verletzungen der Bedingungen eigenständig und für jede Bedingung separat festzulegen. Als Schweregrad müssen mindestens drei verschiedene Kategorien zur Verfügung stehen. Für nicht-kritische Verletzungen mit informativen Charakter. Für nicht-kritische Verletzungen als Warnung und für kritische Verletzungen, die als schwerwiegende Fehler angesehen werden. Als Ergänzung sollte die Möglichkeit bestehen, dass Akteure eigene, deskriptive Fehlermeldungen verfassen können.

Anwendungsfall 3: „Daten prüfen“ (S. 27)

4.3.5 Dateninhalte anzeigen

/BA5/

Das System muss eine Möglichkeit bereitstellen, dass eine Datenansicht der in RDF vorliegenden Daten erzeugt werden kann. Dafür kann das System eine Vorlage erzeugen, die durch ein externes Programm interpretiert und auf die Daten angewendet werden kann. Die daraus entstehende Datenansicht muss einfach zugänglich und für alle Akteure verständlich sein. Die Ansicht soll alle Informationen beinhalten, die den Daten zu entnehmen sind. Dazu zählen insbesondere Objekte und deren Eigenschaften. Die erzeugte Ansicht soll für alle Datenobjekte generiert werden.

Anwendungsfall 4: „Dateninhalt sichten“ (S. 28)

4.3.6 Nachvollziehbarkeit

/BA6/

Die durchgeführten Schritte zur Verarbeitung der Daten müssen nachvollziehbar und deterministisch sein. Um Nachvollziehbarkeit zu gewährleisten, muss das System fähig sein Informationen über jeden Verarbeitungsschritt anzuzeigen und abzuspeichern. Die Ergebnisse (Artefakte) der einzelnen Verarbeitungsschritte müssen mit der Information über den Zeitpunkt der Entstehung bereitgestellt werden. Das System muss nach jedem Arbeitsschritt die Daten speichern und darf die Ausgangsdaten nicht verändern. Ein Akteur muss die Information über die Datenquelle dem System übergeben können. Verarbeitungsschritte sollten Abhängigkeiten zu anderen Prozessen und deren Ergebnisse (Artefakte) formulieren können. Verarbeitungsschritte sollten sich miteinander verketten lassen können.

Anwendungsfall 1: „Daten transformieren“ (S. 25)
Anwendungsfall 2: „Datenstruktur sichten“ (S. 26)
Anwendungsfall 3: „Daten prüfen“ (S. 27)

4.4 Systemanforderungen

Neben den Benutzeranforderungen hat der Kontext und die Systemumgebung einen maßgeblichen Einfluss auf das zu entwickelnde System. Im folgenden Abschnitt werden daher die Bedingungen, die die Implementierung beeinflussen, näher beschrieben. [Bal09, S. 459 ff.]

4.4.1 Interoperabilität und Werkzeugunterstützung

/NF1/

Es sollten bestehende Werkzeuge für spezielle Aufgaben integriert werden können, damit die Anzahl der Neuimplementierungen gering gehalten wird. Diese umfassen externe Softwarebibliotheken für den Umgang mit RDF Daten und Kommandozeilenprogramme, die eventuell notwendige Datenkonvertierungen übernehmen. Das System muss fähig sein Daten an externe Programme weiterzuleiten und die Ergebnisse aus externen Verarbeitungen entgegen zu nehmen und weiterzuverarbeiten.

Anwendungsfall 1: „Daten transformieren“ (S. 25)
Anwendungsfall 2: „Datenstruktur sichten“ (S. 26)

Anwendungsfall 3: „Daten prüfen“ (S. 27)
Anwendungsfall 4: „Dateninhalt sichten“ (S. 28)

4.4.2 Erweiterbarkeit

/NF2/

Über alle Verarbeitungsschritte hinweg soll eine Erweiterbarkeit sichergestellt werden. Dies ist generell für Systeme von Bedeutungen, deren Zweck unterschiedlichen und eventuell noch nicht bekannten Anforderungen zugrunde liegen können. Die bisherigen Erfahrungen sollen in den Workflow einfließen. Da es in Zukunft jedoch weitere Forschungsfragen und zusätzliche Datenquellen mit neuen Anforderungen geben wird, ist der Datenintegrationsprozess so flexibel wie möglich zu gestalten. Das bedeutet, dass neue Prozesse integriert werden können, ohne dass andere Prozesse dafür angepasst werden müssen.

Anwendungsfall 1: „Daten transformieren“ (S. 25)
Anwendungsfall 2: „Datenstruktur sichten“ (S. 26)
Anwendungsfall 3: „Daten prüfen“ (S. 27)
Anwendungsfall 4: „Dateninhalt sichten“ (S. 28)

4.4.3 Linked-Data-Umgebung

/NF3/

Das Projekt diggr setzt überwiegend Linked-Data-Technologien ein, daher muss das System das Datenformat RDF lesen, verarbeiten und speichern können. RDF soll verwendet werden, um die Erweiterbarkeit und Interoperabilität der Daten zu unterstützen.

Anwendungsfall 1: „Daten transformieren“ (S. 25)
Anwendungsfall 2: „Datenstruktur sichten“ (S. 26)
Anwendungsfall 3: „Daten prüfen“ (S. 27)
Anwendungsfall 4: „Dateninhalt sichten“ (S. 28)

4.4.4 Performance

/NF4/

Da auch größere Datenmengen im Projekt verarbeitet werden, spielt die Performance eine Rolle bei der Ausführung der einzelnen Verarbeitungsschritte. Ihr wird jedoch ei-

ne untergeordnete Rolle zugewiesen, da keine kritischen Entscheidungsprozesse damit verbunden sind. Es kann daher keine konkrete Angabe zu akzeptablen Verarbeitungszeiten bestimmter Datenmengen getroffen werden. Die Größenordnung der zu verarbeitenden Daten kann in Abhängigkeit zur Forschungsfrage und damit Datenquelle schwanken. Es sind jedoch keine Datenmengen im zweistelligen Gigabyte-Bereich zu erwarten.

Anwendungsfall 1: „Daten transformieren“ (S. 25)
Anwendungsfall 3: „Daten prüfen“ (S. 27)

4.5 Zusammenfassung der Anforderungen

Welche Anforderungen für die Umsetzung der Anwendungsfälle benötigt werden, ist der Tab. 4.1 zu entnehmen. Anforderungen an die Systemumgebung und die sich damit auf mehrere Anwendungsfälle beziehen, sind mit Klammern gekennzeichnet. Zur Priorisierung der einzelnen Anforderungen wird eine dreistufige Skala gewählt. Es unterteilt die Anforderungen nach „unbedingt erforderlich“ (1), „sollte umgesetzt werde“n (2) und „kann umgesetzt werden“ (3).

Die primären Ziele des Workflows bestehen darin, die Daten für die Integration vorzubereiten und zu prüfen. Die Umsetzung dieser Funktionen wird die höchste Priorität beigemessen. Die bereits im Projekt festgelegte Linked-Data-Ausrichtung ist eine Rahmenbedingung, daher wird die Anforderung „Linked-Data-Umgebung“ in NF 3 auf S. 33 für alle Anwendungsfälle benötigt. Sie hat maßgeblichen Einfluss auf die später verwendeten Software-Bibliotheken und Programme. Dass alle Schritte des Workflows flexibel angepasst werden können, geht aus der Anforderung „Erweiterbarkeit“ in NF 2 auf S. 33 hervor. Die Möglichkeit, Werkzeuge Dritter einzubinden gilt auch für den gesamten Workflow und bezieht sich somit auf alle Anwendungsfälle. Dies ist in Anforderung „Interoperabilität und Werkzeugunterstützung“ in NF 1 auf S. 32 formuliert. Die Performance ist für die Anforderung „Daten transformieren“ in AF 1 auf S. 25 und Anforderung „Daten prüfen“ in AF 3 auf S. 27 zu betrachten. Ihr wird jedoch eine geringe Priorität zugeordnet.

	Datentransformation	Strukturelle Informationen speichern	Datenvalidierung	Prüfberichte erzeugen	Dateninhalte anzeigen	Nachvollziehbarkeit	Interoperabilität und Werkzeugunterstützung	Erweiterbarkeit	Linked-Data-Umgebung	Performance
Daten transformieren /AF1/	x					(x)	(x)	(x)	(x)	(x)
Datenstruktur sichten /AF2/		x				(x)	(x)	(x)	(x)	
Daten prüfen /AF3/			x	x		(x)	(x)	(x)	(x)	(x)
Dateninhalt sichten /AF4/					x		(x)	(x)	(x)	
Priorität	1	1	1	1	2	2	2	2	1	3

Tabelle 4.1: Anforderungsmatrix: Übersicht der Anwendungsfälle und der zur Umsetzung benötigten Anforderungen

5 Spezifikation

Im vorhergehenden Abschnitt wurden Anwendungsfälle skizziert und daraus Anforderungen abgeleitet. Die für die Umsetzung erforderlichen Funktionen, Schnittstellen und der Datenfluss werden in Abschnitt 5.2 beschrieben. Dafür werden im folgenden Abschnitt 5.1 zusätzliche Begriffe, die zum Verständnis beitragen sollen, erläutert. Datenmodelle für die zu verarbeitenden Daten, die Anwendungsprofile und für die Datenansicht werden in Abschnitt 5.2.1 erläutert und spezifiziert. Anschließend wird in Abschnitt 5.2.2 der Workflow in einzelne Verarbeitungsschritte aufgeteilt und in eine logische Abfolge gebracht.

5.1 Terminologie

Für ein einheitliches Verständnis der folgenden Abschnitte müssen zunächst weitere Begriffe definiert werden.

Blank Node Blank Nodes identifizieren keine bestimmte Ressource. Blank Nodes weisen auf eine existierende Beziehung hin, ohne sie genau zu benennen. [CWL14]

Data Graph Jeder RDF Graph ist ein Data Graph.[1] Er bezeichnet den RDF Graph im Kontext der Validierung durch ein SHACL Shape bzw. des Shapes Graph.

Shapes Graph ist ein RDF Graph, der keine oder mehrere SHACL Shapes beinhaltet, die für eine Validation eines Data Graphs genutzt werden.[2]

Task Bezeichnet einen in sich geschlossenen Prozessschritt.

Artefakt Daten, die persistent gespeichert werden. Können Zwischenergebnisse sein.

Datensets Als Datenset wird in dieser Arbeit eine Sammlung von Datenobjekten in einer Datei bezeichnet.

[1] `https://www.w3.org/TR/shacl/#dfn-data-graph`
[2] `https://www.w3.org/TR/shacl/#dfn-shapes-graph`

5.2 Entwurf und Architektur

Um den Entwicklungsaufwand der sich aus den unterschiedlichen Anforderungen ergibt, zu minimieren, sollen an möglichst vielen Stellen bereits bestehende Programme und Bibliotheken eingesetzt werden. Damit einher geht die Voraussetzung, dass die Programme miteinander kommunizieren können und die Daten von einem Verarbeitungsschritt zum nächsten weitergeleitet werden können. Aus den Anforderungen ergibt sich sowohl eine Datenzentrierung als auch eine Prozessorientierung für die Entwicklung. Der bisherige Workflow ist stark datenzentriert. Mit einer zusätzlichen Prozessorientierung werden Inkonsistenzen bei der wiederholten Verarbeitung unterschiedlicher Quellen vermieden.

Aufgrund der Anforderung „Linked-Data-Umgebung“ in NF 3 auf S. 33 wird der gesamte Workflow überwiegend mit Komponenten, die mit RDF arbeiten können realisiert. Eine Anforderung an den Workflow stellt die Anforderung „Erweiterbarkeit“ in NF 2 auf S. 33 und die Anforderung „Interoperabilität und Werkzeugunterstützung“ in NF 1 auf S. 32 des Systems dar. Diese Eigenschaften sollen mit der Verkettung einzelner, eigenständiger Programme zu einer größeren Werkzeugkette (engl. *toolchain*) realisiert werden.

Eine Toolchain wird bei [CLA] definiert als eine Abfolge von Programmen, welche

- sequenziell auf den gleichen Daten arbeiten,
- die Ausgabe des vorhergehenden Werkzeugs als Eingabe übernehmen,
- Informationen zu diesen Eingabedaten kumulativ hinzufügen, und
- weder die Ausgangsdaten noch die Daten, die durch vorhergehende Werkzeuge hinzugefügt wurden, verändern.

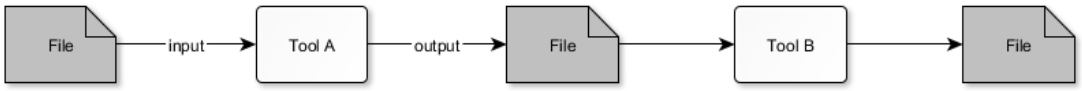

Abbildung 5.1: Einfache Toolchain

Bei dieser Art der Verkettung eigenständiger Programme müssen die Schnittstellen und der Datenaustausch zwischen den einzelnen Komponenten definiert werden. Zur Unterstützung dieser Aufgabe und zur Überwachung der einzelnen Prozesse eignen sich sogenannte Workflow Management Systeme bzw. Plattformen oder Bibliotheken

für Batch Prozesse. Diese stellen Möglichkeiten zur Verfügung Prozesse zu verwalten, zu koordinieren und Abhängigkeiten festzulegen. In Abb. 5.2 auf S. 38 wird die Erweiterung einer Toolchain durch eine zusätzliche Ebene für die Koordination dargestellt.

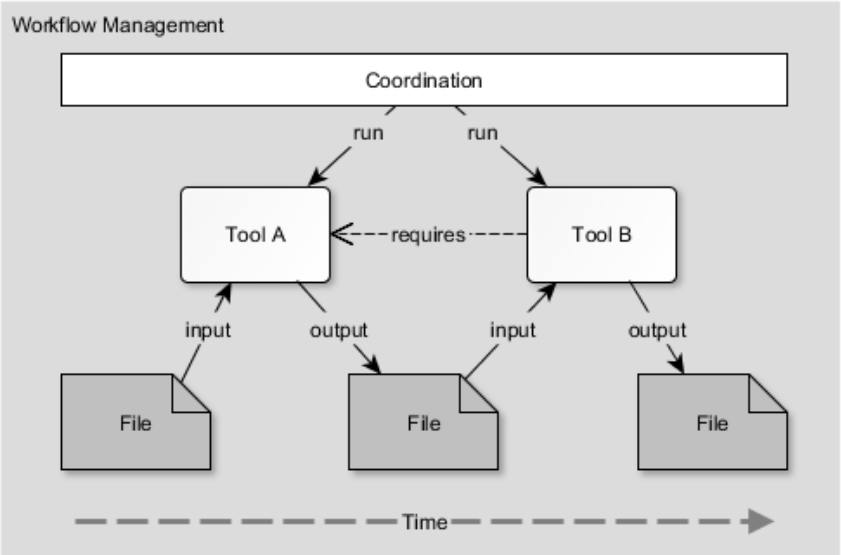

Abbildung 5.2: Toolchain mit Unterstützung durch ein Workflow Management

Aus dem Open Source Bereich zählen unter anderem Azkaban[3], Airflow[4] oder Luigi[5] zu dieser Art von System. Luigi wird bereits seit einigen Jahren an der Universitätsbibliothek Leipzig erfolgreich für den Aufbau eines aggregierten Artikelindex eingesetzt. [Czy] Dabei werden große heterogene Datenmengen aus vielen verschiedenen Bezugsquellen verarbeitet. Der Einsatz von Luigi im Projekt diggr verspricht daher eine höhere Chance auf Akzeptanz, da es als Workflow Management System bereits bekannt ist. Eine ausführliche Beschreibung von Luigi folgt in Abschn. 6.1.1 auf S. 47.

[3] `https://azkaban.github.io`
[4] `https://airflow.apache.org`
[5] `https://github.com/spotify/luigi`

5.2.1 Datenmodelle

Daten

Wie in Abschn. 1.2 auf S. 8 beschrieben, werden die Daten im Projekt diggr überwiegend aus der Domäne der Videospiele bezogen. Als Datenquellen dienen kommerzielle sowie nicht-kommerzielle Online Datenbanken und Foren zu Videospielen. Die Informationen in den Datenbanken werden von Fan-Communities erstellt und gepflegt. Die in dieser Arbeit verwendeten Daten enthalten detaillierte Informationen zu Videospielen und ihre Veröffentlichungshistorie. Im Anhang A werden Auszüge aus den Datensets im Format JSON dargestellt. Aufgrund unterschiedlicher Ziele und Arbeitsweisen der einzelnen Webseitenbetreiber und Communities besteht dabei eine strukturelle Heterogenität zwischen den Daten. [ES13]

Die vollständige Integration der Daten (physisch und logisch) kann zu Interpretationsschwierigkeiten bei der Auswertung der Daten führen (vgl. Abschn. 3.2 auf S. 21). Deshalb soll bei der Überarbeitung des Workflows lediglich eine physische, aber keine logische Integration stattfinden. Die logische Integration wird ausgelagert an die nächsthöhere Ebene, an das Vokabular bzw. an eine Ontologie. Dabei sollen Vorteile, die RDF für die Datenintegration bietet verstärkt genutzt werden. Nach [LPB18], bietet RDF die Möglichkeit, Daten aus verschiedenen Quellen automatisiert in einen Graphen zu integrieren, was unter anderem durch die Nutzung von URIs realisiert wird. Aus diesem Grund muss den Quelldaten zum Zeitpunkt der Transformation in RDF ein individueller Namensraum hinzugefügt werden. Als Basis-URI wird `http://diggr.link/data/` verwendet, an dem ein Bezeichner für die Quelle angehängt wird. In Listing 5.1 wird in Zeile 1 der Präfix und der Namensraum für die Quelle `example` definiert. In den Zeilen 3-10 folgen beispielhaft Triple mit der Auszeichnung des entsprechenden Namensraums.

```
1 @prefix example: <http://diggr.link/data/example/> .
2
3 example:gameRelease123 a example:Releases ;
4     example:publisher example:publisher123 ;
5     example:region "EU" ;
6     example:release_date "2014-06-14T00:00:00" ;
7     example:title "The Last Of Us" .
```

Listing 5.1: RDF Datenmodell und Namensraum

Sobald die Quelldaten mit individuellen Namensräumen in RDF vorliegen, können sie in eine gemeinsame Wissensbasis in Form einer RDF Datenbank importiert werden. Wie die Daten später in Kontext zu einander betrachtet werden können, wird in Abschn. 7.2 auf S. 64 beschrieben.

Das hier beschriebene Datenmodell dient zur Umsetzung der Anforderung „Datentransformation“ in BA 1 auf S. 29

Anwendungsprofile (Shapes)

Die Kriterien nach denen ein Programm die Daten prüft, müssen sowohl maschinenlesbar als auch für Menschen verständlich sein. Natürliche Sprache weist Ambiguitäten auf und ist schwer für Maschinen zu verarbeiten. In einer maschinennahen Sprache (Programmiersprache) muss die Pflege durch verschiedene Personengruppen berücksichtigt werden. Aus diesen Gründen muss ein Mittel zur Verfügung stehen, welches die Balance zwischen beiden Ansätzen schafft. In dieser Arbeit ist die Entscheidung auf den Einsatz von Anwendungsprofilen gefallen. Wie in Abschn. 2.5 auf S. 17 beschrieben, stehen für RDF Daten zwei Standards zur Beschreibung der Daten zur Verfügung. Dabei verfolgt laut [GPBK17], SHACL mehr die Intention zur Validation. „The designers of SHACL aimed at providing a constraint language for RDF. […] The main goal is to verify that a given RDF graph satisfies a collection of constraints.“ Im Vergleich mit ShEX: „The designers of ShEx intended the language to be a grammar or schema for RDF graphs.“ Darüber hinaus besteht bei SHACL die Möglichkeit Fehler, die auftreten wenn Bedingungen nicht erfüllt werden, genauer beschreiben zu können (siehe Anforderung „Datenvalidierung“ in BA 3 auf S. 30). „A SHACL Validation Report can be very useful for detecting and repairing errors in RDF Graphs.“ Für diese Arbeit fiel die Wahl daher auf SHACL zur Beschreibung und Prüfung der Daten. Im folgenden Abschnitt wird das Grundgerüst der Shapes beschrieben, die auf Grundlage des JSON Schema automatisiert erzeugt werden sollen.

SHACL Shapes können verschiedene Ziele (engl. *targets*) definieren die durch das Shape geprüft werden sollen. Neben der Definition einzelner, expliziter Knoten können auch ganze Klassen als Ziel dienen. Das bedeutet, das alle Knoten, die Instanzen dieser Klasse sind gegen das Shape geprüft werden. In Zeile 15 von Listing 5.2 auf S. 41 wird mit `sh:targetClass` diese Zielklasse definiert. Shapes sind benannt (Zeile 1) und beschreiben Eigenschaften, die mit `sh:property` eingeleitet werden. Standardmäßig sollen folgende Eigenschaften erzeugt werden:

sh:path Definiert den Pfad der vom geprüften Knoten zu einem Wert führt. Dabei handelt es sich um Prädikate in Form von URIs.

sh:datatype Definiert den Datentyp von Literalen. Diese basieren auf den Datentypen von XML Schema.

sh:minCount/sh:maxCount Spezifiziert die Kardinalität von eindeutigen Werten und begrenzt die minimale bzw. maximale Anzahl von Werten.

Die Shapes sollen für eine erste automatisierte Prüfung der RDF Daten genutzt werden. Dafür wird für jede Eigenschaft angenommen, dass sie genau einmal pro Ressource vorhanden sein muss.

```
example:Shape_Releases a sh:Shape ;
    sh:property
        [sh:datatype xsd:string ;
            sh:maxCount 1 ;
            sh:minCount 1 ;
            sh:path example:region ],
        [sh:datatype xsd:string ;
            sh:maxCount 1 ;
            sh:minCount 1 ;
            sh:path example:title ],
        [sh:datatype xsd:integer ;
            sh:maxCount 1 ;
            sh:minCount 1 ;
            sh:path example:rating ] ;
    sh:targetClass example:Releases .
```

Listing 5.2: Beispielhafter Auszug aus einem Shapes Graph

Das hier beschriebene Datenmodell dient zur Umsetzung der Anforderung „Datenvalidierung“ in BA 3 auf S. 30

Template für Datenansicht

Die Daten sollen in einer einfachen und strukturierten Ansicht wiedergegeben werden. Mit Hilfe von Schablonen (engl. *templates*) kann sichergestellt werden, dass alle Daten in gleicher Art und Weise präsentiert werden. Für einen einfachen Zugang bietet sich eine Präsentation in Form einer Webseite an.

Für die Darstellung und Erfassung von RDF Daten auf Webseiten gibt es verschiedene Möglichkeiten der Implementierung. Mit Hilfe der Java-Script Bibliothek RDForms[6] können Eingabeformulare für RDF Daten erzeugt werden. Das gleiche Ziel verfolgt RDForm[7] als jQuery Plugin. Beide Implementierung basieren auf der RDFa Notation[8]. RDFa ist eine Erweiterung von HTML, um Webdokumente mit Linked Data anzureichern. Für umfangreiche und nutzerfreundliche Webapplikationen auf Basis von RDF Daten stellt der Linked Data Reactor[9] ein umfassendes Framework zur Verfügung. Die bisher genannten Lösungen bieten sich vor allem für Anwendungen an, die dynamische Webseiten erfordern.

In den letzten Jahren werden zur Erstellung und Pflege von Webseiten auch zunehmend Programme verwendet, die statische Webseiten produzieren. Diese Art verringert den Aufwand für die Pflege von Webseiten, da sie beispielsweise keine Datenbanken verwenden. Eines der bekanntesten Programme ist Jekyll. Dazu beigetragen hat die Entwicklungsplattform GitHub[10], die Jekyll für GitHub Pages[11] einsetzt. Jekyll verwendet Layout-Dateien und Inhaltsdateien um reine HTML Seiten zu generieren. Jekyll bietet auch die Grundlage für die Implementierung von Jekyll-RDF[12], die RDF Daten als Inhaltsdateien verarbeitet und zum Erzeugen statischer HTML Seiten verwendet. Jekyll-RDF steht als Plugin für Jekyll zur Verfügung.

Aufgrund der Anforderung 5 soll ein einfacher Zugang zu den Informationen geschaffen werden. Es werden keine Formulare oder dynamische Elemente benötigt. Die niedrige Priorisierung dieser Anforderung (vgl. Abschn. 4.5 auf S. 34) lässt eine exemplarische Implementierung zu. Aus diesen Gründen wurde sich für den prototypischen Einsatz von Jekyll-RDF entschieden. Für die Schablonen verwendet Jekyll-RDF, wie auch Jekyll, die Liquid Template Language[13].

Um Schablonen für alle Daten einer Quelle zu erstellen, müssen die Daten und ihre Eigenschaften beschrieben werden. Da während des Workflows die Struktur der Daten extrahiert wird, können die Informationen, die in Form der Anwendungsprofile vorliegt, für eine automatisierte Erstellung eines Jekyll-RDF Templates verwendet werden.

[6] http://rdforms.org/
[7] https://github.com/simeonackermann/RDForm
[8] http://www.w3.org/TR/rdfa-primer/
[9] http://ld-r.org/
[10] https://www.github.com
[11] https://pages.github.com
[12] https://github.com/white-gecko/jekyll-rdf
[13] http://shopify.github.io/liquid/

Die Informationen aus den SHACL Shapes können in ein Jekyll-RDF Template, wie in Listing 5.3 auf S. 43 dargestellt, überführt werden.

```
<h2>{{ page.rdf.iri }}</h2>
{% assign type = page.rdf | rdf_property: "<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>" %}
{% if type.iri == "http://diggr.link/data/gamefaqs/data" %}
    Ressource: <strong>Data</strong> (http://diggr.link/data/gamefaqs/Data)<br><br>
    {% assign developer = page.rdf | rdf_property: "<http://diggr.link/data/gamefaqs/developer>", nil, true %}
    {% if developer %}
        <dl>
        <dt>developer</dt>
        {% for each_developer in develope %}
            {% if each_developer.iri %}
                <dd>Link: <a href={{each_developer.page_url}}>{{each_developer}}</a></dd>
            {% else %}
                <dd>{{each_developer }}</dd>
            {% endif %}
        {% endfor %}
        </dl>
    {% endif %}
    {% assign title = page.rdf | rdf_property: "<http://diggr.link/data/gamefaqs/title>", nil, true %}
    {% if title %}
        <dl>
        <dt>title</dt>
        {% for each_title in title %}
            {% if each_title.iri %}
                <dd>Link: <a href={{each_title.page_url}}>{{each_title}}</a></dd>
            {% else %}
                <dd>{{each_title }}</dd>
            {% endif %}
        {% endfor %}
    </dl>
    {% endif %}
{% endif %}
```

Listing 5.3: Jekyll-RDF Template

Das hier beschriebene Datenmodell dient zur Umsetzung der Anforderung „Dateninhalte anzeigen“ in BA 5 auf S. 31

5.2.2 Workflow

Die in Abschn. 4.2 auf S. 25 skizzierten Anwendungsfälle stellen einzelne Schritte für den Prozess der Datenintegration dar. Sie müssen in einer gewissen Abhängigkeit von einander betrachtet werden. Beispielsweise kann eine Validierung der Daten nur stattfinden, wenn die Daten bereits im RDF Format vorliegen und ein Anwendungsprofil beschrieben wurde. In diesem Abschnitt wird ein Workflow skizziert, der die Benutzeranforderungen aufnimmt und in zu realisierende Programmschritte einteilt. Dabei wird beschrieben mit welchen Eingangsformaten (engl. *input*) der Verarbeitungsschritt stattfindet und welche Ausgangsformate (engl. *output*) dabei entsteht. Die Schritte werden anschließend über eine Gesamtübersicht Abb. 5.3 auf S. 46 in eine logische Reihenfolge gebracht.

A. Datenstruktur extrahieren

Die Quelldaten befinden sich im Dateisystem und liegen in einem validen JSON Format vor. Aus den Quelldaten wird die zugrundeliegende Struktur und das Datenschema extrahiert. Das Ergebnis des Prozesses wird als separate Datei im Dateisystem abgespeichert.

B. Datentransformation JSON zu RDF

Die Quelldaten befinden sich im Dateisystem und liegen in einem gültigen JSON-Format vor. Die Daten werden in ein valides RDF Format transformiert. Ein Namensraum, der die Herkunft der Daten kennzeichnet, ist enthalten. Das Ergebnis des Prozesses wird als separate Datei im Dateisystem abgespeichert.

Die eigentliche Datenintegration in ein zentrales Speichersystem wie einer RDF Datenbank kann bereits nach diesem Schritt stattfinden. Da der Datenimport an dieser Stelle unkompliziert ist, wird er in dieser Arbeit nicht näher betrachtet.

C. Erzeugen eines RDF Anwendungsprofils

Aus dem erstellten JSON Schema wird eine Datei zur Beschreibung der Daten in RDF erstellt. Diese beinhaltet die Datenobjekte, ihre Eigenschaften und Bedingungen, die für alle Quellen gleichermaßen geprüft werden sollen. Das Ergebnis des Prozess wird als separate Datei im Dateisystem abgespeichert. Das System meldet auf der Konsole den Status der Verarbeitung.

D. Ergänzung des Anwendungsprofils

Nachdem ein Anwendungsprofil erfolgreich erzeugt wurde, findet eine manuelle Bearbeitung des Anwendungsprofils statt. Sie enthält nach der Bearbeitung weitere Bedingungen, die für eine Validierung genutzt werden sollen.

E. Voraussetzung für die Ansicht der Dateninhalte schaffen

Das Anwendungsprofil wird für die Erzeugung eines Templates für eine Datenansicht verwendet. Als Ergebnis liegt eine Template-Datei vor, die von einem weiteren Programm für die Generierung einer Datenansicht genutzt werden kann.

F. Datenvalidierung durchführen und Prüfbericht erstellen

Ein Anwendungsprofil wird auf die Daten im RDF Format angewendet. Als Ergebnis entsteht ein Prüfbericht in HTML oder einem RDF Format, der als Datei im Dateisystem abgespeichert wird.

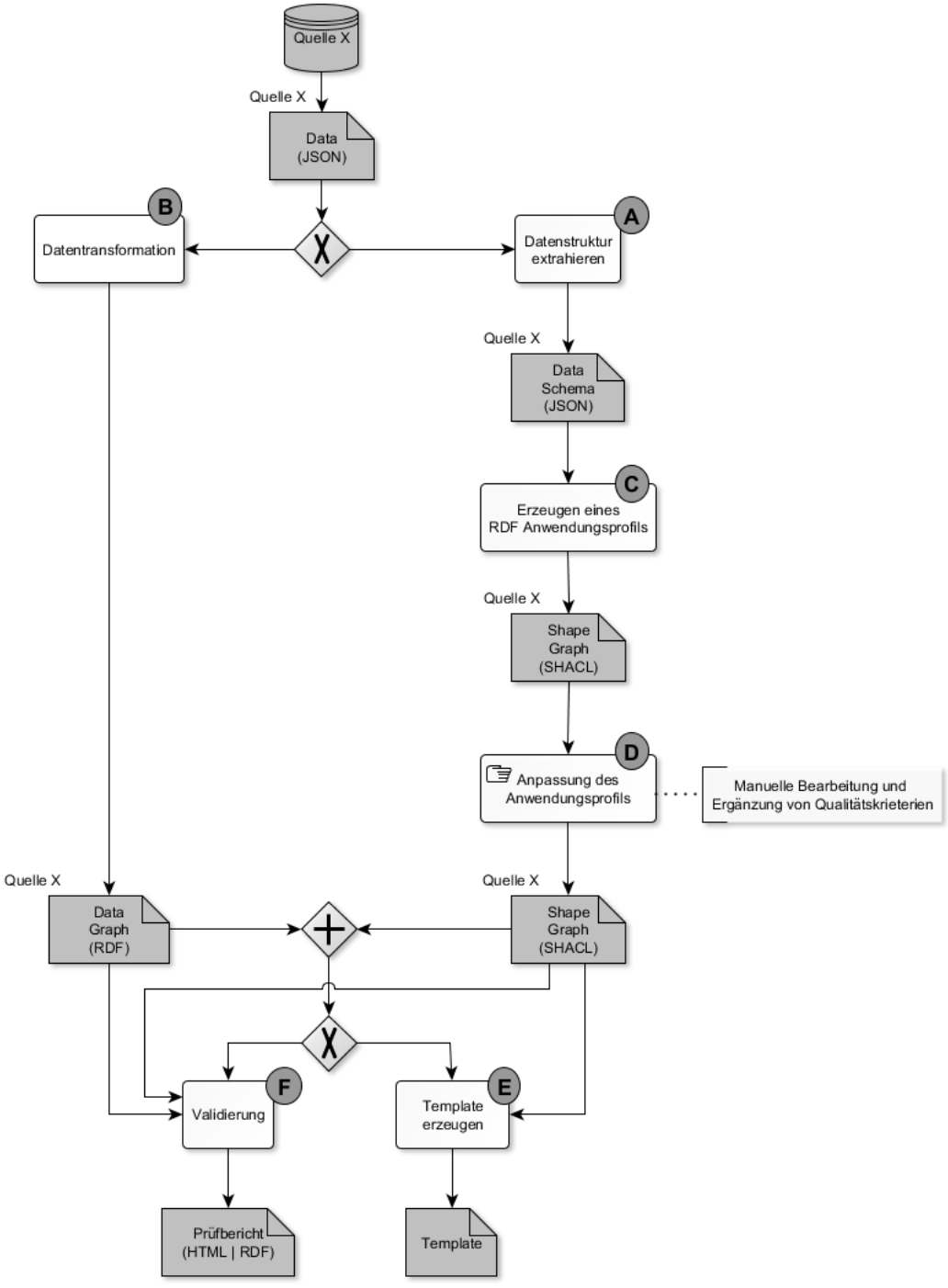

Abbildung 5.3: Gesamtansicht des Workflows (Soll-Zustand)

6 Realisierung

Im Rahmen dieser Arbeit wurde ein Workflow zur Integration und Bewertung heterogener Daten mit dem Schwerpunkt auf den Einsatz von Linked-Data-Technologien entworfen und prototypisch umgesetzt. Er setzt dabei auf die Wiederverwendung von bestehenden Programmcodes durch Bibliotheken. Diese werden im folgenden Abschnitt 6.1 vorgestellt. Im Anschluss daran werden in Abschn. 6.2 auf S. 50 die implementierten Verarbeitungsschritte und ihre Organisation unter Einsatz von Luigi beschrieben.

6.1 Verwendete Bibliotheken und Frameworks

Bei der Umsetzung des Workflows wird überwiegend auf vorhandene Bibliotheken und Frameworks zurückgegriffen, die die Arbeit mit JSON und RDF-Daten erleichtern. Dies ermöglicht eine übersichtliche Nutzung des Quellcodes, in dem die einzelnen Bestandteile separat betrachtet und gepflegt werden können und vermeidet unnötige Neuentwicklungen. Im Folgenden wird der Funktionsumfang der eingesetzten Komponenten kurz beschrieben.

6.1.1 Luigi

Luigi[1] ist eine generische Workflowmanagment Bibliothek in Python. Sie ist von Spotify - einem Musik Streaming Dienst - entwickelt worden und steht als Open Source zur Verfügung.

Mit Luigi lassen sich simple bis komplexe Workflows implementieren. Die Basis bilden Aufgaben (engl. *Tasks*), welche einige grundlegende Eigenschaften mitbringen. Ein

[1] `https://github.com/spotify/luigi`
„Luigi is a Python module that helps you build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization etc.“

Task hat keine, eine oder mehr Abhängigkeiten, welche wiederum Tasks sein können. Weiterhin enthält ein Task Programmcode, welcher die Business Logik beinhaltet. Abschließend hat jeder Task ein oder mehrere Ziele (engl. *targets*). Tasks können durch keinen oder mehrere Parameter erweitert werden. Jeder Workflow kann in einzelne Tasks zerlegt werden. Die Tasks werden über den Programmcode selbst implementiert und miteinander verbunden. Die Workflows lassen sich über die Kommandozeilenintegration sowohl interaktiv als auch automatisiert über Standard Programmen wie *cron*[2] ausführen. [Czy] Als Benutzerschnittstelle bietet Luigi standardmäßig die Ein- und Ausgabe auf der Konsole an. Es ist jedoch auch möglich den integrierten Luigi Server zu starten und über eine Webansicht, die Tasks zu starten und zu überwachen. Für diese Arbeit wurde die Nutzung von Luigi über die Kommandozeile bevorzugt. Die zu definierenden Ziele unterstützen viele unterschiedliche Implementierungen, die meisten Artefakte sind jedoch Dateien. Dateien werden nicht verändert sobald sie einmal geschrieben wurden und bereits erfolgreich durchgeführte Prozessschritte werden standardmäßig nicht wiederholt. Damit verfolgt Luigi unter anderem die Unveränderbarkeit der Daten (engl. *immutability*). Sowohl in Hinblick auf Performance als auch auf die Fehlbedienung durch den Nutzer ergeben sich dadurch Vorteile.

Wie in Abschn. 5.2 auf S. 37 beschrieben, wird Luigi bereits an der Universitätsbibliothek Leipzig eingesetzt. Darüber hinaus erfüllt und unterstützt Luigi mit seinen Funktionen die Umsetzung der Anforderung „Nachvollziehbarkeit“ in BA 6 auf S. 32, Anforderung „Interoperabilität und Werkzeugunterstützung“ in NF 1 auf S. 32, Anforderung „Erweiterbarkeit“ in NF 2 auf S. 33 und Anforderung „Performance“ in NF 4 auf S. 33

Für die Arbeit wurde Luigi mit der Erweiterung `gluish`[3], unter anderem für einen einfacheren Umgang mit Shell Kommandos, eingesetzt.

6.1.2 RDFLib

RDFLib[4] ist eine Python Bibliothek für RDF Daten. Sie stellt Parser und Serialisierungen für verschiedene Formate (RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa) zur Verfügung. RDFLib gilt als Standardbibliothek für die Erzeugung und Verarbei-

[2] `http://unixhelp.ed.ac.uk/CGI/man-cgi?cron`

[3] `https://github.com/miku/gluish`

[4] `https://github.com/RDFLib/rdflib`
„RDFLib is a Python library for working with RDF, a simple yet powerful language for representing information.“

tung von RDF in Python und erfüllt damit die Anforderung „Linked-Data-Umgebung“ in NF 3 auf S. 33 und Anforderung „Datentransformation“ in BA 1 auf S. 29.

6.1.3 RDFUnit

Mit dem in Java implementierten Programm RDFUnit[5] besteht die Möglichkeit RDF Daten gegen ein bestimmtes Schema zu validieren. Es werden automatisiert Tests und Prüfberichte erstellt. RDFUnit unterstützt die meisten[6] SHACL Komponenten, unter anderem auch die Erweiterung durch SHACL SPARQL. Die für Anforderung „Datenvalidierung“ in BA 3 auf S. 30 geforderten Tests werden damit abgedeckt. Tests können sowohl über RDF Dateien als auch über SPARQL Endpunkte stattfinden. RDFUnit bietet die Möglichkeit Prüfberichte in den Formaten HTML, Turtle und JSON-LD zu erzeugen.

Neben RDFUnit existieren weitere SHACL Implementierungen in verschieden Sprachen und mit unterschiedlicher Abdeckung der SHACL Komponenten. Dazu zählen u. a. SHACLex[7] und TopBraid SHACL API[8]. RDFUnit bietet bisher als einzige Implementierung die Ausgabe der Prüfberichte in einer übersichtlichen HTML Darstellung und erfüllt damit die Anforderung „Prüfberichte erzeugen“ in BA 4 auf S. 31. Es verarbeitet ausschließlich Daten in RDF und erfüllt gleichermaßen die Anforderung „Linked-Data-Umgebung“ in NF 3 auf S. 33.

6.1.4 GenSON

Mit Hilfe des Python Pakets GenSON[9] kann das Schema einer beliebigen JSON Datei erzeugt werden. Es erfüllt den Hauptzweck, aus einer großen Anzahl von JSON Objekten ein einziges Schema zu extrahieren.

Dabei folgt der GenSON schema builder drei Regeln:

1. *Jedes* Objekt, das angegeben wird, muss unter dem generierten Schema validiert werden.

[5] `https://github.com/AKSW/RDFUnit`
[6] `https://github.com/AKSW/RDFUnit/issues/62`
[7] `http://labra.weso.es/shaclex/`
[8] `https://github.com/TopQuadrant/shacl`
[9] `https://pypi.python.org/pypi/genson/`

2. *Ein beliebiges* Objekt, das unter einem beliebigen Schema gültig ist, muss auch unter dem generierten Schema validiert werden.
3. Das generierte Schema sollte nach den ersten zwei Regeln so streng wie möglich sein.

Mit GenSON wird die Anforderung „Strukturelle Informationen speichern“ in BA 2 auf S. 30 erfüllt, da es mit dem Kommandozeilenprogramm möglich ist, die Struktur einer JSON Datei zu extrahieren.

Mit `https://jsonschema.net/` steht auch ein Onlinedienst zur Verfügung, der es ermöglicht, über einen Webservice das JSON Schema zu erhalten. Bei den in der Arbeit verwendeten Daten handelt es sich jedoch um Daten, die Dritten nur eingeschränkt verfügbar gemacht werden dürfen, daher wurde der Einsatz von GenSON als lokales Programm bevorzugt.

6.1.5 ShacShifter

Mit dem Kommandozeilenprogramm ShacShifter[10] gibt es eine erste Implementierungen, um SHACL Shapes für die Steuerung von Webansichten einzusetzen. Solche Webansichten können sowohl Formulare, als auch einfach, strukturierte Darstellung von Daten sein. Innerhalb dieser Arbeit wurde ShacShifter um eine Serialisierung nach Jekyll-RDF, wie sie bei Abschn. 5.2.1 auf S. 41 beschrieben wurde, ergänzt. ShacShifter unterstützt die Umsetzung von Anforderung „Dateninhalte anzeigen“ in BA 5 auf S. 31.

6.2 Organisation des Programmcodes

In dieser Arbeit wird Luigi für die Koordination der einzelnen Prozessschritte eingesetzt, es steuert den Datenfluss und definiert Abhängigkeiten. Damit stellt Luigi den Rahmen für eine Toolchain bzw. Tool-Komposition zur Verfügung. Es stellt darüber hinaus grundlegende Funktionen für Informationen über auftretende Fehler in der Ausführungskette bereit. Im folgenden Abschnitt wird der allgemeine Aufbau von Tasks dargestellt. Anschließend werden die implementierten Workflow Komponenten detailliert beschrieben.

[10] `https://github.com/AKSW/ShacShifter`

6.2.1 Luigi Tasks

Jeder Task ist eine eigene Klasse und besteht, je nach Prozess, aus vier Blöcken: Parameter, Abhängigkeiten (`requries`), Business Logik (`run`) und Ergebnis (`output`). Über die Parameter können Eingaben eines Nutzers oder eines anderen Programms an den Task übergeben werden. Externe Programme oder Tasks, die zuvor erfolgreich abgearbeitet sein müssen, lassen sich mit Abhängigkeiten definieren. Die eigentliche Business Logik eines Task erfolgt in einem separaten Block. Der Block für das Ergebnis definiert das Format und den Speicherort. In Listing 6.1 wird das Grundgerüst für Tasks dargestellt.

```
class BeispielTask(Task):
    parameter1 = Parameter(default="test1")
    parameter2 = Parameter(default="test2")

    def requires(self):
        return AndererTask(self.parameter2)

    def run(self):
        return LocalTarget("/tmp/test.txt")

    def output(self):
        with self.output().open('w') as f:
            f.write(self.parameter1)
```

Listing 6.1: Aufbau eines Task

MasterTask

Der `MasterTask` bildet die übergeordnete Einheit für die weiteren Verarbeitungsschritte. Er fungiert als eine Art Gruppierung und definiert übergreifende Parameter, wie den Basispfad unter dem die Ergebnisse der einzelnen Schritte gespeichert werden.

ExtractSchema

Der Task `ExtractSchema` nimmt jede valide JSON Datei über den Parameter `--file` entgegen und extrahiert das verwendete Schema mit Hilfe des Programms GenSON. Dieser Schritt erwartet, dass GenSON in der Systemumgebung vorhanden und ausführbar

ist. Der gesamte Task ist in Listing 6.2 dargestellt. Als Ergebnis wird ein JSON Schema gemeinsam mit der Angabe des aktuellen Datums im Dateinamen gespeichert. Der Task wird über den folgenden Aufruf gestartet:

```
$ PYTHONPATH='.' luigi --module master ExtractSchema --local-scheduler --file
../testdata/gameFAQs_ps4.json --namespace gamefaqs
```

```
class ExtractSchema(MasterTask):
    date = ClosestDateParameter(default=datetime.date.today())
    file = luigi.Parameter(default='../testdata/gameFAQs_ps4.json',
                           description='path json file',
                           significant=False)
    namespace = luigi.Parameter(default='example',
                                description='namespace of data source')

    def requires(self):
        return {
            'genson': Executable(name='genson', message='https://pypi.python↲
              .org/pypi/genson/')
        }

    def run(self):
        output = shellout("""genson {file} > {output}""", file=self.file)
        luigi.LocalTarget(output).move(self.output().path)

    def output(self):
        return luigi.LocalTarget(path=self.path(ext='schema'))
```

Listing 6.2: Task ExtractSchema

Abhängigkeit GenSON (siehe Abschn. 6.1.4 auf S. 49)

Anforderung „Datenstruktur sichten“ in AF 2 auf S. 26

BuildSample

Der Task `BuildSample` bietet die Möglichkeit aus einer Ausgangsdatei `--file` über eine einfache Zufallsauswahl mit `random.sample` (vgl. Listing 6.3) nur eine bestimmte Anzahl von Objekten `--size` als Stichprobe abzuspeichern. Dieser Task wurde für die Tests in Kap. 7 auf S. 59 implementiert. Mit dem folgenden Aufruf wird der Task gestartet:

```
$ PYTHONPATH='.' luigi --module master ExtractSchema --local-scheduler --file
../testdata/mobygames.json --namespace mobygames --size 5000
```

```
def run(self):
    with open(self.file) as in_file:
        data = json.load(in_file)

    with self.output().open('w') as f:
        sample = random.sample(data, int(self.size))
        f.write(json.dumps(sample))
```

Listing 6.3: Ausschnitt aus Task BuildSample

BuildRDFData

Der Task `BuildRDFData` bietet die Funktion beliebige Quelldaten in das Format RDF zu transformieren. Als Parameter müssen ein `--namespace` (default = example) und eine valide JSON Datei mit `--file` an diesen Task übergeben werden. Mit Hilfe des Ausdrucks `@vocab` wird in JSON-LD ein default-Vokabular definiert, welches verwendet wird, sobald keine spezifische URI zu einem Vokabular angegeben wird. Damit wird der Namensraum für diese Datenquelle festgelegt, wobei die Basis URI aus `http://diggr.link/data/` und der Nutzereingabe besteht. Es besteht für den Nutzer die Möglichkeit über den Parameter `--subject-id` eine Eigenschaft der Datenobjekte als Identifier festzulegen und während der Transformation daraus die URI der Ressource erzeugen zu lassen. Da diese Eigenschaft irgendeinen String-Wert enthalten kann, wird über die Funktionen `sanitize_triple` und `url_fix` sichergestellt, dass valide URIs erzeugt werden.

```
context = {"@vocab": "http://diggr.link/data/{}/".format(self.namespace)}

if self.subject_id is not None:
    context = {"@vocab": "http://diggr.link/data/{}/"
                    .format(self.namespace),
                    "@base": "http://diggr.link/data/{}/"
                    .format(self.namespace),
                    "{}".format(self.subject_id): {
                    "@id": "@id"}
                    }
```

Listing 6.4: Erzeugen des JSON-LD Context

Bei der Serialisierung von JSON zu RDF entstehen Blank Nodes, wenn keine eindeutigen Bezeichner für Datenobjekte festgelegt werden. Blank Nodes sind jedoch in der weiteren Verarbeitung der Daten nicht erwünscht. Die Empfehlungen von RDF 1.1[CWL14] schlagen vor, Blank Nodes durch URIs zu ersetzen. Dieser Vorgang wird als Skolemisierung bezeichnet und wurde für den Task (vgl. Listing 6.5) verwendet.

```
skolemGraph = skolemize(fixedgraph, authority="http://diggr.link/",
                        basepath="data/{}/".format(self.namespace))
skolemGraph.namespace_manager.bind(self.namespace,
                        URIRef('http://diggr.link/data/{}/'
                        .format(self.namespace)))
```

Listing 6.5: Blank Nodes durch URIs ersetzen

Durch das Programm werden den Daten Pseudoklassen hinzugefügt. Alle Objekte, die Blank Nodes oder URIs sind, bekommen als Klasse den Namen des verweisenden Prädikats zugewiesen. Für Subjekte, die keine Objekte sind, wird die default-Klasse `Item` vergeben (siehe Listing 6.6). Zur besseren Unterscheidung zwischen Klassen und den Eigenschaften werden die Klassen mit einem großen Anfangsbuchstaben geschrieben. Die Angabe von Klassen dient unter anderem der späteren Prüfung durch den Validationsprozess (vgl. Abschn. 5.2.1 auf S. 40). Darüber hinaus lassen sich die Daten nach der Integration in einer RDF Datenbank leichter abfragen, da es möglich ist, alle Ressourcen einer Klasse abzufragen, ohne eine konkrete Ressource anzusprechen.

```
for s, p, o in fixedgraph:
  if isinstance(o, BNode) or isinstance(o, URIRef):
      predicate = rdflib.namespace.split_uri(p)[1].replace(' ', '_').title()
      fixedgraph.add([o, RDF.type, URIRef(u'http://diggr.link/data/{↲
        namespace}/{predicate}'
                       .format(namespace=self.namespace, predicate=↲
                         predicate))])

# get all nodes
entities = [s for s, p, o in fixedgraph.triples((None, None, None))]
# get all nodes with a RDF type
derived = [s for s, p, o in fixedgraph.triples((None, RDF.type, None))]
rootnodes = list(set(entities) - set(derived))
```

Listing 6.6: Eigenschaften als Klassen zu den Graphen hinzufügen

Der Task kann über folgenden Aufruf gestartet werden:

```
$ PYTHONPATH='.' luigi --module master BuildRDFData --local-scheduler ../testda-
ta/gameFAQs_ps4.json --namespace gamefaqs --subject-id _id
```

Als Ergebnis des Task, wird der Data Graph erstellt und unter Angabe des aktuellen Datums und der angegebenen Parameter im Dateisystem gespeichert.

Anforderung „Daten transformieren“ in AF 1 auf S. 25

GenerateSHACL

Der Task `GenerateSHACL` erzeugt auf Basis des JSON Schema (vgl. Task `ExtractSchema`) rudimentäre SHACL Shapes in einer Turtle Serialisierung. Mit Hilfe des Parameters `--namespace` (default = example) wird der Namensraum festgelegt. Innerhalb dieses Task wird das Python Modul `schema2shacl` aufgerufen. Es enthält die Klassen `ShaclShape` und `Schema2Shacl`. Diese sind dafür zuständig durch das JSON Schema zu iterieren und die SHACL Shapes zu erzeugen. Die SHACL Shapes enthalten für jedes Datenobjekt die obligatorischen Eigenschaften. Die erzeugten Shapes bilden die Grundlage für die automatisierte Prüfung auf Vollständigkeit von Eigenschaften in den Daten und zur manuellen Ergänzung durch einen Menschen. Der Task kann über folgenden Aufruf gestartet werden:
`$ PYTHONPATH='.' luigi --module master GenerateSHACL --local-scheduler ../test-data/gameFAQs_ps4.json --namespace gamefaqs`
Als Ergebnis des Task wird der Shapes Graph erstellt und unter Angabe des aktuellen Datums und des Namensraums im Dateisystem gespeichert.

Abhängigkeit ExtractSchema

Anforderung „Daten prüfen“ in AF 3 auf S. 27

Validate

Für den Task `Validate` müssen zuvor die Tasks `ExtractSchema`, `GenerateSHACL` und `BuildRDFData` erfolgreich durchgeführt worden sein. Der Data Graph und der Shapes Graph müssen vorliegen und der Ablageort für das Programm `RDFUnit` muss konfiguriert sein. Bei Aufruf des Tasks wird durch Luigi sichergestellt, dass alle formulierten Abhängigkeiten erfolgreich durchgeführt sind bzw. durchgeführt werden. Anschließend wird RDFUnit, mit den erstellten Dateien als Input, erstellt. Falls keine bestimmte SHACL Datei mit dem Parameter `--shacl-file` durch den Nutzer übergeben wird, wird die vom Programm erzeugte SHACL Datei mit dem aktuellen Datum für die Validierung

des Data Graph verwendet. Diese enthält bereits Prüfungen auf die obligatorischen Eigenschaften. Das Ergebnis des Tasks, der Prüfbericht, wird innerhalb des Programms RDFUnit gespeichert `RDFUnit/data/results`. Um den Tasks zu starten ist folgender Aufruf möglich:
`$ PYTHONPATH='.' luigi --module master Validate --local-scheduler ../testdata/gameFAQs_ps4.json --namespace gamefaqs`

Abhängigkeit ExtractSchema, GenerateSHACL, BuildRDFData, RDFUnit

Anforderung „Daten prüfen“ in AF 3 auf S. 27

GenerateView

Dieser Task erzeugt ein Jekyll-RDF Template. Es wird mit Hilfe des Programms `ShacShifter` ein einfaches HTML Gerüst erstellt, das die Eigenschaften jeder Ressource auflistet. Da in den Quellen die Terme für die Eigenschaften auch mehrfach für unterschiedliche Datenobjekte benutzt werden, wird durch die Erzeugung eines Universally Unique Identifier (UUID) sichergestellt, dass keine Überschreibungen stattfinden. Die Klasse `JekyllRDFSerializer` erstellt dafür ein HTML Grundgerüst wie in Listing 6.7 gezeigt wird. Der Aufruf des Tasks ist mit folgendem Aufruf möglich:
`$ PYTHONPATH='.' luigi --module master GenerateView --local-scheduler ../testdata/gameFAQs_ps4.json --namespace gamefaqs`

```
uri = propertyShape.path
lowercase_str = uuid.uuid4().hex[:4]
label_jekyll = propertyShape.name.lower() \
    if propertyShape.isSet['name'] else propertyShape.path.lower().rsplit('/↲
      ', 1)[-1]
label_jekyll += "_"+ lowercase_str
label = propertyShape.name.lower() \
    if propertyShape.isSet['name'] else propertyShape.path.lower().rsplit('/↲
      ', 1)[-1]

html += """{{% assign {label} = page.rdf | rdf_property: "<{uri}>", nil, ↲
  true %}}\n""".format(
        uri=uri, label=label_jekyll, type=shapeName.lower())
html += """{{% if {label} %}}\n""".format(label=label_jekyll)
html += """<dl>"""
html += """<dt>{label}</dt>""".format(label=label)
```

```
html += """{{% for each_{label} in {label} %}}\n""".format(label=↲
  label_jekyll)
html += """{{% if each_{label}.iri %}}\n""".format(label=label_jekyll)
html += """<dd>Link: <a href={{{{each_{label}.page_url}}}}>{{{{each_{label↲
  }}}}}</a></dd>\n
            {{% else %}}\n""".format(label=label_jekyll)
html += """<dd>{{{{each_{label} }}}}</dd>\n{{% endif %}}\n""".format(label=↲
  label_jekyll)
html += """\n{% endfor %}\n</dl>\n{% endif %}\n"""
```

Listing 6.7: Auszug aus der Klasse JekyllRDFSerializer

Abhängigkeit ExtractSchema, GenerateSHACL

Anforderung „Dateninhalt sichten“ in AF 4 auf S. 28

6.2.2 Task-Abarbeitung

Die Tasks werden über die Komandozeile gestartet. Luigi gibt standardmäßig Informationen über den Status der Bearbeitung während und nach der Ausführung von Tasks aus. Über einen Zeitstempel kann nachverfolgt werden wann, welcher Task gestartet und beendet wurde. Darüber hinaus erhält der Nutzer Informationen über die verwendeten Parameter und die ausgeführten Kommandozeilenaufrufe. Werden über ein externes, eingebundenes Programm Informationen auf der Kommandozeile ausgegeben, so werden diese Ausgaben während der Verarbeitung ebenfalls angezeigt.

Luigi schließt jeden Task oder eine Abfolge von Tasks durch eine Zusammenfassung ab. Listing 6.8 zeigt die Ausgabe nach der erfolgreichen Ausführung des Tasks `Validate` mit allen Abhängigkeiten.

```
===== Luigi Execution Summary =====
Scheduled 5 tasks of which:
* 1 present dependencies were encountered:
    - 1 Executable(name=genson,
    message=https://pypi.python.org/pypi/genson/)
* 4 ran successfully:
    - 1 BuildRDFData(date=2018-03-01, namespace=example)
    - 1 ExtractSchema(date=2018-03-01)
    - 1 GenerateSHACL(date=2018-03-01, namespace=example)
    - 1 Validate(date=2018-03-01, namespace=example)
```

```
This progress looks :) because there were no failed tasks or
missing external dependencies
===== Luigi Execution Summary =====
```

Listing 6.8: Zusammenfassung von Tasks durch Luigi - erfolgreich durchgeführt

Sollte es zu Problemen bei der Abarbeitung des Workflows gekommen sein, wird durch Luigi gemeldet, in welchem Task der Fehler auftrat und welche Abhängigkeiten erwartet wurden.

```
===== Luigi Execution Summary =====
Scheduled 4 tasks of which:
* 2 present dependencies were encountered:
    - 1 BuildRDFData(date=2018-04-21, namespace=gamefaqs)
    - 1 ExtractSchema(date=2018-04-21, namespace=gamefaqs)
* 1 failed:
    - 1 GenerateSHACL(date=2018-04-21, namespace=gamefaqs)
* 1 were left pending, among these:
    * 1 had failed dependencies:
        - 1 Validate(date=2018-04-21, namespace=gamefaqs,
        rdfunit_location=../../RDFUnit)

This progress looks :( because there were failed tasks
===== Luigi Execution Summary =====
```

Listing 6.9: Zusammenfassung von Tasks durch Luigi - fehlgeschlagene Abarbeitung

Die Artefakte der einzelnen Tasks werden im Dateisystem nach den Namen der Tasks organisiert. Über die Struktur der Tasks wurden die folgenden Ordner angelegt:

```
$ tree luigi/
luigi/
└── Master
    ├── BuildRDFData
    │   ├── date-2018-03-29-namespace-example.ttl
    ├── BuildSample
    │   ├── date-2018-03-29-namespace-example.json
    ├── ExtractSchema
    │   ├── date-2018-03-29-namespace-example.schema
    ├── GenerateSHACL
    │   ├── date-2018-03-29-namespace-example.ttl
    └── GenerateView
        └── ShacShifter.html
```

Abbildung 6.1: Dateiablage der Tasks

7 Tests und Diskussion

Der in den vorhergehenden Kapiteln spezifizierte und implementierte Workflow muss auf seine Eignung untersucht werden. Dazu wird im folgenden Abschnitt mit Hilfe eines dokumentierten Testdurchlaufs die Eignung für unterschiedliche Ausgangsdaten diskutiert. Daten aus drei verschiedenen Quellen in unterschiedlichem Umfang werden mit dem Workflow verarbeitet. Im Anschluss wird die Möglichkeit beschrieben, über alle Quellen hinweg Abfragen zu formulieren, die den Forschenden in die Lage versetzen können, Datenquellen auch im Kontext zu anderen Datenquellen zu betrachten.

7.1 Testfälle

Den Forschenden wurden vor dieser Arbeit bereits Daten zur Verbreitung japanischer Videospiele aufbereitet und bereitgestellt. Aus diesem Anwendungsfall werden die bereits vorhandenen Daten für den Test des hier implementierten Workflows verwendet. Die Quelldaten stammen von den Webseiten Mobygames[1], Online Games Daten Bank (OGDB)[2] und GameFAQs[3]. Die Größe der Dateien und die Anzahl der Datensätze umfassen:

GameFAQs enthält 3.027 Datensätze (Umfang 4 MB) Auszug in Anlage A.1

OGDB enthält 26.071 Datensätze (Umfang 18 MB) Auszug in Anlage A.2

Mobygames enthält 10.000 Datensätze (Umfang 176 MB) Auszug in Anlage A.3

Die Tests fanden auf einer Workstation mit 16 GB RAM, Intel Core i7 3.10 GHz und Windows 10 statt. Für die Quellen GameFAQs und OGDB konnte der Workflow erfolgreich durchgeführt werden. Bei der Quelle Mobygames kam es Aufgrund der größeren

[1] `https://www.mobygames.com`
[2] `http://ogdb.eu`
[3] `https://gamefaqs.gamespot.com`

Datenmenge, im Vergleich zu den anderen Testdaten, zu Problemen. Die ursprüngliche Ausgangsdatei der Quelle Mobygames hatte eine Größe von 1,2 GB. Die Struktur konnte erfolgreich extrahiert und damit das Anwendungsprofil erstellt werden. Es zeigte sich aber, dass die Transformation der Daten von JSON zu RDF erhebliche Performanceprobleme verursachte. Daher wurde die Transformation der gesamten Datei nach fünf Stunden abgebrochen und über den Task `BuildSample` ein Datenauszug mit 10.000 zufällig gewählten Datensätzen erzeugt. Damit konnten erfolgreich RDF Daten erzeugt werden.

7.1.1 Manuelle Bearbeitung der SHACL Shapes

Sobald ein rudimentäres SHACL Shape für eine Quelle erzeugt wurde, kann dieses manuell bearbeitet und ergänzt werden. Je nach Quelle können unterschiedliche Bedingungen formuliert werden gegen die geprüft werden sollen. Für die in BA 3 auf S. 30 und BA 4 auf S. 31 geforderten Funktionen wurden exemplarisch folgende Ergänzungen in den SHACL Shapes vorgenommen:

sh:maxLength Definiert die maximale Anzahl von Zeichen in String-Werten.

sh:severity Der Schweregrad kann für einzelne Bedingungen festgelegt werden. Als Standardwert für Verletzungen wird `Error` verwendet. Daneben stehen in SHACL noch `Violation` und `Info` zur Verfügung.[4]

sh:message Kann für benutzerdefinierte Fehlermeldungen verwendet werden.

```
[ sh:datatype xsd:string ;
  sh:maxCount 1 ;
  sh:minCount 1 ;
  sh:maxLength 50 ;
  sh:message "title ist laenger als 50 Zeichen" ;
  sh:severity sh:Info ;
  sh:path gamefaqs:title ]
```

Listing 7.1: Beispiel für die Verwendung von sh:length, sh:message und sh:severity

sh:pattern Gibt einen regulären Ausdruck an, mit dem der Wert übereinstimmen muss. Sie sind nach der gleichen Art und Weise spezifiziert wie in SPARQL Regex Funktion.[5]

[4] Während der Tests berücksichtigte die verwendete RDFUnit Version 0.8.11-SNAPSHOT `sh:severity` nicht. Aus diesem Grund erscheint in den Prüfberichten an diesen Stellen der Standardwert Error

[5] `https://www.w3.org/TR/sparql11-query/#func-regex`

```
[ sh:datatype xsd:integer ;
  sh:maxCount 1 ;
  sh:minCount 1 ;
  sh:pattern "^\\d\\d\\d\\d$" ;
  sh:message "year nicht im Format YYYY" ;
  sh:path ogdb:year ]
```

Listing 7.2: Beispiel für die Verwendung von sh:pattern

sh:in Definiert eine Liste von Werten, die der Wert annehmen darf.

```
[ sh:datatype xsd:string ;
  sh:maxCount 1 ;
  sh:minCount 1 ;
  sh:message "Unerwarteter Wert in region " ;
  sh:in ("JP" "US" "EU" "AU") ;
  sh:path gamefaqs:region ]
```

Listing 7.3: Beispiel für die Verwendung von sh:in

sh:sparql Erweiterung um komplexe Einschränkungen mit Hilfe von SPARQL Queries zu definieren.

```
sh:sparql [
  sh:message "Spiel in JP nach 2015 veroeffentlicht" ;
  sh:prefixes [
    sh:declare [
      sh:prefix "gamefaqs" ;
      sh:namespace "http://diggr.link/data/gamefaqs/"^^xsd:anyURI ;
    ];
    sh:declare [
      sh:prefix "xsd" ;
      sh:namespace "http://www.w3.org/2001/XMLSchema#"^^xsd:anyURI ;
    ]
  ];
sh:select """
  SELECT $this
  WHERE {
    $this gamefaqs:region "JP" .
    $this gamefaqs:release_date ?date .
    BIND(xsd:date(SUBSTR(str(?date), 0, 11)) as ?releasedate) .
    BIND(YEAR(?releasedate) as ?year) .
    FILTER(?year > 2015).
```

```
    }
  """ ;
];
```

Listing 7.4: Definition eines constraints mit SHACL SPARQL Erweiterung

Die Datentypen in den automatisch erstellten Shapes können in dieser Form nicht zur Validierung eingesetzt werden, da bei der Transformation von JSON nach RDF diese Informationen nicht vorhanden sind. Aus dem JSON Schema heraus lassen sich nur bedingt diese Informationen entnehmen. JSON Schema bietet nur sechs primitive Datentypen (`string`, `number`, `array`, `boolean`, `null`, `object`) an. Weiterhin zeichnete das Programm GenSON in einigen Fällen den Datentyp integer fälschlicherweise als string aus. Um künftig auf Datentypen zu prüfen, muss vor der Datentransformation in den Quelldaten gekennzeichnet werden, um welchen Datentyp es sich handelt.

Eigenschaften, die mit `xsd:object` ausgezeichnet wurden, verweisen auf eine weitere Ressource. Um diese Verknüpfung zu prüfen wurden folgende Angaben ergänzt und `xsd:object` entfernt:

sh:nodeKind Spezifiziert, welche Art hinsichtlich des RDF Datenmodells, der Wert haben muss. Mögliche Werte sind unter anderem sh:IRI, sh:BlankNode, sh:Literal.

sh:class Definiert, dass der Wert eine Instanz einer bestimmten Klasse sein muss.

```
[ sh:minCount 1 ;
  sh:nodeKind sh:IRI ;
  sh:class ogdb:Developer ;
  sh:path ogdb:developer ]
```

Listing 7.5: Beispiel für die Verwendung von sh:nodekind und sh:class

Diese manuelle Bearbeitung, sollte durch die Berücksichtigung bei der Erzeugung der SHACL Shapes aus dem JSON Schema, zukünftig vermieden werden.

7.1.2 Validierung

Auch bei diesem Schritt kam es zu Performanceschwierigkeiten mit den Daten von Mobygames. Die Daten sind gegenüber den anderen Testdaten tiefer verschachtelt

und besitzen mehr Eigenschaften (siehe Anlage A.3). Daraus ergeben sich mehr Prüfungen, die auf die Daten angewendet werden müssen. RDFUnit erzeugte 250 Tests aus dem automatisch erzeugten SHACL Shape. Die damit durchgeführte Prüfung der Mobygames Daten erreichte die Speicherauslastung und brach mit der Fehlermeldung: `An exception occured while executing the Java class. Java heap space` ab. Der Test wurde noch einmal mit einer um die Hälfte (5.000 Objekte) reduzierten Datei durchgeführt. Ohne die manuelle Bearbeitung der SHACL Shapes wurden von RDFUnit für die Quellen folgende Ergebnisse ausgegeben:

Quelle	Anzahl Tests	Erfolgreich	Fehlgeschlagen	Instanzen mit Verletzungen
GameFAQs	61	54	7	23.524
OGDB	62	51	11	91.038
Mobygames (10.000 Datensätze)	250	k.A.	k.A.	k.A.
Mobygames (5.000 Datensätze)	250	199	51	3.306.579

Tabelle 7.1: Anzahl ausgewerteter Tests und entdeckte Verstöße gegen Bedingungen pro Datenquelle

Wie im vorhergehenden Abschnitt erwähnt, müssen die Datentypen aus dem JSON Schema in die Transformation nach RDF übernommen werden. Bei der vorliegenden Implementierung ist dies noch nicht der Fall. Dieser Umstand macht den Großteil der Fehler aus. Die Definitionen wurden im Zuge der manuellen Bearbeitung der Shapes korrigiert und mit den in Abschn. 7.1.1 auf S. 60 ergänzt. Die Validierung wurde mit den bearbeiteten Shapes noch einmal durchgeführt und erzielte folgende Ergebnisse:

Quelle	Anzahl Tests	Erfolgreich	Fehlgeschlagen	Instanzen mit Verletzungen
GameFAQs	69	62	7	4.783
OGDB	67	58	9	33.096
Mobygames (5.000 Datensätze)	268	235	33	2.955.346

Tabelle 7.2: Anzahl ausgewerteter Tests und entdeckte Verstöße gegen Bedingungen pro Datenquelle

Auszüge aus den Prüfberichten befinden sich in Anhang B.

7.1.3 Browsing in den Daten

Um eine einfache Datenansicht zu erzeugen wird Jekyll-RDF eingesetzt. Dafür wurde exemplarisch ein Datenset mit `BuildSample`, das 200 Datenobjekt der Quelle GameFAQs enthält, erstellt. Anschließend konnte mit dem zuvor erstellten SHACL Shape und dem Task `Validate` das Template für Jekyll-RDF erzeugt werden. Die Screenshots in Anhang C zeigen exemplarisch die Präsentation der Daten als statische HTML Seite. Zwischen den einzelnen Ressourcen bzw. Datenobjekten werden Links zur Navigation erstellt.

7.2 Mapping auf Vokabularebene

Die Integration der Daten in eine Datenbank lässt sich durch den Einsatz von RDF problemlos lösen. Aufgrund unterschiedlicher Namensräume ist es möglich verschiedene Datenquellen in Form von Graphen in einer RDF Datenbank zu speichern. Dabei können die Daten sowohl getrennt als auch gemeinsam abgefragt werden. Dafür muss das Vokabular und die Struktur der Daten bekannt sein. Beides kann durch den erstellten Workflow bereitgestellt werden. Für Abfragen über mehrere Quellen hinweg kann auf dieser Grundlage zukünftig ein übergreifendes Vokabular für die Domäne der Videospiele entwickelt werden, welches die einzelnen Quellvokabulare miteinander verbindet. Dieser Prozess kann durch die Anwendung geeigneter Ontology Matching Verfahren, wie in [ES13] beschrieben, unterstützt werden.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf—schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix diggr: <http://vocab.ub.uni—leipzig.de/diggr/> .
@prefix gameFAQs: <http://diggr.link/data/gameFAQs/> .
@prefix mobygames: <http://diggr.link/data/mobygames/> .
@prefix ogdb: <http://diggr.link/data/ogdb/> .

diggr:title a owl:property;
        rdfs:subPropertyOf mobygames:title;
        rdfs:subPropertyOf gamefaqs:title;
        rdfs:subPropertyOf ogdb:name .
```

Listing 7.6: Beispiel für ein gemeinsames Vokabular

8 Zusammenfassung und Ausblick

Der Workflow zur Datenintegration im Projekt diggr ist durch nicht-formalisierte Prüfungen der Daten gekennzeichnet (vgl. Abschn. 3.2 auf S. 21). Daher hat sich diese Arbeit zum Ziel gesetzt, den Workflow hinsichtlich Wiederholbarkeit und Automatisierung zu verbessern (vgl. Abschn. 1.3 auf S. 9). Dafür wird der Workflow durch das Workflow Management System Luigi unterstützt, welches die Verarbeitungsschritte koordiniert und eine teilweise Automatisierung ermöglicht. Darüber hinaus wurde eine Verbesserung des Workflows hinsichtlich Nachvollziehbarkeit und Transparenz verfolgt. Zum Erreichen dieser Ziele kam unter anderem die für Menschen lesbare Schemasprache SHACL für die Prüfung der RDF Daten zum Einsatz. Die in dieser Arbeit entwickelte Lösung ist flexibel und bietet viele Erweiterungsmöglichkeiten.

Zur Realisierung des Workflows wurden Anwendungsfälle beschrieben und die daraus resultierenden Anforderungen formuliert und zusammengetragen (vgl. Kap. 4 auf S. 24). Sie umfassen die wichtigsten Funktionen und Schritte. Auf Basis der Anforderungen wurde ein Entwurf des Systems erarbeitet (vgl. Kap. 5 auf S. 36). Es setzt zur Abarbeitung der einzelnen Teilschritte auf das Workflow Management System Luigi (vgl. Abschn. 5.2 auf S. 37). Für die einzelnen Verarbeitungsschritte werden spezielle Programme und Bibliotheken verwendet (vgl. Abschn. 6.1 auf S. 47). Gemäß der in Abschn. 5.2.2 auf S. 44 definierten Verarbeitungsschritte wurde das System umgesetzt (vgl. Kap. 6 auf S. 47). Es wurde untersucht, ob das System den zuvor beschriebenen Anforderungen entspricht und generisch genug für die Verarbeitung verschiedener Datenstrukturen aus unterschiedlichen Quellen ist (vgl. Kap. 7 auf S. 59). Dazu wurden drei Datensets mit dem Workflow verarbeitet.

Das entstandene System bietet Funktionen, um Daten mit heterogenen Datenschemata nach RDF zu überführen (BA 1 auf S. 29) und sie auf bestimmte Merkmale zu prüfen (BA 3 auf S. 30). Ergebnisse dieser Prüfung werden als HTML Bericht gespeichert und sind damit einfach zugänglich (BA 4 auf S. 31). Strukturelle Informationen aus den Datenquellen werden sowohl als JSON Schema als auch in Form von SHACL Shapes gespeichert (BA 2 auf S. 30). Während der Tests zeigte sich, dass die automatische Erstellung der SHACL Shapes ergänzt werden sollte.

Die erstellten SHACL Shapes bilden die Basis für weitere Tests und für die Erzeugung eines Templates für eine einfache HTML Ansicht der Daten (BA 5 auf S. 31). In dieser Arbeit wurde ein konkretes Anwendungsbeispiel für SHACL Shapes präsentiert. Dieses deckte die Benutzeranforderung 3: „Datenvalidierung“ (S. 30) ab. Durch den Einsatz von SHACL Shapes setzt der gesamte Workflow RDF als primäres Datenformat zur Verarbeitung, Überprüfung und Ausgabe der Daten ein (NF 3 auf S. 33).

Es zeigte sich, dass es bei einzelnen Komponenten zu Performanceproblemen kommt, insbesondere bei der Transformation der Daten von JSON zu RDF und bei der Prüfung der RDF Daten mit dem Programm RDFUnit. Die prototypische Implementierung ist nicht geeignet um JSON Dateien, die größer als 200 MB sind, zu verarbeiten. Damit ist die Anforderung NF 4 auf S. 33 nur bedingt erfüllt. Das System soll zukünftig im Projektrahmen von diggr eingesetzt werden. Dafür muss die Implementierung an diesen Stellen überarbeitet werden. Aufgrund der flexiblen Architektur (NF 2 auf S. 33) können jedoch Verarbeitungsschritte leicht verändert und einzelne externe Komponenten ausgetauscht werden (NF 1 auf S. 32).

Die prototypische Implementierung bietet bereits eine übersichtlichere Abarbeitung des Datenintegrationsprozess (BA 6 auf S. 32) und ist damit transparenter für alle am Workflow beteiligten Akteure. Der Workflow unterstützt damit gezielt den Forschungsprozess und hilft den Forschenden bei der Einschätzung und späteren Interpretation der Daten.

8.1 Erweiterungsmöglichkeiten

Während des Entwurfs, der Entwicklung und der Tests haben sich einige Möglichkeiten gezeigt, durch welche Verarbeitungsschritte das System erweitert werden könnte. In den folgenden Abschnitten werden diese kurz skizziert.

Automatisierte Korrekturen

Eine Komponente könnte auf Grundlage der erzeugten Prüfberichte automatisiert Fehlerkorrekturen durchführen. Dies müsste in Absprache mit den Forschenden geschehen, die entscheiden müssen, in wie weit bestimmte Manipulationen der Daten erwünscht sind. Insbesondere bei der Aktualisierung von Daten könnten Fehler, die bereits bekannt sind, beim Import der Daten automatisiert manipuliert werden.

Kennzeichnung von Provenance

Während der Bearbeitungszeit ist im Projekt diggr das Python Paket provit[1] zur Nachverfolgung der Herkunft von Daten (engl. *provenance*) entstanden. Es nimmt Informationen, beispielsweise den Prozessnamen und den ausführenden (Software-)Agenten für einen bestimmten Verarbeitungsprozess auf und produziert für jede Ergebnisdatei eine strukturierte Textdatei mit den entsprechenden Provenance Informationen. Es verwendet dafür einen Ausschnitt des PROV-O[2] Vokabulars. Die Einbindung dieses Programms in den Workflow könnte die Nachvollziehbarkeit weiter steigern.

Integration Stichproben

Der Task `BuildSample` wurde zunächst aus pragmatischen Gründen implementiert. Er könnte in Zukunft in den gesamten Workflow integriert werden. Es gibt Szenarien, in denen eine Verarbeitung eines gesamten Datensets nicht nötig ist.

[1] `https://pypi.org/project/provit/`
[2] `https://www.w3.org/TR/prov-o/`

9 Quellcode

Der Quellcode zu dem beschriebenen System befindet sich in einer virtuellen Maschine, die mit dieser Arbeit eingereicht wird. Sowohl das System selbst, als auch die notwendigen Programme und Bibliotheken sind darauf installiert. Eine README.md auf dem Desktop beschreibt die wichtigsten Schritte, um das hier beschriebene System nachzuvollziehen.

Literaturverzeichnis

[Bal09] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements-Engineering*. 3. Aufl. Heidelberg : Spektrum Akad. Verl., 2009 (Lehrbücher der Informatik). – ISBN 978–3–8274–1705–3

[Bau13] Bauer, Andreas (Hrsg.): *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung*. 4., überarb. und erw. Aufl. Heidelberg : dpunkt-Verl., 2013. – ISBN 3–89864–785–4

[CLA] CLARIN-D AP 5: *CLARIN-D User Guide*. http://media.dwds.de/clarin/userguide/userguide-1.0.1.pdf

[CWL14] Cyganiak, Richard ; Wood, David ; Lanthaler, Markus: RDF 1.1 Concepts and Abstract Syntax / W3C. 2014. – W3C Recommendation. – http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[Czy] Czygan, Martin: *Design and Implementation of a Library Metadata Management Framework and its Application in Fuzzy Data Deduplication and Data Reconciliation with Authority Data*. http://dx.doi.org/10.1787/578054332028. http://dx.doi.org/10.1787/578054332028

[DAR15] *Handbuch Digital Humanities: [Anwendungen, Forschungsdaten und Projekte]*. Göttingen : DARIAH-DE, 2015 https://handbuch.tib.eu/w/images/2/2c/DH-Handbuch.pdf. – ISBN 9783737568180

[ES13] Euzenat, Jérôme ; Shvaiko, Pavel: *Ontology matching*. 2. ed. Berlin u.a. : Springer, 2013. – ISBN 3–642–38720–9

[Eve15] Evelyn Dröge: RDF Application Profiles und RDF Data Shapes: RDF-Anwendungsprofile oder: Wie validiere ich meine Ontologie? In: *Password* (2015), Nr. 7-8, S. 20. – ISSN 0930–3693

[GPBK17] Gayo, Jose Emilio L. ; Prud'hommeaux, Eric ; Boneva, Iovka ; Kontokostas, Dimitris: Validating RDF Data. In: *Synthesis Lectures on the Semantic Web: Theory and Technology* 7 (2017), Nr. 1, S. 1–328. http://dx.doi.org/10.2200/S00786ED1V01Y201707WBE016. – DOI 10.2200/S00786ED1V01Y201707WBE016. – ISSN 2160–4711

[HFM17] Hoffmann, Tracy ; Freybe, Konstantin ; Mühleder, Peter: Workflows zur datenbasierten Videospielforschung - Am Beispiel der populären Videospielserie Metal Gear Solid. In: Eibl, Maximilian (Hrsg.) ; Gaedke, Martin (Hrsg.): *INFORMATIK 2017*, Gesellschaft für Informatik, Bonn, 2017. – ISBN 978–3–88579–669–5, S. 1113–1124

[Hit08] Hitzler, Pascal: *Semantic Web: Grundlagen*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2008 (eXamen.press). http://dx.doi.org/10.1007/978-3-540-33994-6. http://dx.doi.org/10.1007/978-3-540-33994-6. – ISBN 9783540339946

[KE17] Kuras, Christoph ; Eckar, Thomas: Prozessmodellierung mittels BPMN in Forschungsinfrastrukturen der Digital Humanities. In: Eibl, Maximilian (Hrsg.) ; Gaedke, Martin (Hrsg.): *INFORMATIK 2017*, Gesellschaft für Informatik, Bonn, 2017. – ISBN 978–3–88579–669–5, S. 1101–1112

[KK17] Knublauch, Holger ; Kontokostas, Dimitris: Shapes Constraint Language (SHACL) / W3C. 2017. – W3C Recommendation. – https://www.w3.org/TR/2017/REC-shacl-20170720/

[LGPSB] Labra-Gayo, Jose-Emilio ; Prud'hommeaux, Eric ; Solbrig, Harold ; Boneva, Iovka: *Validating and describing linked data portals using shapes*. http://arxiv.org/pdf/1701.08924v1

[LPB18] Labra Gayo, José E. ; Prud'hommeaux, Eric ; Boneva, Iovka: *Validating RDF data*. 2018 (Synthesis lectures on the semantic web: theory and technology). – ISBN 9781681731667

[Mor10] Morgenstern, Ulf (Hrsg.): *Leipziger Beiträge zur Informatik*. Bd. 21: *Catalogus Professorium Lipsiensis: Konzeption, technische Umsetzung und Anwendungen für Professorenkataloge im Semantic Web*. Leipzig : LIV, 2010. – ISBN 978–3–941608–08–5

[MYB+08] Maler, Eve ; Yergeau, François ; Bray, Tim ; Sperberg-McQueen, Michael ; Paoli, Jean: Extensible Markup Language (XML) 1.0 (Fifth Edition) /

W3C. 2008. – W3C Recommendation. – http://www.w3.org/TR/2008/REC-xml-20081126/

[RB16] Riechert, Thomas ; Beretta, Francesco: Collaborative Research on Academic History using Linked Open Data: A Proposal for the Heloise Common Research Model. In: *CIAN-Revista de Historia de las Universidades* 19 (2016), Nr. 0. http://e-revistas.uc3m.es/index.php/CIAN/article/view/3147. – ISSN 1988– 8503

[RB17] Riechert, Thomas ; Beretta, Francesco: Kollaborative Forschung über Linked Open Data Forschungsdatenbanken der Universitätsgeschichte - Implementierung des Heloise Common Research Model. In: *Abstraktband der Jahreskonferenz des Digital Humanities im deutschsprachigen Raum (DHd) e. V. Bern 2017*, 2017

[RS14] Raimond, Yves ; Schreiber, Guus: RDF 1.1 Primer / W3C. 2014. – W3C Note. – http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/

[SC17] Steyskal, Simon ; Coyle, Karen: SHACL Use Cases and Requirements / W3C. 2017. – W3C Note. – https://www.w3.org/TR/2017/NOTE-shacl-ucr-20170720/

[Tim06] Tim Berners-Lee: *Linked-data design issues: W3C design issue document.* http://www.w3.org/DesignIssues/LinkedData.html. Version: July 2006

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

API Application Programming Interface

DH Digital Humanities

HCRM Heloise Common Research Model

ETL Extract Transform Load

DDL Data Defintion Language

HTML Hypertext Markup Language

XML Extensible Markup Language

RDF Resource Description Framework

JSON JavaScript Object Notation

JSON-LD JavaScript Object Notation for Linked Data

URI Uniform Resource Identifier

ShEx Shape Expressions

SHACL Shapes Constraint Language

SPARQL SPARQL Protocol and RDF Query Language

W3C World Wide Web Consortium

A

A.1 Beispieldaten GameFAQs


```
{
    "retrieved": "2017-10-13T05:51:57.174974",
    "data": {
        "wikipedia": null,
        "esrb": null,
        "developer": {
            "name": "Daedalic Entertainment",
            "slug": "/company/78697-daedalic-entertainment"
        },
        "releases": [
            {
                "region": "EU",
                "rating": "7+",
                "product_id": "",
                "publisher": {
                    "name": "Daedalic Entertainment",
                    "slug": "/company/78697-daedalic-entertainment"
                },
                "title": "AER",
                "release_date": "2017-10-25T00:00:00",
                "dist_barcode": ""
            },
            {
                "region": "US",
                "rating": "",
                "product_id": "",
                "publisher": {
                    "name": "Daedalic Entertainment",
                    "slug": "/company/78697-daedalic-entertainment"
```

```
                    },
                    "title": "AER",
                    "release_date": "TBA",
                    "dist_barcode": "PlayStation Store PS4"
                }
            ],
            "genre": "Action Adventure > General",
            "title": "AER",
            "slug": "/ps4/117418-aer"
        },
        "_id": "ps4/117418-aer"
    }
```


Listing A.1: Datenauszug GameFAQs

A.2 Beispieldaten OGDB


```
{
        "year": 2009,
        "developer": [
            {
                "companyId": 864,
                "alt_names": [
                    "Namco Bandai Games, Inc.",
                    "株式会社バンダイナムコゲームス"
                ],
                "CompanyName": "Bandai Namco Games, Inc.",
                "countryId": 1
            }
        ],
        "name_variants": [
            "Super Pac-Man",
            "スーパーパックマン"
        ],
        "name": "Super Pac-Man",
        "month": 5,
        "region_id": 2,
        "platform": "Microsoft Xbox 360",
        "day": 15,
```

```
        "publisher": [
            {
                "companyId": 5621,
                "alt_names": [
                    "NBGE"
                ],
                "CompanyName": "Namco Bandai Games Europe S.A.S.",
                "countryId": 18
            }
        ],
        "title_id": 14178,
        "distributor": [
            {
                "companyId": 5621,
                "alt_names": [
                    "NBGE"
                ],
                "CompanyName": "Namco Bandai Games Europe S.A.S.",
                "countryId": 18
            }
        ]
    }
```

Listing A.2: Datenauszug OGDB

A.3 Beispieldaten Mobygames

```
{
"_id": "344",
"retrieved": "2017-10-25T09:06:15.260479",
"url": "http://www.mobygames.com/game/mdk",
"_rev": "1-b3a56b274971bd2f9b0ef1ba6ef3d811",
"title": "MDK",
"platforms": [
    {
        "game_id": 344,
        "hints": [
            {
```

```
            "hint": "<pre>\r\nineedabiggun ── Super Chain Gun powerup ↲
              once per level [...]",
            "title": "MDK Cheats"
        }
    ],
    "asins": [],
    "patches": [],
    "platform_name": "DOS",
    "credits": [
        {
            "credits": [
                {
                    "name": "Andrew Brown",
                    "developer_id": 20295
                }
            ],
            "role_type": "Assistant Producer",
            "role": "Associate Producer",
            "role_type_id": 59
        }
    ],
    "ratings": [
        {
            "rating_id": 435,
            "rating_system_id": 33,
            "rating_name": "16",
            "rating_system_name": "USK Rating"
        }
    ],
    "critic_scores": [
        {
            "url": "http://gamesmania.de/pc/action/mdk/1655/gamespace/↲
              test/11207/",
            "language": "German",
            "review_date": "1997",
            "source": "Gamesmania.de",
            "excerpt": "Ein Feuerwerk an Ideen mit grandioser Grafik und↲
               genialem Leveldesign.[...]",
            "score": 90
        }
    ],
```

```
        "attributes": [
            {
                "attribute_category_name": "Sound Devices Supported",
                "attribute_id": 17,
                "attribute_category_id": 1,
                "attribute_name": "Sound Blaster"
            },
            {
                "attribute_category_name": "Sound Devices Supported",
                "attribute_id": 20,
                "attribute_category_id": 1,
                "attribute_name": "Gravis Ultrasound / ACE"
            }
        ],
        "user_reviews": [
            {
                "the_good": "Shiny! What's not to like? This is probably the↲
                   weirdest shooter I've EVER played.[...]",
                "reviewer_url": "http://www.mobygames.com/user/sheet/↲
                  userSheetId,60/",
                "review_date": "1999-11-03",
                "headline": "One of the best games ever.",
                "the_bad": "Zero replayability, which is unfortunate since ↲
                  this game is really, really good.\r\n",
                "rating": 5,
                "reviewer": "Holograph",
                "the_bottom_line": "Am amazing shooter which is without a ↲
                  shadow of a doubt a classic.\r\n"
            }
        ],
        "screenshots": [
            {
                "width": 640,
                "height": 480,
                "caption": "Title Screen",
                "image": "http://www.mobygames.com/images/shots/l/172174-mdk↲
                  -dos-screenshot-title-screen.png",
                "thumbnail_image": "http://www.mobygames.com/images/shots/s↲
                  /172174-mdk-dos-screenshot-title-screen.jpg"
            }
        ],
```

```
        "platform_id": 2,
        "releases": [
            {
                "description": null,
                "companies": [
                    {
                        "company_name": "Shiny Entertainment, Inc.",
                        "company_id": 212,
                        "role": "Developed by"
                    }
                ],
                "product_codes": [],
                "countries": [
                    "United States"
                ],
                "release_date": "1997"
            }
            {
                "description": null,
                "companies": [
                    {
                        "company_name": "Shiny Entertainment, Inc.",
                        "company_id": 212,
                        "role": "Published by"
                    }
                ],
                "product_codes": [
                    {
                        "product_code": "5026102141781",
                        "product_code_type_id": 2,
                        "product_code_type": "EAN-13"
                    }
                ],
                "countries": [
                    "Germany"
                ],
                "release_date": "1997"
            }
        ],
        "first_release_date": "1997",
        "cover_groups": [ ...]
```

```
    }
],
"alternative_titles": [
    {
        "description": "Chinese spelling (simplified)",
        "title": "孤胆⊠手"
    },
    {
        "description": "Chinese spelling (traditional)",
        "title": "亡命暗殺令"
    },
    {
        "description": "Development title",
        "title": "Murder Death Kill"
    }
],
"source": "https://www.mobygames.com/",
"genres": [
    {
        "genre_id": 8,
        "genre_name": "Sci—Fi / Futuristic",
        "genre_category": "Setting",
        "genre_category_id": 10
    },
    {
        "genre_id": 16,
        "genre_name": "3rd—person [DEPRECATED]",
        "genre_category": "Perspective",
        "genre_category_id": 2
    }
]
}
```


Listing A.3: Gekürzter Datenauszug Mobygames

B

B.1 Screenshots Prüfberichte

TestExecution: urn:uuid:886afc2c-2bb2-11b2-80c4-00155da1c650

Dataset	/mnt/c/Users/HoffmannT/workspace/jgames/th-master-shacl/luigi/Master/BuildRDFData/date-2018-04-21-namespace-gamefaqs.ttl
Test execution started	2018-05-08T08:35:31.107Z
-ended	2018-05-08T08:35:42.634Z
Total test cases	69
Succeeded	62
Failed	7
Timeout / Error	T:0 / E: 0
Violation instances	4783

Results

Status	Level	Test Case	Errors	Prevalence
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/gamefaqs/wikipedia> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	1222	3027
Fail	ERROR	Unerwarteter Wert in region	305	12035
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/gamefaqs/developer> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	10	3027
Fail	ERROR	title ist laenger als 50 Zeichen	135	3027
Fail	ERROR	Spiel in JP nach 2015 veroeffentlicht	1286	12035
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/gamefaqs/esrb> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	1186	3027
Fail	ERROR	release_date nicht im Format YYYY-MM-DDT00:00:00	639	12035
Success	ERROR	value not of datatype <http://www.w3.org/2001/XMLSchema#string>	0	3027
Success	ERROR	Maximum cardinality for <http://diggr.link/data/gamefaqs/wikipedia> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	3027
Success	ERROR	Unerwarteter Wert in region	0	12035
Success	ERROR	Unerwarteter Wert in region	0	12035
Success	ERROR	Unerwarteter Wert in region	0	12035
Success	ERROR	value not of datatype <http://www.w3.org/2001/XMLSchema#string>	0	3027
Success	ERROR	Minimum cardinality for <http://diggr.link/data/gamefaqs/_id> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	3027
Success	ERROR	Maximum cardinality for <http://diggr.link/data/gamefaqs/_id> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	3027
Success	ERROR	sh:nodeKind should be <http://www.w3.org/ns/shacl#IRI>	0	3027
Success	ERROR	value not of type <http://diggr.link/data/gamefaqs/Developer>	0	3027

Abbildung B.1: Prüfbericht GameFAQs nach Anpassung der SHACL Shapes

TestExecution: urn:uuid:88502f10-2bb2-11b2-802c-00155da1c650

Dataset	/mnt/c/Users/HoffmannT/workspace/jgames/th-master-shacl/luigi/Master/BuildRDFData/date-2018-04-21-namespace-ogdb.ttl
Test execution started	2018-05-08T08:32:35.542Z
-ended	2018-05-08T08:33:05.781Z
Total test cases	67
Succeeded	58
Failed	9
Timeout / Error	T:0 / E: 0
Violation instances	33096

Results

Status	Level	Test Case	Errors	Prevalence
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/day> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	286	26071
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/publisher> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	63	26071
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/month> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	286	26071
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/alt_names> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	4730	29267
Fail	ERROR	name ist laenger als 50 Zeichen	1486	26071
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/alt_names> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	2693	8758
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/alt_names> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	4830	26162
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/distributor> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	17893	26071
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/developer> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	829	26071
Success	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/companyId> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	26162
Success	ERROR	Maximum cardinality for <http://diggr.link/data/ogdb/countryId> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	8758
Success	ERROR	sh:nodeKind should be <http://www.w3.org/ns/shacl#IRI>	0	26071
Success	ERROR	value not of datatype <http://www.w3.org/2001/XMLSchema#string>	0	26071
Success	ERROR	Minimum cardinality for <http://diggr.link/data/ogdb/name_variants> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	26071
Success	ERROR	Maximum cardinality for <http://diggr.link/data/ogdb/companyId> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	0	8758

Abbildung B.2: Prüfbericht OGDB nach Anpassung der SHACL Shapes

TestExecution: urn:uuid:87843eca-2bb2-11b2-802e-00155da1c650

Dataset	/mnt/c/Users/HoffmannT/workspace/jgames/th-master-shacl/luigi/Master/BuildRDFData/date-2018-04-21-namespace-mobygames.ttl
Test execution started	2018-05-08T08:10:18.916Z
-ended	2018-05-08T08:25:05.066Z
Total test cases	268
Succeeded	235
Failed	33
Timeout / Error	T:0 / E: 0
Violation instances	2955346

Results

Status	Level	Test Case	Errors	Prevalence
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/credits> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	12437	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/cover_groups> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	6794	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/score> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	441	49391
Fail	ERROR	value not of datatype <http://www.w3.org/2001/XMLSchema#number>	48950	49391
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/patches> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	19269	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/first_release_date> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	1	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/role> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	510063	698205
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/name> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	188142	698205
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/comments> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	40044	57334
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/role_type_id> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	579150	698205
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/url> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	9034	49391
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/ratings> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	14237	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/alternative_titles> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	7018	10000
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/hints> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	18885	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/platforms> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	3	10000
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/screenshots> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	11236	19594
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/credits> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	510063	698205
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/description> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	16852	28289
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/description> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	56354	57334
Fail	ERROR	Minimum cardinality for <http://diggr.link/data/mobygames/release_date> is "1"^^<http://www.w3.org/2001/XMLSchema#integer>	1	28289

Abbildung B.3: Prüfbericht Mobygames nach Anpassung der SHACL Shapes

C

C.1 Screenshots mit Jekyll-RDF erzeugte HTML Ansicht

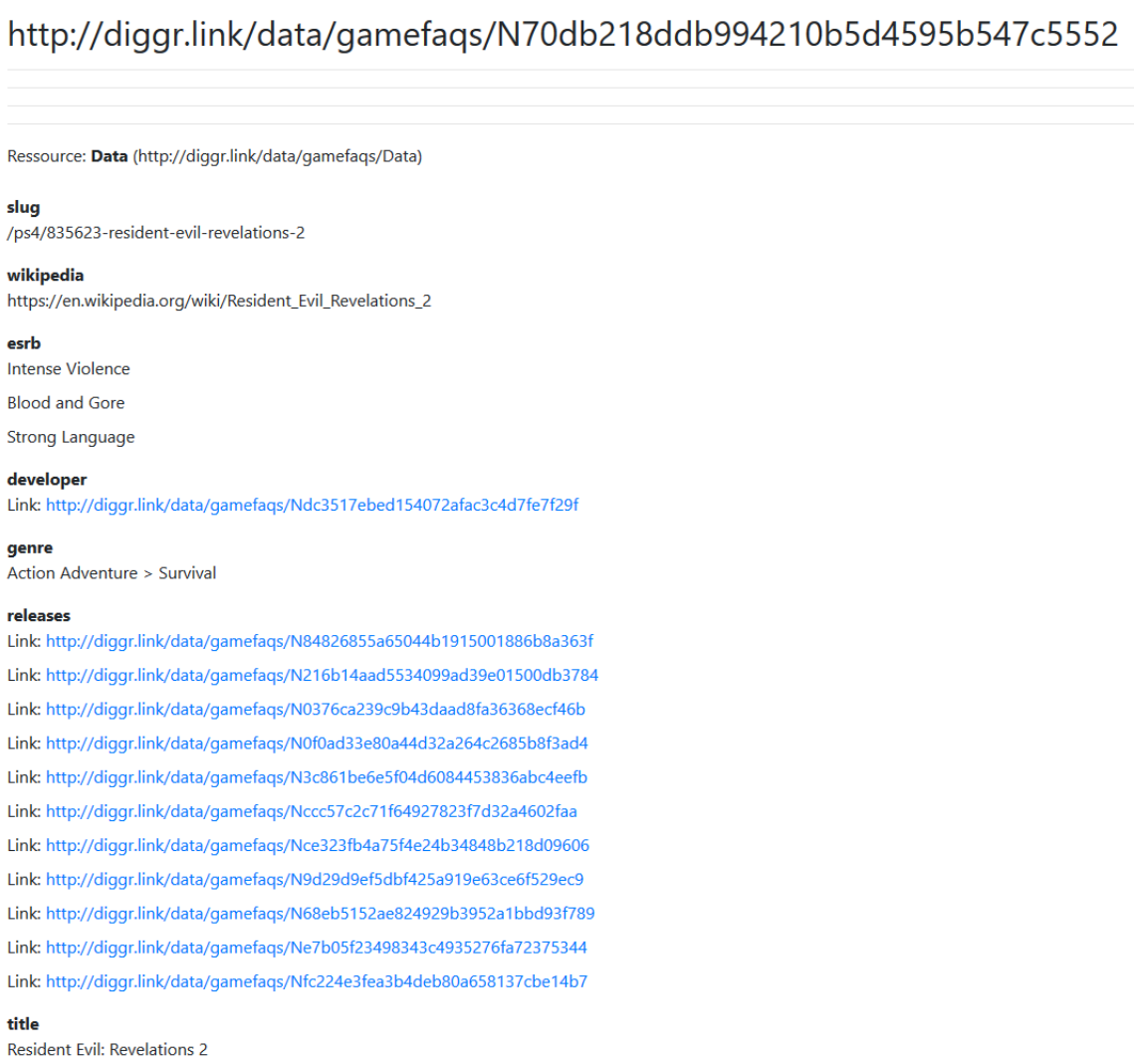

Abbildung C.1: HTML Seite einer Ressource der Klasse gamefaqs:Data mit Links zu weiteren Ressourcen

http://diggr.link/data/gamefaqs/N84826855a65044b1915001886b8a363f

Ressource: **Releases** (http://diggr.link/data/gamefaqs/Releases)

dist_barcode
4976219077095

publisher
Link: http://diggr.link/data/gamefaqs/N20051ef49dbe47f09e06845d1ce1b47c

rating
D

region
JP

release_date
2016-08-04T00:00:00

title
Resident Evil: Revelations 2 (Best Price)

product_id
PLJM-80175

Abbildung C.2: HTML Seite einer Ressource der Klasse gamefaqs:Release

http://diggr.link/data/gamefaqs/N20051ef49dbe47f09e06845d1ce1b47c

Ressource: **Publisher** (http://diggr.link/data/gamefaqs/Publisher)

name
Capcom

slug
/company/2324-capcom

Abbildung C.3: HTML Seite einer Ressource der Klasse gamefaqs:Publisher

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 8. Mai 2018 Unterschrift