Nachhaltige Infrastruktur zur Forschungsdatenpublikation am Beispiel von Hochdurchsatz-Pflanzenphänotypisierungsdaten

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Bioinf. Daniel Arend

geb. am 09.06.1986 in Wippra

Gutachterinnen/Gutachter

Prof. Dr. Gunter Saake
Prof. Dr. Björn Usadel
Dr. Matthias Lange

Magdeburg, den 08.08.2018

Arend, Daniel:
Nachhaltige Infrastruktur zur Forschungsdatenpublikation am Beispiel von Hochdurchsatz-Pflanzenphänotypisierungsdaten
Dissertation, Otto-von-Guericke-Universität Magdeburg, 2018.

Kurzfassung

Der schnelle technische Fortschritt hat unter anderem in den Lebenswissenschaften zu einem sehr großen Anstieg an verfügbaren Forschungsdaten geführt. Die Herausforderungen des „big data“-Zeitalters sind allgegenwärtig. Neben den neuen, vielfältigen Möglichkeiten, die diese Flut an Daten bietet, stellt sie auch neue Anforderungen an vorhandene Infrastrukturen und etablierte Verfahren zum Datenmanagement und zur Datenpublikation.

In einigen Forschungsdomänen, z. B. im Bereich der Genomanalyse gibt es bereits seit vielen Jahren allgemein akzeptierte Datenbanken zum Austausch von Forschungsdaten. Diese müssen jedoch mit immer größer werdenden Datenbeständen zurechtkommen und gleichzeitig eine effektive Nachnutzung der Daten gewährleisten. Andere, vor allem noch junge Forschungsbereiche, z. B. Pflanzenphänotypisierung, sind darüber hinaus mit der Herausforderung konfrontiert, dass es in ihrem Feld kaum etablierte Infrastrukturen zum Datenmanagement gibt und das die starke Heterogenität der produzierten Forschungsdaten die Entwicklung neuer Informationssysteme erschwert. Aber nicht nur der Austausch von Forschungsdaten, auch die effektive und nachhaltige Speicherung und Publikation stellt eine neue Aufgabe dar. Forschungsdaten wurden bisher in reduzierter Form häufig als Supplement einer zugehörigen wissenschaftlichen Publikation veröffentlicht. Obwohl diese Daten meist nur im Bereich von wenigen Megabyte lagen, war die dauerhafte Bereitstellung auch nach Veröffentlichung aufgrund fehlender institutioneller Infrastrukturen bereits eine große Herausforderung. Heutzutage umfassen viele Forschungsdaten jedoch bereits mehrere Gigabyte bis Terabyte, sodass viele Institutionen mit dieser Aufgabe mittlerweile überfordert sind.

In dieser Arbeit wird eine nachhaltige und generische Infrastruktur zum Forschungsdatenmanagement sowie das darin enthaltene Konzept zur Begutachtung und Publikation von Forschungsdaten vorgestellt. Um eine effektive Infrastruktur zu entwickeln, wurden die Anforderungen in der Verarbeitung und Publikation ebenso wie verbreitete Ansätze und Systeme zur Veröffentlichung von Forschungsdaten untersucht. Durch die Verwendung moderner Konzepte der Informatik sowie aktueller Technologien und Softwareframeworks wurde eine dynamische und skalierbare Implementierung realisiert, welche die Schwachstellen existierender Systeme überwindet. Diese folgt dem „Bringing the infrastructure to the data“-Ansatz und ermöglicht die einfache und flexible Einrichtung einer institutionellen Infrastruktur, welche die vorhandene Speicherkapazität zur effektiven Verwaltung und Veröffentlichung von Forschungsdaten nutzen kann. Eine erste produktive Instanz der Infrastruktur wurde zur Publikation von pflanzenphänotypischen Hochdurchsatzdaten am Leibniz-Institut für Pflanzengenetik und Kulturpflanzenforschung (IPK) in Gatersleben erfolgreich etabliert. Sie enthält bereits über 150 verschiedene Forschungsdatensätze und verarbeitet mehrere 1000 Zugriffe pro Monat. Als ein Service im Portofolie der de.NBI-Infrastruktur ist ihre langfristige Bereitstellung gewährleistet. Darüber hinaus konnte Mitte 2018 im Rahmen des DPPN-Projektes eine weitere Instanz auf Basis der entwickelten Infrastruktur am Forschungszentrum Jülich etabliert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Verzeichnis der Abkürzungen

AAI	Authentication and Authorization Infrastructure ... 87
ACID	Atomicity-Consistency-Isolation-Durability ... 21
AOP	aspektorientierte Programmierung ... 23
API	Application Programming Interface ... 16
AWI	Alfred-Wegener-Institut für Polar- und Meeresforschung ... 15
BMBF	Bundesministerium für Bildung und Forschung ... 10
DBMS	Datenbankmanagementsystem ... 20
DCMES	DublinCore Metadata Element Set ... 17
de.NBI	German Network for Bioinformatics Infrastructure ... 3
DFG	Deutsche Forschungsgemeinschaft ... 10
DPPN	Deutsches Pflanzen Phänotypisierungsnetzwerk ... 3
DOI	Digital Object Identifier ... 15
eDAL	electronic Data Archiving Library ... 56
EMBL-EBI	European Bioinformatics Institute ... 2
FAIR	Findable, Accessible, Interoperable, Re-Usable ... 11
FEDORA	Flexible Extensible Digital Object Repository Architecture ... 47
FZJ	Forschungszentrum Jülich ... 3
GCBN	German Crop BioGreenformatics Network ... 3
HDFS	Hadoop Distributed File System ... 122
HQL	Hibernate Query Language ... 22
HSM	hierarchisches Speichermanagement ... 113
HTML	Hypertext Markup Language ... 35
HTTP	Hypertext Transfer Protocol ... 19
HTTPS	Hypertext Transfer Protocol Secure ... 79
IANA	Internet Assigned Numbers Authority ... 16
ID	Identifikator ... 15
IDF	International DOI Foundation ... 15
IDE	Integrated Development Environment ... 90
IPK	Leibniz-Institut für Pflanzengenetik und Kulturpflanzenforschung ... 3
JAAS	Java Authentication and Authorization Service ... 26
JAX-RS	Java API for RESTfull Webservices ... 34
JDBC	Java Database Connectivity ... 21
JNLP	Java Network Launching Protocol ... 89
JPA	Java Persistence API ... 21
JPQL	Java Persistence Query Language ... 21
JSON	JavaScript Object Notation ... 34
JVM	Java Virtual Machine ... 20
LSID	Life Science Identifier ... 16
MARUM	Zentrum für Marine Umweltwissenschaften ... 15
MIME	Multipurpose Internet Mail Extensions ... 67

1 Einleitung

Begünstigt durch den rasanten technischen Fortschritt in fast allen Bereichen der Lebenswissenschaften ist die Verfügbarkeit von Hochdurchsatzdaten in den letzten Jahren stark angestiegen (vgl. Marx, 2013). Wir befinden uns im sogenannten *big data*-Zeitalter. Ein Begriff der heute sehr populär und Thema vieler aktueller Publikationen ist. Auch wenn es keine klare Definition gibt und Forschungsdaten aus verschiedenen Gründen als „big“ betrachtet werden können, ist damit im Allgemeinen die große Menge und damit verbundene Herausforderung der effizienten Verwaltung und Verarbeitung von Daten gemeint (vgl. Lynch, 2008). Bereits nach der Jahrtausendwende wurde aufgrund der schnellen Weiterentwicklung von Computern und stark sinkenden Preise für Speichermedien auf eine bevorstehende Flut an Daten hingewiesen (vgl. Roos, 2001), auch wenn damals noch niemand von big data sprach. Mittlerweile sind die Lebenswissenschaften, neben der Physik und Astronomie (Marx, 2013; Stephens et al., 2015), eine zentrale Domäne des big data-Zeitalters (vgl. May, 2014). Sie umfassen dabei ein sehr breites Spektrum der Forschungsgebiete in der modernen Biologie, die sich jeweils mit der Untersuchung von ähnlichen Strukturkomponenten beschäftigen und durch neue Verfahren, den sogenannten *omics*-Technologien, generiert werden (vgl. Valdes et al., 2013). Dazu gehören etablierte Forschungszweige, wie die *Genomics*, die sich mit der Analyse von Genen und Genomen beschäftigen und durch die Entwicklung der *Next-Generation-Sequencing (NGS)*-Verfahren (Metzker, 2010; Schneeberger und Weigel, 2011) einen enormen Zuwachs an Daten erfahren haben. Es gibt aber auch relativ neue Gebiete wie die *Phenomics*. Diese beschäftigen sich mit der Untersuchung von phänotypischen Charakteristika und gewinnen durch die Entstehung neuer Hochdurchsatz-Bildverfahren immer mehr an Bedeutung (Houle et al., 2010; Fiorani und Schurr, 2013). Aufgrund dessen sind gleichzeitig der Bedarf und das Interesse vonseiten der Wissenschaft, Fördergeber und Verlage nach einem zukunftssicheren und modernen Forschungsdatenmanagement entstanden, um die anfallende Menge an Daten zu bewältigen. Im Fokus dieser Dissertation steht nicht die Leistungsfähigkeit neuer Hardware, sondern der langfristige Umgang mit Forschungsdaten und deren effiziente Handhabung im täglichen wissenschaftlichen Arbeitsprozess. Der Wert von Forschungsdaten und deren potenzielle Nachnutzung muss durch ein geeignetes Forschungsdatenmanagement erhalten werden. Die freie Verfügbarkeit von Forschungsdaten bietet sowohl die Möglichkeit, existierende Ergebnisse zu reproduzieren und zu validieren, als auch neue Erkenntnisse zu gewinnen (vgl. McNutt et al., 2016). Außerdem können die Daten so einfacher in einen interdisziplinären Zusammenhang gebracht werden.

Bisher existiert jedoch noch kein interdisziplinäres und organisationsübergreifendes Forschungsdatenmanagement mit einer dynamischen Infrastruktur, welche die Möglichkeit bietet die vielfältigen Arten von wissenschaftlichen Daten aus allen Domänen langfristig und nachhaltig zu verwalten. Die Gründe dafür sind verschieden. Einerseits gibt es eine Vielzahl an domänenspezifischen Lösungen zum Forschungsdatenmanagement, die sich in unterschiedlichen Forschungszweigen entwickelt und etabliert haben, jedoch auch einige

Schwachstellen besitzen. So ist das European Bioinformatics Institute (EMBL-EBI) seit seiner Gründung in einigen Bereichen der Lebenswissenschaften als Datenspeicherdienst für Forschungsdaten etabliert (Rodriguez-Tomé et al., 1996). Es entwickelt und verwaltet über 40 unterschiedliche Archive für z. B. genomische, transkriptomische oder proteomische Daten und hat seine Gesamtspeicherkapazität in den letzten zwei Jahren auf 120 Petabyte verdoppelt. (vgl. Cook et al., 2016, 2018). Im Bereich der phänotypischen Daten hingegen gibt es bisher kaum etablierte Archive und Datenbanken (vgl. Cook et al., 2018). Andererseits ist die Entwicklung eines allgemeinen Konzeptes zum Forschungsdatenmanagement durchaus komplex, da verschiedene Anforderungen erfüllt und Richtlinien zum nachhaltigen Umgang mit Forschungsdaten durchgesetzt werden müssen. So ist etwa die Entwicklung und Integration eines Verfahrens zur eindeutigen Referenzierung von digitalen Daten notwendig.

Ziel dieser Dissertation ist die Entwicklung einer nachhaltigen, generischen und performanten Infrastruktur zum Forschungsdatenmanagement. Erste konzeptionelle Vorarbeiten für die Infrastruktur wurden bereits im Zuge einer Diplomarbeit (Arend, 2012) betrachtet. Das Hauptaugenmerk des Dissertationsprojektes liegt auf der Veröffentlichung und Referenzierung von Phänotypisierungsdaten. Dafür wurden verschiedene nationale und internationale Standards und Informationssysteme zur Publikation von Forschungsdaten betrachtet. Die Herausforderung bei der Entwicklung der Infrastruktur liegt dabei unter anderem in der Konzeptionierung und Implementierung eines geeigneten Datenbegutachtungs- und Freigabeprozesses. Außerdem ist die Gewährleistung einer ausreichenden Performance und Skalierbarkeit der Infrastruktur ein weiterer Anspruch. Während des Dissertationsprojektes wurden daher folgende Forschungsfragen bearbeitet:

1. Wie werden Forschungsdaten üblicherweise verwaltet und publiziert?
Forschungsdaten sind sehr heterogen und durchlaufen verschiedene Phasen von der Entstehung bis zur finalen Veröffentlichung als Bestandteil einer wissenschaftlichen Publikation. Abhängig von der jeweiligen Forschungsdomäne, dem veröffentlichenden Verlag oder den Erfahrungen der Autoren werden sie in unterschiedlichen Datenbanken und Informationssystemen archiviert und publiziert. Es stellt sich die Frage, welche Anforderungen diese Systeme erfüllen können und wo ihre Schwachstellen liegen.

2. Welche Ansprüche stellen Forschungsdaten an die Infrastruktur?
Aufgrund der Komplexität und Heterogenität von Forschungsdaten stellt sich die Frage der besonderen Funktionen und technischen Eigenschaften einer generische Infrastruktur zur effizienten Verwaltung und langfristigen Publikation. Außerdem ist die Frage wie diese am effektivsten und nachhaltig implementiert werden kann, um im big data-Zeitalter auch mit sehr großen Volumen an Datensätzen umgehen zu können.

3. Was sind die Anforderungen an einen generischen Prozess für die Begutachtung von Forschungsdaten?
Die Bereitstellung und Veröffentlichung von Forschungsdaten ist nur sinnvoll, wenn eine qualitative Begutachtung erfolgt. Nur so kann gewährleistet werden, dass die Daten auch wiederverwendet werden können und reproduzierbar sind. Daraus ergibt sich die Frage nach einem generischen und intuitiven Begutachtungsprozess für die Bewertung von Forschungsdaten und dessen technischer Umsetzung.

Es wurde eine Datenhaltungs- und Publikationsinfrastruktur für Forschungsdaten etabliert, die als ersten Anwendungsfall in den existierenden Forschungsprozess am Leibniz-Institut für Pflanzengenetik und Kulturpflanzenforschung (IPK) in Gatersleben integriert wurde. Diese wurde unter anderem für die Archivierung und Publikation von phänotypischen Daten, die im Rahmen des DPPN-Projektes (Deutsches Pflanzen Phänotypisierungsnetzwerk) erzeugt wurden, genutzt. Darüber hinaus ist das daraus resultierende Repository Bestanstandteil des Service-Portfolios im GCBN-Verbund (German Crop BioGreenformatics Network) des de.NBI-Netzwerks (German Network for Bioinformatics Infrastructure). Die entwickelte Infrastruktur ist jedoch nicht auf Phänotypisierungsdaten beschränkt, sondern kann jegliche Art von Forschungsdaten speichern und veröffentlichen. Sie lässt sich dynamisch und ohne zusätzlichen Entwicklungsaufwand in weitere Prozesse und Anwendungen integrierten und kann auch von anderen Institutionen genutzt werden. Gegen Ende des Dissertationsprojektes konnte so eine weitere Instanz auf Basis der entwickelten Infrastruktur am Forschungszentrum Jülich (FZJ) etabliert werden.

1.1 Überblick und Struktur

Im Kapitel 2 werden die grundlegenden Aspekte und Begriffe zum Thema Pflanzenphänotypisierung und Forschungsdatenmanagement vorgestellt. Diese bilden die Basis für die Bearbeitung des Dissertationsprojektes und werden im Weiteren verwendet. Außerdem werden wichtige technische und informatische Konzepte sowie Softwarebibliotheken und Frameworks vorgestellt, die für die Entwicklung der Infrastruktur relevant sind. In Kapitel 3 werden existierende Datenbanken, Informationssysteme und Publikationsdienste zum Austausch von Forschungsdaten untersucht und miteinander verglichen. Die sich aus den Stärken und Schwächen der vorgestellten Infrastrukturen und aus dem üblichen Publikationsprozess von Forschungsdaten ergebenden Anforderungen an die langfristige Archivierung und Sicherstellung der Reproduzierbarkeit von Forschungsdaten werden im Anschluss in Kapitel 4 beschrieben. Außerdem werden Vorarbeiten, die in Verbindung mit diesem Dissertationsprojekt stehen, erläutert. Das entwickelte Infrastrukturkonzept zum Forschungsdatenmanagement und zur Publikation von wissenschaftlichen Daten sowie dessen Implementierung ist Bestandteil von Kapitel 5. Außerdem wird die in diesem Zusammenhang entwickelte Anwendung zur Einreichung und Publikation von Forschungsdaten detailliert beschrieben. Am Ende des Kapitels wird darüber hinaus auf die Nachhaltigkeit der implementierten Infrastruktur eingegangen. In Kapitel 6 werden die Ergebnisse des Dissertationsprojektes evaluiert, indem die entwickelte Infrastruktur mit existierenden Ansätzen zum Forschungsdatenmanagement verglichen wird. Darüber hinaus werden einige Leistungstests vorgestellt und Nutzerstatistiken der ersten Instanz der Infrastruktur gezeigt. Das letzte Kapitel enthält eine Zusammenfassung der Arbeit sowie Informationen zur langfristigen Nutzung der entwickelten Infrastrukur und einen Ausblick auf zukünftige Entwicklungen.

Hinweise

Alle programmiersprachlichen Klassen, Objekte, Funktionen oder Quelltexte sind mit einer `Festbreitenschrift` gekennzeichnet. Die abgebildeten Quelltexte und Codebeispiele dienen zur Veranschaulichung und enthalten keinen vollständigen, ausführbaren Code. Durch eine *kursive* Hervorhebung werden Fachbegriffe eingeführt und besondere Textelemente betont. Häufig verwendete und gebräuchliche Abkürzungen sind im Abkürzungsverzeichnis mit der Seite des ersten Auftretens gelistet und werden nur einmal im Text vollständig ausgeschrieben.

Alle Dateien, Quelltexte und die Java-Dokumentation der im Zuge der Dissertation entwickelten eDAL-Infrastruktur sind in einen öffentlichen Bitbucket-Repository (eDAL, 2018) verfügbar. Die Projektwebseite mit einer umfangreichen Dokumentation, Anwendungsbeispielen, Publikationen und Präsentationen ist unter `http://edal.ipk-gatersleben.de` erreichbar.

1.2 Eigene Publikationen als Bestandteil des Promotionsvorhabens

Alle Teile dieser Arbeit wurden in einer Reihe von Publikationen in wissenschaftlichen Fachzeitschriften oder Tagungsbänden von verschiedenen Konferenzen veröffentlicht und vorgetragen. Diese Artikel sind im nachfolgenden gelistet.

- Arend D, Lange M, Colmsee C, Flemming S, Chen J, und Scholz U. The e!DAL JAVA-API: Store, share and cite primary data in life sciences. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 4-7 October 2012, Philadelphia, USA*. doi:10.1109/BIBM.2012.6392737
- Arend D, Lange M, Chen J, Colmsee C, Flemming S, Hecht D, und Scholz U. e!DAL - a framework to store, share and publish research data. *BMC Bioinformatics*, 15(1), 2014. doi: 10.1186/1471-2105-15-214.
- Arend D, Colmsee C, Knüpffer H, Oppermann M, Schüler D, Weise S und Lange M. Data management experiences and best practices from the perspective of a plant research institute. In *International Conference on Data Integration in the Life Sciences (DILS) - Series: Lecture Notes in Computer Science, Vol. 8574, 17-18 July 2014, Lisbon, Portugal.* doi:10.1007/978-3-319-08590-6_4
- Arend D, Junker A, Scholz U, Schüler D, Wylie J, und Lange M. PGP repository: a plant phenomics and genomics data publication infrastructure. *Database* (2016) 2016. doi: 10.1093/database/baw033
- Arend D, Lange M, Pape J-M, Weigelt-Fischer K, Arana-Ceballos F, Mücke I, Klukas C, Altmann T, Scholz U und Junker A. Quantitative monitoring of *Arabidopsis thaliana* growth and development using high-throughput plant phenotyping. *Scientific Data* **3**:160055 (2016). doi: 10.1038/sdata.2016.55

2 Grundlagen

In diesem Kapitel werden grundlegende Konzepte und Hintergründe beschrieben, auf denen die vorliegende Arbeit aufbaut. Der erste Teil gibt einen Einblick in theoretische Grundlagen und Begriffe im Zusammenhang mit Forschungsdaten und deren Publikation, die für die Bearbeitung des Themas Voraussetzung sind. Anschließend werden Technologien und Verfahren präsentiert, die für die praktische Umsetzung der Arbeit notwendig sind.

2.1 Pflanzenphänotypisierung

Pflanzenphänotypisierung ist im Allgemeinen ein Prozess zur Untersuchung und Beschreibung von strukturellen und funktionellen Merkmalen von Pflanzen und deren Beziehungen zueinander. Der Phänotyp beschreibt das Erscheinungsbild von Pflanzen oder Pflanzenteilen und wird seit jeher von Landwirten und Züchtern untersucht, um ertragreiche und widerstandfähige Pflanzen zu selektieren sowie nach verschiedenen Sorten zu klassifizieren (vgl. Krajewski et al., 2015). Eine umfangreiche Untersuchung des aktuellen Standes der Forschung im Bereich der Pflanzenphänotypisierung zeigte, dass sich die Begriffe Phänotyp bzw. Phänotypisierung in der Praxis heute nicht mehr ausschließlich auf das äußere Erscheinungsbild beschränken (vgl. Fiorani und Schurr, 2013). So wird häufig auch auf molekularer oder biochemische Ebene von phänotypischen Eigenschaften gesprochen, da sich diese Charakteristika verständlicherweise auch auf das Äußere einer Pflanze auswirken. Daher beinhaltet Pflanzenphänotypisierung heutzutage eine Vielzahl an nichtinvasiven Methoden zur Untersuchung von pflanzlichen Merkmalen und deren Entwicklung auf genetischer, mikroskopischer und makroskopischer Ebene.

Kulturpflanzen und deren Produkte sind die Grundlage für die weltweite Nahrungs- und Futtermittelproduktion. Die stetig wachsende Weltbevölkerung und Klimaveränderungen zwingen uns dazu, deren Produktivität und Resistenz permanent zu verbessern. Eine Untersuchung von 2010 geht davon aus, dass der Bedarf an den drei wichtigsten Getreidesorten Reis, Weizen und Mais bis zum Jahr 2050 um etwa 37 % steigen wird (Tester und Langridge, 2010). Diese Ertragssteigerung ist kaum durch verbesserte Anbaumethoden, effizientere Düngung oder gezielteren Einsatz von Pflanzenschutzmitteln zu realisieren. Um neue Sorten mit höheren Erträgen, gesünderen Inhaltsstoffen und besserem Schutz gegen biotische und abiotische Umweltfaktoren zu entwickeln, ist es notwendig, das Wachstum und die Entwicklung der Pflanzen genauer zu verstehen. Der Phänotyp (P) kann Rückschlüsse auf molekulare und physiologische Eigenschaften liefern und ist im Allgemeinen als das Ergebnis der genetischen Ausstattung, dem Genotyp (G) und den vorherrschenden Umweltbedingungen (E) während der Wachstumsphase (P = G x E) definiert (vgl. Walter et al., 2015; Arend et al., 2016b; Ćwiek-Kupczyńska et al., 2016). In Abbildung 2.1 auf Seite 6 ist dieser Zusammenhang schematisch dargestellt.

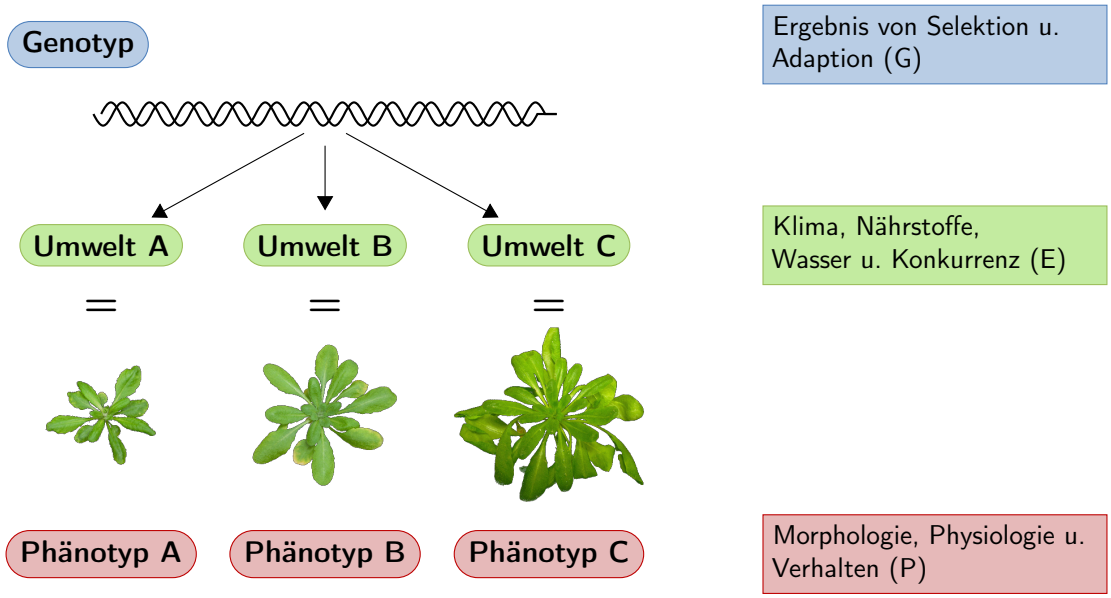

Abbildung 2.1: Zusammenhang zwischen Genotyp und Phänotyp (P = G x E). Der Phänotyp (rot) ist das Ergebnis der genetischen Ausstattung (blau) und den vorherrschenden Umweltbedingungen (grün).

Hochdurchsatz-Phänotypisierung

Die fortschreitende Entwicklung in der Pflanzenphänotypisierung in den letzten Jahren wurde vor allem durch die Entwicklung neuer bildgebender Verfahren, Sensoren und Plattformen sowie moderne Algorithmen und Programme zur Bildanalyse vorangetrieben und führte zu einem enormen Anstieg des Volumens an phänotypischen Daten (vgl. Coppens et al., 2017). In verschiedenen aktuellen Publikationen (Fahlgren et al., 2015; Walter et al., 2015; Rousseau et al., 2015) werden eine Reihe von neuen, vielversprechenden Verfahren zur Pflanzenphänotypisierung vorgestellt. Außerdem sammelten (Lobet et al., 2013) über 140 Tools für unterschiedliche Analysen phänotypischer Bilddaten in einer Datenbank. Das zeigt die vielfältigen Möglichkeiten der Phenomics. Die Erzeugung von phänotypischen Daten im Hochdurchsatzverfahren ist die Grundlage dafür, dass die Phänotypisierung in Zukunft ein noch wichtigerer Bestandteil der Pflanzenforschung sein wird. Ebenso vielfältig, wie die Möglichkeiten sind, genauso vielseitig und heterogen sind die Arten von phänotypischen Daten. (Dhondt et al., 2013) und (Perez-Sanz et al., 2017) beschreiben unter anderem eine Vielzahl an Phänotypisierungsverfahren bzw. Datentypen und deren Unterteilung. So können diese z. B. nach der Art und der Wellenlängen der Sensoren, die zur Aufnahme genutzt werden (sichtbares Licht, nah-infrarotes Licht, Fluoreszenz, Hyperspectral etc.) oder räumlich nach den erfassten Pflanzenteilen (Wurzel, Spross, Blüte etc.) klassifiziert werden. Es gibt Aufnahmen im mikroskopischen und makroskopischen Bereich als auch zwei- und drei-dimensionale Aufnahmen. Abbildung 2.2 auf Seite 7 zeigt eine mögliche Klassifizierung von pflanzenphänotypischen Versuchen bzw. Daten. Dabei sind die Daten, die bei den einzelnen Verfahren entstehen sehr unterschiedlich. So beträgt etwa das Dateivolumen von Aufnahmen im Bereich des sichtbaren Lichts nur wenige Megabyte. Hyperspectral-Aufnahmen können aufgrund der verschiedenen aufgenommenen Wellenlängen bis zu mehreren Gigabyte groß sein.

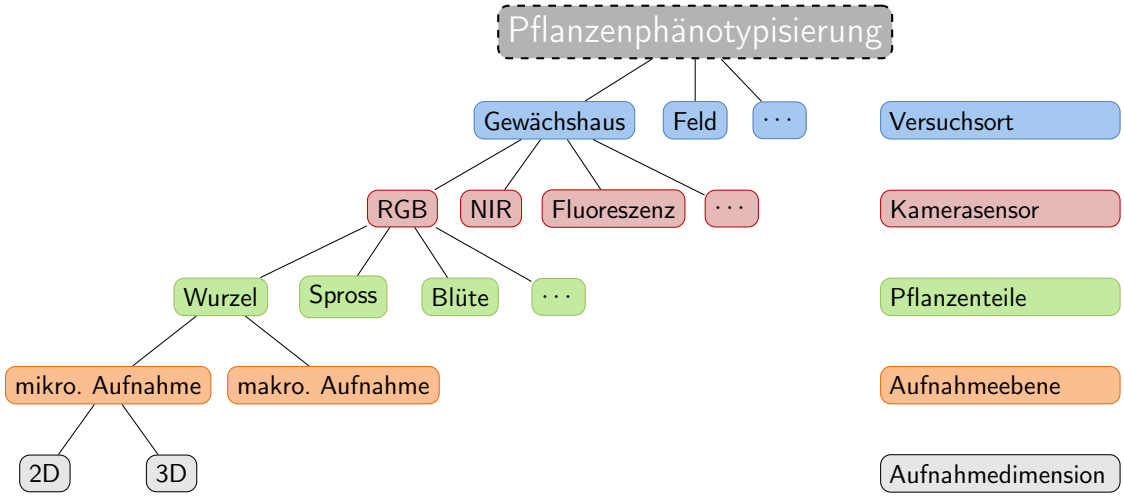

Abbildung 2.2: Schematische Darstellung der verschiedenen Arten von Pflanzenphänotypisierungsversuchen und -aufnahmen (vgl. Dhondt et al., 2013; Perez-Sanz et al., 2017). Die jeweiligen Unterkategorien pro Ebene gelten für alle darüber liegenden Kategorien und wurden zur besseren Übersicht nur für ein Element pro Ebene aufgelistet.

Diese Entwicklung erinnert stark an die rasanten Fortschritte im Bereich der Genomics, die vor allem von der technischen Weiterentwicklung der Sequenzierungsverfahren in den letzten zehn Jahren profitierten und die Forschung in diesem Bereich revolutioniert haben (vgl. Schuster, 2008; Mardis, 2008; Koboldt et al., 2013). Das hängt verständlicherweise auch mit der erwähnten engen Beziehung von Genotyp und Phänotyp zusammen. Die Untersuchung dieser Zusammenhänge ist häufig Bestandteil verschiedenster Züchtungsprojekte. Jedoch gibt es im Bereich der Phenomics noch einige Defizite, was die Verknüpfung von Informationen aus beiden Bereichen erschwert (vgl. Tester und Langridge, 2010). Viele nicht-invasive Verfahren und Technologien ermöglichen zwar die Hochdurchsatzerzeugung von phänotypischen Daten, jedoch gibt es bisher nur sehr wenige standardisierte und von der Community akzeptierte Protokolle und Herangehensweisen. Die meisten Methoden werden an Modellorganismen entwickelt und publiziert, jedoch fehlt es häufig noch an entsprechenden Adaptionen für eine breites Feld an Nutzpflanzen. Gleichzeitig sind die Protokolle der einzelnen Verfahren in unterschiedlichen Einrichtungen und Laboren oft sehr verschieden. Das liegt beispielsweise an unterschiedlicher technischer Ausstattung oder auch daran, dass Experimente mit unterschiedlichen Parametern auf Basis vorausgegangener Erfahrungen durchgeführt werden. Daher ist die Entwicklung von einheitlichen, minimalen Parametern und Bedingungen für bestimmte Protokolle und Experimente notwendig (vgl. Fiorani und Schurr, 2013; Krajewski et al., 2015).

Eine weitere Herausforderung liegt darin, dass es im Gegensatz zu genomischen oder proteomischen Daten im Moment nur sehr wenige akzeptierte und umfangreiche Datenbanken und Archive für phänotypische Daten speziell von Pflanzen gibt (vgl. Fabre et al., 2011; Cruz et al., 2016). So listen (Galperin et al., 2017) über 150 Datenbanken aus verschiedenen biologischen Domänen auf, von denen jedoch nur zwei phänotypische Daten bereitstellen und auch in der Wikipedia sind z. B. knapp 25 Datenbanken mit genomischen Daten und über 20 Archive mit proteomischen Daten gelistet. Phänotypi-

sche Datensätze werden von lediglich zwei Datenbanken archiviert (vgl. Wikipedia, 2018). Ursache dafür ist die angesprochene Heterogenität, aber auch das enorme Volumen phänotypischer Daten. Das schränkt die Reproduzierbarkeit von Forschungsergebnissen ein, da die Autoren bei der Veröffentlichung auf Alternativen zur Bereitstellung von Forschungsdaten zurückgreifen, die nicht immer eine langfristige Verfügbarkeit und Wiederverwendung der Daten garantieren können. Außerdem wird somit eine Auswertung und Verknüpfung von genomischen und phänotypischen Daten erschwert. Die Forschungsgemeinschaft muss neue Wege zur Publikation von Forschungsdaten aus phänotypischen Experimenten und Analysen finden und nutzen (vgl. Nature Genetics Editorial, 2015a).

2.2 Forschungsdatenmanagement

Der Begriff big data wird mittlerweile häufig benutzt und ist wahrscheinlich eines der am meisten verwendeten Schlagworte in verschiedensten Publikationen der letzten Jahre (vgl. Nekrutenko und Taylor, 2012; Chen et al., 2013). Oft wird damit einfach die exponentiell wachsende Menge an digitalen Daten gemeint (vgl. Merelli et al., 2014). Ursächlich dafür ist oft die fortschreitende technologische Entwicklung, aber auch fehlende Datenmanagementrichtlinien, welche die Auswahl und die Aufbewahrungsfristen für Forschungsdaten definieren (vgl. Chen und Zhang, 2014; Cai und Zhu, 2015). So wurden im letzten Jahrzehnt unter anderem im Bereich der Lebenswissenschaften erhebliche technische Fortschritte gemacht und eine Reihe von modernen Hochdurchsatztechnologien entwickelt, die eine Fülle von Forschungsdaten produzieren (vgl. Schofield et al., 2010), sodass deren Volumen exponentiell ansteigt (vgl. Szalay und Gray, 2006; Lynch, 2008). Neben den damit verbundenen neuen Anforderungen an die Analyse und Auswertung, hat auch die Aufgabe der effizienten Verwaltung und Speicherung an Bedeutung gewonnen. In den folgenden Abschnitten werden einige wichtige Aspekte des Forschungsdatenmanagements beleuchtet.

2.2.1 Forschungsdaten

Der Begriff Forschungsdaten entspringt keiner klaren Definition, sondern ist ein sehr inhomogener Sammelbegriff, der je nach Disziplin sämtliche Messdaten, die z. B. im Labor oder von Maschinen erzeugt werden, aber auch Beobachtungen von Experimenten und Studien umfasst. Forschungsdaten bilden die Grundlage für den wissenschaftlichen Arbeitsprozess und Erkenntnisgewinn. Es werden täglich Forschungsdaten in einem Umfang erhoben, dass die Verwaltung und Auswertung der Daten nur durch den Einsatz moderner Computer und Technologien zu schaffen ist. Dadurch können aber auch Daten aus unterschiedlichen Bereichen und Quellen einfacher in Beziehung gesetzt und vernetzt werden. So entstehen neue Möglichkeiten für eine kooperative Wissenschaft, der sogenannte *e-science* (vgl. Hey und Trefethen, 2005; Bardi und Manghi, 2014). Diese sorgte dafür, dass mit der entstehenden Datenflut (english: *data deluge*) ein neues, viertes Forschungsparadigma entstanden ist, das sich mit der durch Daten getriebenen Forschung befasst und neue Erkenntnisse durch Verarbeitung und Analyse von Massendaten gewinnt. Es wurde erstmals vom amerikanischen Forscher Jim Gray beschrieben (vgl. Hey et al., 2009). Die Voraussetzung dafür ist eine Forschungsdateninfrastruktur zur systematischen Speicherung und Verarbeitung der generierten Daten (vgl. Bell et al., 2009). Die zunehmende Bedeutung der

Themen big data und e-science wird auch in Abbildung 2.3 deutlich. Es ist die Anzahl an jährlichen Publikationen mit den jeweiligen Schlagwörtern dargestellt. Die Daten wurden anhand von Artikeln, die in PubMed (NCBI, 2017) gelistet sind, entnommen. PubMed ist eine der größten Datenbanken für Publikationen aus verschiedenen Bereichen der Lebenswissenschaften. So hat sich z. B. die Anzahl an Publikationen mit dem Schlagwort big data in den letzten zehn Jahren verfünffacht. Außerdem hat auch im Allgemeinen der Anstieg an verfügbaren Forschungsdaten auch zu einem Anstieg an wissenschaftlichen Veröffentlichungen geführt (vgl. Lu, 2011).

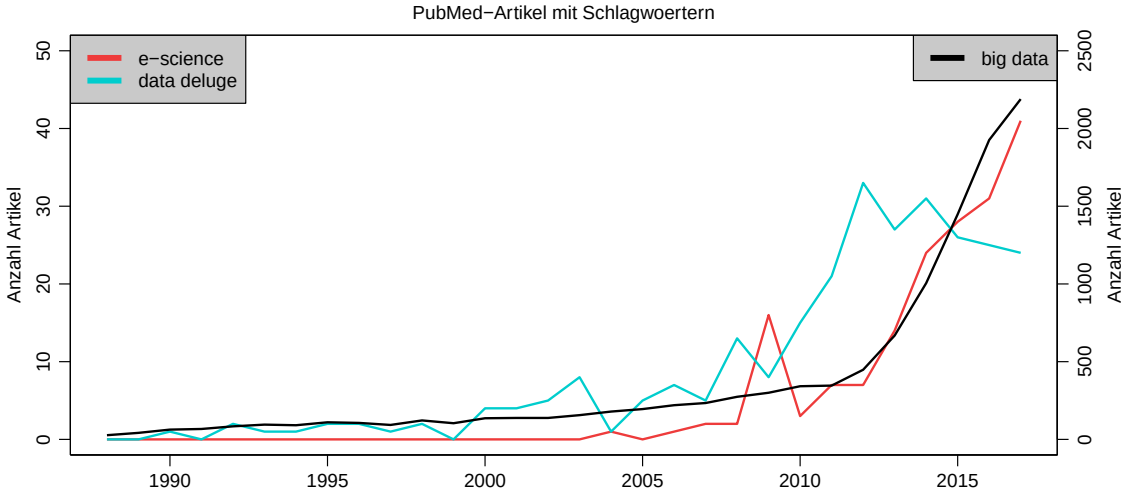

Abbildung 2.3: Entwicklung des Vorkommens von Schlagwörtern zum Thema „big data“, „data deluge“ und „e-sciene“ in PubMed Artikeln der letzten 30 Jahre (vgl. Corlan, 2018).

Umgang mit Forschungsdaten

Jede Institution, die wissenschaftliche Einrichtungen finanziell unterstützt und Forschungsprojekte fördert, ist verständlicherweise daran interessiert, dass nicht nur die daraus resultierenden Ergebnisse, sondern auch die produzierten Daten nachhaltig gespeichert werden und langfristig verfügbar sind (vgl. Lin und Strasser, 2014). Forschungsergebnisse sind sehr vielfältig und werden üblicherweise kondensiert in Form von Publikationen in wissenschaftlichen Journalen oder in Tagungsbänden von Forschungskonferenzen veröffentlicht. Für viele Wissenschaftler ist die Veröffentlichung von Publikationen das oberste Ziel. Artikel und daraus ermittelte Zitierindexe sind das übliche Maß für die Bewertung und Anerkennung der geleisteten wissenschaftlichen Arbeit (vgl. Lawrence et al., 2011). Dies ist durchaus legitim, jedoch sind auch reine Datensätze, egal ob Rohdaten, prozessierte Daten oder im Rahmen des Projektes entwickelte Software es wert veröffentlicht zu werden. Jedoch ist dieses Vorgehen keinesfalls eine verbreitete Praxis und häufig wird diese Aufgabe nur unzureichend erfüllt (vgl. Cragin et al., 2010). Viele Publikationen beinhalten zwar gewöhnlich externe Verweise zu den verwendeten Rohdaten, jedoch zeigte z. B. die Untersuchung eines breiten Spektrums von Artikeln aus PubMed (vgl. Anderson et al., 2006), dass bereits ein Jahr nach deren Veröffentlichung knapp 20 % der verlinkten Daten nicht mehr abrufbar waren. Längerfristige Untersuchungen zeigen sogar, dass nach 20 Jahren teilweise 80 % der Forschungsdaten, die als Bestandteil einer Publikation veröffentlicht

wurden, verloren gehen (vgl. Vines et al., 2014). Grund dafür sind häufig verwaiste URLs oder Webservices, die nicht mehr gepflegt werden (vgl. Wren, 2004; Goodman et al., 2014). Das liegt zum einen oft an den enormen Mengen an Daten, zum anderen stehen für die Wissenschaftler meist die Forschung und nicht die Verwaltung und Veröffentlichung von Daten im Vordergrund, da der Aufwand für die Bereitstellung im Allgemeinen nur wenig wertgeschätzt wird (vgl. Nelson, 2009; Bastow und Leonelli, 2010; Wallis et al., 2013). Außerdem sind sowohl die institutionellen als auch externen Barrieren für die Datenveröffentlichung häufig zu hoch und gleichzeitig ist für viele Forscher der Anreiz dafür nicht ausreichend (vgl. Smith, 2009; Tenopir et al., 2011). Für eine funktionierende Forschungsgemeinschaft sind diese Daten jedoch enorm wichtig, um Ergebnisse zu validieren, eine transparente Forschung zu gewährleisten und weiterführende Analysen vorzunehmen (vgl. Peng, 2011; Fecher et al., 2015; Tellam et al., 2015; Wilkinson et al., 2016). So kann die Veröffentlichung und Publikation von Forschungsdaten zu einer Vielzahl an weiteren Publikationen führen (vgl. Piwowar et al., 2011). Gleichzeitig können die Datenproduzenten damit die nötige Anerkennung für die Zurverfügungstellung ihrer Daten erhalten (Nature Plants Editorial, 2015; McNutt et al., 2016; McNutt, 2016). So konnte unter anderem in verschiedenen Studien (Piwowar et al., 2007; Piwowar und Vision, 2013) gezeigt werden, dass Publikationen mit vollständig veröffentlichten Daten bis zu 70 % häufiger zitiert werden, als Artikel, die ihre zugrunde liegenden Daten nicht veröffentlichen. Da neben dem eigentlichen wissenschaftlichen Artikel, der die gewonnenen Erkenntnisse zusammenfasst, die vollständige Veröffentlichung der zugrunde liegenden Forschungsdaten den Wert der Publikation für die wissenschaftliche Gemeinschaft steigert, wird so auch das Interesse an dem Manuskript und infolge dessen die Anzahl an Zitaten gesteigert (vgl. Botstein, 2010; Drachen et al., 2016). Trotzdem ist die einfache Bereitstellung von Forschungsdaten nur in wenigen Bereichen wie bei genomischen oder astronomischen Daten allgemein üblich. In vielen anderen Bereichen gehen die Vorgehensweisen weit auseinander (vgl. Borgman, 2012).

Fördermittelgeber und Konsortien

Vor allem kleineren und mittelgroßen Institutionen bzw. Forschungsgruppen fehlen oft geeignete Infrastrukturen, die Wissenschaftler bei der Veröffentlichung von Forschungsdaten unterstützen (vgl. Cragin et al., 2010), was ein weiterer Grund für den Verlust von Daten ist. Gleichzeitig mangelt es meist an Strategien und Richtlinien zum Umgang mit Forschungsdaten. Diese Defizite haben mittlerweile auch die Fördergeber erkannt und versuchen, konkrete Rahmenbedingungen und Anforderungen vorzugeben, um Institutionen zu unterstützen und dazu zu drängen, sich mit diesem Thema auseinanderzusetzen. Sie fordern immer häufiger, dass Forschungsergebnisse und Daten aus Projekten, die von öffentlichen Geldern finanziert wurden, auch vollständig veröffentlicht werden und frei zugänglich sind (vgl. Van Noorden, 2013b).

Die Deutsche Forschungsgemeinschaft (DFG) verlangt beispielsweise, dass Einrichtungen, die einen Förderantrag stellen, ein nachhaltiges Forschungsdatenmanagementkonzept vorweisen und unter anderem dafür sorgen, dass produzierte Forschungsdaten zeitnah öffentlich verfügbar sind und langfristig gespeichert werden (vgl. DFG, 2013). Auch das Bundesministerium für Bildung und Forschung (BMBF) fordert, dass Forschungsinstitutionen einen Datenmanagementplan entwickeln und sich dabei an bestimmte Vorga-

ben beispielsweise in Bezug auf die Dokumentation und Archivierung von Forschungsdaten halten (vgl. BMBF, 2018). Außerdem befassen sich auch verschiedene nationale und internationale Verbände und Organisation wie z. B. DataCite (DataCite, 2018a), die Research Data Alliance (RDA) (RDA, 2018a) oder der Arbeitskreis Forschungsdaten der Leibniz Gemeinschaft (WGL) (Leibniz Gemeinschaft, 2018) mit dieser Thematik, um den Umgang und die Wertschätzung von Forschungsdaten zu ändern. Andere Arbeitsgruppen beschäftigen sich darüber hinaus mit Kriterien für die nachhaltige Publikation und Begutachtung von Forschungsdaten und veröffentlichen Empfehlungen diesbezüglich (vgl. Neuroth et al., 2010; Field et al., 2010). Die Europäische Kommission fordert in ihrem Horizon 2020-Programm (European Commission, 2016) ebenfalls, dass alle Einrichtungen, die Förderungen beziehen ihre Forscher dazu anhalten, sämtliche Forschungsdaten zu veröffentlichen (vgl. Lin und Strasser, 2014). Forschungsdaten sollen nach dem Prinzip *Findable, Accessible, Interoperable, Re-Usable (FAIR)* (vgl. Wilkinson et al., 2016) bereitgestellt werden:

- **Findable**:
  Forschungsdaten und dazugehörige Metadaten sollten mithilfe von etablierten Verfahren, wie z. B. dauerhaften Identifikatoren (IDs) langfristig und sicher auffindbar sein.
- **Accessible**:
  Forschungsdaten, die im Rahmen von Projekten erstellt wurden, sollten frei verfügbar und ohne Einschränkungen zugänglich sein. Falls ausgewählte Datensätze z. B. aus Gründen des Datenschutzes nicht frei verfügbar sind, sollte dies detailliert begründet werden.
- **Interoperable**:
  Um den Austausch von Forschungsdaten zwischen Forschern und Institutionen zu ermöglichen, sollten diese in standardisierten Formaten vorliegen und mit etablierten Metadatenschemata und Vokabularien beschrieben werden.
- **Re-Usable**:
  Forschungsdaten sollten eine eindeutige und möglichst weitreichende Lizenz besitzen, um eine umfangreiche Weiternutzung zu gewährleisten. Einschränkungen bezüglich der Nachnutzung sollten detailliert begründet werden.

Die Ideen des FAIR-Prinzips haben sich innerhalb kürzester Zeit weit verbreitet und wurde schon von vielen Fördermittelgebern, Konsortien und Projektträgern in ihre Richtlinien übernommen. Dabei sind die FAIR-Prinzipien nicht unbedingt als feste Norm zu sehen, sondern mehr als ein Art Verhaltenskodex für Systeme und Infrastrukturen, die Forschungsdaten bereit stellen (vgl. Mons et al., 2017).

Fachzeitschriften und wissenschaftliche Gemeinschaft

Lange Zeit haben Fachzeitschriften keine klaren Richtlinien über den Umgang mit Forschungsdaten und ihren Forderungen an die Autoren gehabt (vgl. Cech et al., 2003). Jedoch hat sich dies in den letzten Jahren erheblich geändert und immer mehr Fachzeitschriften fordern den freien Zugang zu Forschungsdaten (Nature Genetics Editorial, 2015b; Roche et al., 2015; Tellam et al., 2015). So akzeptiert etwa die Nature Publishing Group nur Artikel, die alle Forschungsdaten und Materialien, die mit einer Publikation assoziiert sind, vollständig und ohne Einschränkungen zur Verfügung stellen und empfiehlt dafür unter anderem je nach

Zeitschrift und Datentyp eine Reihe von möglichen Datenbanken zur Ablage der Forschungsdaten (vgl. Nature Publishing Group, 2018b; Scientific Data Editorial, 2016). Auch die Fachzeitschriften der Public Library of Science (PLOS) veröffentlichen nur Artikel, deren zugrunde liegenden Forschungsdaten in etablierten Archiven abgelegt oder über dauerhafte IDs auffindbar sind (vgl. PLOS, 2018). Viele weitere Fachzeitschriften haben außerdem ihre Leitfäden für Autoren erweitert und geben ihnen Empfehlungen und Kontrolllisten zur nachhaltigen Bereitstellung ihrer Forschungsdaten (vgl. McNutt, 2014).

Gleichzeitig haben sich einige Fachzeitschriften, wie z. B. ScientificData (Nature Publishing Group, 2018a), GigaScience (Oxford University Press, 2018) oder BMC Research Notes (Springer Nature, 2018) etabliert, die sich auf sogenannte *Data Paper* spezialisiert haben (vgl. Chavan und Penev, 2011; Kratz und Strasser, 2014). Dabei handelt es sich in der Regel um Publikationen, die mit interessanten Forschungsdaten angereichert sind, welche als Referenz bzw. Grundlage für weiterführende Forschungen dienen bzw. Artikel, die sich ausschließlich auf die Beschreibung und Analyse eines bestimmten Satzes an Forschungsdaten spezialisiert haben.

Aber auch von der Seite der wissenschaftlichen Gemeinschaft wird der Ruf nach Verfügbarkeit von Forschungsdaten und Transparenz von Forschungsergebnissen lauter (vgl. Vihinen, 2015) und die meisten Wissenschaftler sind der Meinung, dass dies zu den grundlegenden Normen der wissenschaftlichen Gemeinschaft zählt (vgl. Nosek et al., 2015). Die Verfügbarkeit von Forschungsdaten ist aber nur hilfreich und nützlich, wenn diese auch die notwendige Qualität besitzen, damit sie effektiv verarbeitet und weiter genutzt werden können. Momentan gibt es jedoch keine etablierten Standards und Methoden zu Bewertung der Datenqualität. Die Ursachen liegen in erste Linie im Umfang der Daten und ihrer starken Heterogenität. Wie bereits eingangs erwähnt, steigt das Volumen an produzierten Daten täglich an. Damit steigen aber nicht nur die Anforderungen an die effiziente Analyse und Auswertung der Daten erheblich (Fan et al., 2014), sondern auch der vor allem zeitliche Aufwand für die Bewertungen der Datenqualität. Hinzu kommen unzählige verschiedene Datenformate, die teilweise auch nur sehr kurzlebig sind und die Bewertung erschweren (vgl. Cai und Zhu, 2015).

Reproduzierbarkeit von Forschungsdaten

Mit dem Voranschreiten des big data-Zeitalters hat sich in den letzten Jahren parallel dazu das Thema Reproduzierbarkeit immer mehr in den Vordergrund gedrängt. Jede wissenschaftliche Veröffentlichung gibt immer nur den aktuellen Stand der Forschung wieder und ist keinesfalls eine endgültige Erkenntnis, sondern ist vielmehr erneut Grundlage für weitere Forschung. Daher ist auch die Reproduzierbarkeit von Forschungsergebnissen für den gesamten wissenschaftlichen Prozess ein entscheidender Faktor und wird häufig auch als Goldstandard gesehen (vgl. Jasny et al., 2011). Mit der zunehmenden Flut an Forschungsdaten und dem zusätzlichen Aufwand zur Analyse dieser Daten rückt die Reproduzierbarkeit oft in den Hintergrund. Hauptziel ist gewöhnlich die schnelle Verarbeitung der Daten und die Publikation der Ergebnisse. Außerdem beruht der allgemein etablierte *Peer-Review*-Prozess, z. B. bei der Beurteilung von Publikationen in Fachzeitschriften, auf der Bewertung, ob die vorgestellten Ergebnisse auf Basis der gezeigten Informationen gültig und zuverlässig sind. Es ist üblicherweise nicht die Aufgabe der Gutachter, die Reproduzierbarkeit der Ergebnisse vollständig zu prüfen

(vgl. Borgman, 2012). Deshalb war in den letzten Jahren häufig die Rede davon, dass wir uns in einer Reproduzierbarkeitskrise befinden. So waren etwa 2/3 von knapp 1600 Forschern, die an einer Umfrage der Fachzeitschrift Nature teilgenommen haben (Baker, 2016) der Meinung, dass es bereits eine Reproduzierbarkeitskrise gibt oder wir uns darauf zubewegen. Die Umfrage vermittelt natürlich nur einen subjektiven Eindruck der Teilnehmer, jedoch gibt es eine Vielzahl von Studien dazu aus verschiedenen Forschungsbereichen (Prinz et al., 2011; Begley und Ellis, 2012; Vasilevsky et al., 2013; Freedman et al., 2015), die zwar keine einheitlichen Werte liefern, da sich die Anforderungen an Reproduzierbarkeit je nach Forschungsdomäne teilweise sehr stark unterscheiden (vgl. Nature Editorial, 2016), jedoch wird häufig davon ausgegangen, dass über 50 % der untersuchten Veröffentlichungen nicht reproduzierbar sind.

Dabei sollte jedoch nicht vergessen werden, dass eine vollständige Reproduzierbarkeit nahezu unmöglich ist, da die Vielfalt an Daten und Forschungsdomänen zu groß ist. Informationen, die z. B. durch Messungen oder Beobachtungen bestimmter Umweltfaktoren gewonnen wurden, können nicht reproduziert werden. Daten, die aus komplexen biochemischen Experimenten gewonnen wurden, können schwer exakt reproduziert werden, da bereits winzige Differenzen, etwa in der zeitlichen Abfolge, Auswirkungen auf die Ergebnisse haben können (vgl. Borgman, 2012). Darüber hinaus bedeutet auch die vollständige Reproduzierbarkeit eines Ergebnisses oder einer Publikation nicht automatisch deren Korrektheit. Gleichzeitig ist ein nicht reproduzierbares Ergebnis nicht automatisch falsch (vgl. McNutt, 2014). Trotzdem gibt es viele Publikationen, die von vornherein durch das Fehlen von bestimmten Daten jegliche Reproduzierbarkeit verhindern. Es gibt jedoch eine Reihe von Kriterien bzw. Empfehlungen aus verschiedenen Forschungsdomänen, die für die Reproduzierbarkeit von Forschungsergebnissen und eine möglichst transparente Wissenschaft notwendig sind:

- Für die Reproduzierbarkeit von Forschungsergebnissen ist es notwendig, jegliches verwendete Ursprungsmaterial, z. B. genetische Ressourcen wie Pflanzensamen oder bestimmte Zelllinien, genutzte Peptide oder Antikörperstämme oder untersuchte Organismen, über eindeutige und anerkannte IDs oder Kataloge zu identifizieren (vgl. Vasilevsky et al., 2013).
- Eine möglichst vollständige Reproduzierbarkeit ist nur möglich, wenn durch eine vollständige Versionsverwaltung alle erzeugten Daten und deren Verarbeitungsschritte erfasst und dokumentiert sind, um diese zurückverfolgen zu können und um mögliche Fehler leichter zu finden. (vgl. Ram, 2013; Sandve et al., 2013).
- Eine vollständige und umfangreiche Dokumentation ist nur hilfreich, wenn alle verwendeten und generierten Daten, sowie sämtliche Parameter für die zur Analyse und Auswertung genutzten Geräte und Software zugänglich sind (vgl. Nekrutenko und Taylor, 2012; Sandve et al., 2013).
- Neben der Verfügbarkeit aller verwendeten Daten und Parameter ist darüber hinaus entscheidend, dass diese auch in standardisierten Formaten erfasst werden (vgl. Sandve et al., 2013).

Obwohl wie beschrieben der Druck und Bedarf von allen Seiten zunimmt, gibt es bisher kein verbindliches und disziplinübergreifendes Konzept oder Infrastruktur zum Forschungsdatenmanagement.

2.2.2 Lebenszyklus und Publikationsprozess von Forschungsdaten

Der Lebenszyklus von Forschungsdaten von der Erzeugung bis zur finalen Veröffentlichung ist durch verschiedene Stadien und Formen der Daten charakterisiert und wird häufig durch ein Drei-Domänen-Modell (siehe Abbildung 2.4) beschrieben.

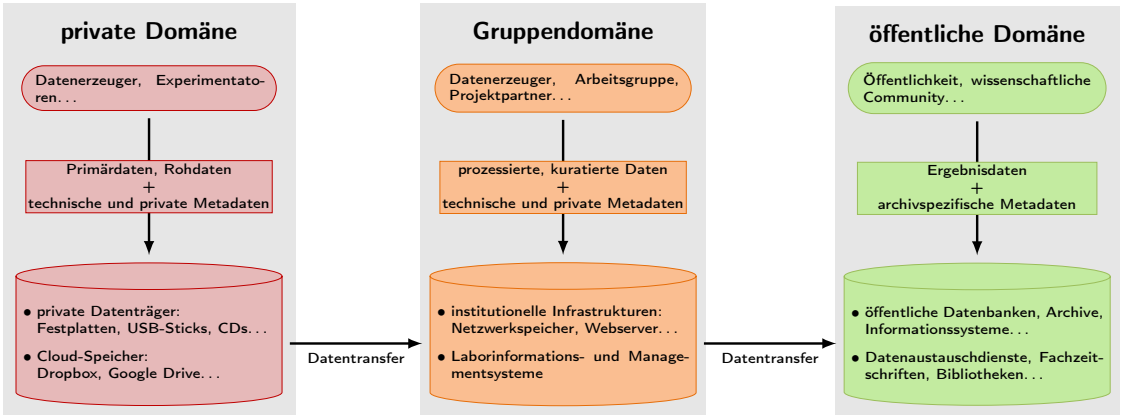

Abbildung 2.4: Schematische Darstellung des Drei-Domänen-Modell des Publikationsprozesses und Lebenszyklus von Forschungsdaten nach (Treloar und Harboe-Ree, 2008).

Zu Beginn des Publikationsprozesses werden die Forschungsdaten in der privaten Datendomäne erzeugt. Diese Primär- oder Rohdaten werden direkt vom Datenerzeuger verwaltet und mit Metadaten angereichert. Zusätzlich sind sie häufig durch technische oder gerätespezifische Metadaten charakterisiert und werden meist auf privaten Datenträgern, die in Regel nur für den Datenerzeuger zugänglich sind, abgelegt (vgl. Neuroth et al., 2010). Unter Umständen werden die Rohdaten auch schon vorverarbeitet ohne das Zwischenergebnisse gespeichert werden oder eine vollständige Verarbeitungskette erfasst wird. Dabei besteht ein hohes Risiko, dass Daten durch Unachtsamkeit verloren gehen, unzureichend beschrieben oder manipuliert werden, wodurch eine Nachnutzung erschwert wird. Außerdem bieten private Speichermedien meist nur unzureichende Mechanismen zum Schutz vor Datenverlust durch technische Defekte. Ist die Analyse der Daten weiter fortgeschritten, werden die Dateien in die Gruppendomäne transferiert. Dafür werden sie beispielsweise auf einen Webserver oder einen Netzwerkspeicher verschoben, auf dem die beteiligten Wissenschaftler und Projektpartner Zugriff auf die Daten erhalten. Hier werden sie weiter untersucht, mit anderen Datensätzen verknüpft oder mit weiteren Metadaten angereichert. Je nachdem, wie die Datensätze ausgetauscht werden, besteht jedoch auch hier die Gefahr, dass Daten verloren gehen. Außerdem lässt sich der Versionsverlauf häufig nur schwer verfolgen. Nach Abschluss der Untersuchung werden die verarbeiteten Daten teilweise in die öffentlichen Domäne übertragen und als Bestandteil einer Publikation veröffentlicht. Viele Datensätze sind aber einfach zu groß um sie als Supplement in einen Artikel zu packen oder es gibt keine geeignete Repräsentation der Daten. Die vielen Veröffentlichungen enthalten nur einen ausgewählten Teil der Daten in Form von kondensierten Grafiken oder Tabellen (vgl. Pop und Salzberg, 2015), der größte Teil bleibt meistens nur für die Datenerzeuger verfügbar. Alternativ werden die Daten häufig über verschiedene Informationssysteme, Datenbanken oder Austauschdienste bereitgestellt. Dies ist jedoch aufgrund des Datenvolumens und der bis dahin meist nur unvollständig gepflegten Metadaten sehr zeitaufwendig. Die Analyse einer Auswahl verbreiteter Systeme wird in Kapitel 3 dargestellt.

Bei der Betrachtung der Domänen wird deutlich, wie komplex der Prozess ist und welche Herausforderungen sich ergeben. Grundsätzliche Schwierigkeiten sind fehlenden Metadaten bzw. nicht standardisierte Formate und die Gefahr des Datenverlustes. Außerdem ist der Versionsverlauf der einzelnen Datensätze oft unvollständig, da Zwischenergebnisse und Teilschritte, die bei der Verarbeitung der Daten entstehen, nicht erfasst werden. Dadurch ist eine exakte Reproduzierbarkeit der Ergebnisse oft nicht möglich (vgl. Sandve et al., 2013). Auch die Suche und Referenzierung von veröffentlichten Forschungsdaten ist häufig aufwendig, da jedes Archiv oder jede Datenbank unterschiedliche und meist proprietäre Metadatenformate und IDs unterstützt. Es gibt bereits einige wenige Systeme bzw. Einrichtungen, die versuchen diese Herausforderung zu bewältigen und Forschungsdaten nachhaltig speichern. So übernimmt z.B das PANGAEA-Informationssystem (Diepenbroek et al., 2002) die Erfassung und Veröffentlichung aller geologischen Daten, die vom Alfred-Wegener-Institut für Polar- und Meeresforschung (AWI) und vom Zentrum für Marine Umweltwissenschaften (MARUM) erzeugt werden. Ein weiteres Beispiel ist die DataONE-Infrastruktur (Michener et al., 2011; Tenopir et al., 2011), die zum Management von Umweltdaten für verschiedene Partner genutzt wird. Beide Systeme sind jedoch auf eine bestimmte Domäne von Forschungsdaten spezialisiert. Ein universelles System, welches das Forschungsdatenmanagement während des gesamten Prozesses gewährleistet, muss sehr flexibel sein damit es in verschiedene Forschungsdomänen verwendet und in existierende Infrastrukturen integriert werden kann.

2.2.3 Persistente Identifikatoren

Damit digitale Forschungsdaten langfristig auffindbar und zitierbar sind, ist ein persistenter Identifikator (ID) notwendig, der online über einen Resolver auflösbar ist. Üblicherweise werden digitale Objekte über Uniform Resource Locators (URLs) referenziert. Diese sind jedoch häufig sehr kurzlebig (vgl. Wren, 2004; Anderson et al., 2006; Vines et al., 2014). Dauerhafte ID-Systeme gewährleisten, dass Objekte langfristig auflösbar sind, auch wenn sich die ursprüngliche URL ändert (vgl. Wimalaratne et al., 2018).

Digital Object Identifier (DOI)

Das bekannteste ID-System ist der Digital Object Identifier (DOI) (vgl. Klump und Huber, 2017). Er wurde erstmals 1997 von der International DOI Foundation (IDF) vorgestellt, die für die Entwicklung und die Verwaltung der DOI-Vergabe verantwortlich ist. Bereits 1998 gab es erste Ideen, dass DOIs sich auch für die Referenzierung von Forschungsdaten eignen (vgl. Mundt, 1998). 2004 wurden dann erstmals DOIs für Forschungsdaten vergeben (vgl. Klump et al., 2016). Im Bereich der digitalen Forschungsdaten werden DOIs vom 2009 gegründeten DataCite-Konsortium (vgl. DataCite, 2018a) registriert und verwaltet. Es ist ein Dachverband von über 20 Mitgliedsorganisationen der sich mit mittlerweile über 600 registrierten Datenzentren, die aktiv DOIs vergeben, etabliert hat (vgl. Brase, 2013). Über den bereitgestellten DOI-Resolver können DOIs dauerhaft aufgelöst werden. Außerdem bietet DataCite eine umfangreiche Metadatensuche über den weltweit registrierten DOI-Bestand. Darüber hinaus ist DataCite an verschiedenen Projekten wie dem ODIN- und THOR-Projekt (vgl. European Commission, 2018a,c) beteiligt, um die Verbreitung und Nutzbarkeit von

DOIs zu erhöhen, sowie die Verknüpfung mit einem eindeutigen Personenidentifikator wie dem Open Researcher and Contributor Identifier (ORCID) (Haak et al., 2012) zu ermöglichen.

DOI-Vergabe Für die kostenfreie Vergabe von DOIs muss sich eine Institution als Datenzentrum im DataCite Konsortium registrieren. In einem Vertrag werden die jeweiligen Ansprechpartner auf der Seite des Datenzentrums und von DataCite festgelegt. DataCite vergibt ein Präfix und gewährt den Zugang zur Registrierungsschnittstelle über eine Weboberfläche (DataCite, 2018b) oder ein *Application Programming Interface (API)*. Im Gegenzug verpflichtet sich das Datenzentrum die langfristige Speicherung der Daten und Erreichbarkeit der URLs, auf welche die DOIs verweisen, sicherzustellen. Bei der Vergabe wird für jede DOI ein Minimalsatz an technischen Metadaten erfasst (siehe Abschnitt 2.2.4).

DOI-Aufbau Der Digital Object Identifier wurde 2012 als ISO-Standard 26324:2012 beschrieben (International Organization for Standardization, 2012), in welchem ebenfalls die Syntax über dessen Aufbau festgelegt wurde (siehe Abbildung 2.5). Jede DOI besteht aus einem Präfix und einem Suffix. Das Präfix beginnt mit dem Indikator „10.“, der die ID als eine DOI innerhalb des Resolving-Dienstes kennzeichnet. Nach dem Indikator folgt ein mindestens vierstelliger Zahlencode, der die vergebende Organisation eindeutig kennzeichnet und üblicherweise von DataCite bei der Registrierung eines Datenzentrums vergeben wird. Das Suffix ist eine frei wählbare Zeichenkette, die vom vergebenden Datenzentrum festgelegt wird und für jedes Präfix eindeutig sein muss. DataCite gibt keine expliziten Konventionen vor, wie ein Suffix aufgebaut sein soll, jedoch sollte dieser im Allgemeinen nach einem konsistenten und leicht verständlichen Schema generiert werden (vgl. Klump et al., 2016).

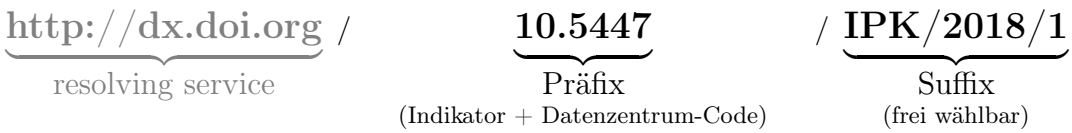

Abbildung 2.5: Schematische Darstellung der DOI-Syntax. Jede DOI besteht aus einem Präfix, das von DataCite vergeben wird, und einem vom Datenzentrum frei wählbaren Suffix.

weitere ID-Systeme

Es gibt noch weitere ID-Systeme, wie der *Uniform Resource Name (URN)* oder der *Life Science Identifier (LSID)*. URNs sind ein Spezialfall der *Uniform Resource Identifiers (URIs)*, die weltweit z. B. in Form von URLs zur Adressierung von Internetressourcen genutzt werden. Die Internet Assigned Numbers Authority (IANA) Organisation hat eine Reihe von *Request for Comments (RFC)* verabschiedet, in denen der Aufbau und die Auflösung von URNs definiert sind (vgl. Neuroth et al., 2010). Die LSIDs sind wiederum eine Spezialisierung der URNs mit einem fest definierten Schema zur Adressierung von Ressourcen aus dem Bereich der Lebenswissenschaften (vgl. Clark et al., 2004). Anders als DOIs haben URNs aber keinen einheitlichen Resolver-Dienst zur Auflösung vergebener IDs. Da beide System im Vergleich zu den DOIs weniger weiter verbreitet sind, wird an dieser Stelle auf eine detaillierte Beschreibung beider Systeme verzichtet.

2.2.4 Metadatenstandards

Um die langfristige Nutzbarkeit und Reproduzierbarkeit von Forschungsdaten zu gewährleisten, müssen neben den eigentlichen Dateien auch ausreichend Metadaten erfasst werden. Sie ermöglichen sowohl die Interpretierbarkeit der Daten als auch eine Katalogisierung, Filterung und Suche. Dafür ist ein standardisiertes Metadatenschema entscheidend.

Im Allgemeinen wird zwischen semantischen und technischen Metadaten unterschieden. Semantische Metadaten sind sehr detailliert und häufig fachspezifisch. Sie haben einen engen Bezug zu den dazugehörigen Datentypen, die sie beschreiben bzw. zur Datendomäne aus der sie stammen. Daher ist eine Standardisierung schwierig und die einzelnen Schemata unterscheiden sich häufig sehr stark. So gibt es einige Publikationen (Brazma et al., 2006; Wierling et al., 2007), die existierende semantische Formate vorstellen und vergleichen, sowie eine Reihe von Organisationen, die umfangreiche Listen und Verzeichnisse (DCC, 2018; RDA, 2018b; FAIRSharing, 2018) mit etablieren semantischen Metadatenformaten aus verschiedenen Bereichen der Lebenswissenschaften bereitstellen. Grundsätzlich ist aber zu beachten, dass nicht alle Formate klar in die semantische oder technische Kategorie fallen, sondern teilweise auch Mischformen sind.

Da im Rahmen des Dissertationsprojektes eine generische Infrastruktur entwickelt wurde, die für verschiedene Arten von Forschungsdaten verwendet werden kann, ist eine Spezialisierung auf ein bestimmtes semantisches Format nicht zweckmäßig. Daher wird an dieser Stelle auf eine detaillierte Beschreibung einzelner Beispiele verzichtet. Die Anzahl an technische Metadatenschemata ist weitaus geringer als die Zahl semantischer Formate, da sie weniger fachspezifisch sind und in erster Linie gewährleisten sollen, dass sich die beschriebenen Dateien mittels gebräuchlicher Software öffnen lassen. Sie enthalten z. B. Informationen über das Dateiformat und die Datenerzeuger. Das wohl bekannteste Format ist das *DublinCore* Metadatenschema, sowie das daraus abgeleitete DataCite Äquivalent.

DublinCore Metadatenschema

Die Entwicklung des DublinCore Metadata Element Set (DCMES), an der Wissenschaftler und Bibliothekare beteiligt waren, begann ursprünglich 1995 und diente der standardisierten Beschreibung von digitalen Internetressourcen. Inzwischen ist das DublinCore Schema ein etablierter Standard für die Beschreibung von digitalen Ressourcen. Es umfasst 15 Attribute, die in Tabelle 2.1 auf Seite 18 dargestellt sind und die eine dauerhafte Lesbarkeit und Nutzbarkeit digitaler Daten gewährleisten sollen. Alle Attribute sind optional und mehrfach verwendbar, wobei kein spezifischer Datentyp zur Beschreibung der Attribute vorgegeben ist. Die einzelnen Bezeichnungen sind leicht verständlich und ermöglichen eine einfache Pflege der Metadaten (vgl. Weibel, 1997).

DataCite Metadatenschema

Das DataCite Metadatenschema ist dem DublinCore Schema sehr ähnlich und beinhaltet viele äquivalente Attribute. Es wird von Arbeitsgruppen im DataCite Konsortium entwickelt und bildet die Grundlage für die Vergabe von DOIs und die dauerhafte Nutzbarkeit der dazugehörigen Daten. Anders als im DublinCore Schema unterscheidet es daher verpflichtende

Attribute	Kurzbeschreibungen
TITLE	Bezeichnung des Objektes
CREATOR	Name des Datenerzeugers
SUBJECT	Schlagwort über den Inhalt der Daten
DESCRIPTION	zusammenfassende Beschreibung des Inhaltes der Daten
PUBLISHER	Name der Instanz, die Daten veröffentlicht
CONTRIBUTOR	zusätzlich beteiligte Personen
DATE	Datum/Zeitspanne der Veröffentlichung
TYPE	Art/Kategorie der Daten
FORMAT	Format der Daten
IDENTIFIER	mögliche ID der Daten, z. B. URL oder DOI
SOURCE	Datenquelle bzw. Datenursprung
LANGUAGE	Sprache des Inhaltes der Daten
RELATION	mögliche Beziehung zu anderen Daten
COVERAGE	mögliche Abgrenzung des Dateninhaltes
RIGHTS	Informationen zu Rechten an den Daten

Tabelle 2.1: Übersicht der Attribute des DublinCore Schemas (vgl. DublinCore, 2018).

und optionale Attribute. Die 5 verpflichtenden Elemente, die für die Beschreibung eines Datensatzes notwendig sind, und ohne die keine DOI vergeben wird, sind in Tabelle 2.2 aufgelistet. Außerdem dienen diese Attribute zur Erstellung eines Zitierformates für einen DOI-referenzierten Datensatz.

Attribute	Kurzbeschreibungen
IDENTIFIER	eindeutige DOI die Datensatz identifiziert
CREATOR	Name des Datenerzeugers
TITLE	Bezeichnung der Ressource
PUBLISHER	Eigentümer der Ressource, z. B. Einrichtung des Datenerzeugers
PUBLICATIONYEAR	Jahr der Veröffentlichung der Daten

Tabelle 2.2: Übersicht der verpflichtenden Attribute des DataCite Metadatenschemas (vgl. DataCite, 2018c).

Mit der Beschränkung auf lediglich 5 verpflichtende Elemente wird die DOI-Generierung vereinfacht. Darüber hinaus bietet das DataCite Schema jedoch eine Vielzahl an weiteren optionalen Elementen (siehe Tabelle 2.3 auf Seite 19), die genutzt werden können, um die Daten noch detaillierter zu beschreiben, und um ein möglichst großes Spektrum an Datendomänen abzudecken. Das umfasst z. B. Attribute wie Dateiformat, eine umfassende Beschreibung, Standortangaben oder Informationen zu Lizenzen (vgl. Starr und Gastl, 2011). Anders als beim DubinCore gibt DataCite dabei sehr detaillierte Empfehlungen zur Beschreibung der einzelnen Attribute vor und legt die zur Verwendung möglichen Datentypen fest (vgl. DataCite, 2018c).

Datensammelschnittstelle

Damit Forschungsdaten effektiv genutzt werden können, ist nicht nur die Verwendung von standardisierten Metadaten notwendig, sondern auch, dass diese automatisiert verarbeitet werden können. Nur so können Verbindungen zwischen Daten beispielsweise über

Attribute	Kurzbeschreibungen
SUBJECT	Schlagwort über den Inhalt der Daten
CONTRIBUTOR	zusätzlich beteiligte Personen
DATE	Datum/Zeitspanne der Veröffentlichung
LANGUAGE	Sprache des Inhaltes der Daten
RESOURCETYPE	Art der Daten
ALTERNATEIDENTIFIER	alternative ID des Datensatzes (nicht DOI)
RELATEDIDENTIFIER	ID eines assoziierte Datensatzes
SIZE	Information über die Größe des Datensatzes
FORMAT	Dateiformat
VERSION	Versionsnummer
RIGHTS	Informationen zu Rechten an den Daten
DESCRIPTION	Beschreibung des Datensatzes
GEOLOCATION	Standortinformationen über den Datensatzes

Tabelle 2.3: Übersicht der optionalen Attribute des DataCite Metadatenschemas (vgl. DataCite, 2018c).

verschiedene Datenbanken hinweg herstellt werden. Daher ist eine einheitliche Schnittstelle zum Sammeln und Verarbeiten von Metadaten sinnvoll. Der wohl bekannteste Ansatz ist das *Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)*, welches seit Anfang dieses Jahrtausends von der *Open Archives Initiative (OAI)* entwickelt wird (vgl. Lagoze und Van de Sompel, 2001; Rusch-Feja, 2002). Es ermöglicht Datenproduzenten ihre Metadaten über diese standardisierte Datensammelschnittstelle in ihrer Infrastruktur bereit zustellen, sodass diese von externen Diensten wie Bibliotheken, Verlagen oder Suchmaschinen ausgelesen und verarbeitet werden können. OAI-PMH basiert dabei auf einer Reihe von *Hypertext Transfer Protocol (HTTP)*-Anfragen. Die aktuelle Version 2.0 bietet folgende Anfragenendpunkte (Open Archives Initiative, 2018):

- **Identify**: Informationen über das Datenrepository, z. B. Name und Version
- **ListMetadataFormats**: Liste aller unterstützter Metadatenformate
- **ListIdentifiers**: Liste mit allen Identifikatoren aller erfassten Datenressourcen
- **GetRecord**: Metadatensatz einer gesuchten Datenressource
- **ListRecords**: alle verfügbaren Metadatensätze
- **ListSets**: alle Organisationsmerkmale der vorhanden Metadatensätze

Durch die Verwendung einer REST-basierte Schnittstelle (siehe Abschnitt 2.3.5) ist eine einfache Integration des Protokolls in vorhanden Infrastrukturen möglich, um so auch große Datenbestände effizient auslesen und verarbeiten zu können.

2.3 Informatik

In den folgenden Abschnitten werden technische Begriffe, Softwarebibliotheken und Technologien erläutert, die zur Bearbeitung des Dissertationsprojektes genutzt wurden.

2.3.1 Objektrelationales Mapping und Datenbanken

Nahezu jede Software benötigt heutzutage den Zugriff auf persistente Daten. Diese werden anders als flüchtige, transiente Daten, die zur Laufzeit im Hauptspeicher vorgehalten werden, auf dauerhaften, nichtflüchtigen Speichermedien wie Festplatten oder SSDs abgelegt und sind somit nach Beenden der dazugehörigen Anwendung oder nach Abschalten des Rechners sicher gespeichert. In der Regel besitzen diese Datenträger ein Dateisystem, in dem Dateien abgelegt werden. Alternativ können die Informationen auch in einem relationalen *Datenbankmanagementsystem (DBMS)* erfasst werden.

Das Konzept der relationalen Datenbank wurde bereits um 1970 entworfen (vgl. Codd, 1970) und ist heute der Standard Datenbanktyp. Eine relationale Datenbank besteht aus verschiedenen Tabellen, auch Relationen genannt, in denen die erfassten Datensätze zeilenweise abgespeichert sind. Dabei bilden die Spalten der Tabellen die Eigenschaften der jeweiligen Relation ab. Außerdem können über Verknüpfungen Beziehungen zwischen bestimmten Relationen bzw. Tabellen abgebildet werden. Relationale Datenbankmanagementsysteme erlauben eine konsistente Erfassung von Daten und sind durch ihre effiziente Anfragesprache für die Verarbeitung von umfangreichen Datensätzen konzipiert und skalieren sehr gut. Dies ist aufgrund der stetig wachsenden Menge an Forschungsdaten eine entscheidende Eigenschaft für die zu entwickelnde Infrastruktur.

Die *objektorientierte Programmierung (OOP)* hat sich in den letzten 20 Jahren schnell entwickelt und ist mittlerweile in vielen Bereichen der status quo in der Entwicklung von Software, auch wenn sich die ursprüngliche Definition von OOP von 1996 (vgl. Kay, 1996) auch heute noch weiterentwickelt. Das Paradigma von OOP basiert darauf, die Architektur einer Anwendung an den jeweiligen Bereich der Realität, der abgebildet werden soll, anzupassen. Daher besteht eine Anwendung aus verschiedenen Klassen mit spezifischen Funktionen und Objekten dieser Klassen. Anders als bei der *prozeduralen Programmierung*, bei der eine Anwendung aus einer Abfolge von Funktionen besteht, ermöglicht die OOP eine einfache und flexible Wiederverwendung von einzelnen Komponenten, was für die Implementierung der im Rahmen des Dissertationsprojektes entwickelten Infrastruktur ein wichtiger Aspekt ist. Auch objektorientierte Programmiersprachen wie Java legen ihre Objekte zur Laufzeit der jeweiligen Java Virtual Machine (JVM) im Hauptspeicher ab. Um auf Daten nach Beenden der JVM zugreifen zu können oder vorherige Objekte, sowie deren Variablenbelegung und Unterobjekte verlustfrei wiederherstellen zu können, müssen sie Informationen persistent speichern können. Die einfachste Möglichkeit ist die Serialisierung von kompletten Objekten und das Schreiben und Ablegen in eine Datei im Dateisystem. Dazu müssen die entsprechenden Java Objekte die Serializable-Schnittstelle (`java.io.serializable`) implementieren. Diese Variante hat jedoch den entscheidenden Nachteil, dass sie schlecht skaliert, da keine Änderungen an Objekten geschrieben werden können. Ändert sich beispielsweise eine Variable des Objektes, muss das gesamte Objekt neu geschrieben werden. Das verschlechtert die Performance und bietet sich daher nur für sehr kleine Objekte und Datenstrukturen an. Um viele Objekte und komplexe Datenstrukturen effizient und persistent zu speichern, werden in der Regel relationale Datenbanken genutzt.

JDBC

Zur Kommunikation und Interaktion mit einem relationalen DBMS beinhaltet Java bereits seit Version 1.1 die Java Database Connectivity (JDBC)-Schnittstelle (`java(x).sql.*`). JDBC unterstützt eine Vielzahl von Datenbanktreibern und ist somit in der Lage mit unterschiedlichen Datenbanken zu interagieren und sämtliche Funktionen eines DBMS zu benutzen. Es können Structured Query Language (SQL)-Anfragen ausgeführt und die Ergebnisse erfasst werden. Außerdem sind die Erzeugung, die Änderung und das Löschen von Tabellen und Werten möglich. Indexe können erstellt und manipuliert werden.

Das manuelle Formulieren von CREATE TABLE-Anweisungen, um Tabellen zu erstellen, die Objekte einer bestimmten Klasse, abbilden bzw. das Erstellen von SQL-Anfragen, um bereits gespeicherte Objekte auszulesen, ist sehr aufwendig und fehleranfällig. Um diese Aufgaben zu vereinfachen, gibt es das Verfahren des *Object-Relational Mapping (ORM)*. Es ermöglicht die Objekte und Klassenhierarchie eines objektorientierten Programmes in einer relationalen Datenbank abzubilden und zu verknüpfen. Für die Anwendung und Entwickler entsteht der Eindruck einer objektorientierten Datenbank. Die Abbildung und Erzeugung der notwendigen Tabellen, Schlüssel und Beziehungen übernimmt ein entsprechendes ORM-Framework. Der Anwender legt lediglich fest, welche Objektklassen und Objektvariablen einer Anwendung in der Datenbank gespeichert werden sollen.

Hibernate und JPA

Eines der populärsten und umfangreichsten ORM-Frameworks für Java ist Hibernate (Bauer et al., 2015). Mit der Java Persistence API (JPA) (`javax.persistence.*`) bietet Java eine Schnittstelle zur Objektpersistierung. In der Regel wird eine Klasse auf eine Datenbanktabelle abgebildet. Die Klassenvariablen entsprechen den Tabellenattributen und eine Klasseninstanz entspricht einer Zeile in der jeweiligen Tabelle. Fremdschlüssel und Beziehungen können über spezielle Annotationen in den jeweiligen Objektklassen definiert werden. Diese Annotationen sind zusätzliche Metadaten im Quelltext, die nicht ausgeführt, aber vom Compiler geprüft werden. Außerdem stellt die JPA die Anfragesprache Java Persistence Query Language (JPQL) bereit, die SQL sehr ähnlich ist, sich jedoch nicht auf Tabellen, sondern auf Klassen bezieht und Java-Objekte zurückliefert. Hibernate ist eine Implementierung der JPA und bietet außerdem einige nützliche Zusatzkomponenten. In den folgenden Abschnitten werden einige ausgewählte Funktionen von Hibernate erläutert, die bei der Implementierung der Forschungsdateninfrastruktur genutzt wurden.

Vererbungshierarchie Die zentrale Klasse des Hibernate-Framework ist die Sessionfactory (`org.hibernate.SessionFactory`), die nur einmal pro Anwendung zentral definiert und initialisiert wird. Sie erfasst alle notwendigen Parameter zur Kommunikation mit dem DBMS über die JDBC-Schnittstelle. Beim Ablauf des Programms kann die Sessionfactory mehrere Sessions (`org.hibernate.Session`) für verschiedene Arbeitsschritte erzeugen. Innerhalb der Session können mehrere Transaktionen (`org.hibernate.Transaction`) ausgeführt werden, um bestimmte Abschnitte zu separieren, die jeweils das grundlegende *Atomicity-Consistency-Isolation-Durability (ACID)*-Prinzip einer relationalen Datenbank (vgl. Saake et al., 2013) erfüllen. Das Speichern oder Löschen von Objektinstanzen, die zu Klassen gehören, die keine Beziehungen zu anderen Klassen haben, ist mit den entsprechenden Funktionen einer Session möglich. Die JPA ist jedoch auch in der Lage komplexe

Klassenhierarchien und -vererbungen durch verschiedene Strategien abzubilden. Haben die Klassen keine Beziehungen zu anderen Klassen ist die Erzeugung einer Tabelle pro Klasse sinnvoll. Existieren jedoch Beziehungen zu allen Hierarchieklassen, wird gewöhnlich eine Tabelle pro Klassenhierachie unter Verwendung einer Diskriminatorspalte erzeugt. Wenn lediglich Beziehungen zu Subklassen existieren, wird pro Subklasse eine Tabelle angelegt. In diesen Fällen ist das Erfassen und Löschen von Objekten komplexer, da die referenzielle Integrität nicht verletzt werden darf. Um den Aufwand für die Einhaltung der referenziellen Integrität zu verringern, bietet Hibernate die Möglichkeit, Objekte transitiv zu persistieren. Dadurch werden beim Speichern oder Löschen von Objekten alle assoziierten Objekte ebenfalls gespeichert bzw. nicht mehr benötigte Objekte gelöscht. Das vereinfacht die Implementierung und vermeidet Fehler. Darüber hinaus bietet Hibernate auch die Möglichkeit komplexe Datenstrukturen wie Listen und Maps, die aus einer Reihe von Schlüssel-Wert-Paaren bestehen, direkt als 1:n- oder n:m-Beziehungen abzubilden.

Objektanfragen Um persistente Objekte aus der Datenbank wiederherzustellen, bietet eine Session verschiedene Funktionen an. Über die eindeutige ID eines Objekts oder Getter-Methoden eines bereits geladenen Referenzobjekts können diese direkt erfasst und geladen werden. Gebräuchlicher ist jedoch die Nutzung von Anfragen. Neben der JPQL bietet Hibernate auch die Möglichkeit reine SQL-Abfragen oder Anfragen in der SQL-ähnlichen, aber objektorientierten Hibernate Query Language (HQL) zu formulieren. Darüber hinaus liefert Hibernate mit der Criteria-API (`org.hibernate.criterion.*`) eine weitere Möglichkeit zur Erfassung von Objekten. Dies ist ebenfalls ein objektorientierter Ansatz, bei dem Objekte über ihre Klasse erfasst und mit sogenannten Restrictions (`org.hibernate.criterion.Restrictions`) selektiert werden (siehe Listing Quelltext 2.1). Dadurch bleibt der Quelltext übersichtlich und es werden Fehler bei der Formulierung von komplexen Anfragen vermieden. Hibernate sorgt intern dafür, dass die Criteria-Anfragen optimal ausgewertet werden. Daher wurde die Criteria-API als bevorzugte Variante zur Anfragen-Formulierung bei der Realisierung der zu entwickelnden Infrastruktur gewählt.

```
Session session = sessionFactory.openSession();

Criteria criteria = session.createCriteria(Person.class);

  criteria.add(Restrictions.eq("name","Daniel"));
  criteria.add(Restrictions.gt("age","30"));

List<Person> persons = (List<Person>) criteria.list();
```

Quelltext 2.1: Beispiel für eine Anfrage über die Criteria-API

Unabhängig davon wie die gesuchten Objekte erfasst werden, unterstützt Hibernate beim konkreten Laden aus der Datenbank zwei verschiedene Verfahren. In der Regel wird das *lazy loading* (deutsch: faules Laden) verwendet. Dadurch werden angeforderte Objekte erst beim Zugriff auf Objektvariablen geladen. Ebenso werden referenzierte Objekte erst beim Aufruf nachgeladen. Bis dies geschieht, werden sogenannte *Proxy*-Objekte als Platzhalter genutzt. Damit wird die Performance verbessert, da die Häufigkeit der Datenbankzugriffe minimiert wird. Alternativ können bestimmte Objekteklassen gezielt per *eager laoding* (deutsch: fleißiges Laden) direkt vollständig wiederhergestellt werden. Dies geschieht jedoch auf Kosten der Performance und sollte nur gezielt in Ausnahmefälle verwendet werden.

Caching Aufgrund der großen Menge an Daten, die mit der im Rahmen der Dissertation entwickelten Infrastruktur, erfasst werden sollen, ist die Performance stark von einer optimalen Auswertung von Anfragen abhängig. Daher ist die Verwendung eines effektiven Caches angebracht, um häufig gestellte Anfrageergebnisse möglichst lange im Hauptspeicher vorzuhalten und so die Zugriffszeiten zu reduzieren. Im Gegensatz zur JPA bietet Hibernate bereits ein integriertes, zweischichtiges Cachesystem. Der First-Level-Cache ist standardmäßig aktiv und kann nicht abgeschaltet werden. Er speichert bereits geladene Objekte zur Laufzeit einer Session. Optional kann ein Second-Level-Cache genutzt werden, der zur Laufzeit einer Sessionfactory existiert und Zustände, wie z. B. Anfrageergebnisse speichern kann. Er eignet sich zum gezielten Speichern von komplexen und zeitintensiven Anfragen. Für die physikalische Implementierung können verschieden Cacheprovider, wie z. B. der populäre Ehcache (Terracotta Inc, 2018) genutzt werden.

Indexierung und Suche Eine weitere wichtige Funktion, um große Menge an Daten zu bewältigen und effektiv nutzen zu können, ist eine umfangreiche Suchfunktion. Bei einer kleinen Anzahl an Datensätzen ist eine einfache Datenbankanfrage über den LIKE-Operator am effektivsten. Mit zunehmender Größe der Datenbank nimmt die Performance dieser Operation jedoch stark ab. Eine Alternative ist die Hibernate Search-API (Bell et al., 2009), die das weitverbreitete Lucene Framework (Hatcher et al., 2010) nutzt. Es ermöglicht eine Volltextindexierung und kann mit nahezu allen textbasierten Daten umgehen, weshalb es sich besonders für die Suche in Webseiten oder Textdokumenten eignet. Hibernate Search nutzt diese Funktionalität und ermöglicht es, textbasierte Variablen von Objekten, die mit Hibernate in einer Datenbank persistent abgelegt werden, zu indexieren und effizient zu durchsuchen. Mittels zusätzlicher Java-Annotationen kann festgelegt werden, welche Objektklassen und -variablen indexiert werden sollen. Die Indexierung erfolgt dabei entweder automatisch beim Abspeichern und Ändern von Objekten oder zu manuell festgelegten Zeitpunkten. Im Gegensatz zu einfachen SQL-basierten Suchanfragen bietet eine indexbasierte Suche auch bei zunehmender Datenbankgröße und steigender Zahl an Datensätzen eine konstante Performance.

2.3.2 Aspektorientierte Programmierung

Die *aspektorientierte Programmierung (AOP)* ist ein etwas jüngeres Konzept als die objektorientierte Programmierung (OOP). Es ist aber kein eigenständiges Programmierparadigma, sondern eine Erweiterung des modularen OOP-Konzepts. Die Schwachstelle von OOP ist die Realisierung von Funktionen außerhalb der Klassenhierarchie, sogenannten *cross-cutting concerns*. Dabei handelt es sich häufig um klassenübergreifende Funktionen, z. B. zur Performancemessung, Objektpersistierung oder zum Informationslogging. Diese müssen in der Regel wiederholt implementiert werden und sind über den ganzen Programmcode verteilt, was einen hohen Entwicklungs- und Wartungsaufwand bedeutet und die Fehleranfälligkeit erhöht. Muss ein Teil dieser Funktionen geändert werden, muss dies unter Umständen an sehr vielen Stellen im Quellcode erfolgen. Mithilfe der AOP wird diese Herausforderung gelöst, indem die entsprechenden Routinen in sogenannte Aspekte gekapselt und nur einmal zentral implementiert werden. Diese Aspekte können dann an definierten Stellen von einem speziellen Compiler in den regulären Programmcode eingewebt und ausgeführt werden (vgl. Kiczales et al., 1997).

AspectJ

Die bekannteste aspektorientierte Programmiersprache ist AspectJ. Sie wurde zusammen mit dem AOP-Konzept entwickelt und prägte Begriffe, die heute allgemeingültig in vielen Sprachen verwendet werden. AspectJ erweitert die Java-Programmiersprache und ermöglicht es Aspekte zu implementieren. Sie liefert einen speziellen, erweiterten Compiler, auch *Weaver* genannt, der definierte Aspekte beim Kompilieren in die entsprechenden Stellen im Quellcode einbaut. Dieses Vorgehen wird *compile-time weaving* genannt.

```
public aspect ExampleAOP {

  pointcut SaveMethods():execution(public * ORM+.save*(..));

   before(): SaveMethods(){
     System.out.print("Storing...");
     Long start = System.currentTimeMillis();
  }

  after(): SaveMethods(){
     Long end = System.currentTimeMillis();
     System.out.print("Storing finished in "+end-start+" ms");
  }
}
```

Quelltext 2.2: Aspekt zur Zeitmessung von ORM-Persistenzoperationen mit AspectJ

Ein Aspekt (siehe Quelltext 2.2) besteht jeweils aus *Pointcuts* und *Advices*. Ein Pointcut wiederum definiert eine oder mehrere *Joinpoints*. Dabei handelt es sich um bestimmte eindeutige Stellen im OOP-Programmcode wie z. B. Methodenaufrufe, Variablenzugriffe oder das Auftreten von Ausnahmen. An diese Stellen webt der Weaver später beim Kompilieren den gewünschten Code des Apektes ein. Dieser Code wird in den Advices definiert und kann vor oder nach einem Pointcut, als auch während dessen Ablaufs ausgeführt werden. Durch die zentrale Implementierung von klassenübergreifenden Funktionen wird Programmcode gespart und die Wartung erleichtert werden. Die Übersichtlichkeit der OOP-Logik bleibt dabei erhalten (vgl. Kiczales et al., 2001).

SpringAOP

Neben AspectJ gibt es noch andere Ansätze zur Realisierung von AOP. Ein weiteres populäres Framework ist SpringAOP (Pivotal Software, 2018). Dieses verwendet ein anderes Verfahren zum Einweben definierter Aspekte. Während AspectJ die Advices beim Kompilieren in das eigentliche Programm einfügt, nutzt SpringAOP das sogenannte *runtime-weaving*. Dabei werden die Aspekte erst während der Ausführung des eigentlichen Programms eingefügt. Diese geschieht über Proxy-Objekte, die als Stellvertreter der eigentlichen Objekte fungieren. So ist die Integration einfacher, da kein zusätzlicher Compiler notwendig ist. Außerdem ist die Definition der Aspekte einfacher, da dies direkt in Java durch die Implementierung von SpringAOP bereitgestellten Schnittstellen erfolgt.

Jedoch hat dieser Ansatz zwei Nachteile. Einerseits ist die Funktionalität von SpringAOP deutlich eingeschränkter, da durch das angewendete runtime-weaving nur Joinpoints definiert werden können, die sich auf die Ausführung von Methoden beziehen. Das Erfassen von

Konstruktoren oder Ausnahmesituationen ist nicht möglich. Darüber hinaus bietet das compile-time weaving von AspectJ eine bessere Performance, da keine zusätzliche Proxy-Objekte erstellt werden müssen und so kein zusätzlicher Laufzeit-Overhead entsteht.

Da die Performance der entwickelten Datenhaltungsinfrastruktur ein entscheidender Faktor bei der Umsetzung des Dissertationsprojektes ist, wurde für die Implementierung der klassenübergreifenden Funktionen AspectJ verwendet. Der etwas höhere Aufwand für die Integration und Definition der Aspekte war aufgrund der besseren Performance und umfangreicheren Funktionalität vernachlässigbar.

2.3.3 Sicherheitstechnologien

Die folgenden Abschnitte beschreiben einige Technologien, die für die Realisierung des Sicherheitskonzepts der entwickelten Forschungsdateninfrastruktur genutzt wurden.

Java Sicherheitsmodell

Die einfache Erweiterbarkeit ist eine der großen Stärken von Java. Neue Klassenbibliotheken und Frameworks können dynamisch geladen werden, um neue Programmcodes zu integrieren. Dies macht es gleichzeitig aber nötig zu prüfen, ob der jeweilige Code sicher ist, um sich vor möglichen Schäden und fremden Zugriffen zu schützen. Das ursprüngliche Java-Sicherheitsmodell war sehr einfach aufgebaut. Lokale Programme wurden grundsätzlich als sicher eingestuft. Code aus fremden Quellen z. B. aus dem Internet wurde hingegen immer in einer geschützten und mit beschränkten Zugriffsrechten ausgestatteten Umgebung, einer sogenannten *Sandbox*, ausgeführt (vgl. Flanagan, 2005).

Policies und Permissions

Mit der Java Version 1.2 wurde ein neues, umfangreicheres Sicherheitskonzept eingeführt, dass sogenannte *Policies* verwendet. Dies sind Sammlungen von Berechtigungen (*Permissions*) für bestimmte Quellen, um einen bestimmten Code auszuführen. Beim Start jeder JVM wird eine Standard-Policy geladen, die einige wenige Berechtigungen zum Auslesen von Systeminformationen enthält. Zusätzliche Policy-Dateien können beim Programmaufruf (`-Djava.security.policy`) oder direkt über die statische Policy Klasse (`java.security.Policy`) im Programmcode geladen werden. Es gibt verschiedene Arten von Berechtigungen, die alle von der Permission Oberklasse (`java.security.Permission`) abgeleitet und generell positiv definiert sind. Sie erlauben immer den Zugriff auf ein Objekt oder die Ausführung einer Funktion. Jede Berechtigung besteht aus dem Namen des Ziels und der erlaubten Aktion. Außerdem werden für jede Berechtigung die Quelle des Codes und eine optionale, digitale Signatur definiert. Ab der Java-Version 1.4 ist es zusätzlich möglich sogenannte Principals (`java.security.Principal`) festzulegen, um Berechtigungen nur für bestimmte Nutzer, die das jeweilige Principal besitzen, freizugeben (siehe Quelltext 2.3).

```
grant signedBy "signer_name", codeBase "URL",
   principal KerberosPrincipal "name", principal NTUserPrincipal "name"{
      permission FilePermission "/files/*", "write";
      permission PropertyPermission "java.version", "read";
   };
```

Quelltext 2.3: Beispiel einer Policy-Datei zur Freigabe von Nutzerberechtigungen

Java SecurityManager

Die Prüfung geladener Policies und der enthaltenen Berechtigungen wird vom *SecurityManager* (`java.lang.SecurityManager`) übernommen. Standardmäßig werden kritische Bereiche wie der Netzwerkverkehr oder der Dateisystemzugriff geschützt. Außerdem muss der SecurityManager explizit durch ein Argument beim Programmaufruf (`-Djava.security.manager`) oder im Programmcode (`System.setSecurityManager()`) aktiviert werden. Intern wird aus jeder gesetzten Berechtigung eine ProtectionDomain (`java.security.ProtectionDomain`) gebildet, die der SecurityManager dann mit der angeforderten Berechtigung der ausführenden Klasse abgleicht. Dafür nutzt er den AccessController (`java.security.AccessController.checkPermission()`), der die oberste und nicht erweiterbare Instanz des Sicherheitssystems bildet und nicht umgangen werden kann. Außerdem ist er die einzige Instanz, welche die Prüfung durch den SecurityManager in Ausnahmefällen aussetzen kann, indem die gewünschte Funktion in einen privilegierten Block gekapselt wird (siehe Quelltext 2.4).

```
public void getPrivilegedAccessToJavaVersion(){
   AccessController.doPrivileged(new PrivilegedAction(){
      public Object run(){
         System.out.print(System.getProperty("java.version");
         return null;
      }
   });
}
```

Quelltext 2.4: Privilegierter Block ohne Prüfung durch den SecurityManager

Java Authentication and Authorization

Mit der Einführung der erwähnten Principals wurde in Java 1.4 auch der Java Authentication and Authorization Service (JAAS) eingeführt, dessen Komponenten im Abbildung 2.6 auf Seite 27 schematisch dargestellt sind. JAAS ermöglicht die flexible Nutzung von austauschbaren Verfahren zur Authentifizierung von Nutzern. Jeder Nutzer wird durch eine Reihe von Principals, die in einem Subject (`java.security.Subject`) zusammengefasst sind, repräsentiert. Um ein solches Subject zu erhalten, muss der Nutzer ein oder mehrere von der Anwendung bereitgestellte Authentifizierungsverfahren, auch *Loginmodule* genannt, erfolgreich ausführen. Die Loginmodule einer Anwendung werden in einer

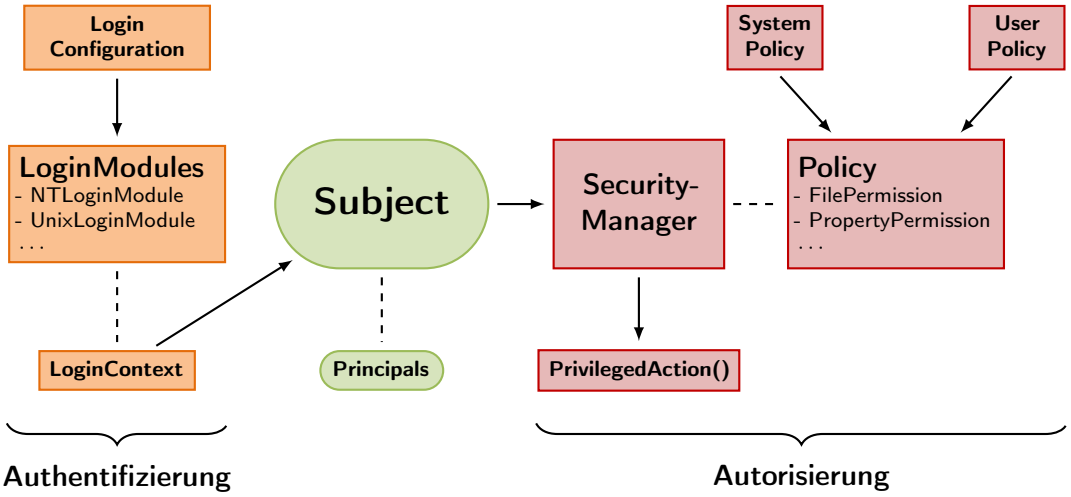

Abbildung 2.6: Schematische Darstellung der Architektur des Java Authentication and Authorization Service. Die Authentifizierungs- und die Autorisierungskomponenten sind gelb bzw. rot gekennzeichnet. Der authentifizierte Nutzer (Subject) ist grün gefärbt.

Konfigurationsdatei (siehe Quelltext 2.5) erfasst und können über ein Argument beim Programmaufruf (`-Djava.security.auth.login.config`) oder direkt im Programmcode (`System.setProperty(java.security.auth.login.config)`) geladen werden.

```
WindowsLogin{
  com.sun.security.auth.module.NTLoginModule required;
}

KerberosLogin{
  com.sun.security.auth.module.Krb5LoginModule required;
}
```

Quelltext 2.5: Auszug aus einer Konfigurationsdatei zur Definition von Loginmodulen

Mit einem LoginContext (`java.security.auth.login.LoginContext`) wird eines der definierten Anmeldeverfahren ausgeführt und der Nutzer identifiziert sich je nach Modul z. B. per Passwort. Die einzelnen Module werden dabei über die gleiche Schnittstelle (`javax.security.auth.spi.LoginModule`) implementiert und ermöglichen so eine flexible Nutzung der zur Verfügung stehenden Authentifizierungsverfahren. So können einfache datei- oder datenbankbasierte Benutzerverzeichnisse oder auch komplexere Authentifizierungsprotokolle, wie z. B. *Kerberos*, genutzt werden, um ein entsprechendes Loginmodul zu entwickeln. Darüber hinaus benötigt jedes Modul einen CallbackHandler (`javax.security.auth.callback.CallbackHandler`), in dem notwendigen Funktionen zur Interaktion mit dem Nutzer implementiert sind. Jeder erfolgreiche authentifizierte Nutzer erhält ein Subject, in dem alle vom Loginmodul bereitgestellten Principals vereint sind und welches dann, wie bereits beschrieben, vom SecurityManager genutzt werden kann, um die Nutzerberechtigung zu prüfen. Dabei wird die zu prüfende Funktion in eine PrivilegedAction (`java.security.PrivilegedAction`) gekapselt (vgl. Oracle, 2018a).

OAuth-Protokoll

Das *OAuth*-Protokoll ist ein offener Standard zur Nutzerauthentifizierung von Client- und Webanwendungen. Die ersten Ideen dafür wurden bereits im Jahr 2006 von Firmen wie Google und Twitter entwickelt. Ziel war es, dass Nutzer einer Webanwendung erlauben, in ihrem Namen auf eine OAuth unterstützte API zuzugreifen, ohne dass die Anwendung den Namen und das Passwort des Nutzers beim Anbieter der API kennen muss. Der große Vorteil ist, dass sich ein Endnutzer mit einem existierenden Account z. B. von Google, bei Anwendungen, die das OAuth-Protokoll unterstützt, anmelden kann, ohne persönliche Daten an die Anwendung preisgeben zu müssen. Ein Account kann so den Zugang zu einer Vielzahl an Anwendungen ermöglichen.

Die erste finale Version des OAuth-Protokolls (Version 1.0) wurde 2010 als RFC-Standard veröffentlicht (Hammer-Lahav, 2010). Bereits zwei Jahre später wurde eine überarbeitete und aktuell akzeptierte Version ebenfalls als RFC-Standard definiert (Hardt, 2012). Ein Hauptgrund dafür war, dass OAuth 1.0 teilweise auf Weiterleitungen im Webbrowser beruht, was eine Integration in native Anwendungen z. B. auf mobilen Geräten schwierig gestaltet, da diese meist keine Browserfenster ansprechen können oder keine Anfragen verarbeiten können. OAuth 2.0 hingegen ist modularer aufgebaut und daher flexibler in der Anbindung an eine Clientanwendung, da es die Definition von sogenannten Clientprofilen ermöglicht bzw. verlangt. Diese definieren dann spezielle Unterprotokolle zur Beschreibung des jeweiligen Anwendungsfalls z. B. für mobile Applikationen (vgl. Boyd, 2012).

Der Ablauf des OAuth-Protokolls, bestehend aus den drei Hauptschritten, ist als UML-Kommunikationsdiagramm in Abbildung 2.7 schematisch dargestellt.

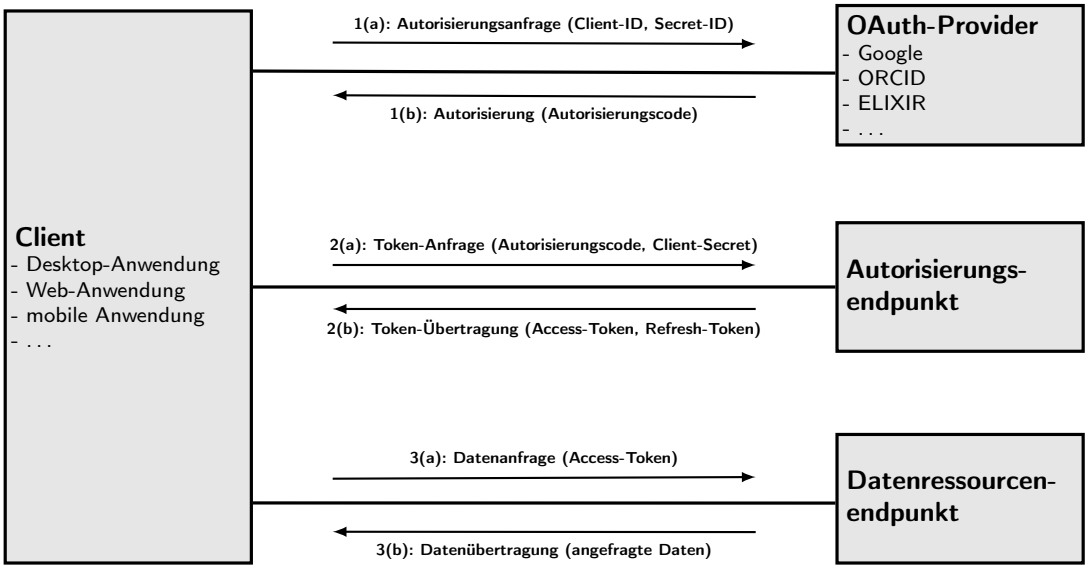

Abbildung 2.7: UML-Kommunikationsdiagramm des allgemeinen OAuth-Protokollablaufs

Im Folgenden wird ein OAuth 2.0 konformer API-Zugriff für eine typische Webanwendung beschrieben, da dies einer der häufigsten Anwendungsfälle für das OAuth-Protokoll ist. Jeder Client, der eine durch OAuth geschützte API ansprechen möchte, muss sich beim jeweiligen Provider registrieren und bekommt eine sogenannte *Client-ID* und ein *Client-Secret*, die

für die Autorisierung notwendig sind. Außerdem muss eine URL hinterlegt werden, auf die Anfrageergebnisse weitergeleitet werden können.

1 - Autorisierung Damit ein Nutzer sich über eine Clientanwendung autorisieren kann, um in seinem Namen auf die gewünschte API zugreifen zu können, stellt er eine Anfrage an die Autorisierungsschnittstelle der API und wird automatisch auf eine Webseite geleitet auf der er sich mit seinem registrierten Account anmeldet. Optional kann eine vom OAuth-Provider definierte Zugriffsebene, auch *scope* genannt, definiert werden, um nur bestimmte Funktionen zu autorisieren. Nach erfolgreicher Autorisierung wird der Nutzer automatisch auf die gewählte URL weitergeleitet. Zusätzlich wird vom OAuth-Provider an die URL ein Autorisierungscode angehängt, welcher spezifisch für den jeweiligen Nutzer und Clientanwendung ist.

```
GET  https://accounts.google.com/o/oauth2/auth?  HTTP/1.1
response_type = code &
client_id = my_client_id &
scope = /read &
redirect_uri = https://www.my_url.de

HTTP/1.1 302 FOUND
Location : https://www.my_url.de?code=12345
```

2 - Access-Token Der übermittelte Autorisierungscode kann anschließend zusammen mit dem Client-Secret des Anwenders genutzt werden, um ein *Access-Token* anzufragen.

```
POST  https://accounts.google.com/o/oauth2/token  HTTP/1.1
content-type = application/json &
client_id = my_client_id &
client_secret = my_client_secret &
grant_type = authorization_code &
code = 12346

HTTP/1.1 200 OK
{"access_token": "abcde",
 "expires_in": 3600,
 "refresh_token": "fghij"}
```

3 - API-Anfrage Mithilfe des erhaltenen Access-Tokens können dann Anfragen an definierte Endpunkte der API gestellt werden, um z. B. das Profil des Nutzers abzufragen. In der Regel hat dieser Token eine zeitlich begrenzte Gültigkeit und kann anschließend mittels des *Refresh-Tokens* erneuert werden.

```
GET  https://api.google.com/user-profile  HTTP/1.1
content-type = application/json &
authorization = Bearer abcde

{"username": "Daniel",
 "email-adress": "arendd@ipk-gatersleben.de",
 "affiliation": "IPK Gatersleben" }
```

2.3.4 Weitere Backend-Technologien

In diesem Abschnitt werden weitere Technologien erläutert, die für die Umsetzung verschiedener Backend-Komponenten der Forschungsdateninfrastruktur genutzt wurden.

Remote Method Invocation (RMI)

Mit der Remote Method Invocation (RMI)-API (`java.rmi.*`) bietet Java die Möglichkeit, dass Programme, die in verschiedenen JVMs auf lokalen oder entfernten Rechner ausgeführt werden, miteinander kommunizieren können. So kann eine lokale Anwendung Funktionen einer entfernten JVM ausführen und eine Server-Client-Infrastruktur realisiert werden, bei der ein Rechner als Server fungiert und seine Funktionen entfernten Clients über eine Netzwerkverbindung zur Verfügung stellt (vgl. Oracle, 2018e). Die grundlegende Funktionsweise einer auf RMI-basierten Server-Client-Architektur ist in Abbildung 2.8 schematisch dargestellt und wird nachfolgend beschrieben.

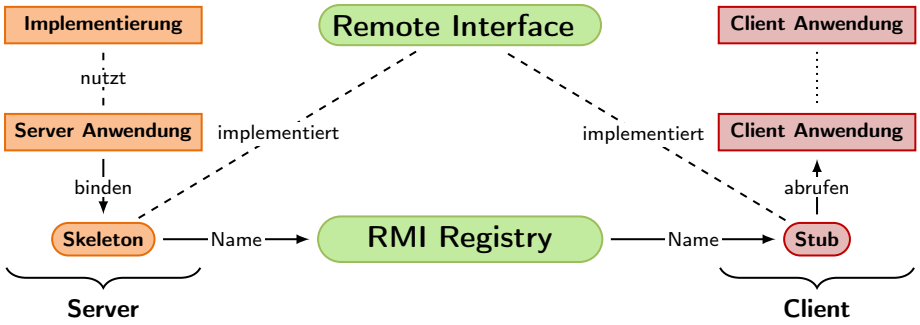

Abbildung 2.8: Schematische Darstellung der RMI-basierten Server-Client-Architektur. Die server- bzw. clientseitigen Komponenten sind gelb bzw. rot gefärbt. Die grünen Elemente stellen die Komponenten der RMI-API dar.

RMI-Server Der Rechner mit der RMI-Server Instanz stellt die per Fernzugriff ausführbaren Methoden bereit und implementiert diese. Alle Funktionen, die von einem Client genutzt werden können, müssen das Remote-Interface (`java.rmi.Remote`) implementieren. Damit kann der Server Remote-Objekte, sogenannte *Skeletons* erzeugen und mit einem eindeutigen Namen in der RMI-Registrierung (`java.rmi.Registry`) anmelden. Zur Kommunikation mit entfernten Rechnern muss beim Initialisieren der RMI-Registrierung ein freier Netzwerkport bereitgestellt werden (siehe Quelltext 2.6).

```
public class Person implements Remote{
   private String name;
   public String getName() throws RemoteException {
       return this.name;
   }
}
public static void main(String args[]) {
   Registry reg = LocateRegistry.createRegistry(1099)
   Person personSkeleton = new Person("Daniel","Arend");
   reg.bind("arendd", personSkeleton);
}
```

Quelltext 2.6: RMI-Server Initialisierung und Registrierung eines Remote-Objektes

RMI-Client Damit sich ein Client mit der RMI-Registrierung des Servers verbinden kann, ist die Netzwerkadresse und der Port der Servers notwendig. Über die vergebenen Namen können die hinterlegten Skeletons abgerufen werden. Der Client erhält dann einen sogenannten *Stub*, eine Objektreferenz auf das Remote-Objekt, das ebenfalls die Remote-Schnittstelle implementiert. Dadurch kann der Nutzer alle bereitgestellten Funktionen aufrufen, als ob er direkt mit den konkreten Objekten arbeiten würde. Die Ausführung der implementierten Funktionen erfolgt dann auf dem Server und nur eventuelle Rückgabewerte werden zum Client übertragen (siehe Quelltext 2.7).

```
public class RmiClient {
...
  public static void main(String[] args) throws RemoteException {

     String rmiServer = "192.123.456.789";
     int rmiPort = 1099;
     Registry reg = LocateRegistry.getRegistry(rmiServer,rmiPort);
     Person personStub = (Person) reg.lookup("arendd");
     System.out.println("Name: " + personStub.getName());
  }
}
```

Quelltext 2.7: RMI Client Initialisierung und Abrufen eines Remote-Objektes

Beim Aufruf von Methoden müssen die an den Server übertragenen Parameter bzw. die an den Client gesendeten Rückgabewerte erst in einen Bytestrom und anschließend auf der Server- bzw. Clientseite wieder in den originalen Datentyp umgewandelt werden. Für primitive Datentypen geschieht dies automatisch. Komplexere Datentypen wie z. B. Dateiströme, Listen oder auch eigene Datentypen müssen erst die bereits erwähnte Serializable-Schnittstelle implementieren, um serialisiert und übertragen zu werden. Dafür gibt es jedoch bereits einige umfangreiche und nützliche Bibliotheken, welche die dafür nötigen Funktionen liefern, wie z. B. die RMIIO-Bibliothek (OpenHMS, 2018), welche die Serialisierung und Übertragung von Dateiobjekte über RMI ermöglicht.

Apache Lucene, Solr und Tika

Das bereits erwähnte Apache Lucene Framework (siehe Abschnitt 2.3.1) ist eine umfangreiche Java-Bibliothek zur Informationsbeschaffung und ermöglicht eine textbasierte Indexierung und Suche. Da es mit allen textbasierten Daten umgehen kann, ist es vielseitig einsetzbar und eignet sich für die Suche in Dokumenten, Webseiten und Datenbanken. Es bietet dafür eine eigene Anfragesprache (vgl. Hatcher et al., 2010), die vielfältige Suchoptionen bereitstellt. Durch seine hohe Flexibilität, schnelle Performance, sowie gute Skalierbarkeit eignet es sich auch für sehr große Infrastrukturen mit riesigen Datenvolumen, wie z. B. Twitter (vgl. Busch et al., 2012).

Eine weitere nützliche Bibliothek, die im Rahmen desselben Projektes entwickelt wird, ist Apache Solr (Grainger und Potter, 2014). Diese baut direkt auf Lucene auf und ermöglicht die Einbindung der Index- und Suchfunktionalität in einen Servlet-Container, um so Anfragen über eine HTTP-Schnittstelle entgegen zu nehmen. So können Suchanfragen mit Unterstützung der Lucene eigenen Anfragesprache an einen Webserver gestellt und ausgewertet werden.

Die dritte umfangreiche Bibliothek, die ursprünglich ebenfalls Bestandteil des Lucene Projektes war und mittlerweile in einem eigenständigen Projekt entwickelt wird, ist Apache Tika (Mattmann und Zitting, 2011). Diese ist darauf spezialisiert, Inhalte und Metadaten von einer Vielzahl an bekannten Dateiformaten auszulesen und zu analysieren. Daraus ergeben sich eine Vielzahl an Anwendungsmöglichkeiten, wie die Analyse von Dokumenten zur Suchmaschinen-Indexierung oder der Bestimmung von Dateitypen.

Quartz API

Die Quartz-API ist eine Java-Bibliothek zur Umsetzung von zeitgesteuerten Prozessen (Terracotta Inc., 2018). So können Funktionen zur Laufzeit einer Anwendung in einem definierten Intervall wiederholend ausgeführt werden. Der Vorteil der Quartz-API ist die einfache und übersichtliche Implementierung. Zur Einrichtung eines Prozesses werden die drei Komponenten *Job*, *Trigger* und *Scheduler* benötigt:

Job Ein Job ist eine Klasse, welche die `org.quartz.Job` Schnittstelle implementiert. Sie enthält Funktionenen, die in einem bestimmten Intervall ausgeführt werden sollen. Ein definierter Job wird dann an ein `JobDetail` Objekt gebunden.

```
public class MyJob implements org.quartz.Job {

  @Override
  public void execute(JobExecutionContext context) throws JobExecutionException{
        System.out.println(System.currentTimeMillis());
  }

  public static void main(String[] args) {
        JobDetail job = JobBuilder.newJob(MyJob.class).build();
  }
}
```

Trigger Ein Trigger (`org.quartz.Trigger`) definiert den Startpunkt und das Intervall, in dem ein Job ausgeführt wird. Dabei gibt es verschiedene Arten von Triggern, die z. B. in einem fest definierten Zeitintervall, etwa alle 24 Stunden einen Prozess ausführen oder Trigger, die immer zu bestimmten Zeitpunkten, etwa am letzten Tag eines Monats, eine Funktion ausführen.

```
public static void main(String[] args) {
   Trigger trigger = TriggerBuilder.newTrigger().startNow()
       .withSchedule(SimpleScheduleBuilder.repeatHourlyForever().build();
}
```

Scheduler Der Scheduler (`org.quartz.Scheduler`) ist die zentrale Komponente, die einen definierten Job mit einem bestimmten Trigger verbindet und die zeitgesteuerte Prozesskette initiiert.

```
public static void main(String[] args) {
     Scheduler scheduler = new StdSchedulerFactory().getScheduler();
     scheduler.start();
     scheduler.scheduleJob(job, trigger);
}
```

2.3.5 Webtechnologien

Im folgenden Abschnitt werden einige Webtechnologien vorgestellt, die für verschiedene Komponenten der Forschungsdateninfrastruktur wie beispielsweise zur Implementierung von Funktionen zur Kommunikation mit externen APIs und Diensten verwendet wurden.

REST-Webservices

Das *Representational State Transfer (REST)*-Paradigma dient zur Programmierung von im Internet bereitgestellten Diensten, sogenannten Webservices. Diese werden zur automatisierten Kommunikation und zum Datenaustausch zwischen Maschinen genutzt. Die REST-Architektur (Fielding, 2000) ist generell unabhängig von bestimmten Protokollen. Sie wird jedoch üblicherweise als Erweiterung des etablierten HTTP implementiert und betrachtet grundsätzlich Ressourcen, die vier allgemeine Anforderungen erfüllen.

- **Adressierbarkeit**:
  Jede Ressource ist über eine eindeutige URI identifizierbar.
- **Zustandslosigkeit**:
  Es werden keine Sitzungsinformationen zwischen Webservice und Nutzer gespeichert.
- **Schnittstellen**:
  Jede Ressource kann über eine Auswahl an standardisierten Methoden abgerufen werden.
- **Repräsentation**:
  Eine Ressource kann über unterschiedliche Repräsentationen verfügen.

REST-Webservices nutzen daher die bekannten Standardmethoden des HTTP-Protokolls (vgl. Richardson und Ruby, 2008):

- **GET**: liest eine Ressource unverändert vom Server
- **POST**: erstellt eine neue Ressource, die noch keine URI hat
- **PUT**: erstellt eine neue Ressource oder verändert eine existierende Ressource
- **PATCH**: ändert Teile einer existierenden Ressource
- **DELETE**: löscht eine existierende Ressource

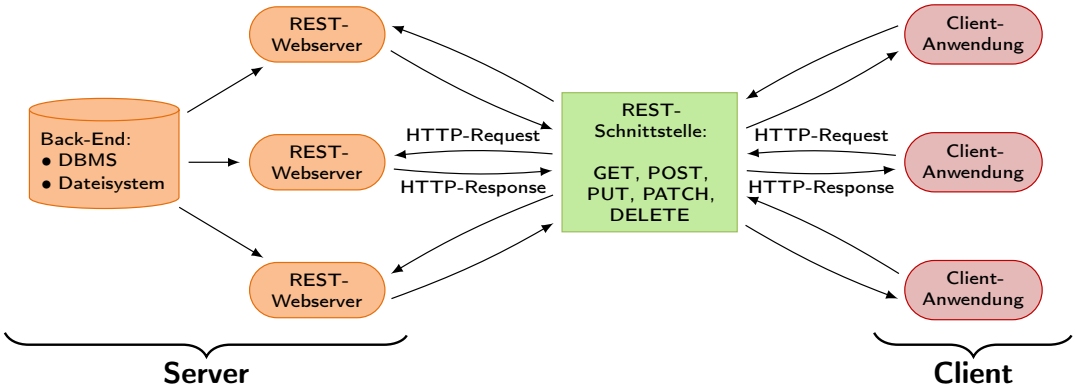

Abbildung 2.9: Schematische Darstellung der Architektur eines REST-Webservices. Die server- bzw. clientseitigen Komponenten sind gelb bzw. rot gefärbt. Die REST-Schnittstelle ist grün gekennzeichnet.

Die Architektur eines REST-Webservices ist in Abbildung 2.9 auf Seite 33 schematisch dargestellt. Die zustandslose Kommunikation zwischen dem Server, der die Ressourcen bereitstellt und verwaltet und dem Client, der die Ressourcen abfragt oder bearbeitet, gewährleistet eine einfache Skalierbarkeit des Systems. Da keine Informationen über die Sitzung eines Clients gespeichert werden, können mehrere aufeinander folgende Client-Anfragen auch von mehreren Servern verarbeitet werden können. Des Weiteren zeichnen sich REST-Webservices durch ihre hohe Flexibilität aus, da jede Ressource über verschiedene Repräsentationen, z. B. als JavaScript Object Notation (JSON)-, Extensible Markup Language (XML)- oder Textdokument abgerufen werden können. Durch die standardisierten Methoden und die einheitliche Adressierung von Ressourcen können verschiedenste Clientanwendungen implementiert werden, die den gleichen REST-Webservice nutzen.

Java API für REST Services

Die Java API for RESTfull Webservices (JAX-RS) (`javax.ws.rs.*`) bietet sowohl standardisierte Funktionen zur Nutzung eines REST-Webservices als auch Methoden zur Einrichtung einer REST-Serverarchitektur (Oracle, 2018b). Dafür nutzt JAX-RS eigene Annotationen im Programmcode, die nicht ausgeführt werden, jedoch zur Laufzeit einer Anwendung abgefragt und vom Compiler überprüft werden (vgl. Burke, 2009).

Es existieren verschiedene Frameworks, welche die Spezifikationen von JAX-RS implementieren. Die gebräuchlichste und umfangreichste Referenzimplementierung liefert das Jersey-Softwarepaket (Oracle, 2018c). Es ermöglicht eine einfache Integration von REST-Webservices innerhalb von Java-Anwendungen sowie die Implementierung von Clientanwendungen zur Nutzung zur Kommunikation mit existierenden REST-Endpunkten.

Jetty Webserver

Jetty ist ein umfangreicher Java-basierter Webserver und Servlet-Container. Neben der klassischen Nutzung als eigenständiger Webserver, der Anfragen entgegen nimmt, Anwendungen ausführt und Ergebnisse zurückliefert, kann Jetty auch vollständig als integrierter Server in existierende Anwendung eingebunden werden. Dies ermöglicht Anwendungen, HTTP-Anfragen von Clients entgegen zu nehmen. Innerhalb der Anwendung können dann sogenannte *Handler* implementiert werden, die die entgegengenommen Anfragen auswerten, verarbeiten und gegebenenfalls Daten zurückliefern oder Filter definiert werden, um Anfragen zu sortieren. Außerdem bietet Jetty bereits eine Reihe von fertigen Handlern und Filtern, um beispielsweise Anfragen zu loggen oder bestimmte Anfragen über einen Proxy umzuleiten. Der große Vorteil von Jetty ist die schlanke Implementierung und gute Skalierbarkeit (Eclipse Foundation, 2017).

2.3.6 Frontend-Technologien

Im letzten Abschnitt werden wichtige Frontend-Technologien vorgestellt, die hauptsächlich für die grafischen Komponenten und Nutzeroberflächen, wie beispielsweise das Webseitenlayout genutzt wurden.

Velocity Template Engine

Die Velocity Template Engine (Velocity, 2017) ist eine umfangreiche Java-Bibliothek, die es ermöglicht statische Vorlagen (*Templates*) mit dynamischen Variablen zu befüllen, um verschiedene Inhalte z. B. Dokumente, Webseiten oder Programmcodes zu generieren. Das hat den Vorteil, dass z. B. beim Erstellen einer Webseite statt vieler ähnlicher Seiten nur einige wenige Templates gespeichert werden müssen, die dann dynamisch nach Bedarf mit Inhalten gefüllt werden. Das erleichtert die Pflege und spart Speicherplatz. Velocity bietet mit der Velocity Template Language (VTL) eine eigene Sprache, die die Generierung vielfältiger Dokumente ermöglicht.

```
<html>
 <body>
  Tabelle aller Mitarbeiter des $institute
  <table>
   <tr>
    <th>Name</th>
    <th>Arbeitsgruppe</th>
    <th>Telfonnummer</th>
   </tr>
   #foreach( $p in $persons )
    <tr>
     <td>$p.getName()</td>
     <td>$p.getGroup()</td>
     <td>$p.getPhone()</td>
    </tr>
   #end
  </table>
 </body>
</html>
```

Quelltext 2.8: VeloCity-Template für eine Webseite mit Mitarbeiter Tabelle „website.vm“

Quelltext 2.8 zeigt ein Beispiel für ein Template einer Hypertext Markup Language (HTML)-Seite zur Abbildung einer Tabelle. Die grundlegende Dokumentenstruktur ist statisch, die Inhalte der abgebildeten Tabelle sind jedoch dynamisch und richten sich nach den Werten, die mit den Variablen übergeben werden. Diese beginnen mit einem „$“ und können Funktionen wie in Java aufrufen. So führt z. B. „$p.getName()“ die Funktion `getName()` für das Objekt hinter Variable „$p“ aus. Darüber hinaus bietet die VTL eine Reihe von Konstrukten, die aus üblichen Programmiersprachen bekannt sind, wie z. B. Bedingungen oder Schleifen, um dynamische Templates zu definieren. So werden im genannten Beispiel alle Objekte in der Variable „$persons“ in einer Schleife durchlaufen und jeweils in eine neue Zeile in der Tabelle eingetragen.

```
List<Person> persons = new ArrayList<Person>();

persons.add(new Person("Daniel Arend","BIT",842))
persons.add(new Person("Matthias Lange","BIT",693))
...
VelocityContext context = new VelocityContext();
context.put("instiute","IPK Gatersleben");
context.put("persons",persons);
...
FileWriter ausgabe = new FileWriter("mitarbeiter.html");
Velocity.mergeTemplate("website.vm",context,ausgabe);
ausgabe.flush()
```

Quelltext 2.9: Erstellung und Belegung eines `VelocityContext` und Einbau in ein definiertes Template zur Erzeugung eines Dokumentes

Um aus dem Template ein Dokument zu erstellen, müssen die verwendeten Variablen in einem VelocityContext (`org.apache.velocity.VelocityContext`) erfasst werden. Anschließend werden sie wie in Quelltext 2.9 auf Seite 35 gezeigt, dynamisch in das Template eingebaut und in eine Datei geschrieben. Dabei ist zu beachten, dass die übergebenen Java Objekte, bzw. deren aufgerufene Funktionen im Template eine Zeichenkette zurückliefern müssen, die in das Dokument eingebaut werden kann.

GeoIP2-API

Die GeoIP2-API (MaxMind Inc., 2018a) ist ein Java-API, die in der Lage ist die Webdienste und Datenbanken des GeoIP2-Services zu nutzen. Diese ermöglichen es bestimmte Informationen anhand von IP-Adressen aus einer von der Firma MaxMind (MaxMind Inc., 2018b) gepflegten Datenbank zu beziehen. So kann die Herkunft, z. B. das Land einer IP-Adresse, ermittelt und entsprechenden Längen- und Breitengraden zugeordnet werden. Während die Nutzung der aktuellsten Datenbank kostenpflichtig und an den Erwerb einer Lizenz gebunden ist, bietet MaxMind auch eine kostenfreie Version ihres Services an. Dieser hat zwar einen geringeren Funktionsumfang und verwendet eine ältere Version der IP-Datenbank, reicht jedoch für viele Anwendungsfälle aus.

2.3.7 Softwareentwicklung

Da die Entwicklung einer umfangreichen Software sehr komplex ist und die Wartung und Behebung von Fehlern mit zunehmender Größe immer aufwendiger wird, gibt es im Bereich der Softwaretechnik eine Reihe von Vorgehensmodellen. Diese organisieren den Prozess der Entwicklung, um ihn überschaubarer zu gestalten und durch eine bessere Strukturierung und Kontrolle der einzelnen Abschnitte von vornherein eine gewisse Qualitätskontrolle zu ermöglichen. In diesem Abschnitt werden drei bekannte und weitverbreitete Modelle erläutert. Diese zählen zu den klassischen Verfahren der Softwareentwicklung. Darüber hinaus gibt es auch noch eine Vielzahl an neueren Modellen, die unter dem Dach der *agilen Softwareentwicklung* zusammengefasst werden. Ziel dieser Verfahren ist es vor allem in komplexen Softwareprojekten, an denen sehr viele Entwickler beteiligt sind, durch Aufweichen der strikten Prozeduren der klassischen Vorgehensmodelle eine höhere Flexibilität besonders innerhalb der Gruppe der Entwickler zu ermöglichen (Dybå und Dingsøyr, 2008). Da die Entwicklung der im Rahmen der Dissertation entworfenen Forschungsdateninfrastruktur eher nach einer klassischen Vorgehensweise implementiert wurde, wird an dieser Stelle auf eine detailliertere Beschreibung agiler Vorgehensmodelle verzichtet.

Wasserfall-Modell

Das sequenzielle Wasserfall-Modell (Boehm, 1981; Royce, 1987) ist eines der ältesten Modelle. Es beschreibt ein nicht-iteratives Vorgehen und ist in Abbildung 2.10 auf Seite 37 schematisch in Form des namensgebenden Wasserfalls dargestellt. Jeder einzelne Abschnitt ist ein bindendes Ereignis für den folgenden Abschnitt. Es kann jeweils nur ein Schritt zurückgegangen werden. Falls im Laufe der Entwicklung komplett neue Anforderungen an die Software auftauchen, muss ein neuer vollständiger Zyklus durchlaufen werden.

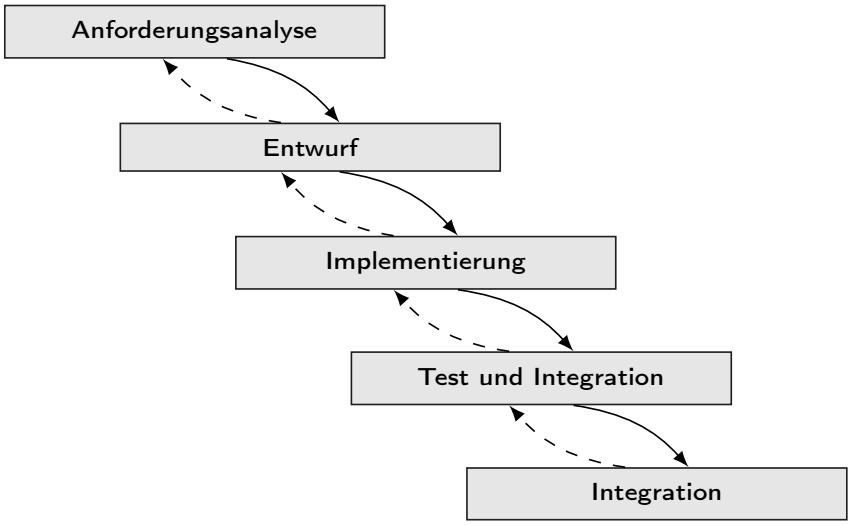

Abbildung 2.10: Schematische Darstellung des Ablaufs des Wasserfall-Modells

Dieses Vorgehensmodell ist daher insbesondere sinnvoll, wenn die Anforderungen an eine Software präzise formuliert und festgelegt sind. Dadurch ermöglicht das Wasserfall-Modell eine hohe Planungssicherheit und Softwarequalität. Aufgrund des sequenziellen Ablaufes sind Änderungen oder Korrekturen zu einem späteren Zeitpunkt jedoch sehr aufwendig. Außerdem werden Fehler häufig erst sehr spät erkannt.

V-Modell

Das V-Modell (Boehm, 1984) ist eine Weiterentwicklung des Wasserfall-Modells und definiert neben den einzelnen Entwicklungsschritten eine Reihe von Testschritten zur Sicherstellung der Softwarequalität. Das Schema in Abbildung 2.11 zeigt den Ablauf des V-Modells. Jeder Entwicklungsphase steht eine entsprechende Testphase, deren Ablauf schon während der Entwicklung definiert wird, gegenüber. Dadurch wird eine hohe Testabdeckung gewährleistet.

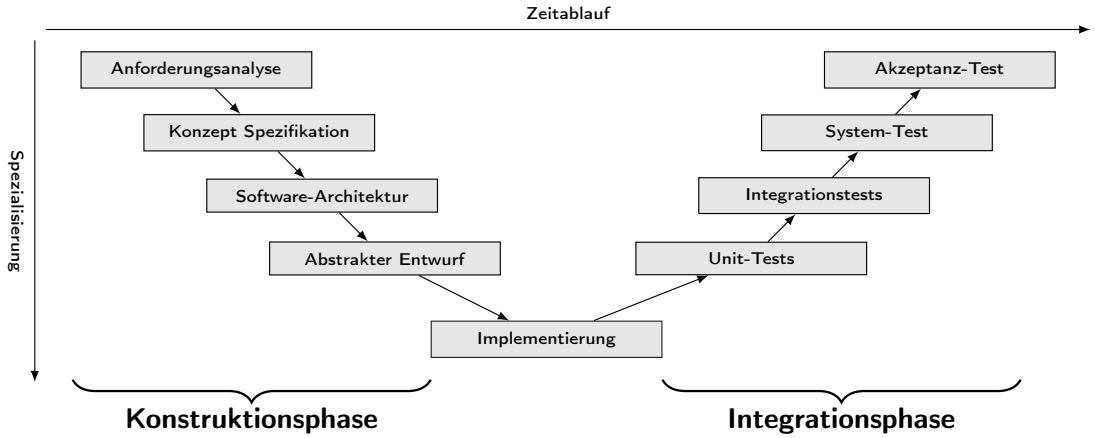

Abbildung 2.11: Schematische Darstellung des Ablaufs des V-Modells

Auch das V-Modell beschreibt ein sinnvolles Vorgehen bei der Entwicklung von Software mit einem klar vorgegebenen Anforderungsprofil. Zusätzlich bietet es aufgrund der hohen Testabdeckung weniger Risiken bei der Entwicklung. Jedoch ist auch hier die eingeschränkte Flexibilität in Bezug auf Anpassungen und das Finden und Beheben von Fehlern durch das nicht-iterative Vorgehen ein entscheidender Nachteil.

Spiral-Modell

Das Spiral-Modell (Boehm, 1988) beschreibt im Gegensatz zu den beiden anderen Modellen ein iteratives Vorgehen und eine schrittweise Entwicklung, wobei der Beginn eines nachfolgenden Schrittes erst durch Validierung des vorherigen Schrittes erfolgt. Dabei können beispielsweise Unit-Tests auch parallel zu Implementierung der eigentlichen Funktion entwickelt werden. Abbildung 2.12 zeigt eine vereinfachte schematische Darstellung des Entwicklungsablaufs gemäß des Spiral-Modells.

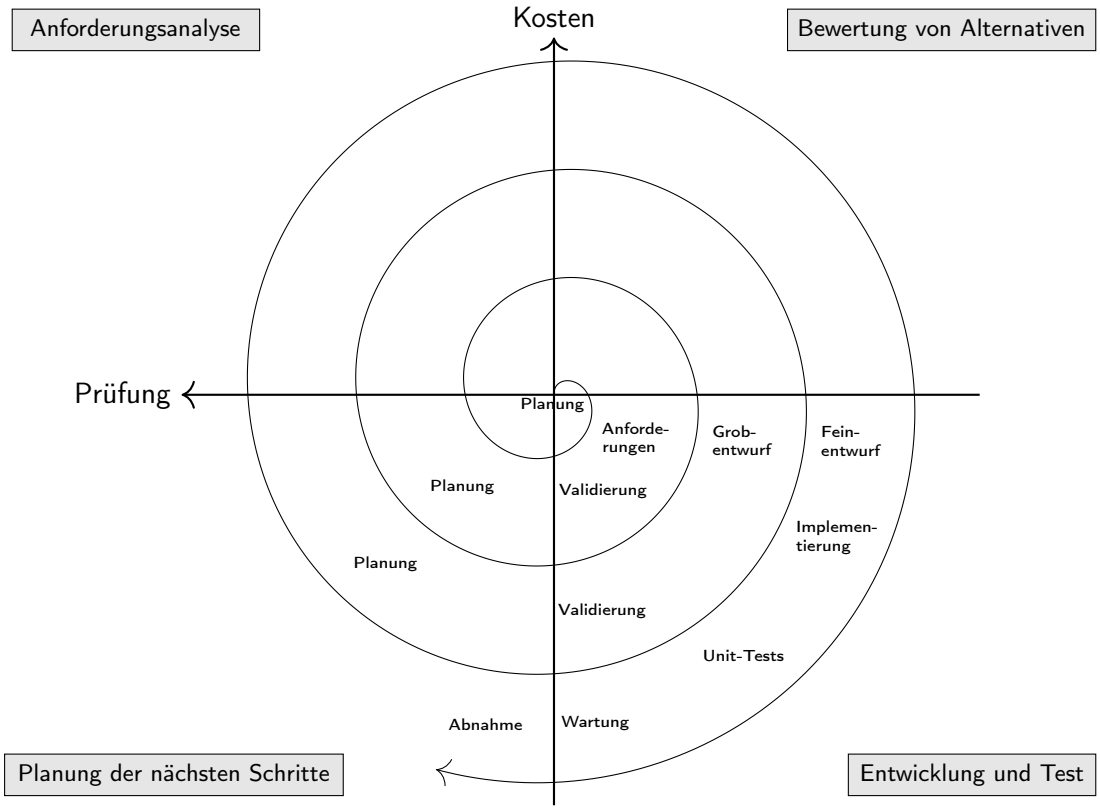

Abbildung 2.12: Schematische Darstellung des Ablaufs des Spiral-Modells

Das Spiral-Modell ist weniger eine Weiterentwicklung des Wasserfall-Modells sondern eher eine iterative Variante davon, da jede einzelne Phase des Modells mehrfach durchlaufen wird. Der größte Vorteil dieser Herangehensweise ist, dass Änderungen oder Erweiterungen des Konzeptes in jeder Entwicklungsphase möglich sind, auch wenn dadurch unter Umständen größere Anpassungen an bereits implementierten Funktionen notwendig sind. Dadurch wird das Gesamtrisiko bzw. die Wahrscheinlichkeit, dass die am Ende entwickelte Software fehlerhaft ist, minimiert, da Fehler früher aufgedeckt und behoben werden können.

3 Systeme zum Forschungsdatenmanagement

Mit der zunehmenden Menge an täglich produzierten Forschungsdaten hat sich nicht nur die Bedeutung etablierter Systeme und Plattformen zum Austausch der Daten erhöht, sondern auch eine Vielzahl an neuen Datenbanken und Informationssystemen entwickelt (vgl. Kılıç et al., 2016). Gleichzeitig werden aber auch andere existierende Infrastrukturen und Anwendungen, die nicht ursprünglich zum Austausch von wissenschaftlichen Daten entwickelt wurden, genutzt.

In diesem Kapitel werden einige existierende Systeme und Plattformen, die häufig zur Verwaltung und Veröffentlichung von Forschungsdaten genutzt werden, beschrieben. Aufgrund der Fülle an genutzten Informationssystemen und Datenbanken handelt es sich hierbei keinesfalls um eine vollständige Übersicht. Es werden Systeme vorgestellt, die während des Dissertationsprojekts und der Literaturrecherche häufig auftraten und in vielen Domänen benutzt werden. Am Ende des Kapitels wird eine Übersicht der jeweiligen Stärken und Schwächen dargestellt. Die Untersuchung und Einschätzung der verschiedenen Systeme hatte Einfluss auf die Entwicklung der in dieser Arbeit vorgestellten Infrastruktur zur Verwaltung und Publikation von Forschungsdaten.

3.1 Domänen- und Projektspezifische Informationssysteme

Im Allgemeinen sind domänen- und projektspezifische Archive und Datenbanken weitverbreitete und nützliche Informationssysteme in der wissenschaftlichen Gemeinschaft (vgl. Bastow und Leonelli, 2010). Außerdem sind sie nicht nur zur Veröffentlichung von produzierten Forschungsdaten und damit oft zur Unterstützung einer wissenschaftlichen Publikation und der darin enthaltenen Ergebnisse wichtig, sondern dienen gleichzeitig auch als eine wichtige Ressource und Ausgangspunkt für weiterführende und auf den Datensätzen aufbauende Forschungen (vgl. Kılıç et al., 2016; Gregory et al., 2018). Sie ermöglichen es, spezielle Forschungsdaten effektiv zu erfassen und zu verteilen, da sie optimal auf die Bedürfnisse der jeweiligen Domäne und deren Nutzer angepasst sind (vgl. Assante et al., 2016). Die gespeicherten Daten durchlaufen gewöhnlich einen umfangreichen Kurationsprozess, bevor sie in das jeweilige System aufgenommen werden (Leonelli et al., 2013). Tabelle 3.1 auf Seite 40 zeigt exemplarisch eine Übersicht etablierter Archive aus verschiedenen Domänen der Lebenswissenschaften.

Alle gelisteten Datenbanken sind auf eine bestimmte Klasse von Daten bzw. Dateiformaten begrenzt. Das sind in der Regel in dieser Domäne etablierte Formate, die im Laufe der Zeit entstanden sind und mithilfe der wissenschaftlichen Gemeinschaft dauerhaft weiterentwickelt werden. Oft sind sie an gebräuchliche Software zur Aufbereitung, Analyse und

Namen / URLs	Datentypen	Formate	Datenvolumen
Sequence Read Archive www.ncbi.nlm.nih.gov/sra (Leinonen et al., 2010)	genomische Daten (DNA reads)	FASTQ, BAM	>1000 TB Sequenzdaten (Kodama et al., 2012)
Gene Expression Omnibus www.ncbi.nlm.nih.gov/geo (Edgar et al., 2002)	Genexpressionsdaten (RNA-Seq,Mirco-Arrays)	Tabellen, TXT, XML	32.000 Studien (Barrett et al., 2013)
UniProt www.uniprot.org (Apweiler et al., 2004)	proteomische Daten (Proteinsequenzen)	FASTA, XML	>47 Mio. Sequenzen (Cook et al., 2016)
MetaboLights www.ebi.ac.uk/metabolights (Haug et al., 2013)	metabolomische Daten und Studien	ISA-Tab	>1000 Assays (Salek et al., 2013)
BioModels Database www.ebi.ac.uk/biomodels (Le Novère et al., 2006)	Biologische Modelle	SBML	>1200 Modelle (Juty et al., 2015)

Tabelle 3.1: Beispiele populärer domänenspezifischer Datenbanken und Archive

Visualisierung gebunden. Viele der bereitgestellten Funktionen der jeweiligen Informationssysteme, wie beispielsweise spezielle Suchfunktionen sind an spezifische semantische Metadaten in dieser Forschungsdomäne und damit an den Bedarf der jeweiligen Community angepasst. Im Allgemeinen werden diese Archive von Fachjournalen als gängige Plattformen zur Bereitstellung von Forschungsdaten akzeptiert und sind oftmals Voraussetzung für die Akzeptanz einer Publikation (vgl. Ball et al., 2004; Alsheikh-Ali et al., 2011).

Handelt es sich bei den Forschungsdaten, die veröffentlicht werden sollen, um komplexe, heterogene und domänenübergreifende Datensätze, die Informationen bzw. Dateien aus verschiedenen Prozessen und Analysen enthalten, ist es häufig notwendig, die Daten zu separieren und in verschiedene Datenbanken zu transferieren. Nachteilig dabei sind die teilweise sehr verschiedenen Dateiformate und häufig proprietären Metadatenschemata, welche die einzelnen Systeme benutzen. Dadurch ist die effiziente Beschreibung der Daten oft sehr komplex und zeitintensiv (vgl. Leonelli et al., 2017). Außerdem bietet jedes System andere Programm- oder Webschnittstellen zum Import von Datensätzen, wodurch die Übertragung unter Umständen ebenfalls sehr zeitaufwendig ist. Bei Datensätzen mit sehr großen Volumen kommt zusätzlicher Zeitaufwand durch den Upload über das Internet hinzu. Neben speziellen Metadatenformaten werden in der Regel auch eigene ID-Systeme benutzt, um Datensätze zu identifizieren. Dadurch ist eine dauerhafte Referenzierung jedoch nicht immer gewährleistet, da sich die Systeme zur Auflösung der IDs meist stark unterscheiden (vgl. Klump und Huber, 2017) und hin und wieder auch IDs geändert werden.

Es gibt zwar eine sehr große Anzahl domänenspezifischer Datenbanken (vgl. Marcial und Hemminger, 2010), jedoch haben sich in einigen jüngeren Forschungsdomänen, beispielsweise im Bereich der phänotypischen Daten (siehe Abschnitt 2.1), bisher kaum öffentlichen Datenbanken und Informationssysteme etabliert (vgl. Coppens et al., 2017), sodass andere Systeme zum Austausch von Forschungsdaten genutzt werden müssen.

3.2 Cloud-basierte Datenaustauschdienste

Eine populäre Alternative zum flexiblen Austausch von Daten sind die sogenannten Cloud-Speicherdienste. Diese haben sich in den letzten Jahren stark weiterentwickelt und sich sowohl auf Desktop-Systemen, also auch auf mobilen Endgeräten, etabliert (vgl. Drago et al., 2012). Auch wenn sie ursprünglich nicht speziell zum Austausch von Forschungsdaten entworfen wurden, werden sie in der Praxis jedoch häufig dafür benutzt. In Tabelle 3.2 ist eine Übersicht einiger bekannter Dienste dargestellt.

Namen / URLs	Plattformen	freie Speicherkapazitäten
Dropbox www.dropbox.com	Windows, Linux, macOS, Android, iOS, Windows Phone	ab 2 Gigabyte
Google Drive www.google.com/drive	Windows, Linux, macOS, Android, iOS, Windows Phone	ab 15 Gigabyte
Microsoft OneDrive onedrive.live.com	Windows, macOS, Android, iOS, Windows Phone	ab 15 Gigabyte
Box www.box.com	Windows, macOS, Android, iOS, Windows Phone	ab 10 Gigabyte

Tabelle 3.2: Beispiele populärer Cloud-basierter Datenaustauschdienste

Ein großer Vorteil und wichtiger Grund für die wachsende Beliebtheit von Cloud-Speicherdiensten ist ihre einfache Einrichtung und Nutzbarkeit. Für nahezu jedes verbreitete Betriebssystem gibt es entsprechende Clientanwendungen, die den Zugang zum eigenen Speicher ermöglichen. Darüber hinaus besitzen sie meistens keine Beschränkungen in Bezug auf die Art und Größe der abgelegten Daten und bieten eine rudimentäre Versionsverwaltung, die es erlaubt den Versionsverlauf einer Datei über einen begrenzten Zeitraum zu verfolgen. Dabei gibt es jedoch die Einschränkung, dass je nach Anbieter nur eine begrenzte Speicherkapazität frei zur Verfügung steht. Wird weiterer Speicherplatz benötigt, fallen in der Regel Kosten an. Über einfache Dateifreigaben können einzelne Dateien oder Verzeichnisse mit anderen Nutzern ausgetauscht werden. Mittels gemeinsam nutzbarer Links können Daten auch vollständig öffentlich freigegeben werden.

Eine dauerhafte Verfügbarkeit der Daten ist dadurch jedoch nicht gewährleistet, da diese Dateifreigaben beispielsweise durch versehentliches Löschen oder Verschieben von Objekten ungültig werden können. Befinden sich die Daten im Bereich des kostenpflichtigen Speicherplatzes, kann es ebenfalls vorkommen, dass diese nicht mehr abrufbar sind, wenn der Besitzer beispielsweise seinen Vertrag auslaufen lässt. Ein weiterer Nachteil ist das unstrukturierte Ablegen von Daten. Da die einzelnen Dienste keine effiziente Möglichkeit zur Beschreibung der abgelegten Dateien mit Metadaten bieten, ist die Auffindbarkeit bzw. effiziente Wiederverwendung oft eingeschränkt und hängt direkt vom Forscher ab, der die Daten bereitstellt. Da es sich außerdem um einen reinen Speicher- und Austauschdienst handelt, gibt es keine Prüfung und Begutachtung der hochgeladenen Daten. Jeder Nutzer kann frei entscheiden welche Daten er bereitstellt ohne das dabei irgendwelche Qualitätsstandards eingehalten werden müssen und so auch keine Nachhaltigkeit gewährleistet ist. Daher eignen sich Cloud-basierte Datenaustauschdienste zwar zum einfachen und schnellen Datenaustausch beispielsweise innerhalb einer Forschungsgruppe, jedoch sind sie zur langfristigen und nachhaltigen Archivierung und Publikation von Forschungsdaten ungeeignet.

3.3 Versionsverwaltungssysteme

Versionsverwaltungssysteme sind eine weitere Möglichkeit, um Forschungsdaten zu verwalten und auszutauschen. Sie erfassen jede Änderung an einer Datei und können diese wiederherstellen, sodass der vollständige Prozess der Entstehung und Verarbeitung eines Datensatzes nachvollziehbar ist. In der Regel besitzen sie keine Dateigrößenbeschränkungen und sind auf keine bestimmten Datentypen oder Dateiformate limitiert. Sie haben ihre Stärken im Bereich von textbasierten Dateien und werden daher häufig in der Softwareentwicklung zur Verwaltung von Quelltexten benutzt. Für die Verwaltung von Binärdateien eignen sie sich weniger, da im Gegensatz zu Textdateien die Erfassung von Änderungen bzw. deren Zusammenführung komplexer und weniger effizient ist. Dadurch können Archive mit vielen Binärdateien unter Umständen sehr viel Speicherplatz belegen und die Performance stark verringern. Diese Limitierung schränkt die Nutzung als generelle Austauschplattform für Forschungsdaten stark ein, jedoch werden sie speziell im Bereich der Bioinformatik auch in den Lebenswissenschaften genutzt. Im Allgemeinen wird zwischen zentralen und verteilten Versionsverwaltungssystemen unterschieden. Zwei verbreitete Vertreter sind Subversion und GIT.

Subversion

Subversion (Apache Software Foundation, 2018b) ist ein zentrales Versionsverwaltungssystem, bei dem es ein zentrales Projekt-Repository gibt, von dem alle Nutzer gespeicherte Dateien beziehen bzw. ihre Änderungen übertragen. Bei jeder eingespielten Änderung eines Nutzers wird direkt eine neue Version im zentralen Archiv erstellt. Vergleiche mit vorherigen Versionen und die Verwaltung von Benutzerrechten sind ebenfalls nur auf dem zentralen Archivserver möglich, was die Nutzbarkeit und Administration des Systems vereinfacht. Der Nachteil ist jedoch, dass die einzelnen Clients vollkommen vom zentralen Server abhängig sind und die Nutzer bei einer Störung oder einem Ausfall des zentralen Repository keinen Zugriff mehr auf das Archiv und die Versionshistorie haben, da alle Funktionalitäten von Subversion nur mit einer dauerhaften Netzwerkverbindung zum Server verfügbar sind (vgl. Pilato et al., 2008).

GIT

GIT (GIT, 2018) ist ein verteiltes Versionsverwaltungssystem, bei dem es keinen einzelnen zentralen Server gibt, der das Projektarchiv verwaltet und speichert. Stattdessen speichert jeder Anwender eine lokale Kopie des gesamten Repository. Alle Änderungen werden jedoch zunächst nur im lokalen Projekt erfasst. Dadurch wird die Performance gesteigert, da kein Netzwerkzugriff notwendig ist, um Änderungen vorzunehmen und neue Versionen zu erstellen. Erst wenn ein Abgleich mit einer anderen, entfernten Kopie des Projektarchives erfolgen soll, ist ein Netzwerkzugriff erforderlich. Ansonsten sind alle Funktionen auch ohne dauerhafte Verbindung zur zentralen Kopie des Repository nutzbar. Das hat den Vorteil, dass die Wahrscheinlichkeit eines Datenverlustes sinkt, da es keinen einzelnen zentralen Server gibt und jeder Nutzer eine lokale Kopie der kompletten Versionshistorie speichert. Ein kleiner Nachteil ist jedoch die fehlende integrierte Rechteverwaltung aufgrund der dezentralen Struktur (vgl. Loeliger und McCullough, 2012).

Versionsverwaltungssysteme sind in Bezug auf die Verwaltung und den Austausch von Quelltexten und Software etabliert, da es sich hier um textbasierte Daten handelt. So sind spezielle Plattformen wie GitHub (GitHub Inc., 2018), SourceForge (Dice Holdings, Inc., 2018) oder Bitbucket (Atlassian, 2018), die auf GIT bzw. Subversion aufbauen, entstanden und ermöglichen den Austausch von Programmcodes und Softwareskripten (vgl. Stodden, 2010; Van Noorden, 2013a; Priem, 2013; Perez-Riverol et al., 2016). Zur generellen Verwaltung des sehr breiten und heterogenen Spektrums von Forschungsdaten sind sie jedoch nur mit Einschränkungen zu empfehlen (Ram, 2013). Sie ermöglichen die Erfassung des kompletten Versionsverlaufes eines Datensatzes, bieten jedoch keine Möglichkeiten, Datensätze effektiv mit Metadaten zu beschreiben. Außerdem gibt es keine Möglichkeit einzelne Dateien oder Datensätze über persistente IDs (siehe Abschnitt 2.2.3) zu referenzieren. Dazu kommt, dass die Integration der Systeme in existierende Infrastrukturen und Anwendungen durchaus komplex ist.

3.4 Datenaustausch- und Datenpublikationsdienste

Neben der fortschreitenden Entstehung neuer spezialisierter Datenbanken und Informationssysteme in den verschiedensten wissenschaftlichen Domänen haben sich parallel dazu einige wenige allgemeine Datenaustauschdienste und Webplattformen entwickelt. Sie ermöglichen es Forschern ihre Daten aus unterschiedlichen Domänen hochzuladen, und sie öffentlich verfügbar zu machen. Die bekanntesten Systeme sind Dryad (Dryad, 2018), figshare (figshare, 2018) und Zenodo (Zenodo, 2018).

Dryad

Dryad ist ein interdisziplinäres Datenrepository zum Speichern und Austauschen von wissenschaftlichen Daten. Ziel von Dryad ist der langfristige Erhalt und freie Zugang von Forschungsdaten für die wissenschaftliche Gemeinschaft als Grundlage für eine qualitative und nachhaltige wissenschaftliche Arbeit. Dabei beschränkt sich Dryad auf Datensätze, die Grundlage für wissenschaftliche Publikationen beispielsweise in Fachjournalen oder Konferenzbeiträgen sind, und arbeitet daher eng mit verschiedenen Verlagen zusammen. Die Forscher sollen motiviert werden, Daten, die nicht direkt als Bestandteil einer Publikation veröffentlicht werden können, beispielsweise aufgrund des Datenvolumens, in Dryad abzulegen und in dem dazugehörigen Artikel zu zitieren. Der Grund dafür ist, dass diese Daten häufig auf eigenen institutionellen Webservern abgelegt werden, um sie zu veröffentlichen. Diese können dann aber unter Umständen nicht dauerhaft gepflegt werden. In Dryad gespeicherte Datensätze werden mit einer DOI (siehe Abschnitt 2.2.3) versehen und können so langfristig zitiert werden. Außerdem wird jeder erfasste Datensatz mit Metadaten, die dem DublinCore Schema (siehe Abschnitt 2.2.4) entsprechen, beschrieben (Vision, 2010).

Dryad bietet zwar keine Beschränkungen auf bestimmte Datentypen, die enge Koppelung des Systems an den Publikationsprozess eines Fachjournals oder Konferenzkomitees, die den wissenschaftlichen Artikel zu den bereitgestellten Daten veröffentlichen, ist jedoch eine große Einschränkung. Nutzer können ihre Daten nur über eine personalisierte URL ablegen und beschreiben, welche sie in Verbindung mit einer Einladung vom jeweiligen

Verlag erhalten. Dadurch wird die Einbindung in den allgemeinen Forschungsprozess erschwert und der Upload und die Beschreibung der Daten ist zeitaufwendig. Außerdem können nur Datensätze bis zu einer Größe von 20 Gigabyte frei abgelegt werden. Wird weiterer Speicherplatz benötigt, fallen zusätzliche Kosten an. Darüber hinaus werden verschiedene Bezahlmodelle für Forschungseinrichtungen geboten. Dryad empfiehlt sich bei der Beschreibung und Dokumentation von bereitgestellten Forschungsdaten an die von der wissenschaftlichen Gemeinschaft akzeptierten Standards, bzw. an die vom Journal der dazugehörigen Publikation vorgegebenen Richtlinien zu halten. Jedoch bietet es keinen expliziten Begutachtungsprozess, um die Einhaltung dieser Vorgaben zu gewährleisten. Abbildung 3.1 zeigt die Inhaltsseite eines Datensatzes im Dryad Repository.

Abbildung 3.1: Screenshot der Inhaltsseite eines Dryad-Datensatzes (Dryad, 2007)

Ein weiterer Nachteil für die Forscher, die ihre Daten bereitstellen, ist auch die verpflichtende Veröffentlichung unter Verwendung der Creative-Commons-Zero (CC0)-Lizenz (Creative Commons, 2018). Dadurch wird die Weiternutzung für andere Forscher vereinfacht, da sie keine rechtlichen oder technischen Beschränkungen bei der Verwendung der Daten beachten müssen. Die Autoren der Daten treten damit jedoch soweit es gesetzlich möglich ist sämtliche Besitzansprüche bzw. Anforderungen an die Weiterverwendung der

Daten ab und geben sie in die Gemeinfreiheit ab. Das entbindet zwar andere Forscher nicht davon sich an gängige wissenschaftliche Richtlinien zu halten, jedoch sind sie unter anderem nicht dazu verpflichtet, den ursprünglichen Datensatz und dessen Autoren zu referenzieren.

figshare

figshare ist ein allgemeines Datenrepository, in dem Forscher jegliche Art von Forschungsdaten veröffentlichen können. Dabei sind die Datensätze nicht an die Publikation eines wissenschaftlichen Artikels gebunden. Abbildung 3.2 zeigt die Inhaltsseite eines Datensatzes auf der figshare Webseite.

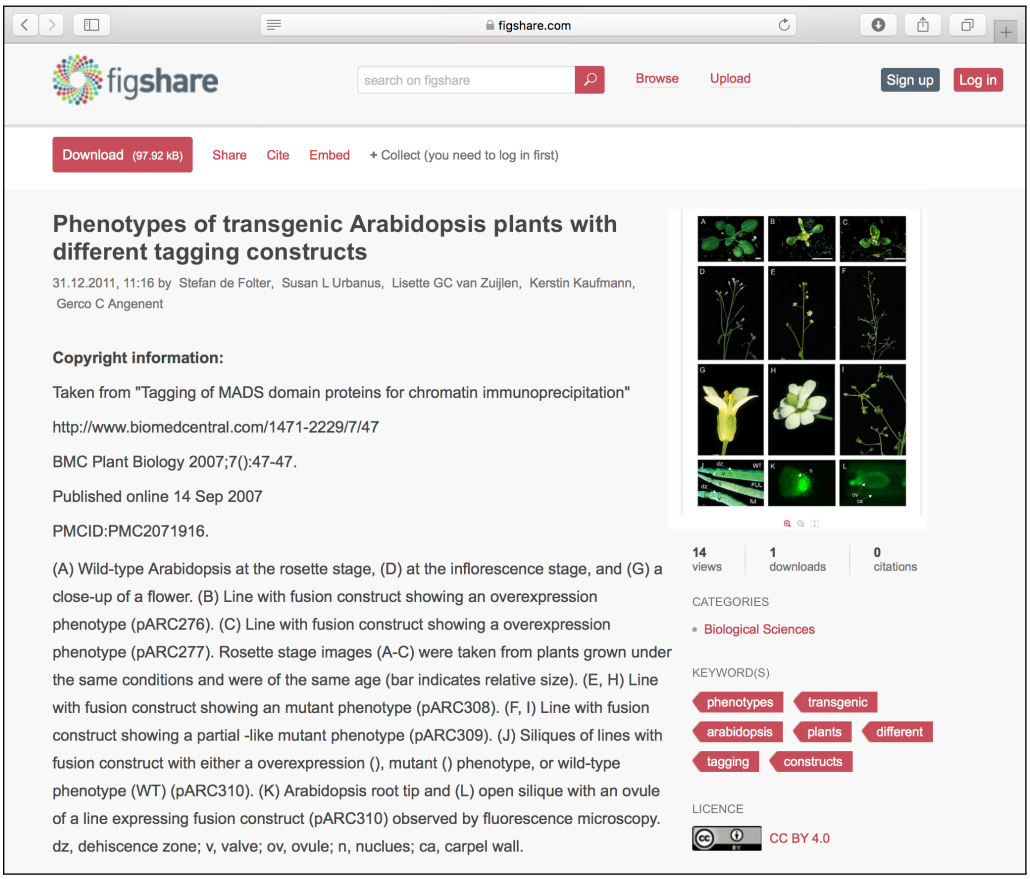

Abbildung 3.2: Screenshot der Inhaltsseite eines figshare-Datensatzes (figshare, 2007)

Nutzer können sich bei figshare registrieren und Datensätze verschiedenster Kategorien wie z. B. Abbildungen, Tabellen, Manuskripte oder Quellcodes hochladen. Dabei gibt es keine Beschränkung auf bestimmte Dateitypen. Außerdem wird eine Programmierschnittstelle angeboten, um figshare in existierende Anwendungen einzubinden. Jedoch hat auch figshare ein privates Speicherlimit von 20 Gigabyte, wobei einzelne Dateien auf 5 Gigabyte limitiert sind. Neben der Möglichkeit Daten nur mit bestimmten Nutzern auszutauschen, können Datensätze auch veröffentlicht und mit einer DOI (siehe Abschnitt 2.2.3) versehen werden. Dadurch sind sie dauerhaft referenzierbar und auffindbar (vgl. Singh, 2011). Nur für die

Vergabe einer DOI müssen auch die von DataCite geforderten Metadaten bereitgestellt werden. Darüber hinaus unterstützt figshare keine standardisierten Metadatenformate zur Beschreibung der erfassten Daten. Die Qualität der Daten liegt in der Verantwortung des jeweiligen Eigentümers. figshare selbst bietet keinen dedizierten Begutachtungsprozess. Die Ausnahme bilden spezielle kostenpflichtige Dienste für Institutionen, die es ermöglichen Vorgaben bei der Kuration von Daten zu verwalten. Anders als Dryad unterstützt figshare verschiedene Lizenzmodelle, unter denen ein Datensatz veröffentlicht werden kann.

Zenodo

Zenodo ist eine noch etwas jüngere Plattform zum Publizieren von Forschungsdaten und wird von der Europäischen Kommission unterstützt. Genau wie figshare sind die erfassten Datensätze nicht an einen Publikationsprozess für einen wissenschaftlichen Artikel gebunden und jeder Nutzer kann seine Daten ohne Einschränkungen hochladen. Zenodo akzeptiert jegliche Art von wissenschaftlichen Daten und gibt keine Beschränkungen in Bezug auf Dateiformat und Dateivolumen vor. Im Unterschied zu figshare ist jedoch jeder erfasste Datensatz über eine DOI öffentlich verfügbar und zitierbar. Abbildung 3.3 zeigt die Inhaltsseite eines Zenodo Datensatzes. Das Hochladen, Speichern und Suchen

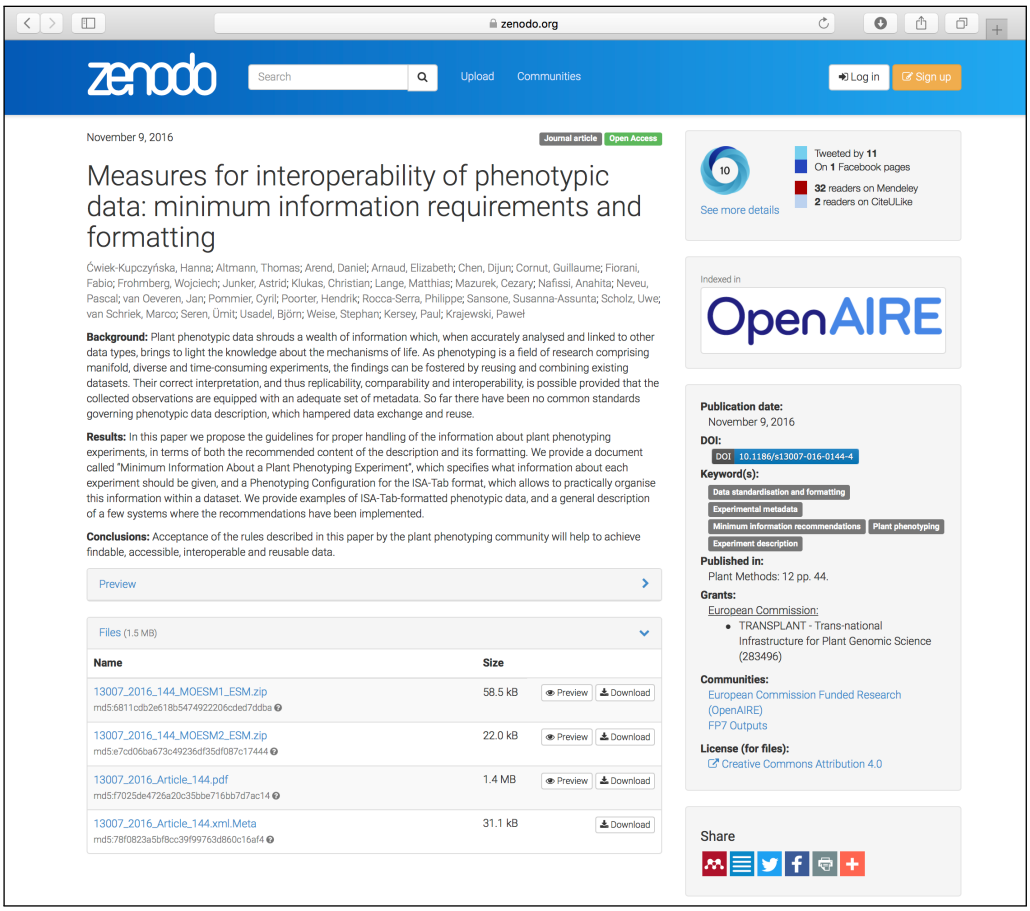

Abbildung 3.3: Screenshot der Inhaltsseite eines Zenodo-Datensatzes (Zenodo, 2016)

von Datensätzen ist über eine Weboberfläche möglich. Ähnlich wie bei figshare werden verschiedene Lizenzmodelle unterstützt, sodass jeder Nutzer entscheiden kann inwieweit seine Daten weiterverwendet werden dürfen. Eine Programmierschnittstelle, beispielsweise zur Einbindung in institutionelle Infrastrukturen und Anwendungen, ist momentan noch in der Entwicklung.

Der größte Vorteil der vorgestellten Datenaustausch- und Publikationsdienste ist die einfache Handhabung. Es muss keine lokale Infrastruktur etabliert werden. Der Nutzer meldet sich bei dem entsprechenden Dienst an, lädt seine Daten hoch und beschreibt diese. Über eine persistente ID (siehe Abschnitt 2.2.3) können die Datensätze anschließend dauerhaft referenziert werden. Der große Nachteil ist jedoch die Limitierung der freien Speicherkapazität. Aufgrund der immer größer werdenden Menge an täglich produzierten Forschungsdaten ist diese kaum ausreichend. Zusätzlicher Speicherplatz muss kostenpflichtig erworben werden. Außerdem kann die Übertragung der Daten je nach Volumen sehr zeitaufwendig sein und die Integration in existierenden Infrastrukturen ist teilweise aufgrund fehlender APIs schwierig.

3.5 Freie Daten-Repository-Systeme

Eine weitere Alternative zur Speicherung und zum effizienten Austausch von Forschungsdaten sind freie Repository-Infrastrukturen für digitale Daten. Diese Softwarepakete bieten die Möglichkeit, ein eigenes, disziplinäres oder institutionelles Repository unter Nutzung der zur Verfügung stehenden Hardware und Infrastruktur zu etablieren. Zwei verbreitete Vertreter sind die FEDORA (FEDORA, 2018) und Dspace (Dspace, 2018).

FEDORA

FEDORA (Flexible Extensible Digital Object Repository Architecture) bietet eine flexible Architektur zur Modellierung und Verwaltung von digitalen Daten. Dabei stellt es eine Vielzahl an Funktionen, Schnittstellen und Erweiterungsmöglichkeiten bereit, um das System an die Bedürfnisse der jeweiligen Einrichtung oder Datendomäne anzupassen. Es bietet die Möglichkeit persistente IDs zu vergeben und unterstützt verschiedene Metadatenformate zur Beschreibung erfasster Daten (vgl. Payette und Lagoze, 1998). FEDORA wird permanent weiterentwickelt und gepflegt. Ein weiterer Vorteil, ist die freie Verfügbarkeit und die fehlende Beschränkung der Speicherkapazität. Da FEDORA als Speicher- und Verwaltungsinfrastruktur fungiert, welche in die jeweilige Nutzerumgebung einer Einrichtung integriert wird, ist sie nur durch die vorhandene Speicherkapazität der zugrunde liegenden Hardware limitiert. Dies bedeutet aber auch unter Umständen einen hohen Aufwand, um FEDORA in eine vorhandene Infrastruktur zu integrieren. Außerdem macht der riesige Funktionsumfang die Wartung sehr komplex und erfordert ein hohes technisches Verständnis, sodass die Einrichtung für kleinere Institutionen häufig nicht realisierbar ist.

DSpace

Dspace ist ein komplettes Softwarepaket zur Erfassung und Publikation von wissenschaftlichen Daten. Es bietet ein umfangreiches Benutzer- und Rechtemanagement und ist grundsätzlich auf keinen Datentyp beschränkt. Es bietet viele vordefinierte Arbeitsabläufe, z. B. zur Eingabe und Verwaltung von Metadaten für textbasierte Dokumente wie wissenschaftliche Publikation oder Berichte. Außerdem ist Dspace in der Lage, persistente IDs zu vergeben und zu verwalten (vgl. Smith et al., 2003). Anders als FEDORA bietet DSpace bereits eine funktionierende Nutzerumgebung. Die Einrichtung einer DSpace Instanz ist jedoch recht komplex, da eine Reihe von Systemvoraussetzungen erfüllt werden müssen. So müssen unter anderem ein DBMS und ein Webserver zur Verfügung gestellt werden.

Insgesamt betrachtet, bieten FEDORA und DSpace die umfangreichste Möglichkeit zur Verwaltung und zum Austausch von Forschungsdaten. Sie bieten eine sehr große Funktionalität und können aufgrund ihrer modularen Implementierung erweitert und an verschiedene existierende Infrastrukturen angepasst werden. Gleichzeitig setzt die Einrichtung eines institutionellen Repository auf Basis einer der beiden Vertreter ein hohes technisches Verständnis und Support voraus. Es gibt auch noch einige weitere, neuere Repository-Infrastrukturen wie z.B. CKAN (CKAN Association, 2018) und DataVerse (IQSS, 2018), die hier aber nicht im Detail beschrieben wurden. Diese ähneln den beiden vorgestellten Systemen in Bezug auf den Funktionsumfang sehr. Jedoch benötigen auch diese Systeme eine Reihe von zusätzlichen Abhängigkeiten, deren Bereitstellung relativ komplex ist.

3.6 Zusammenfassung

Bei der Analyse populärer Systeme zur Verwaltung und Publikation von Forschungsdaten hat sich gezeigt, dass es ein breites Spektrum an verwendeten Datenbanken, Archiven und Informationssystemen gibt. Zusammenfassend konnte festgestellt werden, dass die meisten Systeme und deren Anwendungsfälle sehr spezifisch auf eine bestimmte Forschungsdomäne bzw. Datentypen zugeschnitten sind und sich einige der vorgestellten Möglichkeiten nicht zur generellen Verwaltung aller Arten von Forschungsdaten eignen. Häufig ist auch der fehlende Support von dauerhaften IDs, wie z. B. DOIs ein weiteres Defizit. Diese stellen eine langlebige Zitierbarkeit der Daten sicher, werden jedoch bisher nur von einigen teilweise kostenpflichtigen Publikationsdiensten wie Dryad, figshare oder Zenodo angeboten (siehe Abschnitt 3.4). Außerdem ist die Integration der vorgestellten Systeme in existierende Infrastrukturen und vorhandene Programme oft sehr aufwendig und erfordert teilweise umfangreiche technische Kenntnisse. Dadurch wird die Einbindung in den täglichen wissenschaftlichen Arbeitsprozess erschwert, da viele Forscher diese Anstrengungen bzw. zusätzlichen Zeitaufwand zur konsequenten Beschreibung, Ablage und Publikation der von ihnen erzeugten Daten nicht aufbringen können.

Eine weitere Schwachstelle ist die oft unzureichende Beschreibung von Forschungsdaten mit standardisierten Metadaten, die für eine zuverlässige und dauerhafte Nutzbarkeit essentiell sind. Domänenspezifische Datenbanken (siehe Abschnitt 3.1) nutzen zwar etablierte und von der wissenschaftlichen Gemeinschaft über Jahre hinweg entwickelte, meist semantische Metadatenformate, jedoch erschwert die große Diversität die konstante Pflege und Beschreibung der Datensätze. Viele durchaus populäre Systeme wie beispielsweise Cloud-Datendienste

(siehe Abschnitt 3.2) oder Versionskontrollsysteme (siehe Abschnitt 3.3) bieten jedoch in der Regel keine oder nur unzureichende Möglichkeiten zur Erfassung von Metadaten. Dies hat jedoch häufig historische Ursachen, da die jeweiligen Systeme ursprünglich für andere Anwendungsfälle konstruiert wurden und nur aufgrund ihrer einfachen Nutzung zur Verteilung von Forschungsdaten verwendet werden. Die vielversprechendsten Systeme, die die eingangs erwähnten Mängel nicht aufweisen, sind FEDORA und DSpace (siehe Abschnitt 3.5). Sie sind durchaus gut für die Verwaltung und Publikation aller Arten von Forschungsdaten geeignet und bieten einen sehr großen Funktionsumfang. Sie liefern die Möglichkeit, eine an die institutionellen Bedürfnisse angepasste Infrastruktur zu entwickeln. Dadurch können unter anderem auch vorhandene Speicherkapazitäten genutzt werden. Jedoch ist die Einrichtung, als auch die Wartung einer darauf basierenden Infrastruktur durchaus komplex und erfordert ein gewisses Maß an technischem Verständnis. Außerdem liefern beide Systeme keinesfalls eine fertige und sofort verwendbare Infrastruktur, sondern benötigen zusätzlichen Aufwand um die bereitgestellten Komponenten zu verknüpfen und anzupassen, um die gewünschten Aufgaben zu erfüllen. Daher sind sie für kleinere und mittelgroße Institutionen häufig nur schwer realisierbar.

Die Untersuchung existierender Systeme und Dienste zum Forschungsdatenmanagement und zur Datenpublikation hat sowohl die Stärken der einzelnen Systeme als auch die jeweiligen Defizite aufgezeigt. Diese sind in Tabelle 3.3 zusammengefasst. Die gewonnenen Erkenntnisse sind für die Entwicklung einer Infrastruktur zur Forschungsdatenpublikation sehr wichtig um daraus notwendige Funktionen und Eigenschaften ableiten zu können. Ziel der in dieser Arbeit vorgestellten Infrastruktur ist es die Stärken und Fähigkeiten existierender Systeme zu vereinen und Schwachstellen auszugleichen.

Systeme	**Stärken**	**Schwächen**
Domänen-spezifische Informations-systeme	+ auf spezifische Datentypen zugeschnitten + weit verbreitet und akzeptiert + angepasste Funktionen/Services	– nicht für alle Datendomänen verfügbar – oft proprietäre Metadaten – Import sehr zeitaufwendig
Cloud-basierte Datenaustausch-dienste	+ plattform-übergreifende Einrichtung + keine Datentyp-Beschränkungen + rudimentäre Versionsverwaltung	– begrenzter, kostenfreier Speicher – keine Erfassung von Metadaten – nur einfache Dateifreigaben
Versions-verwaltungs-systeme	+ ideal zur Quellcode-Verwaltung + keine Datentyp- oder Volumen-Beschränkungen + vollständiger Versionsverlauf	– für Binärdateien nur wenig geeignet – keine Metadatenverwaltung – keine Referenzierung über dauerhafte IDs
Daten-publikations-dienste	+ keine Datentyp-Beschränkungen + Vergabe von dauerhafte IDs	– begrenzter, kostenfreier Speicher – Integration in existierende Infrastruk-turen nicht immer möglich – Import sehr zeitaufwendig
Daten-Repository Systeme	+ keine Datentyp- oder Volumen-Beschränkungen + hoher Funktionsumfang + vielfältige Anpassungsmöglichkeiten	– komplexe Einrichtung und Wartung – hoher Auswand für Integration und Anpassung

Tabelle 3.3: Übersicht der Stärken und Schwächen existierender Systeme zur Verwaltung und Publikation von Forschungsdaten

4 Anforderungsanalyse und Vorarbeiten

In diesem Kapitel werden die generellen Anforderungen an eine Infrastruktur zum nachhaltigen Forschungsdatenmanagement zusammengefasst und erläutert. Diese ergeben sich aus der Analyse des gängigen Publikationsprozesses von wissenschaftlichen Daten (siehe Abschnitt 2.2.2) und der Untersuchung von existierenden Systemen zum Forschungsdatenmanagement (siehe Abschnitt 3.6). Dabei werden sie in vier Anforderungsbereiche unterteilt, die jeweils verschiedene Eigenschaftsklassen beinhalten. Diese sind in Abbildung 4.1 grafisch dargestellt und werden im anschließenden Abschnitt genauer erläutert. Im zweiten Abschnitt werden existierende Vorarbeiten, die als Einstieg in die Entwicklung einer Datenhaltungs- und Publikationsinfrastruktur dienen, vorgestellt.

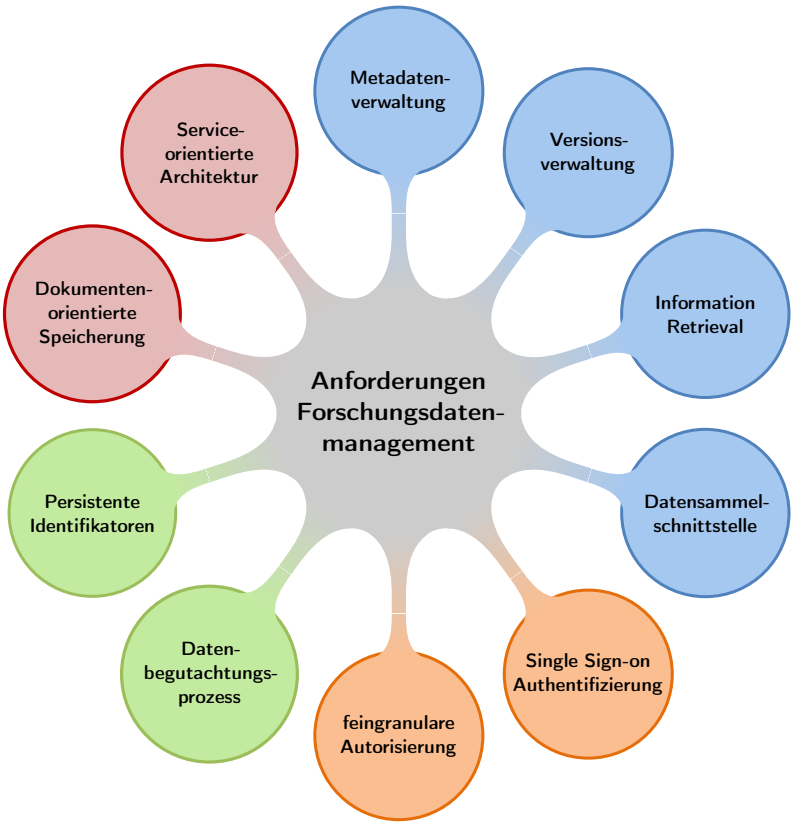

Abbildung 4.1: Anforderungen an das Forschungsdatenmanagement. Schematische Darstellung der generellen Anforderungen und Funktionen einer nachhaltige Infrastruktur zur Verwaltung und Veröffentlichung von Forschungsdaten. Die jeweiligen Farben kennzeichnen die Einstufung der Funktionen zu der entsprechenden Eigenschaftsklasse.

4.1 Anforderungen an Forschungsdatenmanagement

In diesem Abschnitt werden die Eigenschaftsklassen und die dazugehörigen Funktionen einer generellen Infrastruktur zum Forschungsdatenmanagement erläutert. Diese sind notwendig, um die langfristige Wiederverwendung und Referenzierung von Forschungsdaten zu gewährleisten.

4.1.1 Dateisystem ähnliche Datenstruktur

Um eine nachhaltige Datenarchivierung und einen zurückverfolgbaren Publikationsprozess von Forschungsdaten zu gewährleisten, ist eine Dateisystem ähnliche Datenstruktur mit einer Ordner- und Dateihierarchie erforderlich, die darüber hinaus einige zusätzliche Funktionen bereitstellen sollte, die in der Regel von gebräuchlichen Dateisystemen nicht geleistet werden können.

Metadatenverwaltung

Die wichtigste Voraussetzung um eine zukunftssichere Speicherung und eine dauerhaften Nutzbarkeit von Forschungsdaten sicherzustellen, ist die Erfassung von standardisierten, technischen Metadaten. Diese dienen dazu, den Zugriff, die Lesbarkeit und Interpretation der beschreibenden Datensätze zu gewährleisten und außerdem eine automatisierte Verarbeitung zu ermöglichen. Sie können zur Kategorisierung, zur Filterung, sowie zur Suche von spezifischen Daten genutzt werden und dienen weniger zur semantischen Beschreibung und Verarbeitung. Die Metadaten sollten möglichst einfach handhabbar und leicht zu erfassen sein. Ein gewöhnliches Dateisystem bietet diese Funktionalität nicht und so werden Metadaten entweder gar nicht oder häufig nur rudimentär in separaten Dateien erfasst oder teilweise in abstrakten Ordner- und Dateinamen kodiert, was nicht sehr nachhaltig ist.

Versionsverwaltung

Um die Reproduzierbarkeit von Forschungsergebnissen zu ermöglichen, ist es notwendig, die vollständige Historie der zugrunde liegenden Daten von der Erzeugung der Daten, über die verschiedenen Bearbeitungsschritte und Zwischenstufen in der privaten Domäne und Gruppendomäne (siehe Abschnitt 2.2.2) bis hin zur Veröffentlichung für die wissenschaftliche Gemeinschaft zu erfassen. Jede Version eines Datensatzes wird mit einem eindeutigen Satz an standardisierten Metadaten dokumentiert und kann darüber identifiziert werden. So kann der komplette Prozess der Aufbereitung, Verarbeitung und Analyse von Daten nachvollzogen werden. Diese Transparenz ermöglicht eine qualitative Bewertung der aus den Daten gewonnen Ergebnissen und garantiert eine nachhaltige Forschung. Es existieren zwar Dateisysteme, die eine Versionierung von Dateien ermöglichen, jedoch muss diese eng mit der Verwaltung von Metadaten verknüpft sein und diese Funktion wird in der Regel nicht bereitgestellt.

Information Retrieval

Ein effizientes Forschungsdatenmanagement ist aufgrund der wachsenden Zahl an erfassten Datensätzen auf eine performante Suchfunktion angewiesen, um gezielt nach Kriterien und Metadaten, wie beispielweise nach bestimmten Autoren, Themengebieten oder Jahreszahlen, zu suchen. Auch die Integration erweiterter Suchfunktionen wie die Unterstützung von unscharfen Suchen, regulären Ausdrücken oder phonetischen Suchanfragen erhöht die Nutzbarkeit. Um die Leistungsfähigkeit der Suchmaschine zu gewährleisten, sollten spezielle

Datenstrukturen, wie Indexe, verwendet werden. Einige gebräuchliche Dateisystem bieten zwar die Möglichkeit Datensätze zu indexieren, haben jedoch ebenfalls aufgrund der fehlenden Metadatenverwaltung meist nur rudimentäre Suchfunktion z. B. nach Dateinamen oder Dateitypen.

Datensammelschnittstelle

Durch die Unterstützung einer standardisierten Datensammelschnittstelle ist es möglich in erster Linie die Fülle an Metadaten automatisiert, schnell und systematisch zu sammeln. Diese Informationen können beispielsweise genutzt werden um Statistiken über den Datenbestand zu erstellen, oder um Verknüpfungen mit anderen Datenquellen herzustellen, um die Sichtbarkeit und Nutzbarkeit zu erhöhen. Außerdem wird durch die Standardisierungen eine Übertragung von Datensätzen aus einer Quelle zu einer anderen ermöglicht.

4.1.2 Datensicherheit

Um die Akzeptanz der Forscher zu erhalten, ist es wichtig, dass sowohl die Sicherheit ihrer Daten als auch ihre Rechte als Erzeuger gewahrt werden. Forschungsdaten sollen der wissenschaftlichen Gemeinschaft zur Verfügung gestellt werden und dabei aber vor Missbrauch geschützt werden. Sie können aber durchaus auch sensible Inhalte besitzen, die erste Eindrücke in noch nicht veröffentlichte Ergebnisse geben und falsch interpretiert werden können. Außerdem können bestimmte Datentypen auch personenbezogene Informationen enthalten, die nicht für die Öffentlichkeit zugänglich sein sollten. In einigen Fällen werden Forschungsdaten aber auch bewusst zurück gehalten, um Dritten bei der Publikation von Ergebnissen zuvorzukommen (vgl. Tellam et al., 2015). Daher ist es sehr wichtig den Wissenschaftlern, welche die Daten erzeugt haben, die Möglichkeit zu geben, differenziert zu entscheiden, wann und für wen ihre Daten zugänglich sind. Ihre Rechte als Autor der Daten müssen dabei gleichzeitig bewahrt werden.

Single Sign-on Authentifizierung

Um die Nutzung einer Infrastruktur zum Management von Forschungsdaten möglichst einfach zu halten, ist eine einmalige Anmeldung (*Single Sign-on*) notwendig, die den jeweiligen Nutzer eindeutig identifiziert. Dabei ist es wichtig, gängige Authentifizierungsverfahren zu unterstützen. Die Verwendung eines existierenden beispielsweise institutionellen Anmeldeverfahrens eliminiert eine Hürde bei der Nutzung und erspart den Nutzern zusätzliche Arbeit beim Anlegen und Pflegen eines zusätzlichen Benutzerkontos.

Feingranulare Autorisierung

Da Forschungsdaten häufig sehr komplex und heterogen sind, ist es notwendig, dass auch die Autorisierung für deren Nutzbarkeit möglichst feingranular ist. Eine einfache Vergabe von Lese-und Schreibrechten würde diese Anforderung nicht erfüllen. Besonders wenn die Daten sich noch in der privaten Domäne oder Gruppendomäne (siehe Abschnitt 2.2.2) befinden, ist eine feingranulare Rechteverwaltung notwendig, um beispielsweise nur bestimmte Dateien aus einem Datensatz für ausgewählte Nutzer oder Gruppen frei zu geben. Darüber hinaus ist es notwendig, dass verschiedene, gebräuchliche Lizenzen unterstützt werden, mit denen Datensätze versehen werden können. Dies ist vor allem später wichtig, wenn die Daten in die öffentliche Domäne transferiert werden, damit die Rechte der Datenerzeuger geschützt sind.

4.1.3 Datenaustausch und Datenqualität

Eine Hauptaufgabe einer Infrastruktur zur Forschungsdatenverwaltung ist es, Forschungsdaten öffentlich zugänglich und dauerhaft nutzbar zu machen. Das bringt nicht nur Vorteile für die wissenschaftliche Gemeinschaft, die die Daten nutzen will, sondern ermöglicht auch den Datenerzeugern, dass ihre Datensätze zitiert werden können und sie so die Anerkennung für ihre Arbeit erhalten. Bevor Forschungsdaten aber veröffentlicht werden können, müssen sie überprüft werden, da nur hochqualitative Datensätze, die den gängigen Standards der wissenschaftlichen Gemeinschaft in Bezug auf Format und Beschreibung entsprechen, eine Nachnutzbarkeit und hohen Wert für andere Wissenschaftler garantieren.

Persistente Identifikatoren

Zur Publikation von Forschungsdaten müssen diese dauerhaft referenzierbar und zitierfähig sein. Daher ist die Vergabe und Verwaltung von persistenten IDs zwingend notwendig, um eine langfristige Auffindbarkeit zu garantieren (siehe Abschnitt 2.2.3). Einfache Links in Form von URLs können dies nicht langfristig und zuverlässig gewährleisten. Gleichzeitig ermöglicht die Zitierbarkeit, dass die Erzeuger, genau wie bei wissenschaftlichen Artikeln üblich, die Anerkennung für die Bereitstellung der Daten erhalten und neue Metriken und Daten-Zitierindexe erfasst werden können.

Datenbegutachtungsprozess

Zur Sicherstellung einer qualitativen Forschung ist auch die Gewährleistung einer hohen Datenqualität notwendig. Dies kann teilweise automatisiert erfolgen, indem geprüft wird, ob gewisse akzeptierte Datenformate eingehalten werden oder bestimmten Metadaten vorhanden sind, die eine Nachnutzbarkeit ermöglichen. Um eine fachliche Qualität der Daten zu garantieren ist darüber hinaus aber auch eine Prüfung durch unabhängige Gutachter notwendig. Um rechtliche Schwierigkeiten zu vermeiden ist es außerdem angebracht, vor der Veröffentlichung zu prüfen, dass keine Lizenzen, Patente oder andere Rechte beispielsweise in Bezug auf institutionelle Rahmenvereinbarungen oder Kooperationsverträge verletzt werden.

4.1.4 Interoperabilität

Die wichtigste Voraussetzung für eine generelle Infrastruktur zum Forschungsdatenmanagement ist eine hohe Flexibilität in Bezug auf die Verbindung mit existierenden Programmen und die Einbindung in existierende Arbeitsabläufe, die aufgrund der Heterogenität von Forschungsdaten sehr unterschiedlich sein können. Eine einfache und dynamische Integration ermöglicht eine vielfältige Nutzbarkeit und Wiederverwendbarkeit. Außerdem gewährleistet eine benutzerfreundliche Anwendung die Akzeptanz des Systems.

Dokumentenorientierte Speicherung

Aufgrund der starken Heterogenität von Forschungsdaten und der Komplexizität von Datensätzen ist eine dokumentenbasierte Speicherung für eine generelle Forschungsdateninfrastruktur sinnvoll, da diese besonders flexibel ist. Die Vielzahl an verschiedenen Datentypen und Datendomänen macht es sehr schwierig ein allgemeingültiges, relationales Schema zu erstellen. Daher sollten Dateien und Datensätze als Ganzes gespeichert werden und über eine eindeutige ID identifizierbar sein.

Serviceorientierte Architektur

Genauso wichtig wie die Flexibilität in Bezug auf die Datentypen ist auch die Anpassungsfähigkeit an existierende Infrastrukturen und Einbindung in verschiedenen Nutzungsszenarien. Deshalb ist eine serviceorientierte Architektur für eine generelle Datenhaltungsinfrastruktur sinnvoll, da sie einzelne kleine Komponenten liefert, die dann kombiniert werden können, um sich flexibel an die Anforderungen der jeweiligen Anwendung oder Abläufe anzupassen.

4.2 Gegenüberstellung mit existierenden Systemen

Im diesem Abschnitt werden die untersuchten Systeme (siehe Kapitel 3) kurz zusammengefasst und tabellarisch miteinander verglichen, um zu sehen, welche der im vorherigen Abschnitt vorgestellten Anforderungen an eine Forschungsdateninfrastruktur erfüllt werden. Tabelle 4.1 zeigt eine Übersicht der vorhandenen Funktionen der verschiedenen Eigenschaftsklassen der einzelnen Systeme.

	Domänen-spezifische Informations-systeme	Cloud-basierte Daten-austausch-dienste	Versions-verwaltungs-systeme	Datenaustausch- und Publikations-dienste	Freie Daten-Repository-Systeme
Metadaten-verwaltung	(✓)			(✓)	(✓)
Versions-verwaltung	(✓)	✓	✓	(✓)	✓
Information Retrieval	✓	(✓)	(✓)	✓	✓
Datensammel-schnittstelle	(✓)			(✓)	(✓)
Single Sign-on Authentifizierung	(✓)	(✓)	✓	✓	✓
Feingranulare Autorisierung					✓
Persistente Identifikatoren				✓	(✓)
Datenbegutachtungs-prozess					
Dokumentenorientierte Speicherung		✓	✓	(✓)	(✓)
Serviceorientierte Architektur		(✓)	(✓)	(✓)	✓

Tabelle 4.1: Vergleich verschiedener Systeme zur Verwaltung und Publikation von Forschungsdaten. Ist eine Funktion verfügbar, ist der entsprechende Eintrag mit „✓“ gekennzeichnet. Falls eine Funktionalität bzw. Eigenschaft nur teilweise gegeben ist, wird dies durch „(✓)“ dargestellt. Bei einem leeren Eintrag ist die jeweilige Funktion nicht verfügbar bzw. die Eigenschaft nicht erfüllt.

Wie schon in Abschnitt 3.6 erwähnt, besitzen die untersuchten Systeme einige Defizite und können nicht alle Anforderungen erfüllen. Es wird klar, dass es einige freie Daten-Repository-Systeme und Datenpublikationsdienste gibt, die viele der beschriebenen Anforderungen für eine langfristig stabile Infrastruktur zum Forschungsdatenmanagement und zur Datenpublikation zumindestens teilweise erfüllen. Jedoch haben alle Systeme Schwächen im

Bereich der Sicherstellung der Datenqualität, da ein geeigneter Ablauf zur Begutachtung von Daten, die veröffentlicht werden sollen, fehlt. Daher liegt der Fokus der Entwicklung einer allgemeinen Infrastuktur besonders auf der Begutachtung von publizierten Daten zur Sicherstellung einer gewissen Datenqualität.

4.3 electronic Data Archive Library (eDAL)

Im folgenden Abschnitt werden die existierenden Vorarbeiten zur Entwicklung der Datenhaltungsinfrastruktur für Forschungsdaten zusammengefasst. Ein erster konzeptioneller Entwurf der *electronic Data Archiving Library (eDAL)* genannten Infrastruktur-API wurde im Zuge einer Diplomarbeit betrachtet (Arend, 2012). Im Anschluss daran wurde dieser im Rahmen des Dissertationsprojektes weiter verfeinert und erstmals publiziert und auf internationalen Konferenzen vorgetragen (Arend et al., 2012a,b, 2013a,b). Während dieser Zeit wurde die zweite, erweiterte Version der Referenzimplementierung fertig gestellt und in einem umfangreichen Journal-Artikel publiziert (Arend et al., 2014b) und darüber hinaus bei einer weiteren internationalen Tagung vorgestellt (Arend et al., 2014a). Das Konzept, sowie wichtige Details der Referenzimplementierung, die in den genannten Artikeln veröffentlicht wurden, werden im Folgenden erläutert.

Im letzten Abschnitt werden außerdem einige allgemeine Erweiterungen und Verbesserungen der Referenzimplementierung kurz erläutert, da diese unter anderem für die weiteren Entwicklungsschritte von Bedeutung sind. eDAL wurde als plattform-unabhängige Java-API entwickelt, da Java zum einen eine der populärsten Programmiersprachen ist (vgl. TIOBE, 2018) und zum anderen aufgrund seiner flexiblen Erweiterungsmöglichkeiten besonders in den Lebenswissenschaften weit verbreitet ist.

4.3.1 Konzept

In diesem Abschnitt wird das grundlegende Konzept des Aufbaus der Infrastruktur, wie die Organisation der Datenobjekte oder die Erfassung von Metadaten erläutert. Außerdem wird das Sicherheitskonzept der API, sowie die Umsetzung einer Implementierungsschnittstelle erläutert.

Datenstruktur und Versionsverwaltung

Die Organisationsstruktur der in eDAL erfassten Objekte ist der eines gebräuchlichen Dateisystems sehr ähnlich. Datenobjekte werden durch ein abstraktes `PrimaryDataEntity` repräsentiert und in Verzeichnisse (`PrimaryDataDirectory`) und konkrete Dateien (`PrimaryDataFile`) unterteilt. Genau wie in einem Dateisystem können Objekte verschoben werden und innerhalb der Dateistruktur navigiert werden. Diese einfach verkettete Hierarchie, die in verkürzter Form in Abbildung 4.2 auf Seite 57 dargestellt ist, reicht aus um die Beziehungen zwischen Objekten darzustellen und ist sehr effizient und intuitiv. Eine komplettes UML-Klassendiagramm der in der eDAL-Infrastruktur verankerten Datenstruktur ist im Anhang A.1 auf Seite 125 abgebildet.

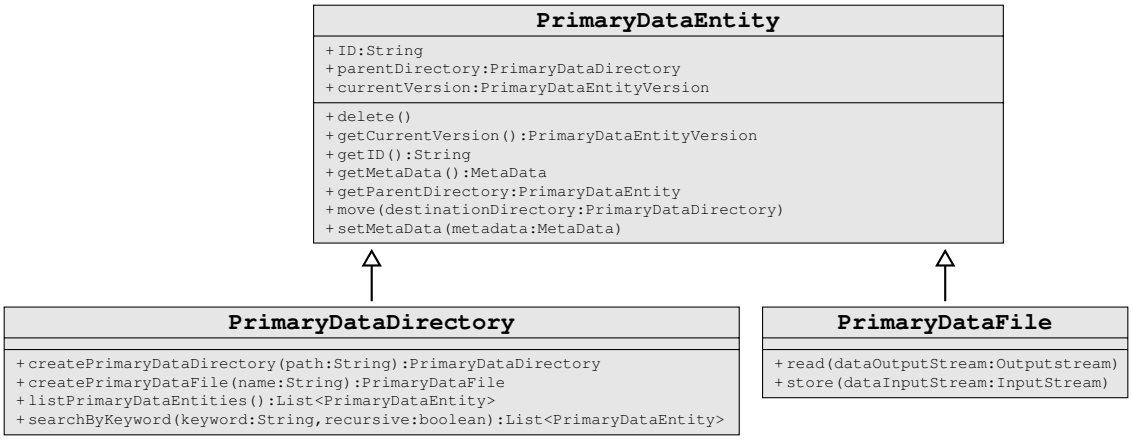

Abbildung 4.2: UML-Klassendiagramm der eDAL-Datenstruktur

Im Gegensatz zu einem gebräuchlichen Dateisystem bietet die eDAL-Infrastruktur auch die Möglichkeit, verschiedene Versionen (`PrimaryDataEntityVersion`) für jedes Datenobjekt zu erfassen und so die komplette Historie eines Datensatzes zu hinterlegen. Neben den grundlegenden Informationen, wie der Versionsnummer oder dem Datum der Erstellung ist jede Version eines erfassten Objektes an einen eigenen separaten Metadatensatz gekoppelt. Bei jeder Veränderung eines Objektes wird automatisch eine neue Version erstellt. Das können einfache Veränderungen sein, wie die Umbenennung eines Objektes oder die Erweiterung der Metadaten, aber auch umfangreiche Änderungen wie beispielsweise die Aktualisierung einer Datei, die mit einem Objekt verknüpft ist. Außerdem kann keine Version physisch gelöscht werden, sondern wird in dem Fall lediglich mit einer „gelöscht“-Marke versehen, die verhindert, dass neue Versionen eines Objektes angelegt werden können. Die Versionskette wird damit abgeschlossen. Alle bis dahin erstellten Versionen bleiben jedoch erhalten und sind weiterhin abrufbar. Des Weiteren besitzt die eDAL-API eine Schnittstelle zur Anbindung an Systeme zur Vergabe von persistenten IDs. So besitzt jede Version eine Liste von möglichen öffentlichen IDs, die über die `PublicReference` Klassen abgebildet werden.

Metadaten und Information Retrieval

Der Metadatensatz, der in der Klasse `MetaData` abgebildet und mit jeder Version verknüpft ist, enthält eine Reihe von technischen Attributen und ist an den DublinCore Standard (siehe Abschnitt 2.2.4) angelehnt. Einerseits sind diese notwendig, um eine langfristige Speicherung und zukunftssichere Nachnutzung zu gewährleisten und andererseits ist die Infrastruktur auf keinen bestimmten Datentypen spezialisiert, daher macht eine Unterstützung domänenspezifischer, semantischer Metadaten von Seite der API wenig Sinn. Der DublinCore Standard gibt keine Datentypen zur Beschreibung der jeweiligen Attribute vor. Um die Pflege von Metadaten zu vereinfachen bietet eDAL aber eine Reihe von gebräuchlichen Datentypen, die zur Beschreibung der Attribute genutzt werden können. Diese sind alle von einer Oberklasse (`UntypedData`) abgeleitet und in Tabelle 4.2 auf Seite 58 aufgelistet.

Die intuitive Datenstruktur von eDAL ermöglicht es durch die gespeicherten Datensätze zu Navigieren und bestimmte Objekte zu finden. Um die Informationen, die in den Metadaten gespeichert sind, effizient zu nutzen und bestimmte Dateien oder Datensätze schnell zu finden, bietet eDAL darüber hinaus eine umfangreiche Suchfunktion über einzelne Attribute oder einen kompletten Metadatensatz. Zusätzlich werden nützliche Suchoptionen, wie beispielsweise eine unscharfe oder rekursive Suche unterstützt.

Datentypen	Kurzbeschreibungen
`UntypedData`	allgemeine Oberklasse aller Datentypen (Freitext)
`CheckSum`	Checksumme für eine konkrete Datei
`DataFormat`	Format des Datenobjektes
`DataSize`	Dateigröße eines Objektes
`DataTape`	Typ eines Datenobjektes nach DublinCore Standard
`DateEvents`	Set von Datumsangaben
`EdalDate`	Datum das ein Ereignis erfasst
`EdalDateRange`	Zeitspanne die ein Ereignis erfasst
`EdalLanguage`	Sprache eines Datenobjektes
`EmptyMetaData`	leerer Datentyp
`Identifier`	eindeutige ID
`IdentifierRelation`	In Beziehung stehende IDs
`LegalPerson`	abstrakte Gruppe (Institution oder Konsortium)
`NaturalPerson`	natürliche Person
`Subjects`	Liste von Schlagwörtern
`UnknownMetaData`	unbekannter Datentyp

Tabelle 4.2: Übersicht und Kurzbeschreibung der von der eDAL-Infrastruktur bereitgestellten Datentypen

Sicherheitskonzept

Bei der Entwicklung des Sicherheitskonzepts für die eDAL-Infrastruktur stand neben dem Schutz der Forschungsdaten auch die Transparenz der Nutzerauthentifizierung und Autorisierung von Rechten sowie eine effektive Integration in die Datenstruktur im Vordergrund. Daher sind alle öffentlichen Methoden, die auf gespeicherten Objekten der Klasse `PrimaryDataEntity` ausgeführt werden können, generell geschützt und nur nach vorheriger Prüfung der jeweiligen Rechte des aktuell angemeldeten Nutzers ausführbar. Mithilfe des Java Authentication and Authorization Service (JAAS) (siehe Abschnitt 2.3.3) werden verschiedene Single-Sign-On Loginmodule unterstützt und jeder Nutzer muss sich einmalig beim Verbinden mit dem System anmelden. Beim Durchlaufen einer Anmeldeprozedur wird ein Subject (siehe Abschnitt 2.3.3) erstellt, das den Nutzer eindeutig identifiziert und in einer lokalen Variable während der gesamten Sitzung eines Nutzers gespeichert wird. Für die Prüfung der Nutzer-Autorisierung ist in der API ein zentraler Sicherheitsaspekt mit AspectJ (siehe Abschnitt 2.3.2) implementiert. Dieser enthält einen Pointcut (siehe Quelltext 4.1 auf Seite 59), der alle öffentlichen Methoden der `PrimaryDataEntity` Objektklasse erfasst und vor der Ausführung prüft, ob der Nutzer die nötigen Rechte besitzt, um die jeweilige Funktion auszuführen. In Anhang A.1 auf Seite 126 ist eine umfangreiche Fassung des Aspektes dargestellt.

```
public aspect PublicPermissionCheck {

  pointcut checkPublicMethods(): execution(public * PrimaryDataEntity+.* (..));

  Object around() throws AccessControlException: checkPublicMethods(){
      checkPermission(thisJoinPoint);
      return proceed();
  }

  public void checkPermission(JoinPoint joinPoint) throws AccessControlException {

      Subject subject = DataManager.getSubject();

      ImplementationProvider ip = DataManager.getImplProv();

      PermissionProvider permissionProvider = ip.getPermissionProvider().newInstance();

      EdalPermission permission = new EdalPermission(uuid,version,actionClass,method);

      boolean checkedPermission = checkPerm(subject,permission);

      if (!permissionProvider.isRoot(subject) && !checkedPermission) {
         throw AccessControllException(...);
      }
  }
}
```

Quelltext 4.1: Auszug aus dem Aspekt zur Überprüfung der eDAL-Nutzerrechte

Standardmäßig erhält jeder Nutzer beim Anlegen eines Datenobjektes die Rechte, um alle Funktionen auf diesem Objekt und den darin enthaltenen Ordnern und Dateien auszuführen. Anschließend kann jeder Nutzer weitere Rechte vergeben und so bestimmten Nutzern erlauben, spezifische Funktionen auf Ordnern, Dateien oder auch Versionen einzuräumen. Damit ist eine sehr feingranulare, aber einfache Autorisierung möglich. Es können Nutzer-Principals (siehe Abschnitt 2.3.3) definiert werden, für die Rechte vergeben werden können, um so z. B. Freigaben nur für einzelne Nutzer oder auch für eine Gruppe zu setzen. Über eine definierte Liste wird festgelegt, welche Principal-Klassen unterstützt werden. Eine Ausnahme bildet der Administrator oder Root-Nutzer des Systems, der standardmäßig keinen Beschränkungen unterliegt und alle Funktionen ausführen darf. Dieser wird in der Regel beim initialen Start des Systems definiert und muss sich zur Sicherheit über einen sogenannten *Double-Opt-In* Prozess identifizieren, indem er seine Identität mittels einer Bestätigungs-Email verifiziert.

Implementierungsschnittstelle und Server-Client-Architektur

Die eDAL-Infrastruktur-API ist sehr umfangreich und bietet eine Vielzahl an Funktionen. Damit die Implementierung der API und die Einbindung in existierende Infrastrukturen möglichst einfach und trotzdem flexibel ist, wurde eine zentrale Implementierungsschnittstelle (`ImplementationProvider`) entworfen. Über diese können die Module für die Implementierung der einzelnen Infrastrukturkomponenten, die sogenannten *Provider*, dynamisch eingebunden werden. So gibt es beispielsweise Provider für die Komponenten zur Speicherung der Datenstruktur oder der Rechte- und Metadatenverwaltung. Diese liefern die jeweiligen Implementierungsklassen für die abstrakten Funktionen der

eDAL-Infrastruktur. Zur Initialisierung des Systems gibt es in der `DataManager` Klasse die statische `getRootDirectory()` Funktion, die eine gültige Implementierung des `ImplementationProvider` und ein Subject, das den Nutzer der aktuellen Sitzung (siehe Abschnitt 4.3.1) identifiziert, benötigt und das Wurzelverzeichnis der Infrastruktur als Einstiegspunkt liefert.

Neben der Nutzung der eDAL-API als lokale Speicherinfrastruktur ist das wohl gebräuchlichste Nutzungsszenario die Einrichtung als zentrale Datenhaltungs- und Datenaustausch-Infrastruktur. Damit mehrere Nutzer parallel auf eine zentrale Instanz zugreifen können, war die Entwicklung einer geeigneten Server-Client-Architektur notwendig. Diese wurde mithilfe der Java RMI-API (siehe Abschnitt 2.3.4) realisiert. Es wurde eine Server-Komponente entwickelt, die es ermöglicht, eine Instanz der eDAL-API zentral zugänglich und jede Funktion über eine Remote-Schnittstelle per Fernzugriff verfügbar zu machen. Eine Herausforderung dabei ist die Anbindung an das Sicherheitssystem, da das Subject zur Nutzeridentifizierung (siehe Abschnitt 4.3.1) an den aktuell ausführenden Thread gebunden ist und zur Laufzeit einer Nutzersession nicht verändert wird. Da RMI-Verbindungen jedoch zustandslos sind und somit keine Informationen zur aktuellen Sitzung speichern, wird jeder Clientaufruf in einem separaten Thread auf dem Server ausgeführt. Daher muss das Subject des Client bei jedem Funktionsaufruf übermittelt werden, damit der Server weiß, welcher Nutzer gerade versucht, eine Methode aufzurufen. Um die Nutzung für die Clients zu vereinfachen und sämtliche Funktionsaufrufe analog zur lokalen Nutzung der Infrastruktur zu ermöglichen, wurde für die Client-Komponente ein Paket mit sogenannten *Wrapper*-Klassen implementiert, die die Übertragung des Subject-Objektes im Hintergrund übernimmt.

4.3.2 Implementierung

Bei der Entwicklung einer Referenzimplementierung für die eDAL-Infrastruktur-API stand in erster Linie eine nachhaltige und vor allem performante Umsetzung im Vordergrund, da dies sehr wichtig für die Akzeptanz der Nutzer ist. Außerdem sollten alle Komponenten komplett in die Implementierung eingebunden sein, sodass kein zusätzlicher Installationsaufwand bei der Einrichtung einer eDAL-Instanz notwendig ist. Die wichtigste Frage dabei ist, wie die erfassten Dateien, die dazugehörigen Versionsinformationen, die Metadaten sowie die Nutzerrechte und Freigaben dauerhaft und effizient gespeichert werden können. Üblicherweise werden Dateien in einem Dateisystem erfasst. Diese unterscheiden sich zwar je nach Betriebssystem, beispielsweise FAT und NTFS für Windows-Betriebssysteme, ext für Linux-Rechner oder HFS und APFS für Mac-Computer, sind aber im Allgemeinen für die Speicherung von großen Mengen an Binärdaten konzipiert. Häufig haben diese jedoch Schwierigkeiten im Umgang mit komplexen, weit verschachtelten Ordnerstrukturen und sehr vielen kleinen Dateien (vgl. Buyya et al., 2016). Außerdem bieten sie üblicherweise keine Möglichkeiten, verschiedene Versionen von Dateien und Metadaten zu erfassen. Dem gegenüber stehen relationale DBMS als Alternative und bieten ein zentrales Datenmanagement und Transaktionsystem für textbasierte Informationen (vgl. Saake et al., 2013). Darüber hinaus skalieren sie auch mit zunehmender Datenmenge und bieten eine effiziente Anfragesprache. Für die Referenzimplementierung der eDAL-Infrastruktur wurden beide Ansätze kombiniert, um von den Vorteilen beider Systeme zu profitieren und eine schnelle und effiziente Infrastruktur zu gewährleisten.

Objektpersitierung

Mithilfe des Hibernate-Frameworks (siehe Abschnitt 2.3.1) werden die eDAL-Datenstruktur, Versionsinformationen und Metadaten persistent in einer Datenbank hinterlegt. Dafür nutzt Hibernate die JDBC-Schnittstelle (siehe Abschnitt 2.3.1), um ein eingebundenes DBMS anzusteuern. Für die Referenzimplementierung wurde eine Java-basierte H2-Datenbank (Müller, 2018) verwendet, da diese sehr schnell ist und direkt aus einer Anwendung heraus erzeugt und genutzt werden kann. Alle notwendigen Parameter zur Datenbankanbindung und Konfiguration von Hibernate sind in einer zentralen XML-Datei definiert und werden bei der Initialisierung des System geladen. Dadurch können diese leicht angepasst werden, um beispielsweise eine anderes DBMS zu nutzen ohne das die restliche Implementierung geändert werden muss.

Datenbankschema Zur Realisierung des Mapping der Java-Klassen der eDAL-API in die entsprechenden Datenbanktabellen wurden die jeweiligen Implementierungsklassen, die im `ImplementationProvider` gebündelt sind, um zusätzliche Hibernate Annotationen erweitert. Dadurch ist Hibernate in der Lage, die notwendigen Tabellen, Datentypen und Beziehungen zur Abbildung der eDAL-Klassen automatisch zu erstellen. Das manuelle Erstellen und Ausführen von entsprechenden CREATE-TABLE Anweisungen in SQL fällt weg. Das erzeugte Schema wurde an einigen Stellen noch durch zusätzliche Parameter für die Standard-Annotationen optimiert, um Speicherplatz zu sparen und Anfragen effizienter auswerten zu können. So wurden beispielsweise für einige Attribute mit begrenzter Länge CHAR-Datentypen mit fester Länge definiert, statt eines VARCHAR-Datentypes der von Hibernate standardmäßig für alle String-Variablen verwendet wird. Bei der Abbildung von Vererbungshierarchien gibt es mehrere Möglichkeiten zur Überführung in ein relationales Schema (siehe Abschnitt 2.3.1). Für die eDAL-Datenstruktur wurde eine Tabelle für alle `PrimaryDataEntity` Objekte erstellt, da die Klassenhierarchie keine weiteren Beziehungen zu anderen Objekten hat. Die Unterscheidung in `PrimaryDataDirectory` und `PrimaryDataFile` Objekte erfolgt über eine Diskriminatorspalte. Ein detailliertes Datenbank-Schema der einzelnen Komponenten der eDAL-Infrastruktur ist in Anhang A.3 auf Seite 127 dargestellt.

Objekte speichern und wiederherstellen Das Speichern von Objekten in die Datenbank bzw. das Wiederherstellen von persistenten Tabelleneinträgen in entsprechende Java Objekte wird ebenfalls von Hibernate übernommen. Dabei wird durch die transitive Persistierung (siehe Abschnitt 2.3.1) sichergestellt, dass alle assoziierten Objekte kaskadierend gespeichert werden, was die Implementierung vereinfacht. Auf der anderen Seite sorgt die standardmäßig verwendete lazy loading-Strategie (siehe Abschnitt 2.3.1) beim Laden von Objekte dafür, dass assoziierte Objekte nur bei Bedarf nachgeladen werden um so die Anzahl an Datenbankzugriffen zu reduzieren und die Performance zu verbessern. Zur Einhaltung der ACID-Regel zum Schutz vor Dateninkonsistenzen, beispielsweise durch abgebrochene Operationen, bietet Hibernate die Möglichkeit, Funktionsaufrufe in Transaktionen zusammenzufassen. Auf diese Weise wurden bei der Implementierung alle Schreiboperationen in möglichst kurze Transaktionen gekapselt, um einerseits Inkonsistenzen zu verhindern und andererseits eine zu lange Blockierung bestimmter Datenbanktabellen während einer Transaktion zu verhindern.

Komplexe Anfragen, die bei der Implementierung einiger Funktionen notwendig sind, wurden mittels der Criteria-API (siehe Abschnitt 2.3.1) formuliert. Dadurch können die Anfragen dynamisch und in einem objektorientierten Kontext definiert werden, wodurch die Implementierung vereinfacht wird und der Quelltext wesentlich übersichtlicher ist, da keine komplexen und unter Umständen sehr langen SQL-Anfragen formuliert werden müssen. Zusätzlich wird der in Hibernate integrierte Second-Level-Cache unter Verwendung der Ehcache Implementierung (siehe Abschnitt 2.3.1) gezielt genutzt, um die Ergebnisse häufiger Anfragen zu cachen und möglichst lange im Speicher zu halten. Dies kommt ebenfalls der Performance zugute und betrifft unter anderem Anfragen, die Teil des Sicherheitssystems sind. So werden alle vergebenen Nutzerrechte auf den erfassten Objekten ebenfalls in der Datenbank abgelegt. Werden bestimmte Objekte häufig aufgerufen, beispielsweise wenn sich ein Nutzer durch ein Verzeichnis bewegt, dann werden die entsprechenden Rechte des Nutzers vor dem Ausführen jeder Funktion abgefragt (siehe Abschnitt 4.3.1). Dies hat zur Folge, dass die gleichen Anfragen unter Umständen in kurzer Zeit häufig ausgeführt werden und so besonders vom Anfrage-Cache profitieren, da keine erneuten Datenbankanfragen notwendig sind. Die jeweiligen Parameter der einzelnen Caches, die jeweils einer bestimmten Anfrage zugeordnet sind, werden in einer separaten XML-Datei definiert, die ebenfalls bei der Initialisierung von Hibernate geladen wird.

Suchanfragen Bei der Implementierung der Suchfunktion zur effektiven Nutzung der Infrastruktur wurde im Gegensatz zur restlichen Implementierung auf die Nutzung von Criteria-Anfragen verzichtet, da der Fokus hier stärker auf einer hohen Performance lag und die Notwendigkeit eines übersichtlichen und verständlichen Quelltextes in den Hintergrund trat. Da die Metadaten der einzelnen Datenobjekte recht umfangreich sind, wird die entsprechende Datenbanktabelle analog in kürzester Zeit sehr umfangreich. Hibernate würde entsprechende Criteria-Anfragen intern mithilfe des SQL LIKE-Operators auswerten, was jedoch nur eine unzureichende Performance liefert. Statt dessen wurde für die Implementierung der Suchfunktionen der eDAL-API eine indexbasierte Suche integriert. Die Hibernate-Search-API (siehe Abschnitt 2.3.1) bietet dafür die nötige Funktionalität. Sie nutzt die verbreitete Lucene Bibliothek für die Indexierung und Suche von textbasierten Daten und wendet diese auf Tabelleneinträge in einer relationalen Datenbank an. Dafür wurden spezielle Hibernate-Search-Annotationen in den Datentypen-Klassen (siehe Tabelle 4.2) der eDAL-API hinzugefügt. Diese legen fest, welche Attribute einer Klasse indexiert werden sollen. Gleichzeitig bietet Hibernate Search entsprechende Funktionen um Suchanfragen auf dem Index auszuführen und erlaubt dabei die Nutzung der Lucene Query Syntax (siehe Abschnitt 2.3.4), die sehr vielfältig ist und beispielsweise eine unscharfe Suche bietet, sowie weitere optionale Suchparameter unterstützt.

Standardmäßig aktualisiert die Hibernate-Search Implementierung automatisch bei jeder Persistenzoperation auch die Indexdateien. Das verhindert, dass Inkonsistenzen zwischen der Datenbank und den Indexdateien entstehen. Außerdem vereinfacht das die Implementierung erheblich, da keine zusätzlichen Funktionen zur Indexierung programmiert werden müssen. Jedoch zeigte sich, dass diese Strategie für die eDAL-Infrastruktur keine zufriedenstellende Performance liefert. Da Forschungsdatensätze häufig sehr umfangreich sind und eine Vielzahl an Dateien enthalten können, würde eine automatische Indexierung beim Speichern den Index nach jeder eingefügten Datei aktualisieren und so die Geschwindigkeit der Anwendung stark verringern. Daher wurde für die Referenzimplementierung eine manuelle Indexie-

rungsstrategie entwickelt. Es wurde ein separater Thread für die Indexierung von erfassten Objekten implementiert, der parallel zur eigentlichen Anwendung läuft. Dieser fragt alle neu hinzu gekommenen `UntypedData` Objekte in der Datenbank ab und indexiert diese. Diese Objekte enthalten die annotierten Attribute mit den Metadaten (siehe Abschnitt 4.3.1). Dabei wird die Dauer der Operation gemessen und daraus zusammen mit einem Faktor, sowie einer Ober- und Untergrenze (500 ms/2000 ms) die Zeit bis zur nächsten Indexierung ermittelt. So ergibt sich beispielsweise für eine Indexierungsdauer von 500 ms eine Wartezeit von 1000 ms bis zur nächsten Indexierung.

$$
\begin{aligned}
\text{Wartezeit}_{\text{Index}} &= \min\left(\max\left(\text{Zeit}_{\text{Index}} \cdot \text{Wartefaktor},\ \text{Wartezeit}_{\min}\right),\ \text{Wartezeit}_{\max}\right) \\
&= \min\left(\max\left(500\ \text{ms} \cdot 2,\ 500\ \text{ms}\right),\ 2000\ \text{ms}\right) \\
&= \min\left(\max\left(1000\ \text{ms},\ 500\ \text{ms}\right),\ 2000\ \text{ms}\right) \\
&= \min\left(1000\ \text{ms},\ 2000\ \text{ms}\right) \\
&= 1000\ \text{ms}
\end{aligned}
$$

Durch die festgelegte Untergrenze wird verhindert, dass eine Indexierung zu Lasten der Performance zu oft ausgeführt wird. Gleichzeitig verhindert die Obergrenze, dass neue Objekte zu lange nicht indexiert werden und damit bei der Suche nicht gefunden werden können. Die Umsetzung der manuellen Indexierungsstrategie erforderte einen erheblichen Aufwand an zusätzlicher Implementierung, da sicher gestellt werden musste, dass alle annotierten Objekte und Attribute erfasst werden und keine Inkonsistenzen zwischen Datenbank und Index entstehen. Jedoch konnte dadurch eine deutlich bessere Performance erzielt werden. So zeigten die Ergebnisse eines Leistungstests, bei dem ein Datensatz mit 1000 Objekten mit manueller Strategie gespeichert wurde, dass dieser nur etwa 1/5 der Zeit benötigt wie der gleiche Test mit der automatischen Indexierungsstrategie (siehe Abbildung 4.3).

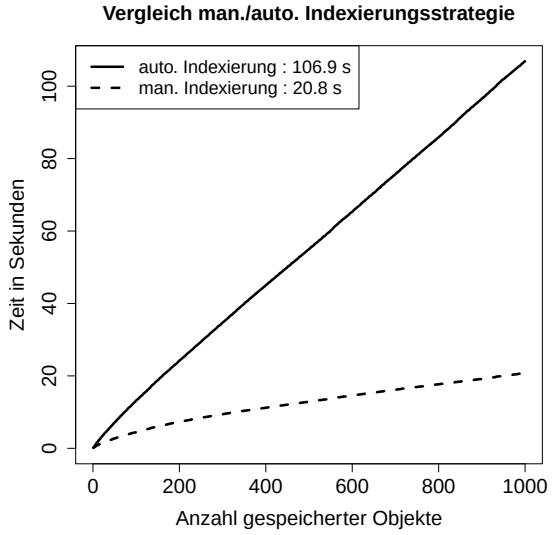

Abbildung 4.3: Vergleich der Laufzeiten mit der automatischen und der manuellen Indexierungsstrategie

Dateisystemanbindung

Wie bereits im vorherigen Abschnitt erwähnt, wurden für die Referenzimplementierung sowohl ein DBMS verwendet, also auch eine Dateisystemanbindung implementiert, die im folgenden Abschnitt erläutert wird. Diese bezieht sich auf die `PrimaryDataFile` Klasse (siehe Abschnitt 4.3.1). Diese repräsentiert eine konkrete Datei, die in einem Dateisystem abgespeichert bzw. wieder ausgelesen wird. Dabei muss sichergestellt werden, dass die richtige Datei mit dem richtigen `PrimaryDataFile` Objekt bzw. der jeweiligen Version verknüpft ist. Dabei bestand die Schwierigkeit weniger in der konkreten Anbindung an ein Dateisystem zum Schreiben und Lesen von Dateien bzw. Datenströmen, sondern viel mehr in der Entwicklung einer effizienten Verzeichnisstruktur zur Dateiablage.

Datei abspeichern und auslesen Die Ablage aller erfassten Dateien in einem einzigen angebunden Ordner im Dateisystem macht wenig Sinn, da mit zunehmenden Datenbestand das Dateisystem Laufzeitprobleme beim Zugriff auf die Dateien bekommen würde. Eine 1:1 Abbildung der originalen Dateistruktur eines jeden gespeicherten Datensatzes ist ebenfalls nicht sinnvoll, da es Datendomänen gibt, in denen Forschungsdatensätze in der Regel aus sehr vielen kleinen Dateien bestehen, die in einer verschachtelten Ordnerstruktur abgelegt sind. Dies würde mit zunehmendem Datenvolumen ebenfalls den Datenzugriff verlangsamen. Um die Ordnerstruktur im Dateisystem möglichst flach zu halten und damit einen schnellen Zugriff zu gewährleisten, legt die Implementierung bei der Dateiablage automatisch eine Ordnerstruktur nach dem Datum der jeweilig gespeicherten Version einer Datei an. Eine Version, die am 01.06.2017 (TT.MM.JJJJ) um 18:00 (hh:mm) angelegt wurde, wird im dynamisch erstellten Verzeichnis „. . . /2017/06/01/18/00/“ (JJJJ/MM/TT/hh/mm) abgelegt. Innerhalb des Ordners wird die Datei über die interne ID des dazugehörigen `PrimaryDataFile` Objektes und der dazugehörigen Versionsnummer des `PrimaryDataEntityVersion` Objektes identifiziert. Dadurch wird verhindert, dass zu viele Dateien in einen einzelnen Ordner abgelegt werden, da jeder Ordner nur maximal so viele Dateien enthalten kann wie pro Minuten gespeichert werden. Außerdem müssen die Pfade zu den jeweiligen Dateien nicht separat persistiert werden, da die Versionsinformationen bereits in der angebunden Datenbank hinterlegt sind und die Pfade daraus abgeleitet werden können. Wenn eine neue Datei gespeichert wird, erstellt die eDAL-API automatisch eine neue Version (siehe Abschnitt 4.3.1) und legt diese Informationen in der Datenbank ab (Abschnitt 4.3.2). Kommt es zu einem Fehler beim Abspeichern der Datenbankeinträge, wird ein *Rollback* ausgelöst, wodurch die ganze Transaktion rückgängig gemacht wird. Anschließend wird auch die dazugehörige Datei aus dem Dateisystem wieder gelöscht, damit keine Inkonsistenzen zwischen Dateisystem und Datenbank entstehen. Beim Auslesen einer gespeicherten Datei ermittelt die eDAL-API den Pfad zur Datei aus den Versionsinformationen der dazugehörigen `PrimaryDataEntityVersion`. Falls die ausgewählte Version mit keiner Datei verknüpft ist, weil sie beispielsweise nur erstellt wurden, da der Nutzer die Metadaten des Objektes aktualisiert hat, wird automatisch die letzte verfügbare Datei aus der Versionskette geladen. Die Anbindung an ein Dateisystem zum Schreiben und Auslesen von Dateien erfolgt mithilfe der nativen Java Filesystem API. Diese ist mit vielen gebräuchlichen Dateisystemen kompatibel und bietet alle nötigen Funktionen im Umgang mit Datenströmen.

Zusammenfassung

In den beiden vorangegangenen Abschnitten wurden die Konzepte der eDAL Infrastruktur erläutert sowie Details der `FileSystemImplementationProvider` genannten Referenzimplementierung, welche die `ImplementationProvider` Schnittstelle (siehe Abschnitt 4.3.1) implementiert, beschrieben. Diese kombiniert die Vorteile eines DBMS und eines Dateisystems, um ein nachhaltiges und performantes Back-End für die Infrastruktur zu bilden (siehe Abschnitt 4.3.2). Außerdem wird eine indexbasierte Suche bereitgestellt, die eine effiziente Auswertung von Suchanfragen über die erfassten Metadaten der Infrastruktur ermöglicht. Die entwickelte manuelle Indexierungsstrategie gewährleistet darüber hinaus eine performante Indexierung (siehe Abschnitt 4.3.2). Die integrierte JAAS-API erlaubt eine flexible Nutzerauthentifizierung durch austauschbare Anmeldemodule. Im `FileSystemImplementationProvider` sind bereits eine Reihe von Funktionen integriert, die die Nutzung von Windows-, Linux- oder Kerberos-Anmeldemodulen unterstützen. Abbildung 4.4 zeigt eine schematische Darstellung der generellen Architektur der eDAL-Infrastruktur-API.

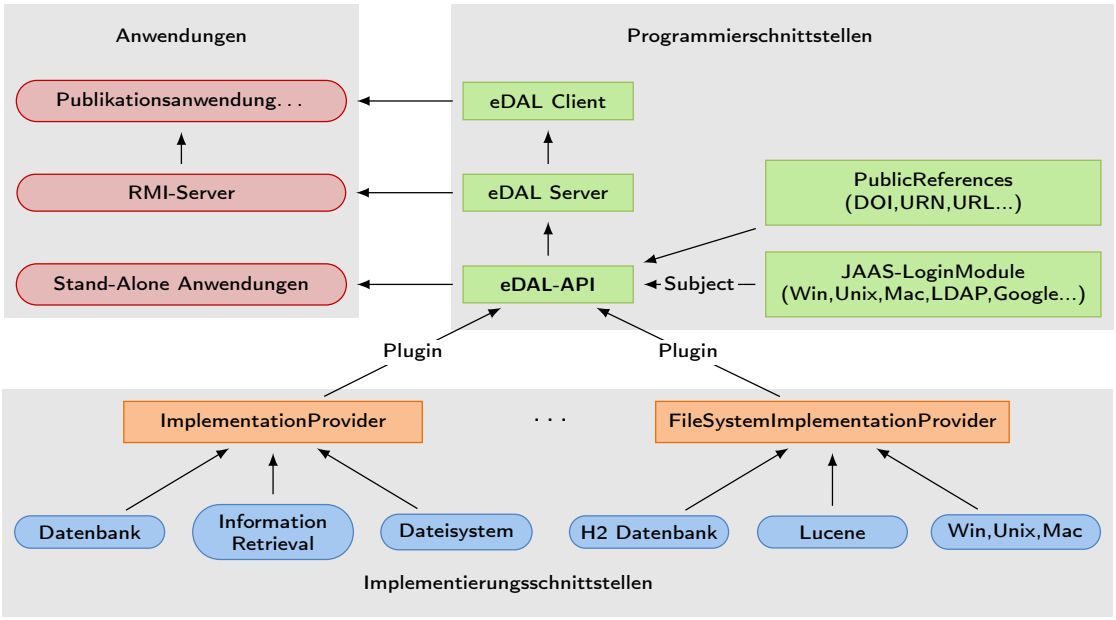

Abbildung 4.4: Schema der generellen Architektur der eDAL-Infrastruktur. An die gelb gekennzeichneten Implementierungsschnittstellen sind die blauen Backend-Komponenten angebunden. Alle Komponenten der eDAL-API sind grün gekennzeichnet. Die roten Komponenten zeigen mögliche Anwendungen der jeweiligen API-Komponenten.

Die eDAL-Infrastruktur bietet zwei grundlegende Nutzungsszenarien. Als lokale Instanz bietet sie ein effizientes Speichersystem zur langfristigen und nachhaltigen Archivierung von Forschungsdaten. Es gibt keine generellen Einschränkungen in Bezug auf Dateigrößen oder Datentypen und es können auch mehrere lokale Anwender auf das System zugreifen. Der weitaus häufigere Anwendungsfall ist jedoch die Nutzung der Infrastruktur über die bereitgestellte Server-Client-Architektur. Diese ermöglicht es, einen zentralen eDAL-Server zu initialisieren, der es mehreren Client-Anwendern erlaubt, parallel auf die Instanz der Infrastruktur zuzugreifen und diese zu nutzen, um Daten zu übertragen oder auszulesen.

4.3.3 Erweiterungen und Verbesserungen

Im Zuge der Verfassung eines Journal-Artikels (Arend et al., 2014b) zur Publikation des Konzeptes und der Umsetzung der eDAL-Infrastruktur-API wurde die Implementierung erweitert und vor allem in Bezug auf die Performance weiter verbessert. Aber auch im Anschluss an die Publikation der ersten Version der eDAL-Infrastruktur und im weiteren Verlauf des Dissertationsprojektes wurde die Referenzimplementierung fortlaufend optimiert. Es wurden neue Funktionen integriert und einige verwendete Bibliotheken ausgetauscht oder aktualisiert. Diese Änderungen werden im folgenden Abschnitt kurz erläutert.

Java Update

Während der Entwicklung des Konzeptes und der Implementierung der eDAL-Infrastruktur wurde die Java-Entwicklungsumgebung auf die Version 7 aktualisiert, jedoch konnte diese zu Beginn aufgrund fehlender Unterstützung einiger essenzieller Bibliotheken und APIs, wie beispielsweise AspectJ, zur Implementierung des Sicherheitssystems (siehe Abschnitt 4.3.1) noch nicht verwendet werden. Mit der Fertigstellung und Publikation der ersten Version der eDAL-API (Arend et al., 2014b) wurden diese notwendigen Abhängigkeiten jedoch aktualisiert und somit konnte infolgedessen auch die Referenzimplementierung auf die zu dem Zeitpunkt neuste Java Version aktualisiert werden. Jedoch hatte dies in erster Linie sicherheitsrelevante Gründe. Es konnte zu diesem Zeitpunkt noch keine der neuen Funktionalitäten der Java 7-Entwicklungsumgebung genutzt werden. Erst später konnten damit wichtige Teile der Referenzimplementierung verbessert werden. So wurde mit Java 7 eine komplett überarbeitete Dateisystem-API (Oracle, 2018d) eingeführt. Diese bietet eine Vielzahl von neuen und verbesserten Funktionen, die dazu genutzt wurden, um die Implementierungskomponenten der eDAL-API zur Anbindung an ein Dateisystem vollständig zu überarbeiten. So wurden unter anderem neu bereitgestellte Funktionen zum Auslesen und Schreiben von Dateiströmen genutzt, welche die Performance beim Speichern und Laden von Objekten in der eDAL-Infrastruktur verbessern konnten. Außerdem wurde eine neue `Path` Klasse (`java.nio.file.Path`) eingeführt, die anders als die bisherige `File` Klasse (`java.io.File`) keine konkrete Datei oder einen Ordner repräsentiert, sondern nur einen Pfad zu einem konkreten Objekt im Dateisystem. Sie bietet eine Reihe von neuen nützlichen Funktionen wie beispielsweise das rekursive Auslesen eines Ordners im Dateisystem. Diese neue Abstraktionsschicht bringt keinen Performancevorteil, war jedoch sehr hilfreich, um den Umfang des Quellcodes der Referenzimplementierung der eDAL-API zu verringern und übersichtlicher zu gestalten.

Die aktuellste Version der eDAL-API wurde bereits auf die Java Version 8 aktualisiert. Dies hatte bisher jedoch keinen Einfluss auf die Referenzimplementierung, da keine der neuen Funktionen oder APIs aus Java 8 Vorteile für die Umsetzung der eDAL-Infrastruktur bieten.

automatische Metadaten-Bestimmung

Bei der Weiterentwicklung der eDAL-API lag ein Hauptaugenmerk darauf, wie die Infrastruktur noch effektiver und nutzerfreundlicher gestaltet werden kann. Dazu sollten unter

anderem möglichst viele Metadaten automatisch bestimmt werden, um den Anwender zu entlasten. So wurde die Apache Tika Bibliothek (siehe Abschnitt 2.3.4) integriert, die in der Lage ist, den Multipurpose Internet Mail Extensions (MIME)-Type einer Datei zu bestimmen, sodass der Nutzer die Information nicht selber eintragen muss, da dies gerade bei sehr umfangreichen Datensätzen aufgrund des Zeitaufwands nahezu unmöglich wäre. Genauso wichtig wie der MIME-Type für die langfristige Nutzbarkeit ist die Größe und Prüfsumme der erfassten Dateien notwendig, um deren Integrität zu gewährleisten. Auch diese Informationen können automatisch mittels nativer Java Funktionen während der Speicherung eines Datensatzes bestimmt werden.

Multi-Thread-Optimierung

Damit das zusätzliche Erfassen von Metadaten beim Abspeichern einer Datei sich nicht zu stark auf die Performance auswirkt, wurde die Funktion zum Speichern von Objekten in der eDAL-API grundlegend verändert. Bisher wurde der übergebene `InputStream` (`java.io.InputStream`) lediglich in einer Datei im Dateisystem abgelegt, um später bei Bedarf wieder ausgelesen zu werden. Diese Funktion wurde mithilfe der nativen Java *Executor-API* erweitert. Beim Initialisieren der Infrastruktur wird ein sogenannter `ExecutorService` (`java.util.concurrent.ExecutorService`) erzeugt, der für die Speicherung von Objekten verantwortlich ist. Dieser kann bei Bedarf mehrere Threads erzeugen und während der Speicherung des Dateistroms einer Datei, deren MIME-Type, ihre Dateigröße und ihre Prüfsumme parallel berechnen und die ermittelten Informationen in den Metadaten des Objektes in der eDAL-Infrastruktur ablegen. Dadurch führen die zusätzlichen Berechnungen unter Verwendung eines aktuellen Mehrkernprozessors zu keiner Verschlechterung der Performance. Die maximale Anzahl an Threads, die der `ExecutorService` parallel verarbeiten kann, wird dabei dynamisch anhand der zur Verfügung stehenden logischen Prozessorkerne ermittelt und beträgt 2/3 der Anzahl an möglichen parallelen Threads. Je mehr logische Threads der Rechner, auf dem die Infrastruktur läuft, verarbeiten kann, desto besser ist die Performance beim Speichern von vielen Dateien bzw. verarbeiten von mehreren parallelen Clientanfragen.

5 Entwurf und Implementierung

In diesem Kapitel wird die Konzeptionierung und Implementierung eines Datenbegutachtungsprozesses sowie die Entwicklung einer grafischen Anwendung zur Einreichung und Publikation von Forschungsdaten erläutert. Die hier vorgestellten Arbeiten stellen den zweiten großen Schwerpunkt des Dissertationsprojektes dar und wurden in einem wissenschaftlichen Journal (Arend et al., 2016a) publiziert. Dabei sind die Prozesse der Konzeptionierung und Implementierung nicht als getrennte, unabhängig Schritte zu sehen, da sich einige Schwachstellen und mögliche Verbesserungen des Entwurfs erst während der Implementierung bzw. nach einer Reihe von Testläufen zeigten. Daher wurde das Konzept im Zuge der Implementierung bis zur finalen Fassung stetig verbessert. Um die Qualität des entwickelten Quellcodes für die eDAL-Infrastruktur, den integrierten Datenbegutachtungsprozess und der dazugehörigen Publikationsanwendung zu gewährleisten, wurde auf eine nachhaltige Entwicklung geachtet. Details dazu werden im letzten Abschnitt erläutert.

5.1 Datenqualität und Datenbegutachtung

Wie bereits in der Einleitung erwähnt, ist es für die Reproduzierbarkeit und Nachhaltigkeit von Forschungsergebnissen nicht ausreichend, die assoziierten Forschungsdaten langfristig zu archivieren und verfügbar zu machen. Es muss sichergestellt werden, dass die Daten auch nutzbar sind. Falsche oder unvollständige Metadaten oder proprietäre Dateiformate können dazu führen, dass verfügbare Forschungsdaten nicht wieder verwendet werden können. Die Sicherstellung einer gewissen Datenqualität muss einerseits technisch gewährleistet werden, indem technische Metadaten wie z. B. Datenformate, Dateigrößen oder Prüfsummen automatisiert erfasst werden, um eine zukünftige Lesbarkeit der Daten zu ermöglichen. Andererseits ist für eine ausreichende Datenqualität eine inhaltliche Begutachtung der Daten durch unabhängige Gutachter sowie eine Prüfung auf mögliche rechtliche Verstöße notwendig. Bei der Untersuchung von existierenden Systemen zum Forschungsdatenmanagement (siehe Abschnitt 3.6) zeigte sich deutlich, dass besonders die unzureichende Datenqualität ein Defizit verbreiteter Systeme ist und es bisher noch keine etablierten Ansätze zur Realisierung eines einfachen und dynamischen Begutachtungsprozesses für Forschungsdatenmanagementsysteme gibt. Im folgenden Abschnitt wird das Konzept eines intuitiven und generischen Datenbegutachtungsprozesses erläutert. Anschließend wird die Erweiterung der eDAL-Infrastruktur zur Integration und Implementierung des Prozesses beschrieben.

5.1.1 Konzept des Datenbegutachtungsprozesses

Die Entwicklung eines Begutachtungsprozesses für Forschungsdaten wurde stark von dem bereits bekannten Prozess der Begutachtung wissenschaftlicher Publikationen beeinflusst

(siehe Abschnitt 2.2.2). Diese werden in der Regel nach einer eingehenden Prüfung durch die Editoren des veröffentlichenden Journals oder dem Komitee einer Konferenz von mehreren unabhängigen, anonymen Gutachtern bewertet. Dieser sogenannte Peer-Review-Prozess ist in der wissenschaftlichen Gemeinschaft jedem Forscher vertraut und als Standard akzeptiert, da er eine unabhängige Beurteilung und eine hohe Forschungsqualität gewährleistet. Daher schien es sinnvoll, einen ähnlichen Prozess für die Bewertung von Daten zu realisieren. Forschungsdatensätze sollen, bevor sie mit einer DOI (siehe Abschnitt 2.2.3) dauerhaft referenzierbar gemacht werden, von mehreren Gutachtern geprüft werden.

Gutachterhierarchie und Auswertung

Da das Ziel eine generische Infrastruktur ist, stellte sich zuerst die Frage, wie die Gutachter, welche die zu publizierenden Daten bewerten, flexibel definiert werden können. Dabei müssen zwei Aspekte unterschieden werden. Einerseits ist eine wissenschaftliche Begutachtung notwendig. Diese erfordert Gutachter mit einer Expertise in einer ähnlichen Domäne wie die zu beurteilenden Daten bzw. der Autor der Daten, um die wissenschaftliche Qualität und Korrektheit zu beurteilen. Andererseits ist eine Bewertung von administrativer Seite notwendig, um zu prüfen, ob der Autor die notwendigen Rechte besitzt, um die Daten zu veröffentlichen. So wird verhindert, dass Lizenzen, Patente oder Kooperationsverträge verletzt werden oder eventuell andere institutionelle Regularien gegen eine Veröffentlichung sprechen. Es gibt zwar keine definierten Standardkriterien, die bei der Begutachtung von Datenpublikationen geprüft werden sollten, jedoch gibt es einige Initiativen (siehe Abschnitt 2.2), die Empfehlungen mit minimal notwendigen Informationen verfasst haben.

- wissenschaftliche Kriterien:
  Hypothese, verwendetes Material, angewandte Methoden, experimentelle Parameter, Ergebnisse, domänenspezifische Datei- und Metadatenformate

- rechtliche Kriterien:
  Eigentümer/Erzeuger der Daten, Verwertung (Lizenz), verwendete Patente

Für die meisten Anwendungsfälle und Datendomänen sollte eine Begutachtung durch zwei wissenschaftliche Gutachter und einen administrativen Gutachter ausreichend sein. Die Bewertung der einzelnen Gutachter wird mithilfe einer Entscheidungstabelle (siehe Tabelle 5.1) ausgewertet und führt zu einer finalen Entscheidung, ob ein Datensatz veröffentlicht werden kann oder geändert werden muss. Dabei kann die Entscheidung des wissenschaftlichen Gutachters und seines Stellvertreters durch den administrativen Gutachter außer Kraft gesetzt werden.

	Entscheidungen			
wissenschaftlicher Gutachter	✓	?	✓	?
stellvertretender wissenschaftlicher Gutachter	✓	✓	✓	✓
administrativer Gutachter	✓	✓	X	X
finale Entscheidung	✓	✓	X	X

Tabelle 5.1: Entscheidungstabelle des Datenbegutachtungsprozesses. Eine positive Begutachtung ist mit „✓“ markiert. Abgelehnte Anfragen sind mit „X“ dargestellt. Anfragen, die nicht innerhalb eines definierten Zeitraums bearbeitet wurden, sind mit „?“ vermerkt.

Diese Gutachterhierarchie und die dazugehörigen Entscheidungsregeln garantieren eine hohe Datenqualität und Verhindern die Veröffentlichung von geschützten Daten. Darüber hinaus sollte die Entscheidungstabelle sowie die Anzahl und die Hierarchie der Gutachter für komplexere Anwendungsfälle anpassbar sein.

Datenübermittlung und Initialisierung der Begutachtung

Zur Initialisierung der Begutachtung muss der Autor seine Daten übertragen und mit einem minimalen Satz an technischen Metadaten beschreiben. Diese gewährleisten eine Nachnutzung der Daten und sind für die Vergabe einer DOI notwendig. Dabei steht die Benutzerfreundlichkeit im Vordergrund, um die Autoren zu unterstützen und die Übertragung und Beschreibung der Daten einfach und intuitiv zu gestalten. Auch dieser Teil des Datenpublikationsprozesses ist an die Publikation von wissenschaftlichen Artikeln angelehnt. Ein formularbasierter Dialog, in dem der Autor die Metadaten in entsprechende Felder einträgt und über einen Auswahldialog im Dateimanager seine zu publizierenden Daten auswählt, erfüllt diese Kriterien und ähnelt den üblichen Dialogen zur Einreichung einer wissenschaftlichen Publikation. Erst wenn der Autor alle notwendigen Informationen eingetragen hat, werden die Daten übertragen und der Begutachtungsprozess gestartet. Die Authentifizierung des Nutzers erfolgt über eine Anmeldemaske. Dabei ist die Unterstützung existierender Authentifizierungsverfahren günstig, weil dies die Hürde für die Nutzung der Infrastruktur reduziert, da kein zusätzliches Benutzerkonto erstellt werden muss.

Neben der Beschreibung der Metadaten und Auswahl der zu publizierenden Daten ist die optionale Vergabe eines Zeitpunktes für ein Embargo sinnvoll, um zu verhindern, dass ein mit einer DOI verlinkter Datensatz vor einem definierten Zeitpunkt abrufbar ist. Dies kann z. B. notwendig sein, wenn die DOI Bestandteil eines wissenschaftlichen Artikels ist. Häufig schreiben Verlage in ihren Richtlinien vor, dass Daten, die Bestandteil einer Publikation sind, nicht vor der finalen Veröffentlichung des Artikels frei verfügbar sein dürfen. Da jedoch für den finalen Druck eines Artikels die DOIs der verknüpften Datensätze bereits vergeben sein müssen, ist dies nicht möglich. Der DataCite Resolving-Service (siehe Abschnitt 2.2.3) bietet selbst keine Möglichkeit dieses Problem zu lösen. Daher ist die Infrastruktur des jeweiligen Datenzentrums, das die Inhaltsseiten der DOIs bereitstellt, dafür verantwortlich, eine entsprechende Lösung zu implementieren. Es muss intern sichergestellt werden, dass die Inhaltsseite einer DOI zwar bereits bei deren Vergabe auflösbar ist, jedoch den Zugang zu den Daten erst nach Ablauf des eingestellten Zeitpunktes des Embargos freigibt.

Datenaustausch und Kommunikation

Nachdem die Anforderungen an die Autoren- und Gutachterseite des Prozesses beschrieben wurden, stellt sich die Frage, wie die beiden Komponenten verknüpft werden und miteinander kommunizieren. Die Gutachter müssen Publikationsanfragen automatisch erhalten und Zugriff auf die Daten bekommen, um sie zu beurteilen und ihre Entscheidung zu übermitteln. Die Autoren wiederum müssen die Ergebnisse der Begutachtung und im Idealfall die finale DOI für ihren Datensatz erhalten. Daher scheint eine Kommunikation per E-Mail am sinnvollsten. Nachdem ein Autor seinen Datensatz erfolgreich übertragen hat, bekommen alle Gutachter eine E-Mail mit personalisierten URLs, über die sie einen beschränkten Zugang

zum Datensatz erhalten und die Anfrage annehmen oder ablehnen können. Die Antworten werden von der Publikationsinfrastruktur ausgewertet. Anschließend erhält der Autor des Datensatzes ebenfalls ein E-Mail mit der Entscheidung der Gutachter. Im positiven Fall ist eine personalisierte URL mit einem vorläufigen Onlinezugang zum Datensatz enthalten. Außerdem gibt es die Möglichkeit die finale DOI zu erstellen oder die Anfrage zurück zu ziehen. Nachdem die Metadaten und die URL, über die der Datensatz abrufbar ist, vom Publikationssystem an den DOI-Resolver (siehe Abschnitt 2.2.3) übertragen wurden, erhält der Autor eine E-Mail mit der vergebenen DOI für seinen Datensatz.

Zusammenfassung

Das vorgestellte Konzept eines Datenpublikations- und Begutachtungsprozesses orientiert sich an den gewohnten Publikationsverfahren für die Einreichung von wissenschaftlichen Artikeln. Um die Hürde für die Autoren zur Veröffentlichung ihrer Forschungsdaten sowie den Aufwand der Bewertung für die Gutachter gering zu halten, ist es nötig, den Prozess so benutzerfreundlich und mit so wenigen Zwischenschritten wie möglich zu gestalten. Ein selbsterklärender, formularbasierter Dialog zur Übertragung und Beschreibung der Daten, eine einfache Kommunikation per E-Mail und eine webbasierte Datenbegutachtung erhöhen die Nutzerakzeptanz. Im Ablaufdiagramm in Abbildung 5.1 sind die einzelnen Schritte des Prozesses zusammengefasst.

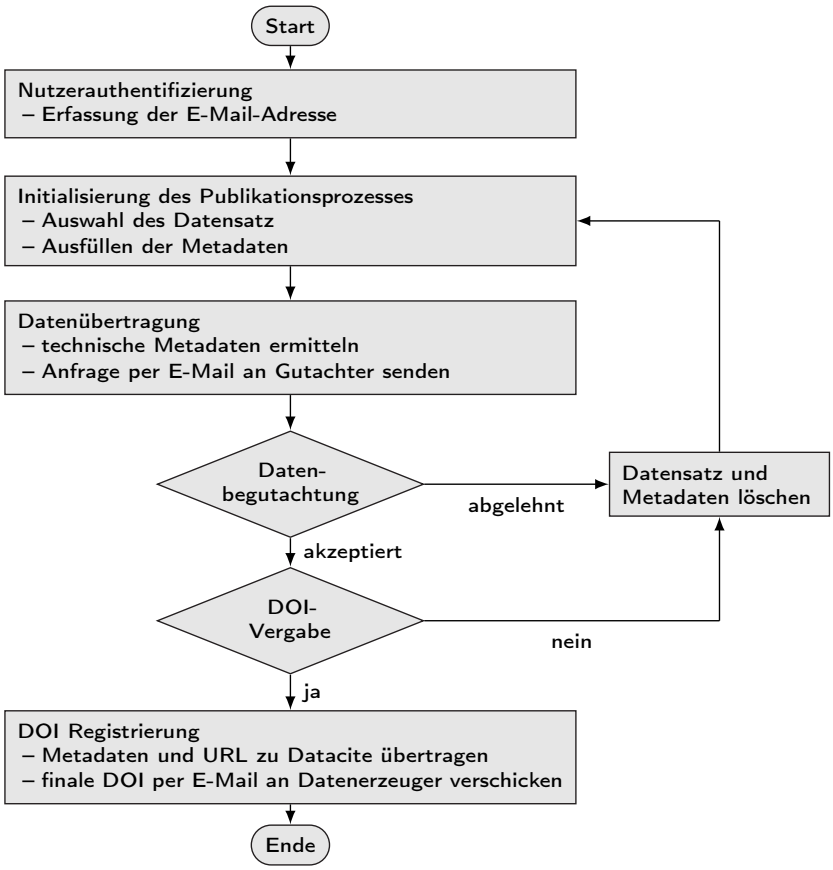

Abbildung 5.1: Ablauf des generischen Datenpublikations- und Begutachtungsprozesses

5.1.2 Implementierung des Datenbegutachtungsprozesses

In diesem Abschnitt wird die Implementierung des entworfenen Datenpublikations- und Begutachtungskonzepts vorgestellt. Nachdem die notwendigen funktionellen Erweiterungen der eDAL-Infrastruktur erläutert wurden, werden im Anschluss die Implementierungsdetails der neuen Komponenten beschrieben. Zum Schluss wird die Entwicklung einer grafischen Anwendung zur Beschreibung und Einreichung der Daten bzw. zur Initialisierung der Datenbegutachtung vorgestellt. Diese bildet das Kernstück des Prozesses.

Erweiterung der eDAL-API

Wie in Abschnitt 4.3.1 beschrieben wurde, besitzt die eDAL-Infrastruktur mit der `PublicReference` Klasse bereits eine Möglichkeit zur Einbindung von persistenten IDs für gespeicherte Datenobjekte bzw. Versionen. Diese wurde im Zuge der Implementierung des Datenbegutachtungsprozesses um zusätzliche Attribute erweitert. Es wurden unter anderem eine Reihe von Variablen zur Erfassung von verschiedenen Zeitpunkten, beispielsweise wann eine persistente ID angefragt und vergeben wurde, hinzugefügt. Für die Implementierung wird eine Verbindung zur REST-API (siehe Abschnitt 2.3.5) des DataCite-Konsortiums zur Vergabe von DOIs realisiert. Um das generische Grundprinzip der eDAL-API bei zu behalten und damit auch andere ID-Systeme zu unterstützen, wurde eine zusätzliche `Referenceable` Schnittstelle definiert. Diese enthält alle notwendigen Funktionen zur Kommunikation mit dem jeweiligen ID-Resolver (siehe Abbildung 5.2).

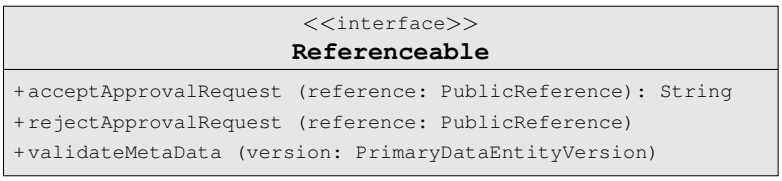

Abbildung 5.2: UML-Klassendiagramm der `Referenceable` Schnittstelle

So können dynamisch weitere Implementierungen für andere ID-Systeme hinzugefügt und genutzt werden. Jede `PublicReference` Instanz enthält daher ebenfalls ein Objekt, das diese Schnittstelle implementiert, und legt damit den Typ der ID fest. Außerdem wird erfasst, welcher Nutzer die ID angefragt hat und wie der aktuelle Status der Freigabe ist. Dieser wird durch ein Objekt des Aufzählungstyps `PublicationStatus` bestimmt und kann vier Zustände annehmen:

- `SUBMITTED`: Es wurde noch keine persistente ID vergeben.
- `UNDER_REVIEW`: Der Begutachtungsprozess wurde initialisiert.
- `ACCEPTED`: Der Begutachtungsprozess wurde erfolgreich abgeschlossen.
- `REJECTED`: Die Publikationsanfrage wurde abgelehnt und keine ID vergeben.

Über die öffentliche `setPublic()` Funktion der `PublicReference` Klasse kann der Begutachtungsprozess initialisiert werden.

Eine weitere Änderung wurde an der Implementierungsschnittstelle der eDAL-API (siehe Abschnitt 4.3.1) vorgenommen. Diese wurde um einen zusätzlichen Provider ergänzt. Er enthält alle notwendigen Backend-Funktionen des Begutachtungsprozesses,

wie z. B. Persistenzoperationen und Funktionen zur Auswertung der Gutachten. Diese `ApprovalServiceProvider` genannte Schnittstelle und deren Funktionen sind in Abbildung 5.3 abgebildet.

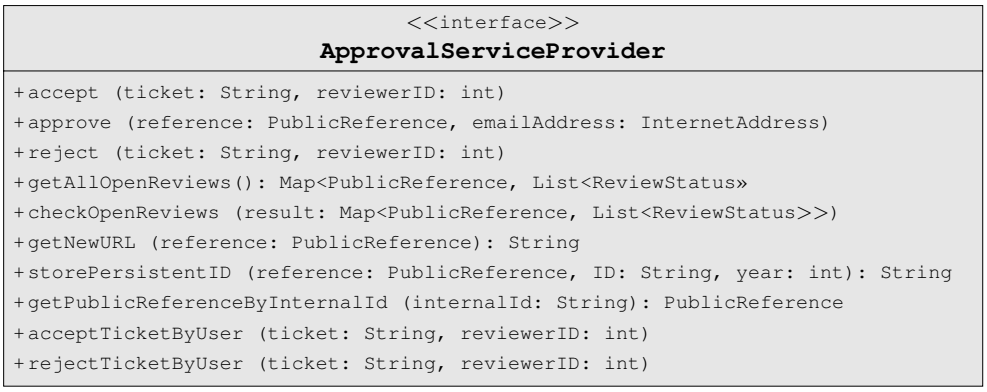

Abbildung 5.3: UML-Klassendiagramm der `ApprovalServiceProvider` Schnittstelle

Für die Auswertung der Beurteilungen der Gutachter wurde die Klasse `ReviewProcess` mit einer `review(List<ReviewStatus>reviewList)` Methode implementiert. Außerdem wird in dieser Klasse über den Aufzählungstyp `ReviewerType` die Liste von Gutachtern definiert und in einer separaten Konfigurationsdatei die Hierarchie der Gutachter sowie die Entscheidungstabelle zur Auswertung festgelegt. Für die Referenzimplementierung ist diese wie im Abschnitt 5.1.1 beschriebenen Konzept aufgebaut und besteht aus einem administrativen Gutachter und zwei wissenschaftlichen Gutachtern. Außerdem werden definierte Zeitbeschränkungen berücksichtigt. Ist z. B. ein Begutachtungsprozess nach sechs Monaten nicht abgeschlossen, weil noch kein Gutachter eine Entscheidung getroffen hat, wird der Prozess abgebrochen und die Anfrage abgelehnt.

Die Ergebnisse der einzelnen Entscheidungen der Gutachter sind in der `ReviewStatus` Klasse abgebildet (siehe Abbildung 5.4 auf Seite 75). Zur Auswertung einer Anfrage wird eine Liste von `ReviewStatus` Objekten benötigt. Insgesamt werden vier verschiedene Status unterschieden, die im `StatusType` Aufzählungstyp abgebildet sind:

- `ACCEPTED`: Die Veröffentlichung des Datensatzes wurde genehmigt.
- `REJECTED`: Die Veröffentlichung des Datensatzes wurde abgelehnt.
- `TIMEOUT`: Die Begutachtung wurde nicht in der vorgegebenen Zeit durchgeführt.
- `UNDECIDED`: Der Datensatz ist noch in der Begutachtung.

Die größte Erweiterung der eDAL-Infrastruktur-API war die Integration eines eingebetteten Webservers. Dieser erfüllt mehrere Funktionen und ist essenziell für die Funktion des Begutachtungsprozesses und die Publikation von Datensätzen. Er ist in erster Linie für die Bereitstellung von webbasierten Seiten zuständig, um erfasste Datensätze darzustellen und URLs für diese zu generieren, auf welche die jeweilig assoziierten DOIs aufgelöst werden. Dabei sind über die Inhaltsseite sowohl die erfassten Metadaten sichtbar als auch die konkreten Dateien des Datensatzes abrufbar. Die zweite wichtige Aufgabe des Webservers ist die Realisierung der E-Mail-Kommunikation zwischen den Autoren und Gutachtern mit der Infrastruktur. Der Webserver stellt personalisierte URLs zur Verfügung, die per E-Mail verschickt werden, und wertet diese aus, sobald sie angeklickt wurden. So können

beispielsweise die Antworten der Gutachter registriert werden, indem sie einen Link für die Annahme und einen für die Ablehnung einer Publikationsanfrage erhalten.

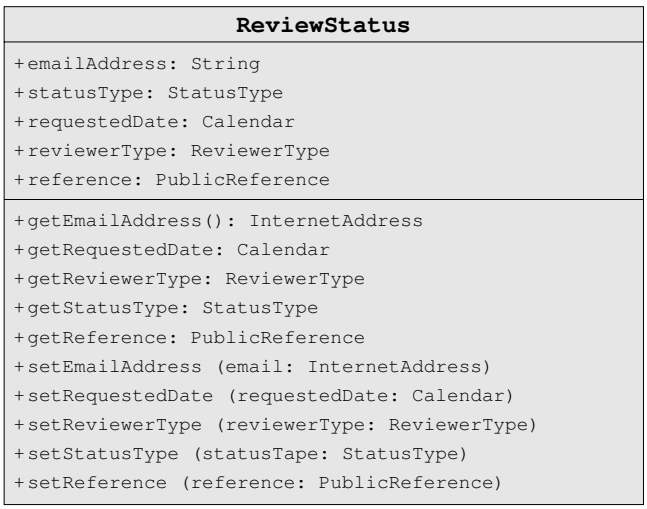

Abbildung 5.4: UML-Klassendiagramm der `ReviewStatus` Klasse

Implementierung

Nachdem die notwendigen Erweiterungen der eDAL-API für die Integration des Datenbegutachtungsprozesses erläutert wurden, wird in diesem Abschnitt beschrieben, wie die jeweiligen Funktionen in der Referenzimplementierung umgesetzt bzw. integriert wurden.

DataCite-REST-API Wie im vorangegangenen Abschnitt erläutert, ist für die Vergabe einer persistenten ID die Kommunikation mit dem Resolver-Service eines ID-Systems, in dem Fall DataCite als Konsortium für die Vergabe von DOIs (siehe Abschnitt 2.2.3), notwendig. Dafür wurde eine `DataCiteReferenceable` Klasse erstellt, welche die vorgestellte `Referenceable` Schnittstelle implementiert. DataCite bietet eine REST-API (DataCite, 2018b), welche die Registrierung von DOIs und den dazugehörigen Metadaten bzw. der zu verweisenden URL ermöglicht und so programmatisch in die Implementierung eingebunden werden kann. Zur Anmeldung an der API ist ein Zugang als registriertes Datenzentrum in DataCite-Konsortium notwendig. Die erforderlichen Zugangsdaten, bestehend aus dem Präfix des Datenzentrums, dem Nutzernamen und dem Nutzerpasswort, wurden als zusätzliche Parameter in die Konfiguration der eDAL-API aufgenommen und werden bei der Initialisierung des Systems abgefragt. Um eine DOI zu registrieren, muss erst ein entsprechender Metadatensatz als XML-Dokument über einen `POST`-Aufruf (siehe Abschnitt 2.3.5) angelegt werden. Anschließend können eine DOI und eine dazugehörige URL, auf die der Resolver später weiterleitet, ebenfalls mittels `POST` erstellt werden. Für die Implementierung der entsprechenden REST-Anfragen wurde die Jersey-API (siehe Abschnitt 2.3.5) genutzt. Diese bietet eine Reihe von Funktionen zur Authentifizierung an einer REST-Schnittstelle und zum Ausführen von Anfragen. Quelltext 5.1 auf Seite 76 zeigt die Nutzung der Jersey-API zur Verbindung mit der DataCite-REST-Schnittstelle und zum Absenden einer Anfrage zur Registrieren einer DOI.

```
Client client = Client.create();

client.addFilter(new HTTPBasicAuthFilter("dataCiteUser","dataCitePasswort"));

WebTarget dataCiteAPI = client.target("https://mds.datacite.org/doi");

String request = "doi=10.5447/IPK/2018/1 url=https://doi.ipk-gatersleben.de/url"

Response response = dataCiteAPI.entity(request).post();

if(response.getStatus() == 200){
   System.out.println("DOI wurde registriert");
}
```

Quelltext 5.1: Jersey-API-Aufruf zum Verbinden mit der DataCite-REST-Schnittstelle und Registrieren einer neuen DOI über die `POST`-Methode

Um das notwendige XML-Dokument mit den beschreibenden Metadaten zu erstellen, bietet DataCite ein entsprechendes XML-Schema an, das auch zur Validierung genutzt werden kann. Vorher wird das `MetaData` Objekt des Datensatzes der veröffentlicht werden soll in das DataCite-konforme Schema übertragen und mithilfe der nativen Java XML-API (`javax.xml.*`) in ein XML-Dokument umgewandelt. Anschließend wird es validiert, um sicherzustellen, dass alle notwendigen Attribute enthalten sind. Da eine lokale Validierung eine wesentlich bessere Performance bietet als eine Onlinevalidierung über die DataCite-Webseite und die Dateien des XML-Schemas sehr klein sind, wurden diese direkt als Ressourcen in die Referenzimplementierung eingebunden. Der einzige Nachteil dabei ist, dass bei einem Update des Schemas diese Dateien ausgetauscht werden müssen. Da das DataCite-Schema jedoch sehr stabil ist und nur selten minimal aktualisiert wird, ist dieser Nachteil im Vergleich zur besseren Performance vertretbar. Außerdem werden auch ältere Versionen des Schemas weiterhin unterstützt, sodass die Dateien nicht umgehend aktualisiert werden müssen, sobald ein neues Schema veröffentlicht wird. DataCite gibt keine Regeln für das Benennungsschema, nach dem das Suffix einer DOI erzeugt werden soll, vor. Jedes Datenzentrum entscheidet selbst, welches Schema am sinnvollsten erscheint. Für die Referenzimplementierung wurde ein generisches Schema gewählt. Eine DOI wird mit dem bei DataCite registrierten Kürzel des veröffentlichenden Datenzentrums und der aktuellen Jahreszahl benannt und dann fortlaufend nummeriert (siehe Abbildung 5.5). Damit das

Abbildung 5.5: Generisches Benennungsschema des DOI-Suffix der eDAL-Infrastruktur

Suffix einer DOI dynamisch erstellt werden kann und keine zusätzlichen Informationen wie beispielsweise die aktuelle Zahl an veröffentlichten DOIs pro Jahr gespeichert werden müssen, nutzt die Implementierung die ebenfalls von DataCite bereitgestellte Such-API (DataCite, 2018d). Dies ist eine Solr-basierte Schnittstelle (siehe Abschnitt 2.3.4), die eine Vielzahl an verschiedene Suchanfragen über alle registrierten DOIs ermöglicht. So kann, wie in Quelltext 5.2 auf Seite 77 gezeigt, die Anzahl an registrierten DOIs für ein Datenzentrum im aktuellen Jahr angefragt werden, um die fortlaufende Nummer für die nächste DOI

dynamisch zu ermitteln. Die Solr-API ist Bestandteil der Lucene-Bibliothek, welche bereits durch die Einbindung der Hibernate-Search-API Teil der Referenzimplementierung ist (siehe Abschnitt 4.3.2). Über die bereitgestellten Funktionen konnten die Anfragen an die DataCite Such-API gesendet und ausgewertet werden.

```
HttpSolrClient solrClient = HttpSolrClient.build("https://search.datacite.org/api");

ModifiableSolrParams solrParams = new ModifiableSolrParams();

solrParams.set(CommonParams.Q, "10.5447"+" and doi:"+"10.5447"+"*"+IPK+"*"+"2018"+"*");
solrParams.set(CommonParams.FL, "doi");

int nextDoiNumber = 0;

    if (solrClient.query(solrParams).getResults().size() != 0) {
       nextDoiNumber = solrClient.query(solrParams).getResults().size() - 1;
    }

System.out.println("Die naechste DOI ist: " + nextDoiNumber);
```

Quelltext 5.2: Suchanfrage über die Solr-API an die DataCite Such-Schnittstelle zur Erfassung der Anzahl an registrierten DOIs im aktuellen Jahr

ApprovalServiceProvider Die umfangreichste Aufgabe bei der Implementierung des erwähnten `ApprovalServiceProviders` war die Programmierung zusätzlicher Persistenzoperationen. Wie in Abschnitt 4.3.2 beschrieben, werden in der Referenzimplementierung alle notwendigen Objektklassen mit Hibernate in eine H2-Datenbank gespeichert. Für die Realisierung des Datenbegutachtungsprozesses mussten zusätzliche Klassen persistent gespeichert werden, sodass auch die ursprüngliche Hibernate-Konfiguration des `FileSystemImplementationProviders` erweitert wurde. Die Tabelle für die `PublicReference` Klasse wurde um die neu hinzugefügten Attribute erweitert und es wurde eine zusätzliche Tabelle für Objekte der `ReviewStatus` Klasse erstellt. Außerdem wurden zwei interne Klassen `Reviewer` und `Ticket` hinzugefügt, die ebenfalls in der Datenbank erfasst werden. Sie dienen zum Speichern der Informationen über die Gutachter, wie z. B. deren E-Mail-Adressen, die ebenfalls in die Konfiguration der eDAL-Infrastruktur aufgenommen wurden und beim Initialisieren des Systems definiert werden müssen. Außerdem wird jedem Begutachtungsprozess ein Ticket zugeordnet, um unter anderem zu prüfen, welche Anfragen noch unbeantwortet sind, und ob gegebenenfalls nach Ablauf eines konfigurierbaren Zeitintervalls eine Erinnerung an die Gutachter verschickt werden soll. Das standardmäßige Intervall wurde auf 14 Tage gesetzt. Das erweiterte Datenbankschema mit den neuen Klassen ist in Anhang A.4 auf Seite 128 abgebildet.

Viele Persistenzoperationen in der Implementierung des `ApprovalServiceProviders` werden durch den Webserver angesteuert. Dies geschieht z. B., wenn ein Gutachter eine der personalisierten URLs in einer E-Mail mit einer Publikationsanfrage anklickt. Die von ihm getroffene Entscheidung, ob ein Datensatz veröffentlicht werden darf, wird durch die entsprechende Funktion im `ApprovalServiceProvider` in der Datenbank gespeichert und später ausgewertet. Dafür wird bei der Initialisierung des `ApprovalServiceProviders` mithilfe der Quartz-API (siehe Abschnitt 2.3.4) ein zeitgesteuerter Prozess gestartet, der in einem festgelegten Intervall offenen Anfragen, die in der Datenbank abgelegt sind, prüft.

Dabei kontrolliert er, ob neue Bewertungen der Gutachter hinzugefügt wurden, um gegebenenfalls eine neue Auswertung über die Entscheidungstabelle in der `ReviewProcess` Klasse zu starten. Führt die Auswertung dazu, dass ein offener Prozess abgeschlossen werden kann, werden anschließend die dafür notwendigen Funktionen ausgeführt. Wird eine Publikationsanfrage akzeptiert, löscht der `ApprovalServiceProvider` die Anfrage und das dazugehörige Ticket aus der Datenbank und ruft die Funktion zur Registrierung einer DOI in der `DataCiteReferenceable` Klasse auf. Falls eine Anfrage abgelehnt wird, wird diese ebenfalls gelöscht und der Datenerzeuger, der die Anfrage übermittelt hat, wird per E-Mail darüber informiert.

Der Versand von E-Mails im Zuge des Datenbegutachtungsprozesses an die Gutachter bzw. Autoren ist daher eine weitere Aufgabe des `ApprovalServiceProviders`. Mittels der nativen Java Mail-API (`javax.mail.*`) können die entsprechenden Nachrichten erzeugt und verschickt werden. Zur einfachen Konfiguration wurde eine Funktion implementiert, die automatisch beim Initialisieren der eDAL-Infrastruktur nach verfügbaren Simple Mail Transfer Protocol (SMTP)-Servern im Netzwerk sucht, die zum E-Mail-Versand genutzt werden können. Wird kein Server gefunden oder schlägt der Aufbau einer Testverbindung aus bestimmten Gründen fehl, muss die Adresse und der Port eines geeigneten E-Mail-Servers manuell beim Starten des Systems definiert werden. Da jede E-Mail eine Reihe von individuellen Inhalten enthält, wie z. B. die URLs für den eingeschränkten Datenzugriff für die Gutachter bzw. die URLs zum Akzeptieren oder Ablehnen einer Publikationsanfrage, ist es schwierig, fertige E-Mail-Nachrichten zu speichern. Diese müssen dynamisch nach Bedarf erstellt werden und jeweils mit den notwendigen Informationen und Links versehen werden, die später vom Webserver ausgewertet werden können. Daher wurden für die Implementierung Templates für die entsprechenden E-Mail-Nachrichten erstellt. Diese werden dann bei Bedarf mithilfe der Velocity-API (siehe Abschnitt 2.3.6) dynamisch mit den individuellen Werten gefüllt und verschickt. Quelltext 5.3 zeigt eine gekürzte Fassung des Templates für eine E-Mail-Benachrichtigung an einen Gutachter zur Prüfung eines Datensatzes. Dies ermöglicht einerseits eine übersichtliche und einfach zu pflegende Implementierung, da bei Änderungen nur einige wenige Templates bearbeitet werden müssen, und macht andererseits eine Speicherung von fertigen E-Mail-Nachrichten überflüssig.

```
<html>
<head><title>Please approve the public reference</title></head>
  <body>
    Dear eDAL-$reviewerType reviewer,

    we recieved a data publication request by $principal. Please check the
    submission carefully and state if it can be published as $id data publication.

     preview: <a href="$landingPageURL">$infoString</a>

    As eDAL scientific reviewer you have the option to accept or reject the request.

    <a href="$acceptURL">YES</a>,the data can be published as $id citable reference.
    <a href="$rejectURL">NO</a>,I have major doubts and reject the submission.

   Thanks for your kind support, the eDAL research data publication system!
  </body>
</html>
```

Quelltext 5.3: VeloCity-Template für die E-Mails an die Gutachter. Die fett hervorgehobenen Variablen werden dynamischen beim Erzeugen einer E-Mail gesetzt.

Webserver Die wichtigste Komponente des Datenbegutachtungsprozesses, die gleichzeitig den größten Aufwand bei der Implementierung erforderte, ist der integrierte Webserver. Er ist für die Auslieferung von webbasierten Inhalten, welche die publizierten Datensätze darstellen, sowie für die Auswertung der Kommunikation zwischen den Gutachtern und Datenerzeugern mit der Infrastruktur verantwortlich. Da die Referenzimplementierung der eDAL-Infrastruktur alle notwendigen Komponenten enthalten und ohne zusätzliche externe Abhängigkeiten lauffähig sein soll, wurde für die Entwicklung ein Jetty-basierter Webserver (siehe Abschnitt 2.3.5) verwendet. Dieser ermöglicht es aus einer Anwendung heraus, einen eingebundenen Webserver zu starten, um HTTP-Anfragen entgegen zunehmen und auszuwerten. Bei der Initialisierung der Infrastruktur wird der Webserver, der in der `EdalHttpServer` Klasse implementiert ist, automatisch gestartet und unterstützt in der Standardkonfiguration das HTTP-Protokoll zur Datenübertragung. Da die Datensicherheit sehr wichtig ist, wurde darüber hinaus ebenfalls eine Möglichkeit zur verschlüsselten Datenübertragung über das Hypertext Transfer Protocol Secure (HTTPS)-Protokoll implementiert. Die dafür notwendigen Secure Sockets Layer (SSL)-Zertifikate werden über ein sogenannten Java *KeyStore* eingebunden. Über zusätzliche Parameter für den Pfad und das Passwort des KeyStore werden diese bei der Initialisierung der Infrastruktur gebunden. Der Webserver wird mit dem `EdalHttpHandler` assoziiert, welcher die Funktionen zur Verarbeitung von Anfragen enthält. Damit dieser weiß, welche Funktionen ausgeführt und welche Daten bzw. Webseiten ausgeliefert werden sollen, sind alle verfügbaren Funktionen in einen Aufzählungstyp `EdalHttpFunctions` definiert:

- `ACCEPT`: Akzeptieren eines zu publizierenden Datensatzes durch einen Gutachter
- `REJETC`: Ablehnen eines zu publizierenden Datensatzes durch einen Gutachter
- `DOI`: Anzeigen der Inhaltsseite für einen mit einer DOI veröffentlichten Datensatz
- `REPORT`: Anzeigen einer Übersichtsseite mit Zugriffszahlen aller erfassten Datensätze
- `DOWNLOAD`: Initialisierung des Downloads für eine Datei in einem Datensatz
- `ZIP`: Initialisierung des Downloads eines ZIP Archivs für einen Ordner eines Datensatzes
- `USER_ACCEPT`: Erzeugen der finalen DOI durch den Datenerzeuger
- `USER_REJECT`: Ablehnen der Anfrage durch den Datenerzeuger

Der `EdalHttpHandler` kann nur Anfragen verarbeiten, deren URL mit einer dieser Funktionen beginnt. Die weiteren Bestandteile der Anfrage enthalten dann die notwendigen Parameter für die entsprechende Funktion, wie beispielsweise die ID eines `PublicReference` Objektes auf das die jeweilige Funktion angewendet und welche Antwort zurück geliefert werden soll.

In den meisten Fällen liefert eine Clientanfrage an den Webserver eine Antwort in Form einer Webseite zurück. Das kann eine Inhaltsseite sein, auf die eine DOI verlinkt ist, welche die Metadaten eines Datensatzes anzeigt und alle enthaltenen Dateien verlinkt, oder auch eine einfache Statusseite, die kurze Benachrichtigungen beispielsweise über den Eingang einer Begutachtung ausliefert. Da eine DOI eine dauerhafte und unveränderliche ID auf ein digitales Objekt ist und auch die eDAL-Datenstruktur (siehe Abschnitt 4.3.1) keine Änderungen an einer einmal angelegten Version eines Objektes erlaubt, sind auch die Inhaltsseiten eines Datensatzes unveränderlich. Daher wäre es möglich, beim Anlegen eines Objektes ebenfalls eine entsprechende Inhaltsseite als HTML-Dokument zu erzeugen und dauerhaft abzuspeichern, um diese bei einer Anfrage über den Webserver auszuliefern. Dies hätte zwar bei einer Anfrage einen Performancevorteil, der jedoch aufgrund der geringen

Größe einer solchen HTML-Datei eher gering ausfällt. Gleichzeitig würde sich die Performance beim Anlegen eines Objektes minimal verschlechtern. Da jedoch viele Datensätze häufig aus mehreren einzelnen Dateien bestehen, müsste für jedes dieser Objekte ebenfalls eine Inhaltsseite gespeichert werden, sodass mit zunehmender Größe des Datenbestands eine große Menge zusätzlicher Speicherplatz notwendig wäre, um alle HTML-Dokument abzuspeichern. Um dies zu vermeiden, wurde wie bei der Generierung von E-Mails die bereits integrierte Velocity-API (siehe Abschnitt 2.3.6) genutzt. Diese kann alle HTML-Dokumente dynamisch bei Bedarf auf Basis des entsprechenden Templates generieren und auszuliefern. Abbildung 5.6 zeigt einen Screenshot einer solchen Inhaltsseite.

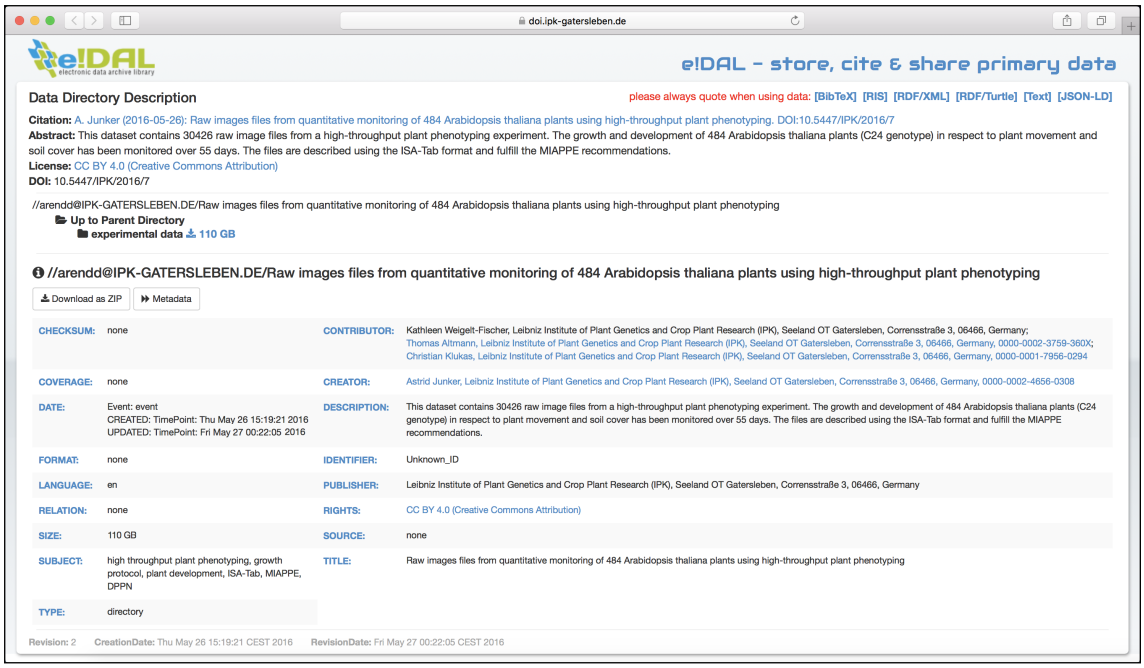

Abbildung 5.6: Inhaltsseite für einen DOI verlinkten Datensatz der eDAL-Infrastruktur. Im oberen Bereich ist die inhaltliche Zusammenfassung des Datensatzes abgebildet. Darunter können die einzelnen Dateien des Datensatzes durchsucht werden. Im unteren Abschnitt sind die jeweiligen Metadaten des Objektes aufgelistet.

Um den geringen Performanceverlust beim dynamischen Erzeugen der HTML-Seiten zu reduzieren und um außerdem häufig angefragte Datensätze möglichst schnell auszuliefern, wurde dafür mithilfe der bereits genutzten Ehcache-API (siehe Abschnitt 4.3.2) ein separater Cache speziell für angefragte Webseiten integriert. Dies ist sinnvoll, da sich wie erwähnt die Webseiten für einen Datensatz nicht verändern und so nur einmal zur Laufzeit der Infrastruktur dynamisch erstellt werden. Danach können sie schneller aus dem Cache bezogen werden, da keine erneuten Anfragen an die angebundene Datenbank erfolgen müssen. Neben der Auslieferung von Inhaltsseiten für veröffentlichte Datensätze, ermöglicht der Webserver auch, dass die vollständigen Datensätze oder einzelne Dateien bzw. Ordern davon über entsprechende Links auf der Inhaltsseite heruntergeladen werden können. Dabei können einzelne Verzeichnisse komplett verpackt und komprimiert als ZIP-Archiv geladen werden, wodurch sich die Größe des Downloads und damit die Zeit, die zum Herunterladen benötigt wird, verringert. Da jede in der eDAL-Infrastruktur erfasste Datei in ihrer ursprünglichen Form im angebundenen Dateisystem abgelegt wird (siehe Abschnitt 4.3.2),

können keine fertigen ZIP-Archive für alle Ordner gespeichert werden. Diese werden stattdessen ebenfalls dynamisch beim Download generiert. Da die eDAL-API zur Anbindung an ein Dateisystem Datenströme nutzt, können diese mit der nativen Java API direkt in einen `ZipOutputStream` (`java.util.zip.ZipOutputStream`) geleitet und beim Download übertragen werden.

In der `EdalRequestHandler` Klasse wurde außerdem ein weiterer Handler entwickelt und mit dem `EdalHttpServer` assoziiert. Dieser zeichnet alle Anfragen, die an den Webserver gestellt werden, auf und speichert sie in einer täglichen Logdatei ab. Darin enthalten sind beispielsweise Informationen wie die IP-Adresse einer Anfrage und die aufgerufene DOI. Diese Angaben sind zur Evaluierung nützlich und können etwa Aufschluss darüber geben welche Datensätze häufig angefragt werden und wie viele Nutzer Daten abgerufen haben. Zur Darstellung bietet der Webserver die `REPORT`-Funktion, welche die Daten aus den Logdateien mit dem Datenbestand abgleicht und die Zugriffsdaten in einer übersichtlichen Reportseite darstellt. Außerdem wird mithilfe der GeoIP2-API (siehe Abschnitt 2.3.6) anhand der IP-Adresse der jeweiligen Clientanfrage deren ungefähre geografische Herkunft ermittelt und auf einer Weltkarte dargestellt.

In Abbildung 5.7 ist ein Screenshot der dynamisch generierten Reportseite abgebildet. Neben der Nutzung zur Evaluierung, bietet diese auch für jeden Wissenschaftler eine interessante Möglichkeit, die Häufigkeit der Datenabrufe zu sehen. Dadurch erhöht sich sowohl die Akzeptanz der Infrastruktur als auch die Motivation, Forschungsdaten zu publizieren.

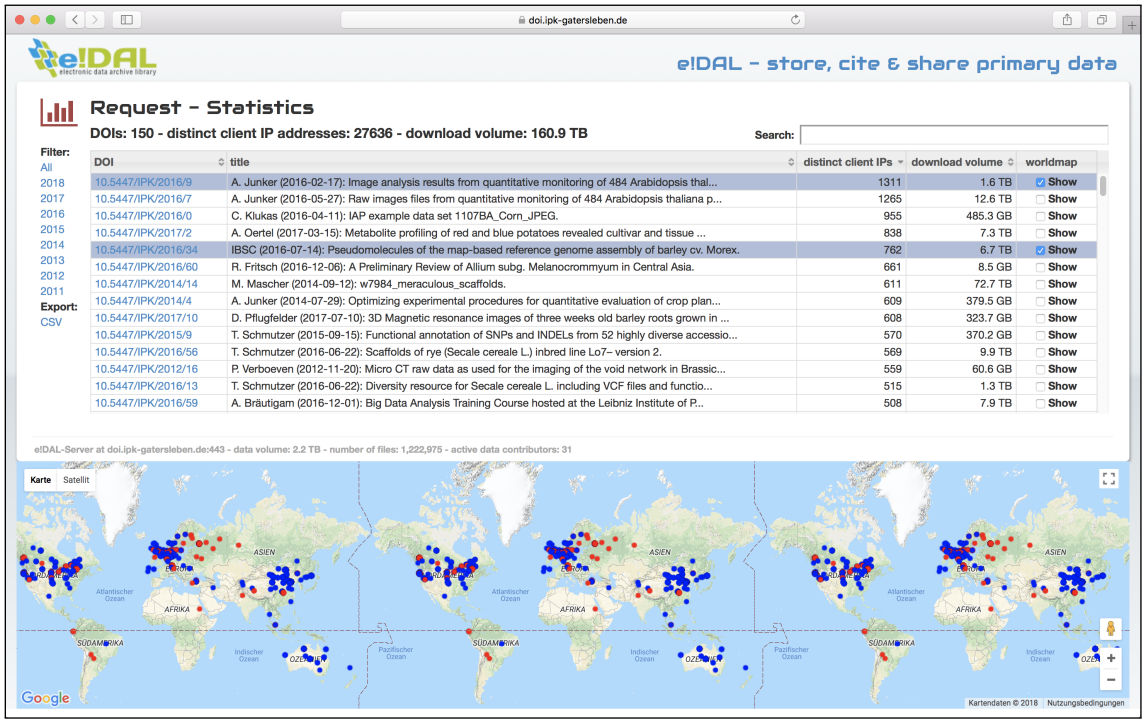

Abbildung 5.7: Integrierte Reportseite mit Zugriffszahlen und Downloadvolumen aller veröffentlichten Datensätze der eDAL-Infrastruktur. Auf der linken Seite können die Datensätze nach dem Jahr der Veröffentlichung gefiltert werden. Im unteren Bereich wird eine Karte mit der Herkunft der Zugriffe angezeigt.

Automatische Serverkonfiguration Wie in den vorangegangenen Abschnitten erläutert wurde, war für die Entwicklung einer Datenpublikationsinfrastruktur und eines integrierten Begutachtungsprozesses eine umfangreiche Erweiterung der eDAL-Infrastruktur notwendig. Dies hatte zur Folge, dass vor allem die Einrichtung eines eDAL-Servers, der als zentrale Komponente dient und Anfragen und Daten von verschiedenen Clients entgegen nimmt, komplexer wurde. Bisher wurde mit dem Start des Servers lediglich eine Instanz der eDAL-Infrastruktur auf Basis der Referenzimplementierung (siehe Abschnitt 4.3.2) initialisiert und über Remote-Schnittstellen (siehe Abschnitt 4.3.1) zur Verfügung gestellt. Daher war die Einrichtung eines eDAL-Servers relativ einfach. Die meisten Startparameter, wie beispielsweise die Pfade zum verknüpften Dateisystem und der eingebundenen Datenbank, waren nur optional und bereits mit Standardwerten belegt. Dieses Konzept der möglichst einfachen und nutzerfreundlichen Einrichtung war von Anfang an ein Hauptaugenmerk bei der Entwicklung der eDAL-Infrastruktur und sollte auch weiterhin bestand haben, um die Nutzbarkeit und die Einbindung in existierende Infrastrukturen zu erleichtern.

Aufgrund der Integration vieler neuer und umfangreicher Komponenten und Funktionen wie dem Webserver, der DataCite-REST-Schnittstelle und dem E-Mail-Versand sind auch eine Reihe von zusätzlichen Parametern zur Initialisierung der Infrastruktur notwendig. Es müssen Parameter für den Begutachtungsprozess, wie die E-Mail-Adressen der Gutachter, Parameter für den E-Mail-Versand, wie Adressen und Zugangsdaten für einen SMTP-Server, und Parameter für die Nutzung der DataCite-API, wie die Zugangsdaten des registrierten Datenzentrums, gesetzt werden. Dazu kommt, dass alle diese Komponenten eine Internetverbindung benötigen, sodass unter Umständen Netzwerk- oder Firewall-Parameter, wie Adressen für Proxy-Server oder freie Ports für den Webserver, benötigt werden. Um die Nutzung weiterhin so einfach wie möglich zu halten, wurde eine umfangreiche automatische Konfigurationsroutine implementiert, die beim Start eines eDAL-Servers bereits eine Vielzahl von Parametern selbstständig ermittelt. So wurde etwa eine Funktion programmiert, die plattformübergreifend die Netzwerk-Konfiguration des Betriebssystems überprüft und nach möglicherweise konfigurierten Proxy-Servern sucht. Anschließend testet das System, ob eine Verbindung zu den DataCite-Schnittstellen aufgebaut werden kann bzw. ob die Authentifizierung gültig ist, damit die DOI-Vergabe funktioniert. Außerdem wird geprüft, ob die Standardports zur Bindung des Webservers verfügbar sind. Eine ähnliche Funktion übernimmt auch die Konfiguration des SMTP-Dienstes zum Versand von E-Mail-Benachrichtigungen für den Begutachtungsprozess. Sie prüft, ob in der Netzwerkdomäne ein SMTP-Server definiert ist und testet, ob dieser zugänglich ist und zum Versand genutzt werden kann. Dies vereinfacht die Initialisierung eines eDAL-Servers erheblich, sodass bei entsprechend konfigurierten Netzwerkeinstellungen lediglich die E-Mail-Adressen der Gutachter und die DataCite-Zugangsdaten des Datenzentrums als Parameter über den Konstruktor gesetzt werden müssen. Falls eine der genannten Funktionen nicht in der Lage ist, die benötigten Einstellungen automatisch zu ermitteln, können diese manuell über einen erweiterten Konstruktor oder die entsprechenden Getter-Methoden bzw. Kommandozeilenparameter beim Starten eines eDAL-Servers gesetzt werden.

Die automatische Konfiguration der Parameter zur Initialisierung der Infrastruktur erleichtert damit nicht nur die Nutzbarkeit und Etablierung der Infrastruktur, sondern gewährleistet gleichzeitig durch das Testen aller Konfigurationen die korrekte Funktionsweise der bereitgestellten Funktionen und damit der gesamten Infrastruktur.

5.2 Anwendung zur Einreichung und Publikation von Forschungsdaten

Mit der Implementierung des Datenbegutachtungsprozesses als Erweiterung der eDAL-API wurde die Grundlage für eine Datenpublikationsinfrastruktur geschaffen. eDAL kann als Java-API in existierende Anwendungen integriert werden, um den Datenbegutachtungs- und Publikationsprozess programmatisch zu nutzen. Für eine benutzerfreundliche und produktive Instanz einer Datenpublikationsinfrastruktur war jedoch eine grafische Anwendung notwendig, um den Datenproduzenten eine einfache Möglichkeit zu geben Datensätze zu archivieren und eine Publikationsanfrage zur Vergabe einer DOI zu starten. Die Implementierung der Anwendung wurde mithilfe der nativen Java Swing-API (`javax.swing.*`) realisiert. Diese war zum Zeitpunkt der Implementierung die Standard-API für grafische Oberflächen. Da es sich um eine Clientanwendung handelt, wurde die Implementierung in die Clientkomponente der eDAL-Infrastruktur integriert. Das bedeutet, dass zum Starten der Anwendung Adresse und Port eines eDAL-Servers, auf dem eine Instanz der eDAL-Infrastruktur läuft, angegeben werden müssen. Die Anwendung stellt dann die Verbindung zum Server her und startet die grafische Oberfläche. Das Design und die Implementierung der genannten Anwendung werden in den folgenden Abschnitten beschrieben.

Die in diesen Abschnitten gezeigten Grafiken sind alle aus der ersten konkreten Implementierung der Datenpublikationsanwendung für ein Repository auf Basis der eDAL-Infrastruktur am IPK Gatersleben entnommen. So enthält die Anwendung unter anderem einige spezifische Texte und Bilder. Die Implementierung der Anwendung ist jedoch wie die eDAL-API generisch aufgebaut, sodass das äußere Erscheinungsbild und spezielle Komponenten der grafischen Oberfläche über definierte Konfigurationsdateien und Templates manipuliert werden können. So kann die Anwendung leicht für weitere Instanzen, die z. B. von Partnerinstituten betrieben werden, angepasst werden.

5.2.1 Design

Das Ziel der Entwicklung war es, eine intuitive und selbsterklärende Anwendung bereitzustellen, die der Anwender ohne zusätzliche Installation nutzen kann und die ihn bei der Datenpublikation unterstützt, indem so viele Schritte wie möglich automatisiert erfolgen, um Fehler zu vermeiden und den Prozess zu beschleunigen. Dabei sollte das Erscheinungsbild an ein übliches Formular bzw. Anwendung zur Einreichung und Übermittlung eines wissenschaftlichen Artikels, z. B. für ein Fachjournal, angelehnt sein, um so den Prozess für den Wissenschaftler zu vereinfachen und die Akzeptanz des Systems zu gewährleisten. Die Anmeldung erfolgt über einen üblichen Anmeldedialog mit Nutzernamen- und Passwortabfrage unter Verwendung der von der eDAL-Infrastruktur bereitgestellten Loginmodule. Die Benutzeroberfläche ermöglicht es, alle Metadaten, die zur Beschreibung und Generierung einer DOI erforderlich sind, einzugeben. Über einen Dateiauswahldialog können die zum Datensatz gehörigen Dateien oder Ordner ausgewählt werden. Nachdem alle notwendigen Felder ausgefüllt wurden, kann der Nutzer den Dateiupload auf den eDAL-Server starten und über eine Fortschrittsanzeige verfolgen. Nach erfolgreichem Abschließen des Prozesses wird die Datenbegutachtung initiiert und der Datenerzeuger wird über die weiteren Schritte, wie in den vorangegangenen Abschnitten beschrieben, per E-Mail informiert.

Anmeldung

Für die Anmeldung eines Nutzers werden die von der eDAL-API implementierten Loginmodule genutzt und in einen grafischen Dialog eingebunden. Neben der Identifizierung des Nutzers ist es notwendig, eine gültige E-Mail-Adresse zu erhalten, da diese für die Kommunikation während des Begutachtungsprozesses benötigt wird. Sie kann z. B. bei der Verwendung des Kerberos-Loginmoduls (siehe Abbildung 5.8) aus den Nutzerinformationen, die in den Principals des authentifizierten Subjects (siehe Abschnitt 2.3.3) enthalten sind, extrahiert werden. Voraussetzung dafür ist jedoch, dass der dazugehörige Authentifizierungsdienst entsprechend konfiguriert ist und diese Information hinterlegt. Um eine universelle Anmeldeprozedur bereitzustellen, können zusätzliche Loginmodule in der eDAL-API implementiert werden. Dabei ist zu beachten, dass die implementierten Verfahren aus Sicherheitsgründen eine geprüfte E-Mail-Adresse zurückliefern. Nur so kann gewährleistet werden, dass der angemeldete Nutzer die Nachrichten, die im Laufe der Begutachtung erstellt werden, zuverlässig erhält. Eine manuelle Eingabe der E-Mail-Adresse durch den Nutzer würde einen zusätzlichen Double-Opt-In Prozesse erfordern, um sicherzustellen, dass keine fremde E-Mail-Adresse genutzt wird.

Abbildung 5.8: Grafische Oberfläche des Anmeldedialog des IPK-Kerberos-Loginmoduls für die Anwendung zur Einreichung und Publikation von Forschungsdaten

Datenbeschreibung

Nach erfolgreicher Anmeldung öffnet sich die eigentliche Benutzeroberfläche, die in verschiedene Felder entsprechend den Attributen zur Beschreibung des Datensatzes, unterteilt ist (siehe Abbildung 5.9 auf Seite 85). Bevor der Nutzer erstmalig einen Datensatz hochladen kann, ist es notwendig, dass er vorher die daten- und lizenzrechtlichen Bestimmungen liest. Diese sind als Verweis direkt in der Kopfzeile der Anwendung oder als PDF-Dokument abrufbar. Erst danach wird die komplette Oberfläche freigeschaltet. Neben Freitextfeldern, z. B. für den Titel und die Beschreibung des Datensatzes, sind einige Felder mit speziell auf die Datentypen der Attribute zugeschnittenen Tabellen oder Auswahldialogen versehen. Diese werden bei der Auswahl eines Feldes eingeblendet. So enthält z. B. das Autorenfeld eine Tabelle in der die entsprechenden Angaben zu den assoziierten Autoren, wie Name, Funktion, Adresse oder ORCID, eingegeben werden kann. Dabei unterstützt ein integrierter Suchdialog (siehe Abbildung 5.10 auf Seite 85) den Anwender bei der Eingabe der ORCIDs der Autoren. Über eine Abfrage an die REST-API der ORCID-Infrastruktur prüft die Anwendung, ob eine gegebene ORCID zum jeweiligen Autor gehört, bzw. schlägt bei Eingabe eines Namens die gefundenen Einträge aus der ORCID-Datenbank vor. So ist die Eingabe für den Nutzer komfortabel und gleichzeitig wird verhindert, dass versehentlich falsche Angaben eingetragen werden. Das Feld zum Festlegen eines optionalen Zeitpunktes für ein Embargo, bis zu welchem ein Datensatz trotz eventuell bereits vergebener DOI vom System nicht bereitgestellt werden soll, enthält einen Kalender-Dialog zur Auswahl eines gewünschten

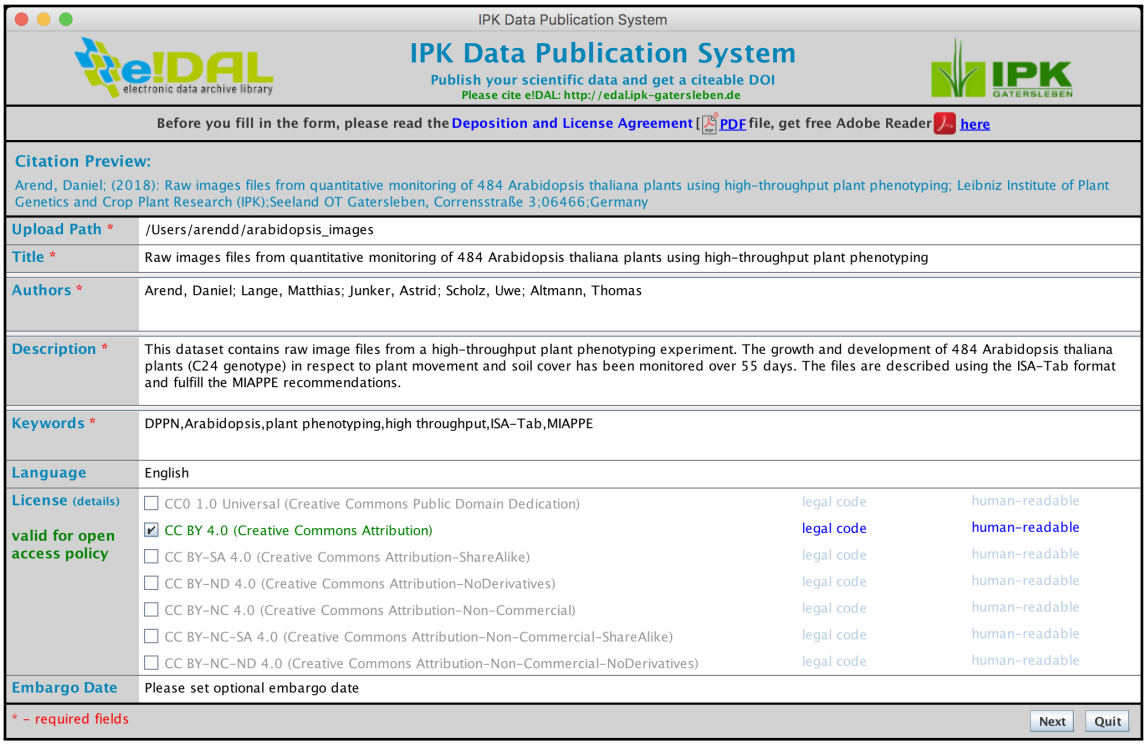

Abbildung 5.9: Formularbasierte Benutzeroberfläche der Anwendung zur Einreichung und Publikation von Forschungsdaten

Zeitpunktes. Im obersten Feld wird der Datensatz, der publiziert werden soll, über einen Dateiauswahldialog selektiert. Zur Vergabe einer Lizenz unterstützt die Anwendung die aktuellen Lizenzmodule der Creative Commons Organisation (Creative Commons, 2018), da diese weitverbreitet sind und sich auch für Forschungsdaten eignen.

Wenn der Anwender alle notwendigen Informationen ausgefüllt hat, kann er seine Angaben bestätigen und mit dem Upload der Daten beginnen. Dabei prüft die Anwendung, ob alle Werte gesetzt wurden, und weist den Nutzer auf eventuelle Fehler hin, indem leere Felder rot markiert werden. Außerdem wird kontrolliert, ob der ausgewählte Datensatz versehentlich leere Dateien oder symbolische Links enthält, und informiert den Nutzer darüber. Zusätzlich verhindert die Anwendung, dass komprimierte Archive hochgeladen werden, da dies später verhindern würde, dass der Datensatz vollständig und direkt auf der Inhaltsseite der dazugehörigen DOI angesehen und durchsucht werden kann. Außerdem bieten die späteren Inhaltsseiten, wie in Abschnitt 5.1.2 beschrieben, bereits die Möglichkeit, einen kompletten Datensatz oder einzelne Ordner direkt als ZIP-Archiv herunterzuladen.

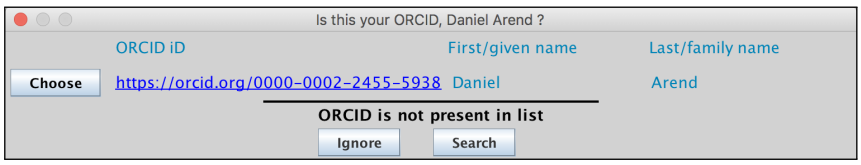

Abbildung 5.10: Grafische Oberfläche des Dialogs zur Prüfung und Bestätigung einer ORCID für die Anwendung zur Einreichung und Publikation von Forschungsdaten

Upload

Der Upload-Dialog, der sich nach erfolgreicher Prüfung der Metadaten öffnet, fasst die wichtigsten Informationen des Datensatzes für den Anwender noch mal zusammen (siehe Abbildung 5.11). Außerdem muss der Nutzer bestätigen, dass er die bereits gelesenen rechtlichen Bestimmungen akzeptiert. Anschließend kann mit dem Upload der Daten begonnen werden.

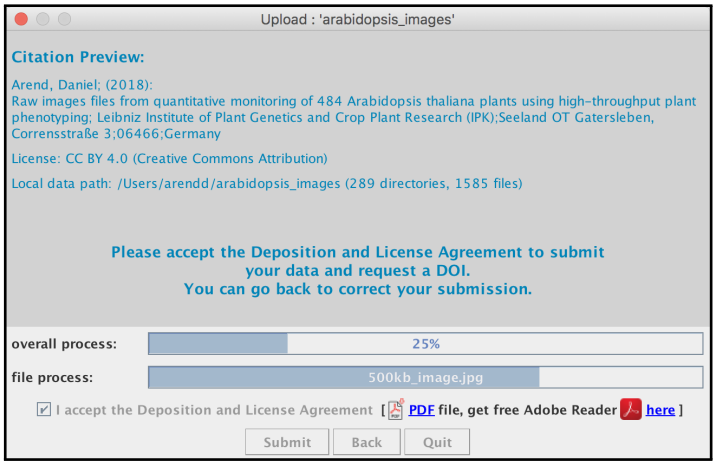

Abbildung 5.11: Grafische Oberfläche des Upload-Dialog der Anwendung zur Einreichung und Publikation von Forschungsdaten

Neben den Metadaten, die vom Nutzer geliefert werden, ermittelt die Anwendung beim Upload auf den eDAL-Server ebenfalls weitere technische Metadaten, wie die Dateitypen, Prüfsummen und die Größen aller Dateien in einem Datensatz und trägt diese Werte ebenfalls in die dazugehörigen Metadaten der Objekte in der eDAL-Infrastruktur ein. Während des Uploadvorgangs wird der Nutzer über den Fortschritt des gesamten Vorgangs bzw. der jeweils aktuellen Datei informiert. Nach Abschluss des Prozesses kann die Anwendung beendet oder ein weiterer Datensatz hochgeladen werden. Da die Publikationsanwendung keine Installation benötigt, werden alle Metadaten mit Ausnahme des Titels und der Beschreibung von der Anwendung nach einem erfolgreichen Upload in einer lokalen Datei im Nutzerverzeichnis gespeichert. So bleiben die Eingaben des Nutzers nach dem Beenden der Anwendung erhalten und können bei einem erneuten Start wieder in die entsprechenden Felder geladen werden. Die Einreichung weiterer Datensätze wird dadurch beschleunigt, da bereits verwendete Angabe wie z. B. die Liste der Autoren oder Schlagwörter, bei Bedarf wiederverwendet werden können.

5.2.2 Implementierung

Wie bereits eingangs beschrieben, wurde die Publikationsanwendung größtenteils mit der nativen Java Swing-API entwickelt. Aufgrund des Umfangs wird an dieser Stelle auf eine vollständige Beschreibung aller implementierten Klassen und Funktionen verzichtet. Stattdessen wird im Folgenden auf einige Ausnahmen und spezielle Implementierungstechniken genauer eingegangen.

Anmeldung

Da die erste produktive Instanz der eDAL-Infrastruktur am IPK Gatersleben etabliert wurde, worauf später noch genauer eingegangen wird, war zu Beginn nur das Loginmodul auf Basis des institutionellen Kerberos-Dienstes verfügbar. Dadurch war die Einreichung und Beschreibung von Forschungsdaten zur Publikation nur für Wissenschaftler am IPK Gatersleben möglich. Um die Einreichung auch für externe Wissenschaftler z. B. von Partnerinstituten zu öffnen bzw. die Etablierung weiterer Instanzen zu vereinfachen, wurden weitere universelle JAAS-Loginmodule auf Basis des OAuth-Protokolls (siehe Abschnitt 2.3.3) realisiert. So wurden Loginmodule zur Authentifizierung über die Google- und ORCID-API sowie über die ELIXIR- Authentication and Authorization Infrastructure (AAI) implementiert. Dabei vereint das letztere Loginmodul Anmeldeverfahren für alle mit ELIXIR assoziierten Organisationen (ELIXIR, 2018) wie z. B. für die Institute im Rahmen des DPPN-Projektes. Da OAuth eine webbasierte Authentifizierung ist und über Weiterleitungen im Webbrowser funktioniert, musste eine Möglichkeit gefunden werden, um HTTP-Seiten innerhalb einer Java Anwendung anzuzeigen und HTTP-Anfragen zu verarbeiten.

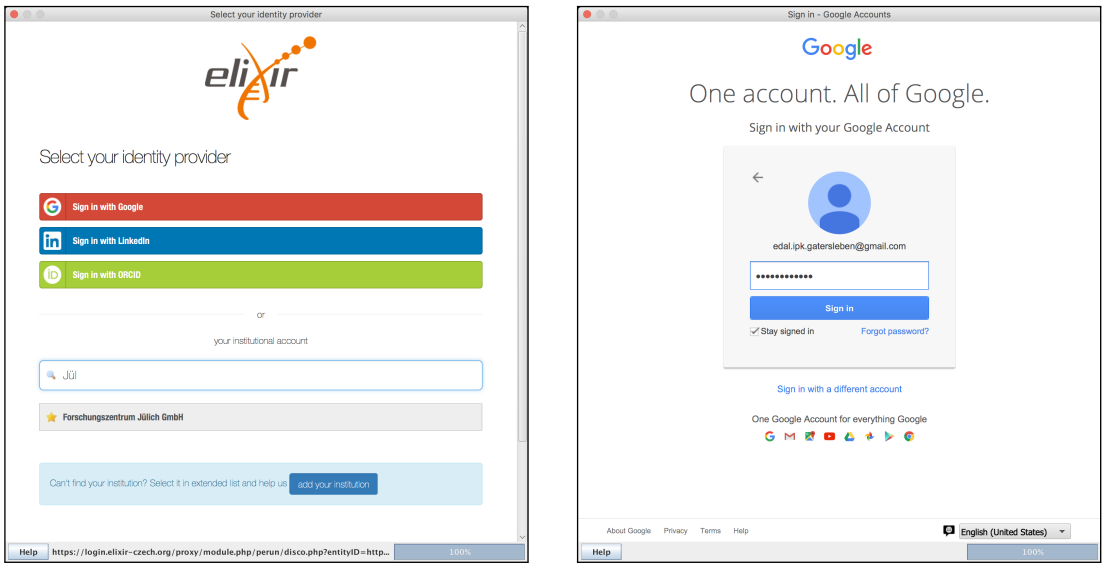

Abbildung 5.12: JavaFX - Anmeldedialog der integrierten ELIXIR- und Google-Loginmodule der Anwendung zur Einreichung und Publikation von Forschungsdaten

Die native Swing-API ist zwar in der Lage, HTML-Dokumente darzustellen, bietet jedoch keine Möglichkeit zur Abbildung eines Webbrowsers und zum Umgang mit Nutzerinteraktion im selbigen. Diese Funktionalität kann nur über umfangreiche, zusätzliche Bibliotheken bereitgestellt werden. Mit der neueren nativen JavaFX-API, die in Zukunft die Swing-API als Standard für die Entwicklung von grafischen Anwendungen ablösen soll, bietet Java jedoch bereits eine eigene weiterentwickelte API, die mit der Webengine Klasse (`javafx.scene.web.WebEngine`) einen integrierten Webbrowser zu Einbindung in grafische Oberflächen liefert (siehe Abbildung 5.12). Diese kann Webseiten anzeigen und auch von diesen Seiten ausgehende Anfragen und Weiterleitungen verarbeiten. Daher wurde für die Implementierung der OAuth-basierten Authentifizierungsverfahren die JavaFX-API verwendet, um die Interaktion zwischen dem Nutzer mit den über OAuth bereitgestellten Webseiten zu steuern und die angefragten Informationen während der Anmeldeprozedur

im Subject des Nutzers abzulegen. Die für OAuth notwendigen HTTP-Anfragen werden intern mithilfe der Jersey-API, die schon für die REST-Kommunikation mit der DataCite-API (siehe Abschnitt 5.1.2) genutzt wurde, verarbeitet. Auf diese Art wurde ein spezieller CallbackHandler implementiert, der die Ausführung einer OAuth-Authentifizierung (siehe Abschnitt 2.3.3) ermöglicht und auch dynamisch für weitere OAuth-basierte Anmeldeverfahren genutzt werden kann.

Layout und Oberfläche

Wie bereits erwähnt, sollte das äußere Erscheinungsbild der Publikationsanwendung dynamisch anpassbar sein, sodass es für die Etablierung weiterer Instanzen wiederverwendet werden kann. Deshalb wurden für allgemeine Parameter, die das optische Erscheinungsbild bestimmen, wie beispielsweise Fenstergrößen, Schriftarten oder Farben, eine Reihe von Konstanten sowie eine Konfigurationsdatei für Variablen zur Bezeichnung von bestimmten Feldern definiert. So können diese Parameter einfach und dynamisch angepasst werden. Für komplexere Komponenten, wie Inhalte von bestimmten Fenstern, Dialogen oder Menüs wurde ein anderer Weg gegangen. Diese werden dynamisch über Templates mit der Velocity-API erzeugt, da sie sich bereits bei der Implementierung des Datenbegutachtungsprozesses bewährt hat (siehe Abschnitt 5.1.2).

Quelltext 5.4 zeigt beispielsweise ein Template, mit welchem die Kopfzeile der Benutzeroberfläche in der Publikationsanwendung (siehe Abbildung 5.9 auf Seite 85) generiert wird. Durch diese Implementierung ist einerseits der eigentliche Programmquellcode und die objektorientierte Logik von den inhaltlichen und layoutspezifischen Komponenten strikt getrennt, was die Übersichtlichkeit verbessert und die Wartung der Anwendung vereinfacht. Andererseits kann das Erscheinungsbild der Benutzeroberfläche leicht angepasst werden.

```
<html>
   <body bgcolor=rgb($bgcolor.getRed(),$bgcolor.getGreen(),$bgcolor.getBlue())
    style="color:rgb($fgcolor.getRed(),$fgcolor.getGreen(),$fgcolor.getBlue())">
      <table>
        <tr>
           <th><a href="http://edal.ipk-gatersleben.de"><img src="$edal"/></a></th>
           <th>
              <h1>IPK Data Publication System</h1>

              <h4>Publish your scientific data and get a citeable DOI
               <a href="http://edal.ipk-gatersleben.de">
               Please cite eDAL:http://edal.ipk-gatersleben.de</a></h4>
           </th>
           <th><a href="http://www.ipk-gatersleben.de"><img src="$ipk"/></a></th>
        </tr>
      </table>
   </body>
</html>
```

Quelltext 5.4: VeloCity-Template für die Kopfzeile der Benutzeroberfläche der Anwendung zur Einreichung und Beschreibung von Forschungsdaten. Die fett hervorgehobenen Werte sind die individuellen Variablen, die beim Erzeugen der grafischen Oberfläche dynamisch gesetzt werden.

Installation und Ausführung

Die Implementierung der Anwendung zur Einreichung und Publikation von Forschungsdaten ist ein Bestandteil der Clientkomponente der eDAL-Infrastruktur. Da diese bereits alle notwendigen Bibliotheken, Konfigurationsdateien und andere Komponenten mitliefert und keine zusätzlichen Abhängigkeiten existieren, kann das Programm direkt als Java-basierte Anwendung ausgeführt werden und benötigt keine separate Installationsroutine. Dies vereinfacht die Weiterentwicklung und Veröffentlichung neuer Programmversionen und ist außerdem für den Anwender sehr komfortabel. Um die Nutzung noch weiter zu vereinfachen und auch Forschern ohne Java Kenntnisse die Ausführung der Anwendung zum Veröffentlichen ihrer Daten zu ermöglichen, wird das Programm als Java-Webstart zur Verfügung gestellt. So muss der Clientrechner, auf dem der Nutzer die Publikationsanwendung ausführt, nur eine Java-Laufzeitumgebung installiert haben. Der Benutzer kann die Anwendung so plattformunabhängig mit nur einem Klick starten. Zum Ausführen einer Java-Webstart Anwendung wird eine Java Network Launching Protocol (JNLP)-Datei definiert, die den Speicherort der eigentlichen Anwendung definiert und festlegt, mit welchen Parametern diese gestartet werden soll. In Quelltext 5.5 ist ein Auszug aus der JNLP-Datei für die Publikationsanwendung dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase="http://edal.ipk-gatersleben.de/">
   <information>
       <title>eDAL-Publication-Module</title>
       <vendor>IPK Gatersleben</vendor>
       <description>eDAL-Publication-Module</description>
   </information>
   <resources>
       <j2se version="1.8+"/>
       <jar href="eDAL-Project-$version-webstart.jar" download="eager"/>
   </resources>
   <application-desc main-class="...publication.IPKPublicationModule"/>
</jnlp>
```

Quelltext 5.5: Auszug aus der JNLP-Datei der Java-Webstart Applikation für die Anwendung zur Einreichung und Publikation von Forschungsdaten.

Die JNLP-Datei wird auf der Projektwebseite zur Verfügung gestellt und ist nur wenige Byte groß. Sie kann direkt aus dem Browser heraus ausgeführt werden. Bevor die eigentliche Anwendung ausgeführt wird, wird geprüft, ob die notwendige Java-Laufzeitumgebung vorhanden ist. Die Anwendung lädt dann automatisch bei Bedarf die aktuellste Version der Publikationsanwendung herunter und startet diese. Ist die aktuellste Version bereits im internen Java Webstart-Cache vorhanden, wird diese direkt gestartet. Der Nutzer wird über die Gültigkeit des Anwendungszertifikates informiert (siehe Abbildung 5.13 auf Seite 90) und kann nach der Bestätigung sofort mit der Anmeldung und der Beschreibung seiner Daten beginnen.

5.3 Nachhaltige Softwareentwicklung

Viele Institutionen sind oft mit der Herausforderung konfrontiert, dass Software und Informationssysteme, die im Laufe eines Projektes entwickelt wurden, sich nach Ablauf

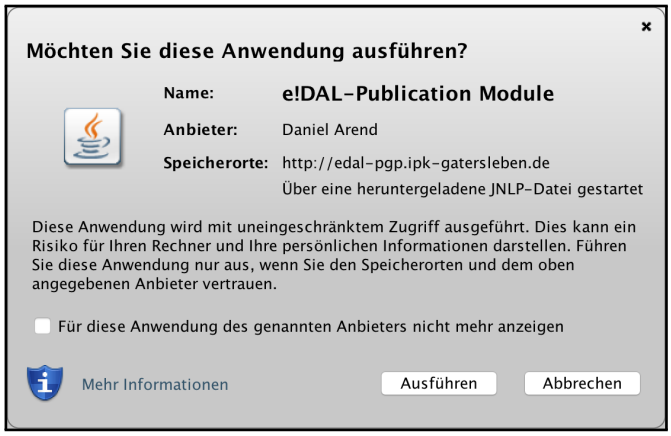

Abbildung 5.13: Bestätigungsdialog des Java-Webstart Anwendungszertifikates zum Ausführen der Anwendung zur Einreichung und Publikation von Forschungsdaten

des Finanzierungszeitraums nur noch schwierig warten lassen (vgl. Khondhu et al., 2013). Der Grund dafür liegt oft in unzureichender Softwarebeschreibung und Quellcodedokumentation, da die meisten Entwickler selbst dafür verantwortlich sind, dass ihre Software möglichst nachhaltig ist. Jedoch können dies aufgrund fehlender Kenntnisse und Zeitdruck durch limitierte Arbeitsverträge oder kurze Projektlaufzeiten nur wenige gewährleisten (vgl. Ahmed et al., 2014; Adams et al., 2014). Wenn die ursprünglichen Entwickler nicht mehr zu Verfügung stehen, muss der Support meistens eingestellt werden und viele Systeme können nicht weiter gepflegt werden.

Um dies zu verhindern, wurde während der Entwicklung der eDAL-Infrastruktur bzw. des integrierten Datenbegutachtungsprozesses sowie der Anwendung zur Einreichung und Publikation von Forschungsdaten darauf geachtet, dass neben der Realisierung der beschriebenen Anforderungen und Funktionen auch nachhaltige Standards und Empfehlungen im Bereich der Softwareentwicklung (vgl. Leprevost et al., 2014; Goodman et al., 2014) eingehalten werden. Alle entwickelten Komponenten sind *open source* und damit frei verfügbar. Außerdem wurden für die Umsetzung der Implementierung ebenfalls nur freie Bibliotheken und Frameworks genutzt, da dadurch eine transparente Entwicklung gewährleistet ist (vgl. Prlić und Procter, 2012). So wird der Nutzen für die wissenschaftliche Gemeinschaft durch die bessere Nachhaltigkeit und Reproduzierbarkeit der entwickelten Software erhöht. Gleichzeitig wird durch zusätzliches Feedback die Weiterentwicklung gefördert (vgl. Vandewalle, 2012). Im folgenden Abschnitt werden einige wichtige Aspekte und Vorgehensweise der Implementierung beleuchtet, die eine nachhaltige Pflege und Weiterentwicklung der eDAL-Infrastruktur sicherstellen.

5.3.1 Entwicklungsumgebung und Source-Code Management

Aufgrund der umfangreichen Funktionalitäten und Komplexität der eDAL-Infrastruktur ist der Umfang des Quellcodes entsprechend groß. Daher bietet sich die Verwendung einer integrierten Entwicklungsumgebung (englisch: Integrated Development Environment (IDE)) an, da diese eine Reihe von Werkzeugen und Funktionen bereitstellt, welche die Programmierer bei der Entwicklung unterstützen. Eine der Bekanntesten und auch im

Rahmen des Dissertationsprojekts verwendete IDE ist Eclipse (Eclipse Foundation, 2018). Diese wurde ursprünglich für Java-basierte Anwendungen entworfen und liefert eine Vielzahl an hilfreichen Unterstützungen und Funktionen wie Verwaltung von Bibliotheken oder automatische Quellcodedokumentation. Außerdem ist es flexibel erweiterbar und kann mittels zusätzlicher Plug-ins neue Funktionen integrieren.

Während der gesamten Entwicklungszeit entsteht eine Vielzahl an Zwischenschritten und Versionen eines Quellcodes. Unter Umständen arbeiten auch mehrere Entwickler parallel am gleichen Projekt. Daher ist ein System zur Verwaltung und Versionierung von Quelltexten nahezu unabdingbar, da so die komplette Historie der Entwicklung nachhaltig erfasst werden kann (vgl. Leprevost et al., 2014) und sie eine kollaborative Entwicklung und Prüfung ermöglichen. In Abschnitt 3.3 wurden bereits zwei populäre Versionsverwaltungssysteme im Zusammenhang mit der Untersuchung existierender Systeme zur Verwaltung von Forschungsdaten erläutert. Anders als für allgemeine Forschungsdaten eigenen sie sehr gut für die Verwaltung von Quelltexten, für die sie auch ursprünglich entwickelt wurden. Der Quelltext der eDAL-API, Referenzimplementierung, Serverkomponenten sowie Clientanwendungen werden über öffentliche GIT-Repository (Loeliger und McCullough, 2012) auf der Bitbucket Plattform (Atlassian, 2018) verwaltet:

- `https://bitbucket.org/ipk_bit_team/electronicdataarchivelibrary`

Damit sind der komplette Quellcode und die Entwicklung der eDAL-Infrastruktur frei zugänglich. Außerdem bietet Bitbucket bereits ein integriertes Ticketsystem. Darüber können Fehler gemeldet und Anfragen für neue Funktionen erstellt werden.

5.3.2 Erstellungsprozess

Einer der größten Vorteile der Java-Programmiersprache ist die große Flexibilität und die vielfältigen Erweiterungsmöglichkeiten. Über eingebundene externe APIs und Bibliotheken kann die Funktionalität umfangreich erweitert werden. Mit zunehmender Größe eines Projektes kann jedoch genau dieser Vorteil eine zusätzliche Herausforderung werden. Zum Einbinden zusätzlicher Abhängigkeiten müssen diese alle einzeln in den Klassenpfad der eigentlichen Anwendung geladen werden. Werden genutzte Bibliotheken aktualisiert, müssen diese ausgetauscht werden. Außerdem kann es passieren, dass bestimmte APIs wiederum andere Bibliotheken verwenden. Dies kann beispielsweise bei Überschneidungen von gleichen Bibliotheken mit unterschiedlicher Version zu erheblichen Problemen beim Kompilieren führen. Der Entwickler muss daher beim Erstellen seiner Anwendung darauf achten, dass alle notwendigen Bibliotheken eingebunden sind und keine Versionskonflikte vorliegen, die eventuell zu Fehlern beim Übersetzen oder Ausführen der Anwendung führen können.

Um diesen Aufwand zu verringern und Fehler zu vermeiden, wurde der Erstellungsprozess der eDAL-Infrastruktur mithilfe von Apache Maven (Apache Software Foundation, 2018a) automatisiert. Dabei handelt es sich um ein umfangreiches und etabliertes Entwicklungsmanagementprogramm zur Automatisierung des Erstellungsprozesses von Java Anwendungen. Es ist in der Lage benötigte Bibliotheken und APIs dynamisch in ein Projekt einzubinden und stellt dabei sicher, dass mögliche Versionskonflikte frühzeitig entdeckt werden. Außerdem bindet Maven alle benötigten Abhängigkeiten automatisch in den Klassenpfad der Anwendung ein und lädt die notwendigen Bibliotheken automatisch aus zentralen öffentlichen oder privaten Archiven und speichert diese lokal zwischen. Beim Durchlaufen

des Erstellungsprozesses werden dann alle benötigten Klassen geladen und in die finale Anwendung eingebunden. Dabei werden im sogenannten *Build Lifecycle* verschiedene Phasen bei der Erstellung durchlaufen:

- **validate** : prüft, ob alle Information für den Erstellungsprozess vorhanden sind
- **compile** : kompiliert den Quelltext des Projektes
- **test** : testet den kompilierten Quelltext mittels enthaltener Test-Klassen
- **package** : packt alle kompilierten Klassen in ein *-jar Paket
- **verify** : prüft, ob das erzeuge Paket vollständig ist
- **install** : installiert das Paket im lokal Repository zur Weiternutzung
- **deploy** : kopiert das finale Paket in ein zentrales Repository

In einer zentralen Konfigurationsdatei „pom.xml“ werden alle Einstellungen des jeweiligen Projektes definiert. Das beinhaltet z. B. Pfadangaben zu notwendigen Ressourcen oder alle für das Projekt notwendigen Bibliotheken und aus welchen Repository diese geladen werden sollen. Im Quelltext 5.6 ist eine stark gekürzte Version der Konfigurationsdatei für die eDAL-API abgebildet. Darüber hinaus gibt es eine Vielzahl an zusätzlichen Plug-ins, die genutzt werden können, um in einer bestimmten Phase des Erstellungsprozesses eine definierte Aufgabe zu erledigen. So gibt es z. B. Plug-ins zum Entfernen von privaten oder sensiblen Informationen, die zwar zur Entwicklung des Projektes notwendig sind, aber später nicht im finalen Anwendungspaket enthalten sein sollen.

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
   <artifactId>eDAL-MetaDataAPI</artifactId>
   <version>2.5.0</version>
   <packaging>jar</packaging>
   <url>http://edal.ipk-gatersleben.de</url>
      <build>
              <plugin>
                  <groupId>org.apache.maven.plugins</groupId>
                  <artifactId>maven-compiler-plugin</artifactId>
                  <version>3.1</version>
                  <configuration>
                      <source>1.8</source>
                  </configuration>
              </plugin>
      </build>
      <dependencies>
          <dependency>
              <groupId>junit</groupId>
              <artifactId>junit</artifactId>
              <version>4.10</version>
          </dependency>
      </dependencies>
</project>
```

Quelltext 5.6: Auszug aus einer pom.xml Konfigurationsdatei für den Maven-Erstellungsprozess der eDAL-Infrastruktur-API

Auf diese Weise konnten die Vielzahl an externen Bibliotheken, die für die Implementierung der eDAL-Infrastruktur, des Datenbegutachtungsprozesses und der Publikationsanwendung verwendet wurden zentral und übersichtlich definiert werden. So wurden Versionskonflikte vermieden. Wenn bestimmte Bibliotheken aktualisiert werden müssen, kann dies einfach

durch Anpassung der entsprechenden Einträge in der Maven-Konfigurationsdatei erfolgen. Ein häufig erwähnter Nachteil vieler open source Projekte ist, dass sie nicht immer nachhaltig sind und bei einer fehlenden, dauerhaften Finanzierung nicht alleine durch die wissenschaftliche Gemeinschaft zuverlässig gepflegt werden können (vgl. Khondhu et al., 2013). Daher wurde während der gesamten Entwicklung darauf geachtet, dass alle verwendeten Bibliotheken nicht nur etabliert und möglichst zuverlässig weiterentwickelt werden, sondern auch im zentralen Maven-Repository (Sonatype Inc., 2018a) erfasst sind, um die zuverlässige und langfristige Einbindung in das Projekt sicherzustellen. Da eine einfache Integration in existierende Infrastrukturen und Anwendungen eine zentrale Anforderung an die Datenhaltungs- und Publikationsinfrastruktur ist, wurden alle Komponenten der eDAL-API ebenfalls im zentralen Maven-Repository (Sonatype Inc., 2018b) veröffentlicht und können so einfach in andere Projekte als Abhängigkeiten integriert und genutzt werden (siehe Quelltext 5.7).

```
<project>
...
  <dependencies>
  ...
     <dependency>
      <groupId>de.ipk-gatersleben</groupId>
      <artifactId>eDAL-MetaDataAPI | -Server | -Client</artifactId>
      <version>2.5.5</version>
     </dependency>
   ...
  <dependencies>
...
</project>
```

Quelltext 5.7: Auszug aus einer pom.xml Konfigurationsdatei mit den Abhängigkeiten der eDAL-Infrastruktur zur Einbindung in ein existierendes Projekt

Gegen Ende des Dissertationsprojektes und im Zuge des mittlerweile sehr großen Umfangs der genutzten externen Bibliotheken, des eigentlichen Quellcodes und der Testklassen, auf die im nächsten Abschnitt noch genauer eingegangen wird, zeigt sich jedoch eine verschlechterte Performance beim Durchlaufen des Erstellungsprozesses. Da Maven alle Schritte des Erstellungsprozesses sequenziell abarbeitet, dauert ein vollständiger Prozess vom Einbinden der Abhängigkeiten, Kompilieren des Quellcodes, Ausführen der Softwaretests und Veröffentlichen der Anwendung mehrere Minuten. Deshalb wurde gegen Ende der Dissertation mit Gradle (Gradle Inc., 2018) ein weiteres, neueres Entwicklungsmanagement-Framework integriert. Es funktioniert ähnlich wie Maven und wird ebenfalls über eine zentrale Konfigurationsdatei gesteuert. Außerdem unterstützt es ebenfalls die Einbindung externer Abhängigkeiten über die gleichen Archive wie Maven, wodurch die Umstellung auf den neuen Entwicklungsmechanismus sehr einfach war. Der große Vorteil von Gradle ist die deutlich bessere Performance, die unter anderem dadurch zustande kommt, dass Gradle in der Lage ist, mehrere Softwaretests parallel auszuführen und so vor allem auf Systemen mit modernen Mehrkernprozessoren deutlich schneller als Maven ist. Dadurch konnte der tägliche Entwicklungsprozess wesentlich beschleunigt werden.

Des Weiteren wird die Pflege der Projektwebseite (siehe Abbildung 5.14 auf Seite 94) über ein Gradle-Plug-in realisiert und an den Erstellungsprozess gekoppelt. Dadurch ist sichergestellt, dass bei Erstellung und Veröffentlichung einer neuen Version der eDAL-Infrastruktur auch gleichzeitig die Projektwebseite aktualisiert wird. Die Webseiten werden dabei dynamisch

generiert und sind als Templates in den Projektressourcen gespeichert. Gradle erstellt daraus automatisch die entsprechenden Webseiten, ersetzt definierte Variablen z. B. für die aktuelle Versionsnummer, und überträgt die fertigen HTML-Dateien auf den festgelegten Webserver.

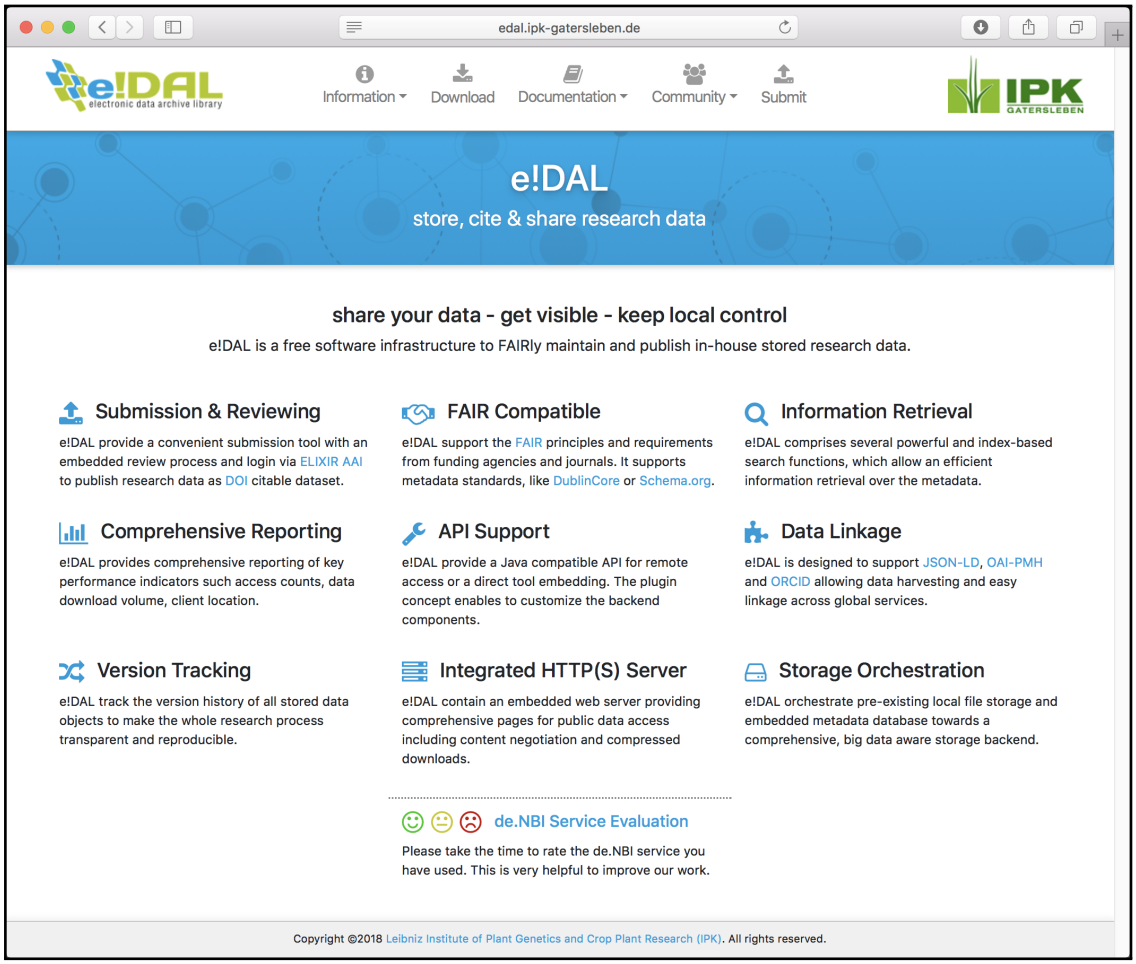

Abbildung 5.14: Screenshot der Projekt-Webseite der eDAL-Infrastruktur

5.3.3 Softwaretests

Um eine hohe Qualität der Implementierung zu gewährleisten und um sicherzustellen, dass alle Funktionen der eDAL-Infrastruktur wie gewünscht funktionieren, ist die Integration einer umfangreichen Testbibliothek notwendig. Da die Infrastruktur mit zunehmender Entwicklung komplexer wird und viele Funktionen auf bereits implementierten Methoden aufbauen oder diese nutzen, muss garantiert werden, dass mit der Implementierung neuer Komponenten keine Fehler in der bisherigen Implementierung verursacht werden.

Eine Testbibliothek ist eine Sammlung von kleinen Programmabschnitten, sogenannten *Unit-Tests*, die eine bestimmte Funktion oder Komponente einer Anwendung auf ihre korrekte Funktion prüft. Eine Testfunktion sollte möglichst einfach implementiert sein und

schnell ein Ergebnis liefern, da die gesamte Testbibliothek gewöhnlich vor der Installation oder Veröffentlichung einer neuen Anwendung ausgeführt wird, um deren Funktionsfähigkeit sicherzustellen. Idealerweise sollten nach Möglichkeit alle Funktionen einer Implementierung geprüft werden, was jedoch aufgrund der Größe und Komplexität eines Projektes nicht immer möglich ist (vgl. Leprevost et al., 2014).

Für die Implementierung einer umfangreichen Sammlung von Unit-Tests für die eDAL-API und Referenzimplementierung wurde das verbreitete JUnit-Framework (JUnit, 2017) genutzt. Es wurde speziell für Java Anwendungen entwickelt und bietet eine Reihe von Methoden zum Testen von implementierten Funktionen und zur Auswertung der Ergebnisse. Darüber hinaus lässt es sich sehr einfach in den Entwicklungsprozess, der im vorangegangenen Abschnitt erläutert wurde, einbinden. Dabei unterscheidet JUnit allgemein nur zwei mögliche Ergebnisse: ein Test gelingt oder schlägt fehl. Quelltext 5.8 zeigt einen Auszug aus einer Test-Suite der eDAL-Infrastruktur. Über die `@Test` Annotation wird eine Testfunktion gekennzeichnet. Das `Assert` Paket der JUnit-API liefert eine Vielzahl an Methoden zur Prüfung, ob eine gegebene Funktion die gewünschten Ergebnisse liefert und damit die Testkriterien erfüllt.

```
public class CompareableTests {

   private static final String FAIL = "compareTo() function failed";

   @Test
   public void testDataSizeCompareTo() {

    DataSize smallDataSize = new DataSize(new Long(100));
    DataSize largeDataSize = new DataSize(new Long(1000));

    /* Unit-Test fail when one of the conditions is wrong */
    Assert.assertTrue(FAIL, smallDataSize.compareTo(largeDataSize) < 0);
    Assert.assertTrue(FAIL, largeDataSize.compareTo(smallDataSize) > 0);
    Assert.assertTrue(FAIL, smallDataSize.compareTo(largeDataSize) == 0);
   }
}
```

Quelltext 5.8: Auszug aus einer JUnit-Test-Suite für eine Funktion der eDAL-Infrastruktur

Insgesamt wurden für die Komponenten der eDAL-API, Referenzimplementierung und Server-Client-Architektur im Laufe der Entwicklung über 50 Unit-Tests implementiert, die eine hohe Softwarequalität gewährleisten und die Stabilität der bisherigen Implementierung während der Weiterentwicklung sicherstellen.

Im Idealfall werden Unit-Tests während einer sogenannten *testgetriebenen Entwicklung*, eine Methode der agilen Softwareentwicklung (siehe Abschnitt 2.3.7) genutzt (vgl. Beck, 2003; George und Williams, 2004; Janzen und Saiedian, 2005). Dabei werden die Unit-Tests vor der eigentlichen Implementierung der zu testenden Funktion geschrieben, weil dadurch die Wahrscheinlichkeit verringert wird, dass eine fehlerhafte Funktion implementiert wird, da nichts umgesetzt wird, was nicht vorher den dazugehörigen Unit-Test bestanden hat. So wird die Qualität des Quellcodes erhöht. Dieses Vorgehen ist jedoch nicht immer zu realisieren, da es besonders bei großen Projekten oft sehr schwierig und zeitaufwendig ist, einen ausreichenden Unit-Test zu entwickeln, für den es noch keine

implementierte Funktion gibt. Dies erfordert ein gewisses Maß an Erfahrung und Übung (vgl. Balaji und Murugaiyan, 2012). Die gesamte Entwicklung der Komponenten der eDAL-Infrastruktur und des Datenbegutachtungsprozesses im Rahmen des Dissertationsprojektes erfolgte nach dem Spiral-Modell (siehe Abschnitt 2.3.7). Da sowohl das Konzept der eDAL-Infrastruktur als auch des Datenbegutachtungsprozesses nicht von Anfang an als final betrachtet wurde, sondern im Laufe der Entwicklung noch verbessert und durch Tests und Erfahrungen von Nutzern beeinflusst wurde, war dieses Entwicklungskonzept aufgrund der höheren Flexibilität besser geeignet als z. B. ein klassisches Vorgehen nach dem Wasserfall-Modell (Abschnitt 2.3.7). Die einzelnen Komponenten wurden nacheinander implementiert und getestet, bevor der nächste Entwicklungsschritt erfolgte. Diese Herangehensweise hatte dabei auch Einfluss auf die für die Implementierung verwendeten Technologien. So wurde das Hibernate-Framework (siehe Abschnitt 4.3.2) auch wegen der flexiblen und einfachen Erweiterungsmöglichkeiten verwendet. Dadurch konnten Funktionen, die erst zu einem späteren Zeitpunkt implementiert wurden und Einfluss auf die Objektpersistierung hatten, mit wenig Aufwand umgesetzt werden, da Hibernate eine einfache Erweiterung bzw. Anpassung des Datenbankschemas ermöglicht. Ein weiteres Beispiel ist der Einsatz der Velocity-API (siehe Abschnitt 5.1.2). Diese ermöglicht die einfache Definition von Templates, um dynamisch Inhalte zu generieren. Durch den umfangreichen Einsatz in verschiedenen Komponenten der Infrastruktur und der Anwendung zur Einreichung und Beschreibung von Forschungsdaten ist es so möglich Anpassungen beispielsweise an der Nutzeroberfläche, einfach und schnell durchzuführen.

6 Evaluierung

In diesem Kapitel werden die Ergebnisse des Dissertationsprojektes evaluiert. Es wird gezeigt, dass die entwickelte Infrastruktur zur Verwaltung und Publikation von Forschungsdaten sowohl aus pflanzenphänotypischen Experimenten, als auch für Forschungsdaten anderen Ursprungs, eignet. Außerdem werden einige Performancetests vorgestellt, die die Leistungsfähigkeit der Implementierung zeigen und Nutzerstatistiken der ersten Instanz auf Basis der entwickelten Infrastruktur am IPK Gatersleben vorgestellt.

6.1 Anforderungen an eine nachhaltige Infrastruktur

In Kapitel 4 wurden die Anforderungen an eine generelle Infrastruktur zur Verwaltung und Publikation von Forschungsdaten erläutert. Im folgenden Abschnitt wird beschrieben, welche dieser Anforderungen bei der Entwicklung der eDAL-Infrastruktur im Rahmen des Dissertationsprojektes umgesetzt wurden und zusammengefasst, wie diese realisiert wurden. Daher ist die Gliederung dieses Abschnittes mit der in Abschnitt 4.1 identisch.

6.1.1 Dateisystem ähnliche Datenstruktur

Für die Realisierung einer geeigneten Datenstruktur wurde in der eDAL-API eine Dateisystem ähnliche Hierarchie mit Verzeichnissen und Dateien konzipiert. Für die Implementierung der Datenstruktur wurde eine Kombination aus einem DBMS und einer Dateisystemanbindung gewählt, um eine möglichst effektive und performante Speicherung zu gewährleisten. Dadurch wird sichergestellt, dass die Infrastruktur auch mit zunehmender Menge an erfassten Daten skaliert, worauf im nächsten Abschnitt dieses Kapitels genauer eingegangen wird.

Metadatenverwaltung

Für die Erfassung von technischen Metadaten wurde in eDAL ein Schema, welches an das DublinCore- bzw. DataCite-Metadatenformat (siehe Abschnitt 2.2.4) angelehnt ist, integriert. Um die Beschreibung der einzelnen Attribute zu erleichtern, wurden eine Reihe von spezifischen Datentypen entwickelt (siehe Abschnitt 4.3.1). In der Referenzimplementierung werden sämtliche Metadaten in der angebundenen H2-Datenbank persistent hinterlegt (siehe Abschnitt 4.3.2). Die Verwendung des besagten Metadatenschemas erleichtert außerdem die Registrierung von DOIs als persistente ID, da bereits alle notwendigen Metadaten erfasst werden und so einfach übertragen werden können. Die Referenzimplementierung der eDAL-Infrastruktur bietet des Weiteren einige integrierte Funktionen, die bestimmte Metadaten wie Dateityp, Dateigröße und Prüfsumme automatisch beim Erfassen eines Datensatzes bestimmen und abspeichern.

Versionsverwaltung

Die eDAL-Infrastruktur ist in der Lage, den gesamten Versionsverlauf aller erfassten Dateien zu speichern, wobei jede Version durch einen vollständigen Satz an Metadaten beschrieben wird. Die jeweiligen Versionsinformationen, wie beispielsweise das Erstellungsdatum oder die Versionsnummer, werden ebenfalls im angebundenen DBMS erfasst. Auch Informationen über die Ordnerstruktur der gespeicherten Datensätze werden in der Datenbank hinterlegt. Die konkreten Dateien werden in einer flachen, effektiven Hierarchie im Dateisystem abgespeichert.

Information Retrieval

Durch Indexierung der Metadaten der in eDAL erfassten Datensätze ist es möglich, schnell über die Attribute der Metadaten nach bestimmten Dateien zu suchen. Dank der index-basierten Suche ist auch mit zunehmender Anzahl an gespeicherten Informationen eine gute Skalierbarkeit und Performance gewährleistet. Die Implementierung einer effizienten Indexierungsstrategie stellt darüber hinaus sicher, dass neu erfasste Dateien schnell indexiert werden und auffindbar sind.

Datensammelschnittstelle

Mit der Implementierung einer Schnittstelle zur DataCite-API zur Registrierung von DOIs profitiert die eDAL-Infrastruktur nicht nur von der Vergabe dauerhafter IDs, sondern auch von der umfangreichen Vernetzung der DOIs und den Diensten von DataCite. Da das DataCite Metadatenschema an den DublinCore angelehnt ist (siehe Abschnitt 2.2.4), bietet DataCite bereits eine Schnittstelle zur Sammlung von Daten auf Basis des OAI-PMH Protokolls (siehe Abschnitt 2.2.4). Somit sind alle in eDAL erfassten Datensätze, die über eine DOI veröffentlicht sind, über die Schnittstelle automatisch auslesbar. Dadurch können alle Informationen des gesamten Datenbestands maschinell erfasst und verarbeitet werden, um beispielsweise Beziehungen zu anderen Datenquellen herzustellen und die Sichtbarkeit der Daten zu erhöhen.

6.1.2 Datensicherheit

Die Entwicklung eines geeigneten Sicherheitskonzepts war ein zentraler Punkt bei der Konzeptionierung und Implementierung der eDAL-Infrastruktur, da dies ein Schlüssel für die Akzeptanz der Nutzer ist. Einerseits stand die Entwicklung eines geeigneten Authentifizierungsmechanismus im Vordergrund, um eine flexible Identifizierung von Nutzern oder Gruppen zu ermöglichen. Andererseits musste für die Autorisierung ein geeignetes Konzept gefunden werden, um die Sicherheitsprüfungen zum Schutz der einzelnen API-Funktionen effektiv zu implementieren, da es sich hier um klassenübergreifende Methoden handelt. Mithilfe der JAAS-API (siehe Abschnitt 4.3.1) konnte ein dynamisches Authentifizierungssystem entwickelt werden, welches es ermöglicht, verschiedene, leicht austauschbare Module zur Benutzeridentifikation zu verwenden. Durch Nutzung aspektorientierter Programmierung konnten die Sicherheitsfunktionen in einen Aspekt gekapselt werden, wodurch der Wartungsaufwand verringert wurde und der Quellcode übersichtlich bleibt.

Single Sign-on Authentifizierung

Die in der eDAL-Infrastruktur verwendete JAAS-API bietet eine universelle Schnittstelle zur Implementierung von Anmeldeverfahren. Dadurch ist es möglich unterschiedliche Authentifizierungsmethoden zu verwenden, bzw. neue Verfahren leicht zu implementieren und

mit der gleichen Infrastruktur zu nutzen. Für die Referenzimplementierung der eDAL-API wurde ein Loginmodul (siehe Abschnitt 2.3.3) implementiert, das den Kerberos-Dienst der IPK Gatersleben nutzt und es so Mitarbeitern ermöglicht, sich mit ihrem gewohnten institutionellen Zugang anzumelden. Um die Infrastruktur darüber hinaus flexibel nutzbar zu machen, wurden des Weiteren eine Reihe von Loginmodulen unter Nutzung des OAuth-Protokolls (siehe Abschnitt 2.3.3) implementiert. So ist die Benutzerauthentifizierung auch über externe Dienste wie Google, ORCID oder ELIXIR möglich.

Feingranulare Autorisierung

Durch die Nutzung aspektorientierter Programmierung ermöglicht die eDAL Infrastruktur eine sehr differenzierte Vergabe von Nutzerrechten. Die umfangreiche Funktionalität von AspectJ erlaubt es, dass alle öffentlichen Methoden auf Objekten, die in der Infrastruktur erfasst werden, freigegeben oder gesperrt werden können. Dabei können die Rechte sowohl für einzelne Nutzer als auch Gruppen über die Vergaben von verschiedenen JAAS-Nutzerprincipals gesetzt werden (siehe Abschnitt 4.3.1). Durch Nutzung des compile-time weaving (siehe Abschnitt 2.3.2) wird dabei die Performance der Infrastruktur kaum beeinflusst. Die eigentliche Sicherheitsfunktion zur Prüfung der Nutzerrechte wird nur einmal zentral in einem Aspekt implementiert. Sie prüft vor dem Ausführen jeder geschützten Funktion, ob der aktuelle Nutzer, der sich zuvor über ein JAAS-Loginmodul angemeldet hat, berechtigt ist, die jeweilige Funktion auszuführen. Die konkreten Rechte der jeweiligen Benutzer werden dabei in der angebundenen H2-Datenbank hinterlegt. Damit auch die Datenbankanfrage zur Kontrolle der Nutzerrechte vor dem Aufruf jeder Funktion die Performance der Infrastruktur nicht beeinflusst, wurde für diese Anfrage ein separater Anfragen-Cache implementiert (siehe Abschnitt 4.3.2).

Das entwickelte Sicherheitssystem der eDAL-API ist sehr eng in die Infrastruktur eingebunden und bietet durch die effektive Verwendung verschiedenen Komponenten und Technologien eine komfortable und performante Nutzung und gleichzeitig hohe Sicherheit.

6.1.3 Datenaustausch und Datenqualität

Neben der effektiven Verwaltung von Forschungsdaten war eine weitere wichtige Aufgabe der entwickelten Infrastruktur die Vergabe von dauerhaften IDs, um die abgelegten Daten nicht nur innerhalb der Infrastruktur für freigegebene Nutzer zur Verfügung zu stellen, sondern diese auch öffentlich bereitzustellen und referenzierbar zu machen. DOIs sind weitverbreitet und in sehr vielen Domänen akzeptiert. So ist auch das IPK Gatersleben als Datenzentrum im DataCite-Konsortium registriert und berechtigt DOIs zu vergeben. Daher lag es nahe, DOIs als persistente IDs ebenfalls im Rahmen des Dissertationsprojektes zur Realisierung der Referenzimplementierung der eDAL-API zu nutzen, um Forschungsdaten dauerhaft verfügbar zu machen.

Persistente Identifikatoren

Über eine REST-API ermöglicht das DataCite-Konsortium die programmatische Vergabe von DOIs. Mithilfe der entsprechenden Schnittstelle in der eDAL-API (siehe Abschnitt 5.1.2) wurde die Verbindung zur DataCite-API etabliert. Darüber werden die nötigen Metadaten, die für die Vergabe einer DOI notwendig sind, sowie die URL zur Inhaltsseite des jeweiligen Datensatzes in der eDAL-Infrastruktur, übertragen. Die Inhaltsseiten werden dabei über integrierte Templates dynamisch erstellt und über einen integrierten Webserver ausgeliefert.

Dadurch müssen keine festen Webseiten gespeichert werden, wodurch Speicherplatz gespart wird. Die Nutzung eines separaten Caches erhöht die Performance bei der Auslieferung der Webseite. Außerdem ist so die Pflege und Erweiterung des Systems sehr einfach und Änderungen, beispielsweise am Layout der Inhaltsseiten, können leicht durch Anpassungen der entsprechenden Templates durchgeführt werden (siehe Abschnitt 5.1.2).

Datenbegutachtungsprozess

Es gibt weder vonseiten des DataCite-Konsortiums konkrete bzw. kontrollierte Anforderungen an die Qualität von Forschungsdaten, die mit einer DOI versehen werden sollen, noch gibt es einen etablierten und generischen Ansatz zur Bewertung von Forschungsdaten. Daher wurde ein Begutachtungsprozess entwickelt, der sich an den etablierten Peer-Review-Prozess zur Begutachtung von wissenschaftlichen Publikationen orientiert. Dadurch wird eine gewisse Datenqualität sichergestellt, da jeder Datensatz vor der Vergabe einer dauerhaften DOI von unabhängigen Gutachtern geprüft wird. Mit der Implementierung einer nutzerfreundlichen, grafischen Anwendung zur Einreichung von Forschungsdaten (siehe Abschnitt 5.2) und durch eine einfache E-Mail basierte Kommunikation zwischen Gutachtern und Datenproduzenten (siehe Abschnitt 5.1.2) wird eine transparente und schnelle Begutachtung gewährleistet.

6.1.4 Interoperabilität

Neben den genannten funktionellen Eigenschaften ist für die Akzeptanz einer generischen Infrastruktur zum Forschungsdatenmanagement eine möglichst einfache Integration in existierende, institutionelle Infrastrukturen und Programme der ausschlaggebende Punkt, der darüber entscheidet, ob das System von den Wissenschaftlern angenommen und genutzt wird. Die Infrastruktur muss möglichst nutzerfreundlich sein und den Anwender nicht durch zusätzlichen Aufwand den täglichen Arbeitsablauf erschweren. Nur so sind die Wissenschaftler gewillt, ihre Daten abzulegen. Daher wurde die eDAL-Infrastruktur mit Java realisiert, da es plattformunabhängig und besonders in den Lebenswissenschaften in vielen Domänen weit verbreitet ist, was die Integration in existierende Systeme erleichtert.

Dokumentenorientierte Speicherung

Die Vielfältigkeit der täglich anfallenden Forschungsdaten ist eine große Herausforderung. Die eDAL-Infrastruktur bietet eine dokumentenorientierte Speicherung und besitzt keine Beschränkungen auf Datentypen oder Dateigrößen. Alle erfassten Dateien werden unverändert als Datenstrom im angebundenen Dateisystem abgelegt und die dazugehörigen Metadaten und Versionsinformationen parallel dazu im DBMS hinterlegt, wodurch eine langfristige und persistente Speicherung gewährleistet ist.

Serviceorientierte Architektur

Die eDAL-Architektur bietet verschiedene, sehr flexible Nutzungsszenarien. Neben der Nutzung als rein lokale Dateispeicherinfrastruktur durch Verwendung der API-Komponente und der Referenzimplementierung ist vor allem die Verwendung als zentrales Repository über die Server-Client-Architektur der Hauptanwendungsfall. Durch die Implementierung als Java-API ist eine programmatische Integration in existierende Anwendungen sehr einfach. Gleichzeitig sind alle Komponenten ebenfalls als Maven-Abhängigkeit im zentralen Maven-Repository hinterlegt (siehe Abschnitt 5.3.2). Dadurch kann die API mit vielen verbreiteten Entwicklungsmanagementsystemen wie Maven (Apache Software Foundation, 2018a) oder

Gradle (Gradle Inc., 2018) genutzt werden. Die einzelnen Komponenten der API (siehe Abschnitt 4.3.2) sind dabei als separate Abhängigkeiten verfügbar, sodass nur die benötigten Komponenten geladen werden müssen.

6.1.5 Zusammenfassung

In den vorangegangen Abschnitten wurde beschrieben, wie die entwickelte Forschungsdateninfrastruktur die in Abschnitt 4.1 vorgestellten Anforderungen umsetzt. Dadurch ist ein Vergleich mit existierenden Infrastrukturen (siehe Kapitel 3) möglich. Zur Veranschaulichung wird die gleiche tabellarische Darstellung (siehe Tabelle 4.1 auf Seite 55) gewählt.

	Domänen-spezifische Informations-systeme	Cloud-basierte Daten-austausch-dienste	Versions-verwaltungs-systeme	Daten-austausch- und Publikations-dienste	Freie Daten-Repository-Systeme	eDAL
Metadaten-verwaltung	(✓)			(✓)	(✓)	✓
Versions-verwaltung	(✓)	✓	✓	(✓)	✓	✓
Information Retrieval	✓	(✓)	(✓)	✓	✓	✓
Datensammel-schnittstelle	(✓)			(✓)	(✓)	✓
Single Sign-on Authentifizierung	(✓)	(✓)	✓	✓	✓	✓
Feingranulare Autorisierung					✓	✓
Persistente Identifikatoren				✓	(✓)	✓
Datenbegutachtungs-prozess						✓
Dokumentenorientierte Speicherung		✓	✓	(✓)	(✓)	✓
Serviceorientierte Architektur		(✓)	(✓)	(✓)	✓	✓

Tabelle 6.1: Vergleich verschiedener existierender Systeme zur Verwaltung und Publikation von Forschungsdaten mit der im Rahmen der Dissertation entwickelten eDAL-Infrastruktur. Ist eine Funktion verfügbar, ist dies durch „✓“ gekennzeichnet. Falls eine Eigenschaft nur teilweise erfüllt ist, wird dies durch „(✓)“ dargestellt. Bei einem leeren Eintrag ist die jeweilige Funktion nicht verfügbar bzw. die Eigenschaft nicht erfüllt.

In Tabelle 6.1 ist erkennbar, dass die eDAL-Infrastruktur alle ermittelten Anforderungen erfüllt und in der Lage ist, die Schwachstellen existierender Systeme zu überwinden. Dies wird besonders im Bereich Datenpublikation und Datenqualität deutlich. Viele der existierenden Infrastrukturen bieten keine Möglichkeit, dauerhafte IDs für Forschungsdatensätze zu vergeben. Dies ist jedoch essenziell, um einerseits die langfristige Auflösbarkeit der Daten zu garantieren und andererseits den Datenproduzenten die Möglichkeit zu geben durch Referenzierung die Anerkennung für die Bereitstellung ihrer Daten zu erhalten. Sowohl die vorgestellten Datenpublikationsplattformen wie figshare oder Zenodo als auch die freien Datenrepository Systeme wie FEDORA oder Dspace sind in der Lage DOIs zu vergeben oder in Verbindung mit den entsprechenden Schnittstellen der ID-Systeme zu treten. Jedoch

bietet kein System eine integrierte Funktion oder die Möglichkeit zur Einbindung eines geeigneten Prozesses zur Prüfung und Begutachtung der veröffentlichten Daten. Dies ist ein sehr großer Vorteil und Alleinstellungsmerkmal der entwickelten eDAL-Infrastruktur.

Eine weitere Schwachstelle der Datenaustauschplattformen ist deren begrenzter freier Speicherplatz. Dieser ist aufgrund des täglich wachsenden Volumens an produzierten Forschungsdaten kaum ausreichend. Dadurch sind Forscher und Institutionen dazu gezwungen, zusätzlichen Speicher kostenpflichtig zu erwerben, während bereits vorhandene lokale Speicherkapazitäten nicht dafür genutzt werden können, um die Daten langfristig zu archivieren und öffentlich zur Verfügung zu stellen. Freie Daten-Repository-Systeme zur Etablierung einer institutionellen Datenmanagementinfrastruktur haben dieses Defizit zwar nicht, sind jedoch von der Einrichtung und Wartung her sehr komplex und erfordern ein hohes Maß an technischem Verständnis. Die entwickelte eDAL-API, ist in der Lage beide Defizite zu überwinden. Die Einrichtung einer zentralen eDAL-Infrastruktur ist sehr einfach, da in der Referenzimplementierung der API (siehe Abschnitt 4.3.2) alle notwendigen Komponenten wie ein Datenbankmanagementsystem oder ein Webserver integriert sind und keine zusätzlichen Abhängigkeiten zur erfolgreichen Einrichtung notwendig sind. Für die Initialisierung ist lediglich ein DataCite-Zugang zur Vergabe von DOIs notwendig. Die dafür benötigen Zugangsdaten bekommt jede Institution über eine Registrierung als Datenzentrum im DataCite-Konsortium. Dabei werden einige rechtliche Aspekte und Verantwortlichkeiten geklärt. Das System prüft bei der Initialisierung automatisch, ob alle Parameter richtig gesetzt sind und kontrolliert die notwendigen Verbindungen zu den Schnittstellen der genutzten Dienste. Treten etwa Verbindungsprobleme auf, z. B. aufgrund einer fehlerhaften Netzwerk- oder Proxy-Konfiguration, wird der Nutzer darüber informiert, um die entsprechenden Parameter zu korrigieren (siehe Abschnitt 5.1.2). Darüber hinaus wird nur ein Zugang zu einem E-Mail Server benötigt, um E-Mails, die beispielsweise während des Datenbegutachtungsprozesses erstellt werden, zu versenden (siehe Abschnitt 5.1.2).

Die eDAL-API bietet damit eine einfache Möglichkeit, lokal vorhandenen, institutionellen Speicherplatz effizient zu nutzen und liefert eine leistungsstarke Infrastruktur zur Verwaltung und Publikation beliebiger Forschungsdaten. Der Ansatz der eDAL-Infrastruktur ist es, die notwendige und generische Infrastruktur zu den bereits vorhanden Daten zu bringen und nicht die Daten zu einer entfernten Infrastruktur zu übertragen.

6.2 Leistungstests

Wie bereits erwähnt, wurde bei der Programmierung der Referenzimplementierung der eDAL-API darauf geachtet, eine möglichst hohe Performance des Systems zu gewährleisten, da dies eine entscheidende Voraussetzung für die Nutzerakzeptanz der Infrastruktur ist. Durch die stetig zunehmende Menge an Forschungsdaten ist es notwendig, eine schnelle Laufzeit zu erreichen und eine hohe Skalierbarkeit zu gewährleisten, um den Aufwand für die Wissenschaftler möglichst gering zu halten und die langfristige Stabilität der Infrastruktur zu garantieren. Daher wurden im Laufe des Dissertationsprojektes eine Reihe von Laufzeittests entwickelt und durchgeführt, um die Performance zu überwachen und mögliche Schwachstellen der Implementierung aufzudecken. In den folgenden Abschnitten werden einige ausgewählte Tests vorgestellt und deren Ergebnisse ausgewertet.

Je nach Testszenario wurde sowohl die Performance der Infrastruktur als lokale Datenhaltungsinfrastruktur sowie die Nutzung als ein zentrales Repository in einer Server-Client-Architektur untersucht. Dafür wurden zwei lokale Rechner am IPK Gatersleben verwendet. Auf beiden Systemen wurde mittels VirtualBox (Oracle, 2018f) die gleiche virtuelle Linux-Distribution (Ubuntu Desktop 16.04 LTS) als Betriebssystem verwendet, die jeweils mit den folgenden Parametern konfiguriert wurden:

System A (lokale Tests und Serversystem)

- Intel Core i7 Prozessor; 4-Kerne (4 parallele Threads)
- 8 GB Hauptspeicher (DDR4)
- SATA-3 SSD für DBMS und Indexdateien (512 GB)
- SATA-3 HDD (5400 RPM) für Speicherung der konkreten Dateien (1 TB)

System B (Clientsystem)

- Intel Core i5 Prozessor; 2-Kerne (2 parallele Threads)
- 4 GB Hauptspeicher (DDR3)
- SATA-3 SSD (512 GB)

Beide Systeme verfügen über eine 1-Gbit schnelle Netzwerkanbindung und jede JVM wurde mit 1 GB Speicher initialisiert. Alle vorgestellten Tests wurden mithilfe von AspectJ (siehe Abschnitt 2.3.2), welches schon für die Implementierung des Sicherheitssystems der eDAL Infrastruktur (siehe Abschnitt 4.3.1) genutzt wurde, realisiert. Es wurde ein neuer Aspekt implementiert, um die exakten Laufzeiten der verschiedenen untersuchten Funktionen der API, die jeweils durch entsprechende JoinPoints definiert wurden, zu erfassen und in Logdateien zu schreiben. Die Auswertung und die Erstellung der grafischen Abbildungen der gemessenen Werte wurden mit R (R Core Team, 2018) durchgeführt.

6.2.1 Daten abspeichern und auslesen

Der erste Laufzeittest untersucht die Performance beim Abspeichern und Auslesen von Objekten, da diese zu den wichtigsten und am häufigsten verwendeten Funktionen der Infrastruktur gehören. Für die Tests wurden vier verschiedene Datensätze, jeweils mit einem Gesamtvolumen von 1 GB erzeugt. Es wurde dann bei jedem Durchlauf ein Datensatz von einem Nutzer bzw. alternativ dreimal der gleiche Datensatz von drei verschiedenen Nutzern parallel in der eDAL-Infrastruktur abgelegt und anschließend wieder ausgelesen. Zwischen jedem Versuch wurde die Infrastruktur gelöscht und neu initialisiert, sodass jeder Test mit leerem Datenbestand erfolgte. Um die Laufzeit und Skalierbarkeit des Systems zu prüfen, beinhalteten alle Datensätze eine unterschiedliche Anzahl an Dateien, jeweils gleichmäßig auf Unterordnern verteilt, damit mögliche Einflüsse durch das verwendete Dateisystem ausgeschlossen werden konnten. Tabelle 6.2 zeigt die Zusammensetzung der Datensätze.

Datensatz	Ordner	Dateien pro Ordner	Dateigröße	Gesamtvolumen
Datensatz 1	10	10	10,24 MB	1 GB
Datensatz 2	10	100	1,024 MB	1 GB
Datensatz 3	100	100	104,8 KB	1 GB
Datensatz 4	100	1000	10,48 KB	1 GB

Tabelle 6.2: Übersicht der verwendeten Datensätze für den Laufzeittest zur Bewertung der Geschwindigkeit zum Ein- und Auslesen von Dateien in die eDAL-Infrastruktur

Alle Testläufe wurden dreimal durchgeführt. Daraus wurde jeweils ein Mittelwert pro Messung gebildet, um einzelne kurze Schwankungen oder zufällige Ausreißer, die während eines Testlaufes, beispielsweise durch im Hintergrund ablaufende Betriebssystemprozesse, Netzwerkverkehr, den installierten Virenscanner oder die Konfiguration der Firewall auftreten können, auszugleichen. Anschließend wurden die Laufzeiten der einzelnen Schritte dann nacheinander addiert, um einen Verlauf zu ermitteln.

Lokaler Laufzeittest - Einzelnutzer

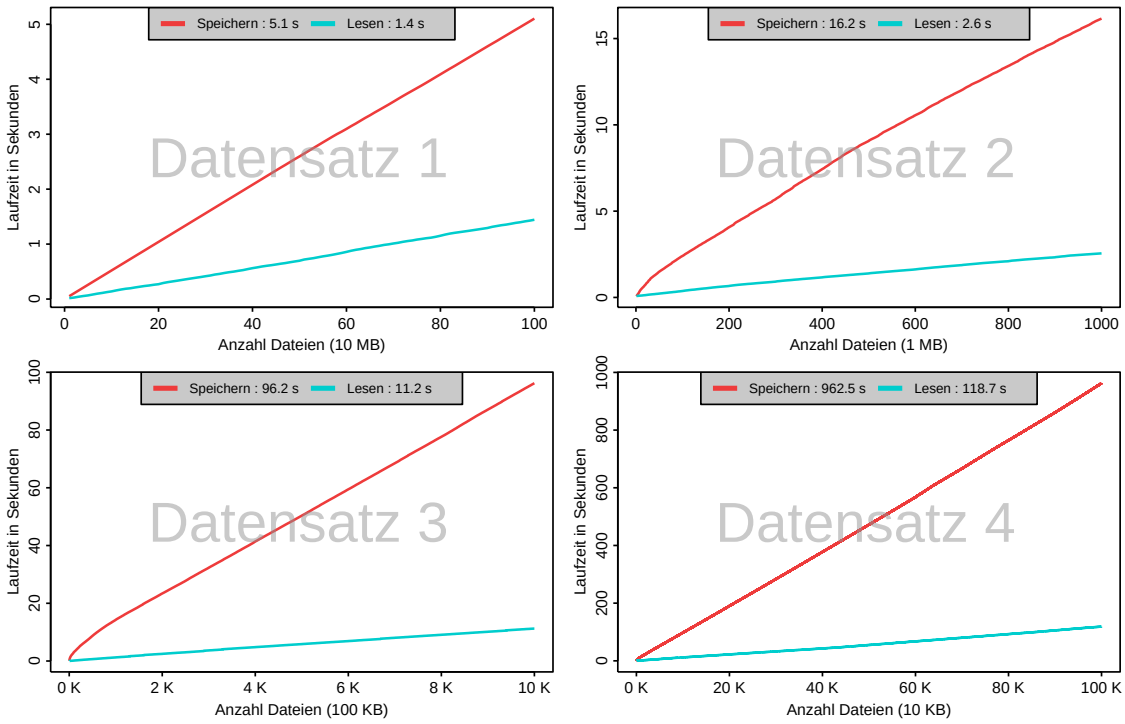

Abbildung 6.1: Ergebnisse des lokalen Laufzeittests für das Speichern und Lesen von Objekten in die eDAL-Infrastruktur durch einen Nutzer. Die x-Achsen zeigen die Anzahl an erfassten bzw. geladenen Dateien (für Datensatz 3 und 4 auf 1000 skaliert) und die y-Achsen die Laufzeit in Sekunden. Die Gesamtlaufzeit ist in der jeweiligen Legende abgebildet.

In Abbildung 6.1 sind die Ergebnisse des lokalen Laufzeittests mit einem einzelnen Nutzer grafisch dargestellt, um die generelle Performance des Systems zu prüfen. Bei diesem ersten Test wird deutlich, dass die eDAL-Infrastruktur trotz eines gleichen Gesamtvolumens für die umfangreicheren Datensätze, die mehr Objekte enthalten, im Allgemeinen mehr Zeit zum Abspeichern und Auslesen benötigt als für Datensätze mit weniger Objekten. So dauert das Speichern und Auslesen von 100 Objekten etwa 5 Sekunden bzw. 1,5 Sekunden, während es für 100.000 Objekte etwa 962 Sekunden bzw. 118 Sekunden dauert. Das liegt daran, dass für die Erfassung von mehr Objekten auch mehr Operationen notwendig sind, die Zugriff auf die angebundene Datenbank erfordern. Es müssen öfters Metadaten, wie z. B. der MIME-Type oder die MD5-Prüfsummen der erfassten Dateien, ermittelt und in der Datenbank abgelegt werden. Außerdem müssen häufiger Zugriffsrechte durch Datenbankabfragen geprüft werden und die Indexdateien für die Suchfunktionen öfters aktualisiert werden. Zusätzlich muss

vor dem Erstellen eines neuen Objektes in einem Ordner geprüft werden, ob es bereits Objekte mit dem gleichen Namen gibt. Je mehr Objekte bereits in einem Ordner sind, desto aufwendiger ist dabei die Prüfung.

Des Weiteren wird in den Diagrammen sichtbar, dass die Laufzeit für das Auslesen erfasster Objekte geringer ist, da im Allgemeinen beim Laden weniger Operationen ausgeführt werden. Es muss lediglich geprüft werden, ob der aktuelle Nutzer die Erlaubnis hat, auf die jeweiligen Dateien zuzugreifen. Gleichzeitig wird bei diesem Test deutlich, wie gut die Infrastruktur skaliert und die Laufzeit linear mit der Anzahl der Objekte ansteigt. Somit stellt auch das Erfassen von umfangreichen Datensätzen keine Herausforderung dar.

Lokaler Laufzeittest - Mehrnutzer

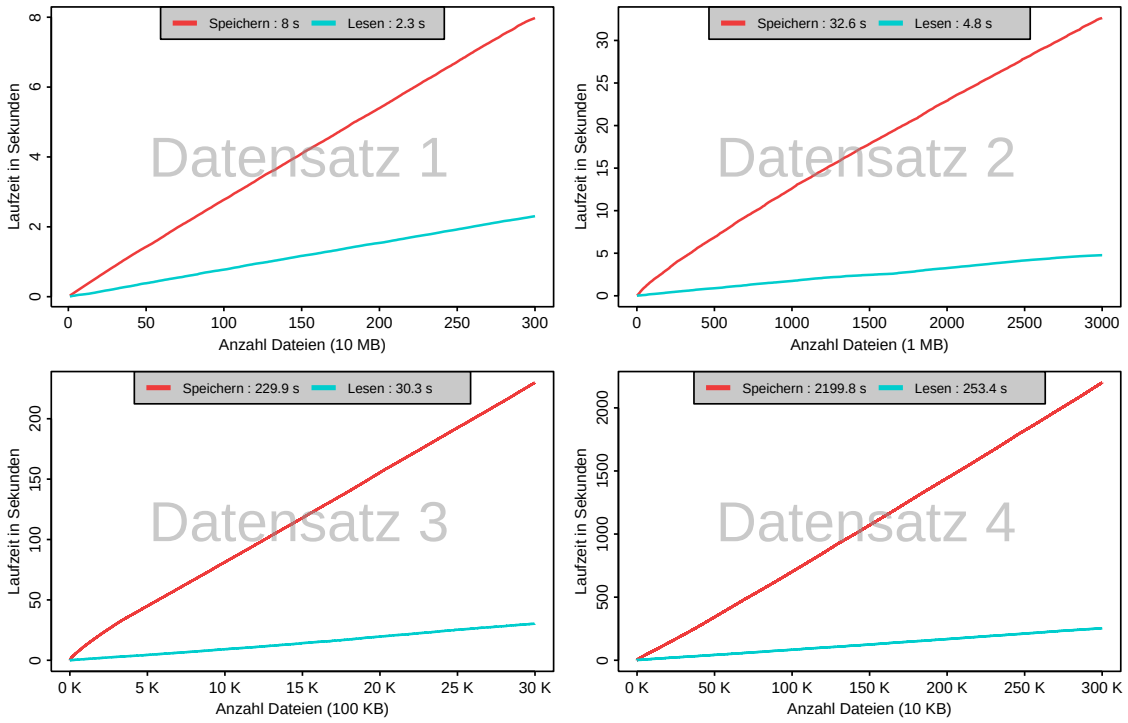

Abbildung 6.2: Ergebnisse des lokalen Laufzeittests für das Speichern und Lesen von Objekten in die eDAL-Infrastruktur durch mehrere Nutzer. Die x-Achsen zeigen die Anzahl an erfassten bzw. geladenen Dateien (für Datensatz 3 und 4 auf 1000 skaliert) und die y-Achsen die Laufzeit in Sekunden. Die Gesamtlaufzeit ist in der jeweiligen Legende abgebildet.

Abbildung 6.2 zeigt die Ergebnisse des lokalen Laufzeittests mit mehreren Anwendern. Für diesen Test wurden die gleichen Datensätze von drei verschiedenen Nutzern parallel abgespeichert und wieder ausgelesen, um die Performance im Mehrnutzerbetrieb bzw. die Effektivität der Multi-Thread-Implementierung (siehe Abschnitt 4.3.3) zu evaluieren. Bei der Betrachtung der Grafiken wird erkennbar, dass sich die Verläufe denen der vorangegangenen Tests mit nur einen Nutzer sehr ähneln. Obwohl drei Nutzer gleichzeitig Daten ablegen und auslesen und so der Datenbestand, den die Infrastruktur verarbeitet dreimal so groß ist,

verlaufen die Kurven linear und die Infrastruktur skaliert sehr gut. In allen vier Durchläufen zeigte sich, dass nicht dreimal so viel Zeit zum Erfassen der Datensätze benötigt wird, sondern jeweils etwas weniger. Dies zeigt die Vorteile der Implementierung und deren Optimierung für Systeme mit Mehrkernprozessoren. Dadurch können die verschiedenen Funktionen, die beim Abspeichern von Objekten in die eDAL-Infrastruktur aufgerufen werden, parallel ablaufen und so einen Laufzeitvorteil liefern. Auf der anderen Seite gab es bei der Laufzeit für das parallele Auslesen der drei Datensätze nur einen geringen Geschwindigkeitsvorteil. Dies könnte jedoch auch drauf zurückzuführen sein, dass der Test auf einem einzelnen, lokalen Rechner durchgeführt wurde. Die Durchläufe für die drei Nutzer wurden zwar jeweils in verschiedenen Threads gestartet, jedoch schreiben alle Nutzer auf die gleiche Festplatte, wodurch es zu Verzögerungen beim parallelen Auslesen kommen kann, da dies auch gleichzeitig noch für das Auslesen der Daten aus der eDAL-Infrastruktur verwendet wird. Eventuell würde der gleiche Test mit drei Nutzern, welche die Datensätze auf jeweils separate Festplatten auslesen, ein besseres Ergebnis liefern.

Zentraler Laufzeittest - Einzelnutzer

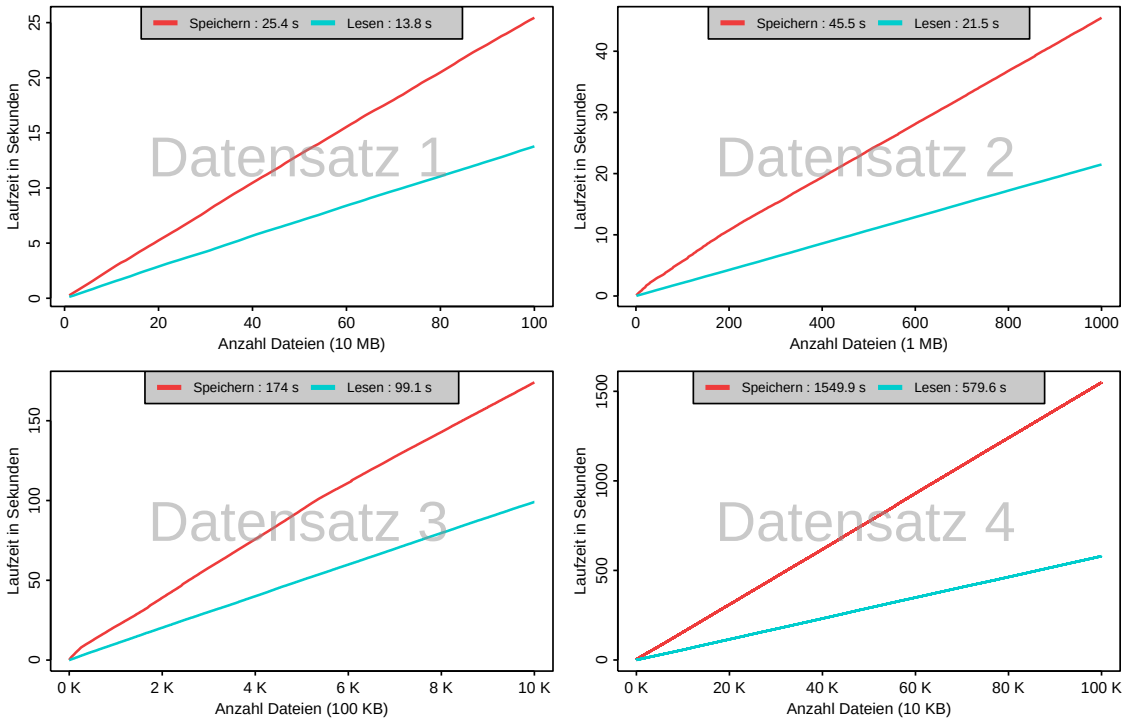

Abbildung 6.3: Ergebnisse des Laufzeittests für das Speichern und Lesen von Objekten in eine zentrale eDAL-Infrastruktur durch einen Nutzer. Die x-Achsen zeigen die Anzahl an erfassten bzw. geladenen Dateien (für Datensatz 3 und 4 auf 1000 skaliert) und die y-Achsen die Laufzeit in Sekunden. Die Gesamtlaufzeit ist in der jeweiligen Legende abgebildet.

Im zweiten Abschnitt der ersten Laufzeittests wurden die beiden vorangegangen Versuche mit den exakt gleichen Parametern und Datensätzen wiederholt. Diesmal wurde die eDAL-API jedoch nicht als lokale Datenhaltungsinfrastruktur genutzt, sondern als zentrales Repository auf dem Serversystem (System A, siehe Abschnitt 6.2) initialisiert, da es sich dabei um das

gebräuchlichere Nutzungsszenario der Infrastruktur handelt. Die einzelnen Testdurchläufe wurden dann über die Funktionen der Server-Client-Architektur der eDAL-API auf dem Clientsystem (System B, siehe Abschnitt 6.2) gestartet. In Abbildung 6.3 auf Seite 106 sind die Ergebnisse der Laufzeittests der Server-Client-Architektur der eDAL-Infrastruktur mit jeweils einem Clientnutzer abgebildet. Auf den ersten Blick wird sofort deutlich, dass das allgemeine Laufzeitverhalten der Infrastruktur, wie erwartet, den Ergebnissen der vorangegangenen Tests entspricht. Das System skaliert auch bei umfangreichen Datensätzen und benötigt mehr Zeit zum Abspeichern neuer Objekte, als für das Auslesen erfasster Datensätze. Insgesamt ist die Laufzeit jedoch für beide Operationen etwas höher, was durch den zusätzlichen Transfer der Daten über das Netzwerk verursacht wird. Darüber hinaus müssen die Objekte und Datenströme bei der Übertragung über RMI (siehe Abschnitt 4.3.1) erst serialisiert werden. Dies beeinflusst die Laufzeit zusätzlich.

Zentraler Laufzeittest - Mehrnutzer

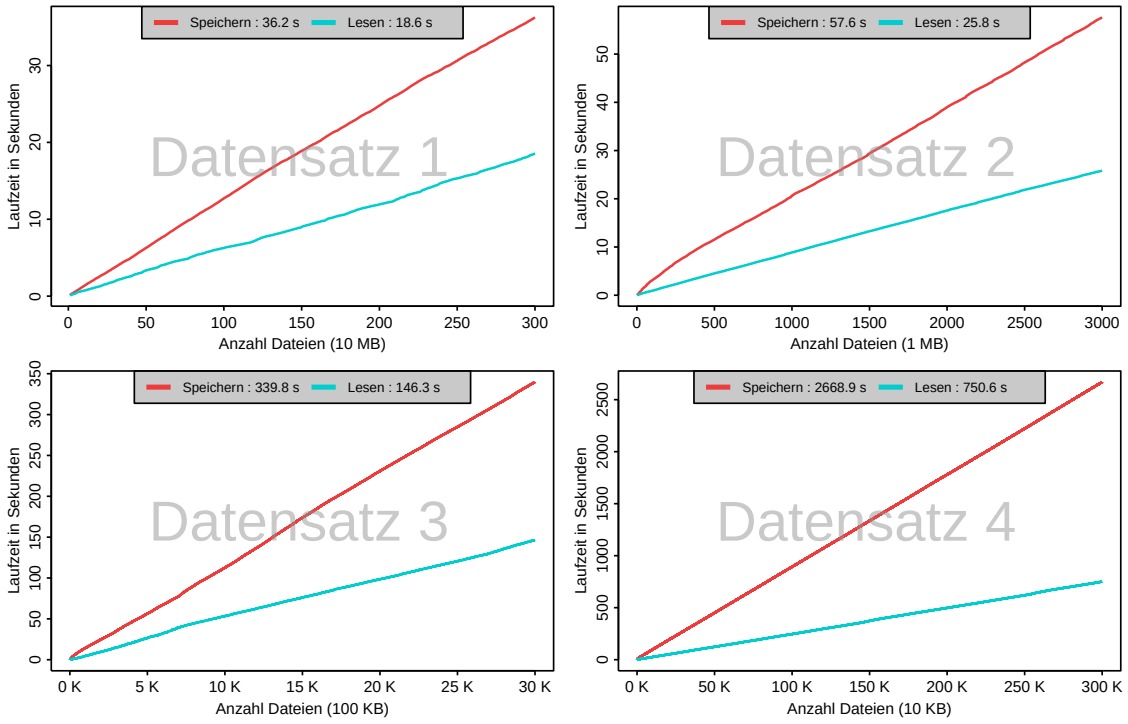

Abbildung 6.4: Ergebnisse des Laufzeittests für das Speichern und Lesen von Objekten in eine zentrale eDAL-Infrastruktur durch mehrere Nutzer. Die x-Achsen zeigen die Anzahl an erfassten bzw. geladenen Dateien (für Datensatz 3 und 4 auf 1000 skaliert) und die y-Achsen die Laufzeit in Sekunden. Die Gesamtlaufzeit ist in der jeweiligen Legende abgebildet.

Abbildung 6.4 zeigt die Resultate der Laufzeittests der Server-Client-Architektur der eDAL-Infrastruktur mit jeweils mehreren Clientnutzern. Genau wie bei der ersten Versuchsreihe wurden die Tests auch mit drei parallelen Clientanwendern wiederholt. Auch diese Versuche benötigen aufgrund des Netzwerktransfers und der notwendigen Serialisierung der Objekte und Dateiströme eine höhere Laufzeit als die Versuche mit einer lokalen Infrastruktur. Jedoch zeigt sich hier wie bereits vermutet, dass der Laufzeitvorteil für das parallele

Auslesen in diesem Test höher ist als beim analogen Versuch mit der lokalen Instanz der eDAL-Infrastruktur. Zwar wurden alle drei Clientanwender wieder auf dem gleichen Rechner in unterschiedlichen Threads initialisiert und schreiben auf die gleiche Festplatte, jedoch läuft die eDAL-Infrastruktur auf einem zentralen Server mit einer separaten Festplatte und beeinflusst die Laufzeit beim Auslesen so weniger.

Laufzeittest - verschiedene Dateigrößen

Im nächsten Abschnitt der Laufzeittests wurde der Einfluss der Größe des Datenvolumens auf die Performance der eDAL-Infrastruktur untersucht. Dafür wurden ebenfalls verschiedene Testdatensätze generiert, die erst in die Infrastruktur abgelegt und anschließend wieder abgerufen wurden. Diesmal wurden jedoch Datensätze mit der gleichen Anzahl an Objekten (100.000), aber mit unterschiedlicher Größe der einzelnen enthaltenen Dateien (siehe Tabelle 6.3) gewählt, sodass sich jeweils ein unterschiedliches Gesamtvolumen pro Datensatz ergibt.

Datensatz	Ordner	Dateien pro Ordner	Dateigröße	Gesamtvolumen
Datensatz 1	100	1000	10,48 KB	1 GB
Datensatz 2	100	1000	104,8 KB	10 GB
Datensatz 3	100	1000	1,024 MB	100 GB

Tabelle 6.3: Übersicht der Datensätze für den Laufzeittest zur Bewertung der Geschwindigkeit zum Ein- und Auslesen von verschiedenen Dateigrößen in die eDAL-Infrastruktur.

Der Test wurde wie bei den vorangegangenen Versuchen einmal sowohl mit einer lokalen Instanz der Infrastruktur als auch mit einer zentralen Serverinstanz durchgeführt. Es wurde aber nur jeweils ein Datensatz von einem Nutzer gespeichert bzw. ausgelesen. Abbildung 6.5 auf Seite 109 zeigt die Ergebnisse des lokalen Laufzeittests für das Speichern und Lesen verschiedener Datensätze mit unterschiedlichem Volumen. Die Erkenntnisse, die in den vorangegangenen Tests gewonnen wurden, konnten durch die Ergebnisse dieses Versuchs bestätigt werden. Die Dauer für das Erfassen von Objekten in die eDAL-Infrastruktur hängt sehr stark von der Anzahl der enthaltenen Objekte und weniger von der Größe des Datensatzes bzw. der einzelnen Dateien ab. Wie bei den bisherigen Versuchen zeigt sich, dass das Auslesen im Allgemeinen schneller ist, als das Abspeichern von Objekten, da unter anderem aufwendigere Datenbankoperationen notwendig sind. Jedoch zeigt sich beim Vergleich der Datensätze miteinander, dass hier die Unterschiede wesentlich geringer sind. So benötigt die Ablage von 100 GB nur knapp 6 Minuten oder etwa 35 % mehr Zeit als die Ablage von 1 GB, obwohl das Volumen 100-mal so groß ist. Der Unterschied zwischen 1 GB und 10 GB ist mit etwa 40 Sekunden noch geringer. Beim Vergleich der Laufzeit für das Auslesen der einzelnen Datensätze sind die Unterschiede noch geringer, da hier wie schon erwähnt weniger zeitintensive Datenbankoperationen ausgeführt werden müssen. So dauert das Auslesen des 100 GB großen Datensatzes mit 4,8 Minuten weniger als das Doppelte der Laufzeit für den 1 GB Datensatz. Die Ergebnisse des analogen Tests mit einem zentralen eDAL-Repository zeigten ähnliche Ergebnisse. Die erhaltenen Werte unterscheiden sich jedoch um einen zusätzlichen Faktor für die Laufzeit, der durch die notwendige Serialisierung und Übertragung der Daten über das Netzwerk hinzukommt. Außerdem sind die Abstände zwischen den Datensätzen etwas anders verteilt als bei den lokalen Versuchen. So wird für das Erfassen des 100 GB großen Datensatzes mit rund 70 Minuten mehr als doppelt

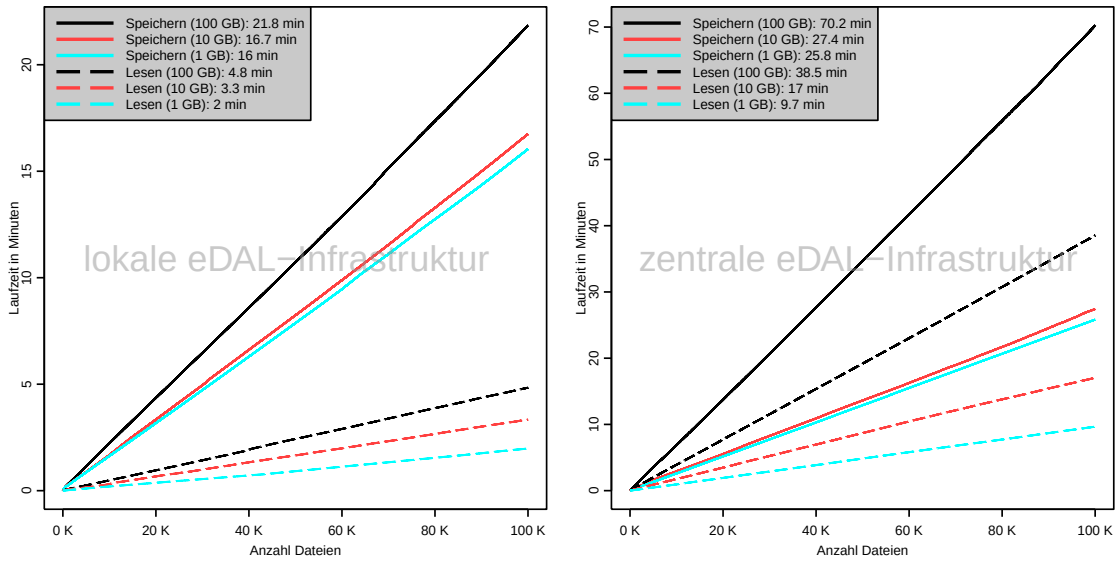

Abbildung 6.5: Ergebnisse des lokalen und zentralen Laufzeittests für das Speichern und Lesen von Datensätzen mit unterschiedlichem Volumen bei gleicher Anzahl an Objekten

solange wie für den 10 GB großen Datensatz benötigt. Jedoch hat dieser auch nur 1/10 des Gesamtvolumens. Es zeigt sich hier, dass die Serialisierung der Dateiströme bei der Übertragung über das Netzwerk für große Dateien mehr Zeit benötigt als für kleinere Dateien. Diese Unterschiede zeigten sich auch bei der Laufzeit für das Auslesen der Objekte, die im Vergleich zum lokalen Test wesentlich weiter auseinanderliegen, da die Serialisierung erheblichen Einfluss auf die Laufzeit hat.

Vergleich mit Dateisystem

In Abschnitt 4.3.2 wurde erläutert, dass für die Referenzimplementierung der eDAL-Infrastruktur eine Kombination aus Dateisystem und DBMS gewählt wurde, um von den Vorteilen beider Infrastrukturen zu profitieren. Während Metadaten, Versionsinformationen und Zugriffsrechte in der Datenbank abgelegt werden, wird das Dateisystem zur Speicherung der konkreten binären Dateien sowie für die Dateien des Suchindexes genutzt. Daher wurde in einem letzten Test ein Vergleich der eDAL-Infrastruktur mit einem üblichen Dateisystem (System A, siehe Abschnitt 6.2) durchgeführt. Dafür wurde ein weiterer Datensatz, der genau wie bei den ersten Tests 1 GB groß ist, erstellt. Dieser enthält jedoch 1 Million Dateien, die auf 1000 Unterordner verteilt wurden und je 1,048 KB groß sind. Damit sollte in erster Linie geprüft werden, ob die Infrastruktur auch bei noch umfangreicheren Datensätzen skaliert. Parallel dazu wurde ein vergleichbarer Versuch implementiert, bei dem der gleiche Datensatz mit Java über die native API in einem Dateisystem gespeichert und anschließend wieder ausgelesen wurde. Beide Versuche wurden wie die vorangegangenen Versuche in zwei Varianten ausgeführt. Einmal mit einem einzelnen Nutzer, der genau einen Datensatz abspeichert und ausliest und einmal mit drei Nutzern, die dies jeweils mit einem Datensatz parallel tun. Abbildung 6.6 auf Seite 110 zeigt die Ergebnisse der Versuche.

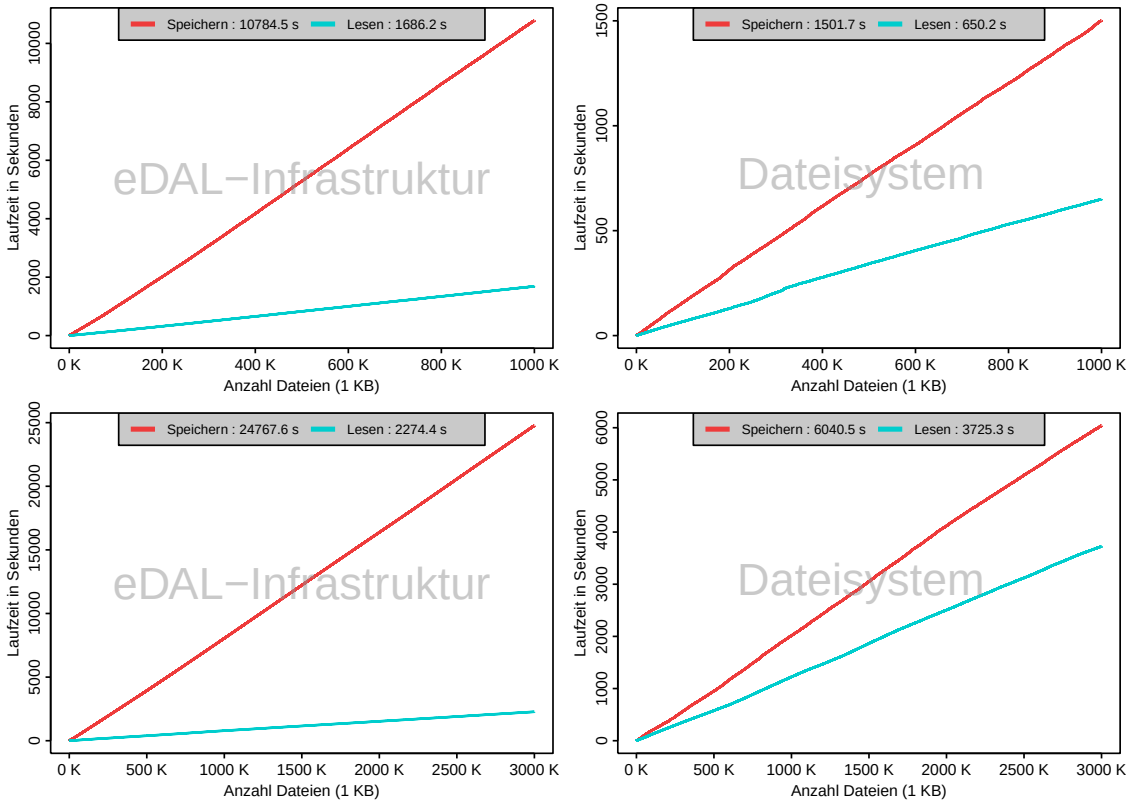

Abbildung 6.6: Ergebnisse des lokalen Laufzeittests für das Speichern und Lesen von Datensätzen im Vergleich mit einem Dateisystem. Die oberen Diagramme stellen die Laufzeit für einen einzelnen Nutzer dar. Die Ergebnisse des Versuchs mit drei parallelen Nutzern sind in den unteren Grafiken abgebildet.

Das wichtigste Ergebnis dieser Versuche ist, dass die eDAL-Infrastruktur auch bei sehr großem Datenbestand von 3 Millionen Objekten eine lineare Laufzeit aufzeigt und skaliert. Die Gesamtlaufzeit liegt zwar über der eines normalen Dateisystems, jedoch lässt sich dieses Ergebnis dadurch erklären, dass ein Dateisystem die Dateien des Datensatzes lediglich in einem Verzeichnis im Dateisystem ablegt bzw. aus diesem ausliest. Die eDAL-Infrastruktur erfasst hingegen beim Abspeichern umfangreiche Versionsinformationen und bestimmt technische Metadaten wie Dateigröße, MIME-Type und die MD5-Prüfsumme jedes Objektes. Diese Informationen werden parallel beim Übertragen der Dateien in das angebundene Dateisystem in der integrierten Datenbank hinterlegt. Darüber hinaus werden beim Speichern existierende Nutzerrechte geprüft, um sicherzustellen, dass der Nutzer diese neuen Objekte auch anlegen darf, und gleichzeitig werden neue Nutzerrechte in der Datenbank abgelegt. Zusätzlich wird parallel ein Suchindex über die Metadaten erstellt, um später eine performante Suche zu gewährleisten. Dadurch ermöglicht die eDAL-Infrastruktur im Gegensatz zu einem einfachen Dateisystem eine langfristige Archivierung und Nutzbarkeit der erfassten Daten. Ein weiterer Unterschied ist das Verhältnis zwischen den Versuchen mit einem einzelnen Nutzer und drei Nutzern, die parallel Daten ablegen. Die eDAL-Infrastruktur bietet, wie schon bei den vorangegangenen Tests gezeigt, einen Vorteil beim parallelen Verarbeiten von Daten. So benötigt der Versuch mit drei parallelen Nutzern nur etwa das 2,5-fache der Zeit, während das Dateisystem hier langsamer ist und etwa das 4-fache der Zeit benötigt, um die drei Datensätze zu erfassen.

Zusammenfassung

Die in diesem Abschnitt vorgestellten Laufzeittests für das Erfassen und Auslesen von Objekten haben gezeigt, dass die entwickelte eDAL-Infrastruktur nicht nur die funktionellen Anforderungen erfüllt und sich ideal als generische Infrastruktur zur langfristigen Erfassung und Publikation von verschiedenen Arten von Forschungsdaten eignet, sondern auch eine hohe Performance aufweist und sehr gut skaliert. Dies ist essenziell für die Akzeptanz der Nutzer und für die Gewährleistung einer langfristigen und stabilen Nutzbarkeit der Infrastruktur. Die einzelnen Maßnahmen zur Optimierung der Referenzimplementierung wie z. B. die Verwendung von verschiedenen Caches für häufige Datenbankabfragen, die Implementierung einer manuellen Indexierungsstrategie sowie die Optimierungen für Mehrkernprozessoren vor allem für das Erfassen und Verarbeiten von neuen Dateien, bewirken, dass die Infrastruktur auch sehr umfangreiche Forschungsdatensätze, die unter anderem im Bereich der pflanzenphänotypischen Daten häufig auftreten, erfassen und verarbeiten kann.

6.2.2 Daten suchen

Die Suchfunktion ist eine weitere bedeutende Funktion, deren Performance sehr wichtig für die Funktionalität und Akzeptanz der Infrastruktur ist. Mit zunehmender Größe des Datenbestands ist eine schnelle Suchfunktion notwendig, um eine effektive Nutzung der Infrastruktur zu gewährleisten. Dies wurde mit den folgenden Tests evaluiert. Dabei wurden zwei Aspekte untersucht. Zum einen wurde der Einfluss der Anzahl an gefundenen Ergebnissen einer festgelegten Suchanfrage auf die Dauer der Ausführung der Suche untersucht. Andererseits wurde geprüft, wie die Größe des Bestands an erfassten Objekten in der eDAL-Infrastruktur die Geschwindigkeit bei der Ausführung einer Suchanfrage beeinflusst. Für den ersten Test wurde ein Testdatensatz mit 100.000 Objekten (aufgeteilt auf 1000 Ordner mit je 100 Dateien) in einer lokalen Instanz der eDAL-Infrastruktur erfasst. Anschließend wurden eine Reihe von Suchanfragen ausgeführt, die aufsteigend 1 bis 1000 Objekte finden und zurückliefern. Für den zweiten Test wurde eine Suchanfrage definiert, die genau 10 Objekte als Ergebnis liefert. Danach wurde der gleiche Datensatz wie für den ersten Test (100.000 Dateien) in die Infrastruktur geladen. Dabei wurde nach jedem Ordner, also nach genau 100 neuen Objekten die erwähnte Suchanfrage wiederholt.

Die Ergebnisse der beiden Tests sind in Abbildung 6.7 auf Seite 112 dargestellt. Alle Versuche wurden wie bei den vorangegangenen Tests dreimal durchgeführt, um die Mittelwerte zu berechnen und zufällig auftretende Ausreißer auszugleichen. Das linke Diagramm zeigt die Ergebnisse des ersten Tests. Dabei wird deutlich, dass die Geschwindigkeit der Suchfunktion klar von der Anzahl der gefundenen Ergebnisse abhängt. Mit zunehmender Zahl an Objekten steigt auch die Dauer der Ausführung der Suchfunktion. Während für das Finden und Zurückliefern von 100 Objekten etwa 80 ms notwendig sind, benötigt die Suchfunktion, die 1000 Objekte liefert, schon ca. 800 ms. Dies liegt jedoch in erste Linie nicht an der Implementierung der Suchanfrage, sondern am objektrelationalen Mapping durch Hibernate (siehe Abschnitt 4.3.2). Konkret benötigt die Suchfunktion den größten Teil der Laufzeit für das Wiederherstellen der gefundenen Objekte aus der Datenbank. Das Finden der gesuchten Objekte mithilfe der indexbasierten Suche ist hingegen sehr schnell, was auch im zweiten Test deutlich wird, dessen Ergebnisse im rechten Diagramm dargestellt sind. Die Laufzeit

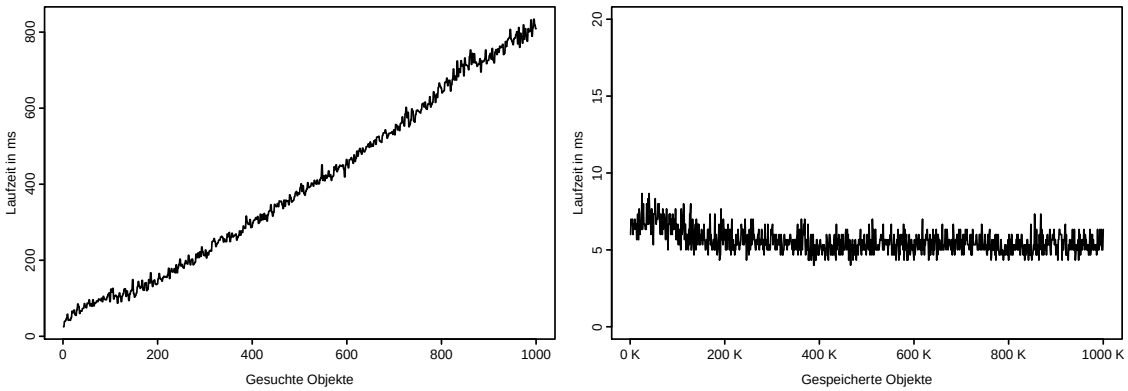

Abbildung 6.7: Ergebnisse des Laufzeittests für das Suchen von Objekten in einer lokalen eDAL-Infrastruktur. Die x-Achse zeigt die Anzahl an gefunden Objekten (links) bzw. erfassten Objekten (rechts). Die y-Achse zeigt jeweils die Laufzeit in Millisekunden.

für die fest definierte Suchanfrage, die genau 10 Objekte liefert, liegt konstant bei ca. 8 ms, wie bereits in ersten Test gezeigt wurde. Die Laufzeit verändert sich auch mit zunehmender Größe des erfassten Datenbestands der eDAL-Infrastruktur kaum. Die Ausreißer im Verlauf der Kurve sind im Bereich von wenigen ms und daher vernachlässigbar. Der Grund dafür ist, dass an dieser Stelle die ausgezeichnete Performance des Suchindex zum Tragen kommt, da dieser selbst bei einer sehr großen Anzahl an indexierten Objekten ein schnelles Ergebnis liefert. Da immer genau 10 Objekte aus der Datenbank geladen und wiederhergestellt werden müssen, hat der Faktor bei diesem Test keinen Einfluss auf Geschwindigkeit.

Zusammenfassung

Sämtliche Laufzeittests der Suchfunktion wurden ebenfalls mit der zentralen eDAL-Infrastruktur über die Server-Client-Architektur durchgeführt. Die grundsätzlichen Ergebnisse unterscheiden sich im Allgemeinen jedoch nicht, sondern die Laufzeit ist lediglich minimal höher, da bei der Ausführung einer Suchanfragen durch einen Client die gefundenen Objekte zusätzlich serialisiert und über das Netzwerk vom Server übertragen werden müssen. Das generelle Verhalten ist aber identisch mit den bereits gezeigten Ergebnissen für die lokale Instanz der Infrastruktur. Daher wird an dieser Stelle auf eine grafische Darstellung und detaillierte Beschreibung verzichtet. Insgesamt hat sich gezeigt, dass die Suchfunktion der Referenzimplementierung gut funktioniert und die indexbasierte Suche auch mit zunehmendem Datenbestand skaliert und eine sehr gute Performance liefert. Zwar steigt die Laufzeit mit zunehmender Zahl an Ergebnissen, da die Infrastruktur mehr Zeit benötigt, die gefundenen Objekte und die dazugehörigen Metadaten aus der Datenbank wiederherzustellen, jedoch ist die eigentliche Suche der jeweiligen Objekte über den Index so schnell, dass die Gesamtlaufzeit selbst bei mehreren hundert Ergebnissen immer noch im Bereich von Millisekunden liegt.

6.3 Zugriffszahlen und Datenbestand

Mit der im Rahmen des Dissertationsprojektes entwickelten eDAL-Infrastruktur zur Verwaltung und Publikation von Forschungsdaten und dem integrierten Datenbegutachtungsprozess sowie der implementierten grafischen Anwendung zur Einreichung von Forschungsdaten wurde der Grundstein für die Etablierung einer ersten produktiven Instanz der Infrastruktur gelegt. So wurde Ende 2015 im Rahmen des DPPN-Projekts am IPK Gatersleben das Plant Genomic and Phenomic Research Data Repository (PGP) auf Basis der eDAL-Infrastruktur eingerichtet und publiziert (Arend et al., 2016a). Dafür wurde ein öffentlich zugänglicher Server bereitgestellt, auf dem eine Instanz der eDAL-Infrastruktur etabliert wurde. Außerdem werden auf dem Server direkt die Dateien des integrierten DBMS sowie die während der Laufzeit erstellten Dateien des Indexes für die Suchfunktionen abgelegt. Zur Abspeicherung der konkreten Forschungsdaten wird jedoch nicht das Dateisystem des Servers benutzt, da dieses nur einen sehr begrenzten Speicherplatz bietet. Die erfassten Daten werden auf dem hierarchischen Speichermanagementsystem (HSM) des Instituts abgelegt. Dieses garantiert durch die Kombination von schnellen Festplatten und Magnetbändern mit sehr großer Speicherkapazität eine sichere Archivierung der Daten. Jede Datei ist in mehrfacher Kopie abgelegt und kann so beim Ausfall einer Festplatte oder eines Magnetbandes wiederhergestellt werden. Außerdem kann das HSM einfach erweitert werden, um den zukünftigen Speicherbedarf zu decken.

Die eDAL-Infrastruktur besitzt keine Limitierung auf bestimmte Dateitypen bzw. Datendomänen. Das Hauptaugenmerk des PGP-Repository liegt auf der Veröffentlichung von genetischen und vor allem pflanzenphänotypischen Daten. Diese sind im Fokus des DPPN-Projektes und haben, wie eingangs erwähnt, in den letzten Jahren zunehmend an Bedeutung gewonnen. Jedoch gibt es noch keine verbreiteten und etablierten Datenbanken in dieser Domäne. Im Folgenden werden eine Reihe von Statistiken über Nutzer- und Zugriffszahlen sowie die Entwicklung des Datenbestands des PGP-Repository dargestellt.

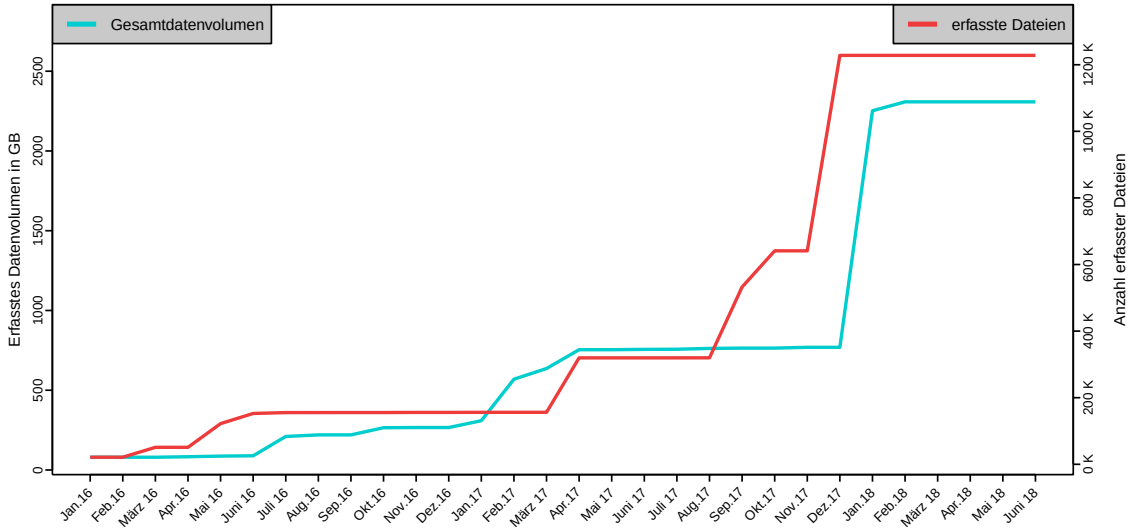

Abbildung 6.8: Datenbestands des PGP-Repository von Januar 2016 bis Juni 2018 aufgeteilt nach Gesamtvolumen der erfassten Daten und Anzahl an gespeicherten Dateien

Die Abbildung 6.8 auf Seite 113 zeigt die Entwicklung des Datenbestands aufgeteilt nach Datenvolumen und Anzahl an erfassten Dateien seit Januar 2016. Im Moment stellt das PGP-Repository über 2,2 Terabyte an Forschungsdaten, bestehend aus über 1.200.000 Dateien, öffentlich zur Verfügung. Diese sind Bestandteil von aktuell 150 Datensätzen, die mit einer DOI versehen und publiziert sind. Im Diagramm wird sichtbar, dass der Datenbestand nicht bei 0 beginnt. Das liegt daran, dass die Erfassung der Kennzahlen für die Datenzugriffe und des Datenbestands erst Anfang 2016 implementiert wurde und die vorherigen Werte nicht vollständig erfasst wurden. Außerdem ist das IPK Gatersleben bereits seit 2011 als Datenzentrum im DataCite-Konsortium registriert. Da es bis zu diesem Zeitpunkt noch keine geeignete Infrastruktur zur Vergabe und Verwaltung von DOIs gab, wurden bis 2015 nur etwa 20 DOIs manuell registriert und über URLs auf einem Webserver verwaltet. Diese DOIs wurden bei der Einrichtung des PGP-Repository direkt migriert und haben daher nicht den integrierten Datenbegutachtungsprozess durchlaufen. Mit dem PGP-Repository auf Basis der eDAL-Infrastruktur sowie der benutzerfreundlichen Anwendung zur Einreichung und Publikation von Forschungsdaten hat sich die Anzahl der veröffentlichten DOIs seitdem deutlich erhöht.

Das in Abbildung 6.9 gezeigte Diagramm stellt die Entwicklung der Zugriffszahlen und des Volumens an heruntergeladenen Daten des PGP-Repository seit Anfang 2016 dar. Auch diese Werte beginnen aus den bereits genannten Gründen nicht bei 0. Seit der Veröffentlichung der Publikation über das PGP-Repository (Arend et al., 2016a) wird eine konstant wachsende Zahl an Zugriffen und Downloads registriert. Gleichzeitig stieg auch die Menge des abgerufenen Datenvolumens parallel dazu stetig an. Kurzfristige, starke Anstiege der Kurven sind häufig mit der Veröffentlichung von wissenschaftlichen Publikationen, die Forschungsdatensätze aus dem PGP-Repository referenzieren, zu erklären. Wenn ein neuer Artikel veröffentlicht wird, steigt im folgenden Zeitraum die Anzahl an Zugriffen und Downloads besonders stark an, da die enthaltenen Datensätze von vielen Forschern abgerufen werden.

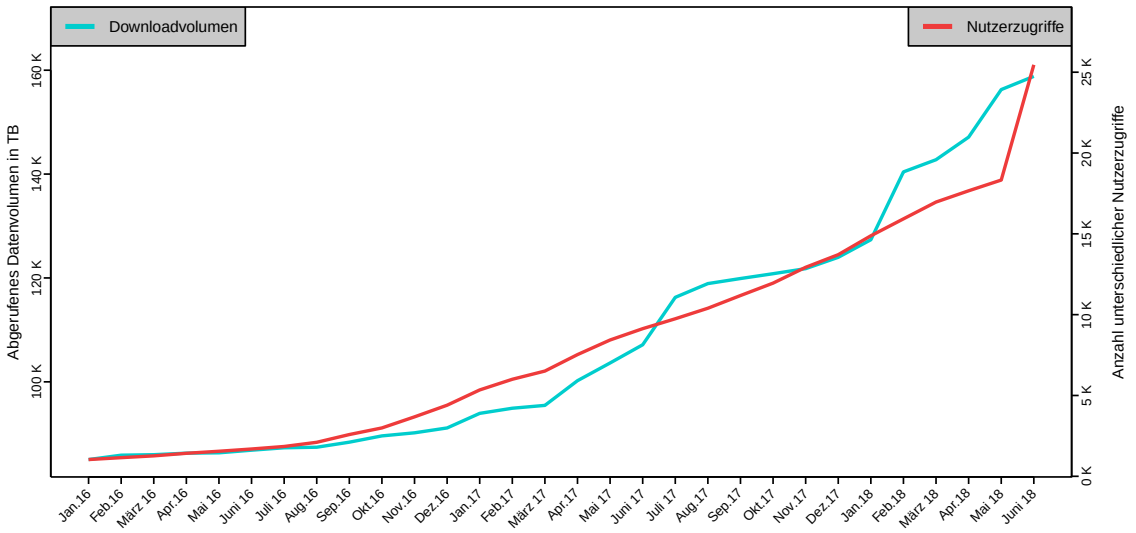

Abbildung 6.9: Übersicht der Anzahl an eindeutigen Zugriffen und des Volumens an heruntergeladenen Daten des PGP-Repository von Januar 2016 bis Juni 2018

Außerdem wurde das PGP-Repository nach dessen Veröffentlichung vom Nature Scientific Data Journal (Nature Publishing Group, 2018a) und vom GigaScience Journal (Oxford University Press, 2018) nach detaillierter Prüfung als ein empfohlenes Repository zur nachhaltigen Speicherung und Publikation von Forschungsdaten akzeptiert. Infolgedessen konnten unter anderem in beiden Zeitschriften zwei Publikationen, die erstmals umfangreiche pflanzenphänotypische Forschungsdatensätze enthielten, verfasst und veröffentlicht werden (Arend et al., 2016b; Chen et al., 2018). Die eDAL-Infrastruktur bzw. das PGP-Repository wurden zwar im Rahmen des DPPN-Projektes entwickelt, jedoch handelt es sich um eine generische Infrastruktur, welche die Möglichkeit bietet, alle Arten von Forschungsdaten zu erfassen, und zu publizieren. Deshalb wurden auch weitere Datensätze aus teilweise sehr unterschiedlichen Datendomänen mit PGP-Repository abgelegt. So wurden bisher bereits über 70 wissenschaftliche Publikationen veröffentlicht, die ihre verwendeten Forschungsdaten im PGP-Repository abgelegt haben oder daraus bezogen und referenziert haben. Dabei hat sich die Anzahl seit der Etablierung der eDAL-Infrastruktur am IPK Gatersleben und der Veröffentlichung der Anwendung zur Einreichung von Forschungsdaten für das PGP-Repository in den letzten Jahren stetig gesteigert (siehe Abbildung 6.10).

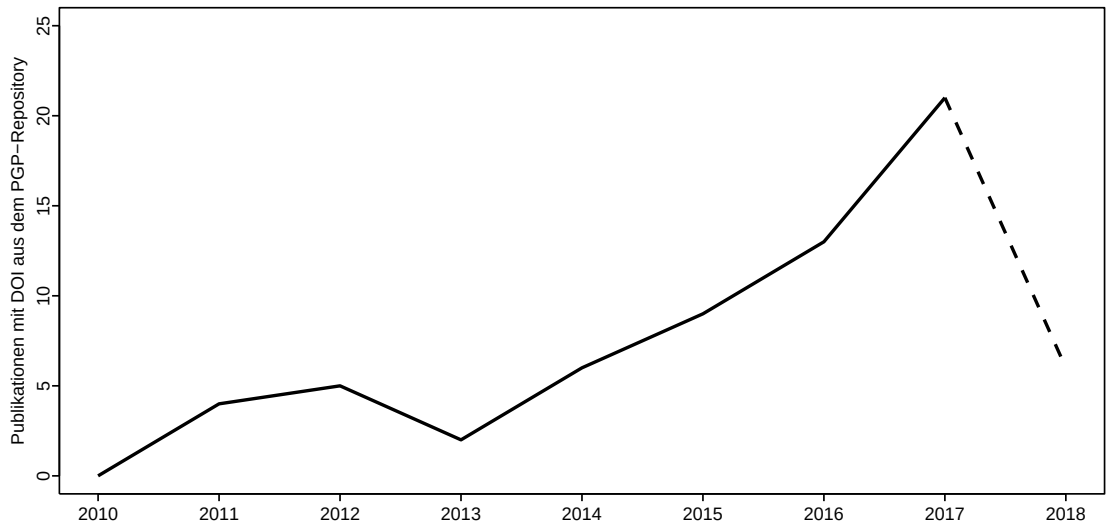

Abbildung 6.10: Anzahl an Publikationen, die einen Datensatz aus dem PGP-Repository mittels DOI referenzieren. Die Werte wurden mit der Dimensions-Suche extrahiert (Digital Science and Research Solutions Inc, 2018).

Darüber hinaus wurde das PGP-Repository (siehe Abbildung 6.11 auf Seite 116) in das Verzeichnis der re3data.org-Plattform für Forschungsdateninfrastrukturen (Pampel et al., 2013) und die kurierte FAIRSharing.org-Datenbank für Infrastrukturen und Systeme, die den FAIR-Prinzipien (siehe Abschnitt 2.2) folgen (McQuilton et al., 2016), aufgenommen. Auch die von der Europäischen Union finanzierte OpenAIRE-Infrastruktur (European Commission, 2018b) für die Vernetzung von frei zugänglichen Forschungsdaten und Plattformen hat das PGP-Repository als Forschungsdateninfrastruktur aufgenommen.

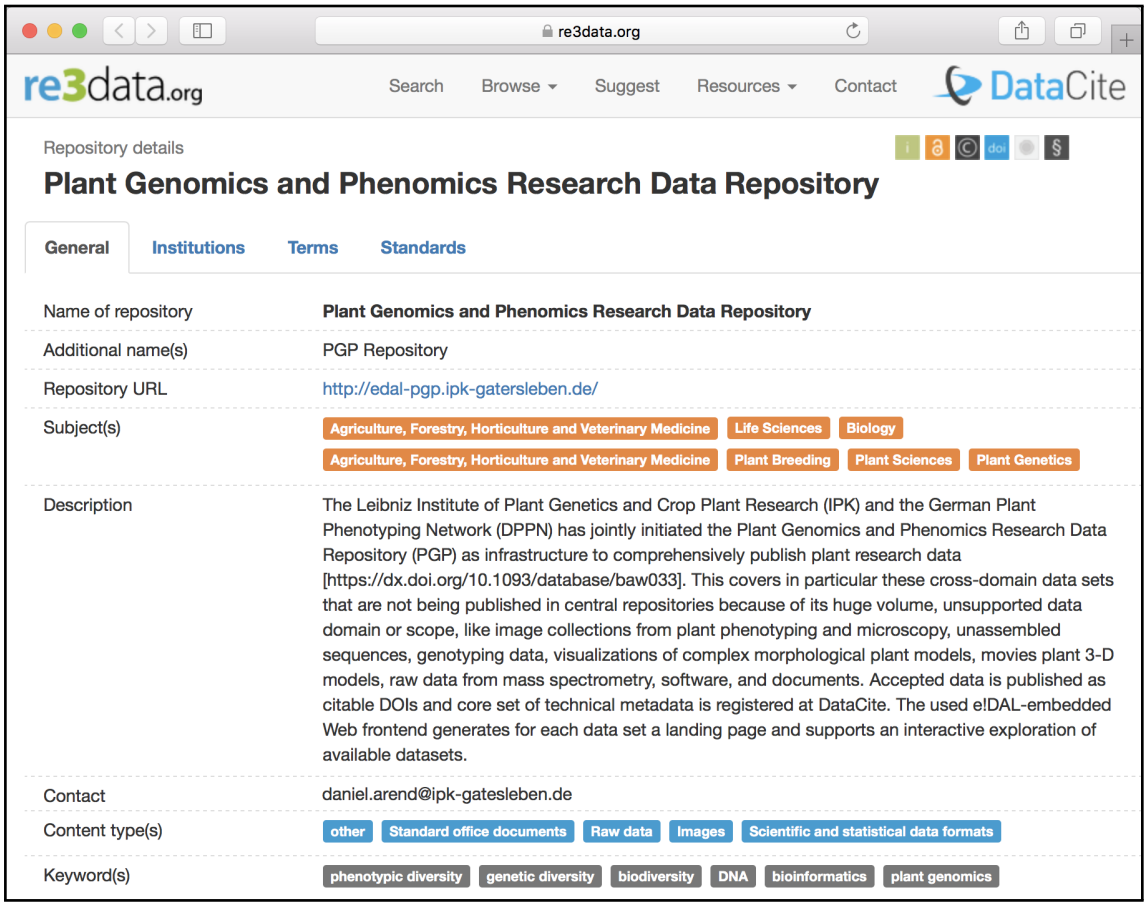

Abbildung 6.11: Screenshot der re3data.org-Webseite des Eintrags über das PGP-Repository (re3data, 2018)

7 Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse der vorliegenden Arbeit zusammengefasst. Dabei wird Bezug auf die am Anfang beschriebenen Aufgaben genommen und die erwähnten Forschungsfragen beantwortet. Außerdem wird im Folgenden ein Ausblick auf zukünftige und weiterführende Aufgaben und Herausforderungen gegeben.

7.1 Ergebnisse

Die im Rahmen der Dissertation entwickelte eDAL-Infrastruktur ist eine generische Forschungsdatenmanagement- und Publikationsinfrastruktur. Sie eignet sich für alle Arten von Forschungsdaten und hat keine Beschränkungen in Bezug auf Datentypen oder Volumen. Mit dem PGP-Repository konnte die erste produktive Instanz der Infrastruktur zur Publikation von pflanzenphänotypischen und -genotypischen Daten veröffentlicht werden. Bei der Realisierung des Projektes konnten die anfangs erwähnten Forschungsfragen beantwortet werden:

1. Wie werden Forschungsdaten üblicherweise verwaltet und publiziert?
Der allgemeine Verwaltungs- und Publikationsprozess von Forschungsdaten ist sehr komplex und es gibt in jeder Domäne oft mehrere unterschiedliche Ansätze zum Umgang mit Forschungsdaten. Daher wurden eine Reihe von verschiedenen Datenbanken, Archiven, Informationssystemen und Softwareinfrastrukturen untersucht und in Bezug auf die sich ergebenden Anforderungen während dieses Prozesses miteinander verglichen. Es zeigte sich, dass viele Systeme erhebliche Schwächen haben und sich nicht als generische Lösung zur Publikation von Forschungsdaten eignen. Einige Plattformen besitzen zwar einen hohen Funktionsumfang, bieten jedoch nur begrenzten Speicherplatz (z. B. figshare oder Dryad) oder benötigen ein hohes technisches Verständnis und zusätzlichen Aufwand zur Integration in existierende Prozesse (z. B. Fedora oder Dspace). Außerdem liefert keines der untersuchten Systeme einen Ansatz zur Prüfung und Begutachtung der Qualität der veröffentlichten Daten.

2. Welche Ansprüche stellen Forschungsdaten an die Infrastruktur?
Die Nachhaltigkeit und Reproduzierbarkeit von Forschungsdaten zu gewährleisten ist wegen der vielen verschiedenen und komplexen Datentypen eine Herausforderung. Daher ist die Beschreibung mit standardisierten und technischen Metadaten über den kompletten Verarbeitungszeitraum von Forschungsdaten sowie die Vergabe dauerhaft referenzierbarer IDs notwendig. Nur so können diese langfristig archiviert und publiziert werden. Aufgrund des stetig wachsenden Volumens an Forschungsdaten ist eine hohe Performance besonders für das Erfassen und Abrufen von Objekten wichtig. Außerdem bestehen viele Datensätze z. B. Phänotypisierungsdaten häufig aus einer Vielzahl an Dateien, sodass eine hohe Skalierbarkeit für die Infrastruktur notwendig ist. Diese beiden Faktoren sind neben dem Schutz der

Daten vor Missbrauch die entscheidenden Aspekte für die Akzeptanz der Nutzer sowie eine produktive und effiziente Integration in den täglichen Forschungsprozess der Wissenschaftler.

3. Was sind die Anforderungen an einen generischen Prozess für die Begutachtung von Forschungsdaten?
Da alle untersuchten Systeme zum Forschungsdatenmanagement keinen Ansatz zur Bewertung der Qualität ihrer Daten besitzen und es auch allgemein keinen etablierten, generellen Prozess zur Datenbegutachtung gibt, wurde im Laufe des Dissertationsprojektes ein Konzept zur Begutachtung von Forschungsdaten vor deren Publikation entwickelt. Dieses orientiert sich am Peer-Review-Konzept, dass auch für die Begutachtung von wissenschaftlichen Artikeln in Zeitschriften und Konferenzbänden angewendet wird, und in nahezu allen Forschungsdomänen etabliert ist. Einerseits gewährleistet es die Prüfung der wissenschaftlichen Qualität, Korrektheit und Einhaltung von gängigen Standards, andererseits stellt es durch eine administrative Prüfung sicher, dass alle rechtlichen Vorgaben eingehalten werden. Die intuitive Nutzung und die flexible Konfiguration ermöglichen eine einfache Integration in existierende Infrastrukturen und Abläufe.

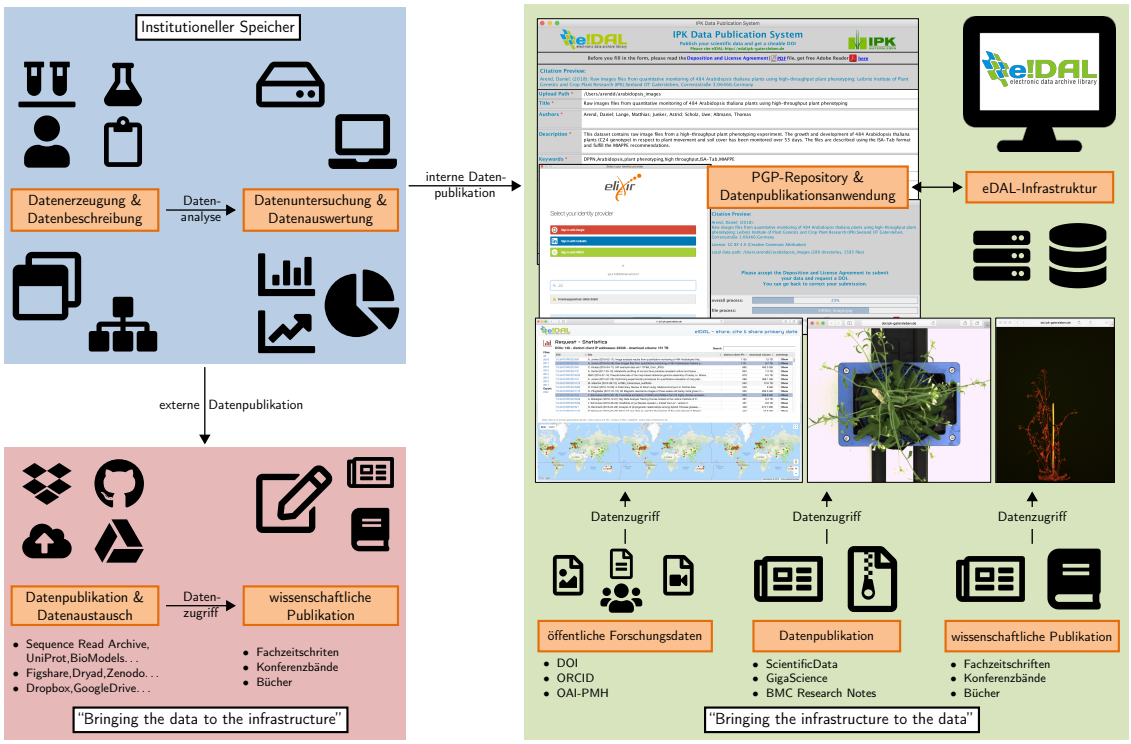

Abbildung 7.1: Schematische Darstellung des „Bringing the infrastructure to the data"-Konzepts. Alle verwendeten Symbolgrafiken sind der FontAwesome-Bibliothek entnommen (Fonticons Inc., 2018).

Der große Unterschied der eDAL-Infrastruktur im Vergleich zu vielen gebräuchlichen Systemen zum Forschungsdatenmanagement ist das unterschiedliche Grundkonzept. Üblicherweise werden Forschungsdaten nach der Verarbeitung, Analyse und dem Durchlaufen verschiedener Domänen in externe Datenbanken und Archiven oder Publikations- bzw. Cloud-Speicherdienste übertragen. Anschließend werden sie dann häufig in reduzierter Form als Teil einer wissenschaftlichen Publikation veröffentlicht. Abgesehen davon, dass erwähnte

Archive und Infrastrukturen funktionelle Schwächen haben, ist die Übertragung zeitaufwendig und ermöglicht keine Nutzung vorhandener Speicherkapazitäten. Die eDAL-Infrastruktur verfolgt hingegen einen „Bringing the infrastructrue to the data"-Ansatz (deutsch: „bring die Infrastruktur zu den Daten"), der auch schematisch in Abbildung 7.1 auf Seite 118 dargestellt ist. Dieses Konzept ermöglicht es, lokale, interne Speicherkapazitäten zu nutzen, um eine Infrastruktur zu etablieren, welche vorhandene Forschungsdaten effizient verwalten und publizieren kann. Freie Repository-Systeme, wie z. B. die vorgestellten Dspace oder FEDORA bieten zwar ebenfalls die Möglichkeit, institutionelle Informationssysteme zu entwickeln, benötigen jedoch einen hohen Aufwand für die Anpassung bzw. Integration und bieten keine fertige Software an. Die eDAL-Infrastruktur bietet eine umfangreiche und lauffähige Lösung zur Verwaltung und Publikation von Forschungsdaten, die in der Lage ist, die Schwächen existierender Systeme zu überwinden. Zum Betreiben eines Repository auf Basis der eDAL-Infrastruktur sind lediglich ein öffentlich zugänglicher Server mit ausreichender Speicherkapazität und ein DataCite-Zugang, der durch die Registrierung als Datenzentrum vergeben wird, notwendig.

Es konnte gezeigt werden, dass durch die Nutzung verschiedener informatischer Konzepte, moderner Technologien und spezialisierter Frameworks eine Infrastruktur implementiert wurde, die in der Lage ist, auch sehr umfangreiche Datensätze zu verwalten und daher auch für die Publikation von phänotypischen Daten genutzt werden kann. Darüber hinaus ist sie jedoch generisch und eignet sich für alle Arten von Forschungsdaten. Die hohe Skalierbarkeit gewährleistet eine langfristige und stabile Nutzung auch bei steigendem Datenvolumen. Da die Infrastruktur bereits eine vollständige Referenzimplementierung liefert, die ohne zusätzliche Abhängigkeiten lauffähig ist und eine Reihe von Funktionen zur automatischen Konfiguration verschiedener Parameter wie z. B. Netzwerk- und Proxy-Einstellungen bietet, werden technische Hürden reduziert und eine einfache Integration in existierende Abläufe und Infrastrukturen ermöglicht. Die Verwendung von DOIs als dauerhafte Identifikatoren bietet sowohl für die wissenschaftliche Gemeinschaft als auch für die Datenproduzenten viele Vorteile. Die Annotation mit standardisierten Metadaten und die stabile Referenzierbarkeit ermöglichen allen Forschern die Wiederverwendung und Reproduzierbarkeit der Daten. Dadurch ist eine offene und transparente Forschung gewährleistet. Für die Erzeuger der Daten ergibt sich dabei eine zusätzliche Möglichkeit, Anerkennung für ihr Arbeit durch die Bereitstellung ihrer Daten zu erhalten. Gleichzeitig erhöht die Vernetzung der DataCite Infrastruktur und die Integration in verschiedene Plattformen und Netzwerke die Sichtbarkeit der Daten und der Forschung der Autoren erheblich. Darüber hinaus bieten DOIs vielfältige Möglichkeiten, um die damit assoziierten Forschungsdaten zu publizieren. Neben der üblichen Veröffentlichung als Anhang in einer wissenschaftlichen Publikation können sie auch als eigenständige Datenpublikation in drauf spezialisierten Fachzeitschriften veröffentlicht werden. Dies wurde auch, wie gezeigt, für eine Reihe von Datensätzen, die im PGP-Repository erfasst wurden, getan.

Durch die Erfassung umfangreicher Nutzungsstatistiken konnte außerdem gezeigt werden, dass die Anzahl an veröffentlichten Forschungsdaten im PGP-Repository sowie die Anzahl an Zugriffen und Downloads dieser Datensätze kontinuierlich steigt, was den Bedarf der wissenschaftlichen Gemeinschaft nach qualitativen Forschungsdaten widerspiegelt. Der intuitive Prozess zur Begutachtung der zu veröffentlichenden Daten, die benutzerfreundliche Oberfläche der Anwendung zur Einreichung und Publikation von Forschungsdaten und die schnelle und einfache Bewertung durch die Gutachter sorgen für eine hohe Ak-

zeptanz bei den Wissenschaftlern. Mit der Integration der ELIXIR-AAI als universelles Authentifizierungsverfahren wurde die Voraussetzung geschaffen, um weitere Repository auf Basis der eDAL-Infrastruktur, z. B. durch Partnerinstitute, die mit ELIXIR assoziiert sind, einzurichten. So konnte Mitte 2018 im Rahmen des DPPN-Projektes eine weitere Instanz am Forschungszentrum Jülich (FZJ) etabliert werden. Das FZJ ist einer der Kooperationspartner des Projektes. Abbildung 7.2 zeigt die Projektwebseite des Repository. Da die im Rahmen der Dissertation entwickelte Referenzimplementierung der eDAL-Infrastruktur bereits alle notwendigen Komponenten enthält, musste lediglich ein öffentlich zugänglicher Server eingerichtet werden. Aufgrund der einfachen Installation und automatischen Konfiguration, sowie der Unterstützung der ELIXIR-AAI konnte die Integration am FZJ ohne zusätzlichen Programmieraufwand in sehr kurzer Zeit realisiert werden.

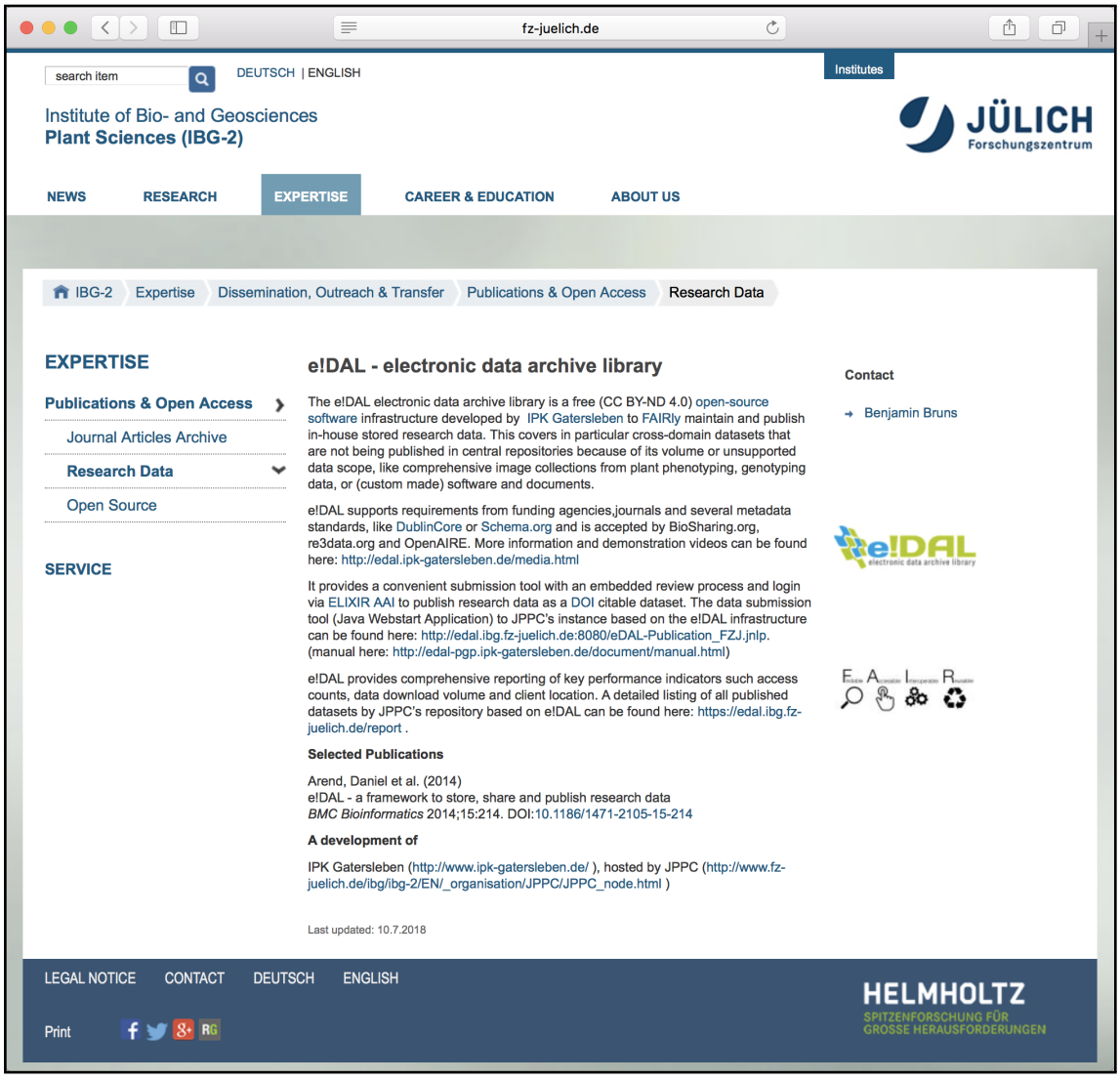

Abbildung 7.2: Screenshot der Projekt-Webseite der Instanz der eDAL-Infrastruktur am Forschungszentrum Jülich

7.2 Nachhaltigkeit

Die Thematik der Nachhaltigkeit war ein wichtiger Bestandteil bei der Entwicklung der eDAL-Infrastruktur im Laufe des Dissertationsprojektes. Einerseits gab es den Aspekt der langfristigen Archivierung und Publikation von Forschungsdaten durch eine geeignete Datenstruktur und die Vergabe von persistente und referenzierbaren DOIs. Andererseits stand eine nachhaltige Softwareentwicklung und Dokumentation im Fokus. Darüber hinaus stellt sich aber auch die Frage wie die Infrastruktur bzw. die daraus etablierten Repositories dauerhaft pflegt werden können.

Für die Nutzung der DataCite-Services zur Vergabe und Auflösung der DOIs die mit der eDAL-Infrastruktur erzeugt und verwaltet werden ist ein DataCite-Account notwendig. Dieser wird über die Registrierung als Datenzentrum vergeben. Dabei wird vertraglich geregelt, dass das Datenzentrum, welches die DOIs registriert, sich dazu verpflichtet die notwendigen Inhaltsseiten zum Abrufen der Datensätze, die mit einer DOI verlinkt wurden, bereitzustellen und zu verwalten. Gleichzeitig akzeptiert das Datenzentrum damit die Verantwortung die Datensätze langfristig und nach den Regeln guter wissenschaftlicher Praxis in der betreffenden Datendomäne zur Verfügung zustellen. Somit übernimmt im Fall des PGP-Repository das IPK Gatersleben die Verantwortung für die veröffentlichten Datensätze und stellt eine langfristige Verfügbarkeit der DOIs bzw. der zugrunde liegenden Infrastruktur sicher. Es gibt eine Reihe von Leitlinien zur Einhaltung guter wissenschaftlicher Praxis, unter anderem von verschiedenen Forschungsgemeinschaften (MPG, 2009; WGL, 2015) oder Universitäten (OVGU, 2018; MLU, 2009; FSU, 2006). Im Allgemeinen orientieren diese sich diese jedoch stark an den weitverbreiteten Richtlinien der DFG (DFG, 2013). Diese sehen vor, dass Forschungsdaten mindestens zehn Jahre aufbewahrt werden.

Darüber hinaus wurde das PGP-Repository in das Service-Portfolio des GCBN-Verbunds aufgenommen (vgl. Bolger et al., 2017). Dies ist einer von acht Serviceverbänden im de.NBI-Netzwerk (Tauch und Al-Dilaimi, 2017). Ziel des GCBN-Verbunds ist unter anderem die Bereitstellung wichtiger pflanzengenetischer Ressourcen und phänotypischer Informationen. Diese sollen in entwickelten Infrastrukturen und Informationssysteme integriert und öffentlich bereitgestellt werden (vgl. Schmutzer et al., 2017). Des Weiteren ist Deutschland seit 2016, vertreten durch de.NBI, Mitglied der internationalen ELIXIR-Infrastruktur (ELIXIR, 2018) mit dem Ziel die Bereitstellung wichtiger biologischer Kernressourcen und Infrastrukturen dauerhaft zu gewährleisten (vgl. Durinx et al., 2017; Tauch und Al-Dilaimi, 2017).

7.3 Zukünftige Aufgaben

In Abschnitt 5.2 wurden das Konzept und die Entwicklung der Anwendung zur Einreichung und Publikation von Forschungsdaten beschrieben. Da zu Beginn der Entwicklung die Java Swing-API die Standardschnittstelle zur Implementierung grafischer Oberflächen in Java war, wurde diese für die Programmierung genutzt. So konnte das Dissertationsprojekt erfolgreich realisiert werden. In den letzten Jahren bzw. im Laufe der produktiven Nutzung der Anwendung zur Übertragung in das PGP-Repository zeigten sich jedoch auch einige Schwachstellen, wie beispielsweise die bereits erwähnte fehlende Unterstützung für die Verarbeitung und Kommunikation mit HTML-basierten Inhalten. Daher wurde auch für

die Entwicklung der OAuth-basierten Authentifizierungsmodule auf die neuere JavaFX-API zurückgegriffen. Da die JavaFX-API die Swing-API mittlerweile als Standard zur Entwicklung grafischer Oberflächen ablöst, ist es sinnvoll, in Zukunft auch die komplette Anwendung mit JavaFX zu realisieren, um auch hier den aktuellen Technologiestandard zu verwenden. Leider ist auch die JavaFX-API nicht frei von Schwächen, so ist die API nicht Bestandteil jeder verfügbaren Java Laufzeitumgebung. Daher muss der Anwender darauf achten, eine entsprechende Java Version installiert zu haben, da ansonsten bestimmte Komponenten nicht geladen werden können.

Die Bereitstellung der Anwendung als Java Webstart-Anwendung ist eine gängige Variante zur Ausführung und Verteilung von Java-basierter Software. Sie bietet den Vorteil, dass der Nutzer die eigentliche Anwendung nicht installieren muss oder irgendwelche Komponenten runter laden muss. Diese wird von Java selbst übernommen und die Anwendung kann direkt ausgeführt werden. Es ist aber immer noch notwendig, dass eine aktuelle Java Laufzeitumgebung auf dem Rechner der Anwender installiert ist. Eine vielversprechende Alternative wäre die komplette Neuentwicklung der Anwendung zum Einreichen von Forschungsdaten als webbasierte Anwendung. Dies würde gleich mehrere Vorteile bieten. Der Nutzer könnte die Anwendung direkt über einen Webbrowser starten und ausführen. Damit wären die erwähnten Schwierigkeiten mit JavaFX gelöst und der Nutzer muss keinerlei Abhängigkeiten erfüllen. Allerdings wären diese Anpassungen sehr umfangreich und hätten sehr wahrscheinlich auch tiefer liegende Änderungen in der eDAL-Infrastruktur zur Folge. Eventuell wäre eine zusätzliche Implementierung der Server-Client-Architektur über eine REST-Schnittstelle angebracht, um eine vollständig webbasierte Anwendung zu entwickeln, was aber ebenfalls sehr aufwendig ist. Dafür wäre eine REST-Schnittstelle flexibler und bietet mehr Potenzial für weitere Entwicklungen als die Java RMI-API.

Die Verwendung von Java für die Implementierung der entwickelten Forschungsdateninfrastruktur hat sich während des Dissertationsprojekts bewährt. Die Plattformunabhängigkeit sowie die flexiblen und einfachen Erweiterungsmöglichkeiten durch externe Bibliotheken ermöglichte es, schnell verschiedene neue Komponenten und Ideen zu testen, um so die optimale Implementierung zu realisieren. Die aktuelle Version der eDAL-Infrastruktur ist mit Java 8 realisiert. Gegen Ende des Dissertationsprojektes wurde die neue Java 9 Entwicklungsumgebung veröffentlicht. Aufgrund der bis dahin noch sehr geringen Verbreitung wurde bisher auf eine Aktualisierung verzichtet. Mit zunehmender Verbreitung der neuen Java Version ist ein Umstieg nicht nur aufgrund sicherheitstechnischer Gründe angebracht, sondern bietet darüber hinaus nach genauer Prüfung eventuell neue Möglichkeiten, um die Implementierung der eDAL-API noch weiter zu verbessern. So könnte die mit Java 9 neu eingeführte File-Streaming-API einige Performancevorteile im Umgang mit Dateisystemen bieten und so die Geschwindigkeit der Infrastruktur noch weiter verbessern.

In Abschnitt 6.2 wurde gezeigt, dass die entwickelte eDAL-Infrastruktur in der Lage ist, auch sehr umfangreiche Datensätze zu verarbeiten und gut skaliert. Trotzdem wird das Volumen an täglich produzierten Daten im Zuge der fortschreitenden technischen Entwicklung weiter anwachsen und so auch der Umfang an Forschungsdaten, die langfristig archiviert und publiziert werden, weiter steigen. Daher wird es notwendig sein, sich darüber Gedanken zu machen, wie die Performance und Leistungsfähigkeit der Infrastruktur in Zukunft noch weiter verbessert werden kann. Ein möglicher Ansatz wäre die Etablierung eines verteilten Systems auf Basis des Konzeptes der eDAL-Infrastruktur. Das wahrscheinliche bekannteste Framework in diesem Bereich ist Hadoop. Dessen bereitgestelltes Hadoop Distributed

File System (HDFS) ermöglicht es, große Datenmenge auf mehrere entfernte Rechner zu verteilen. Es ist in der Lage, auch mit mehreren hundert Millionen Dateien umzugehen (vgl. Shvachko et al., 2010). Dadurch könnte die Geschwindigkeit und Skalierbarkeit der eDAL-Infrastruktur nochmals deutlich erhöht werden.

A Anhang

A.1 UML-Klassendiagramm

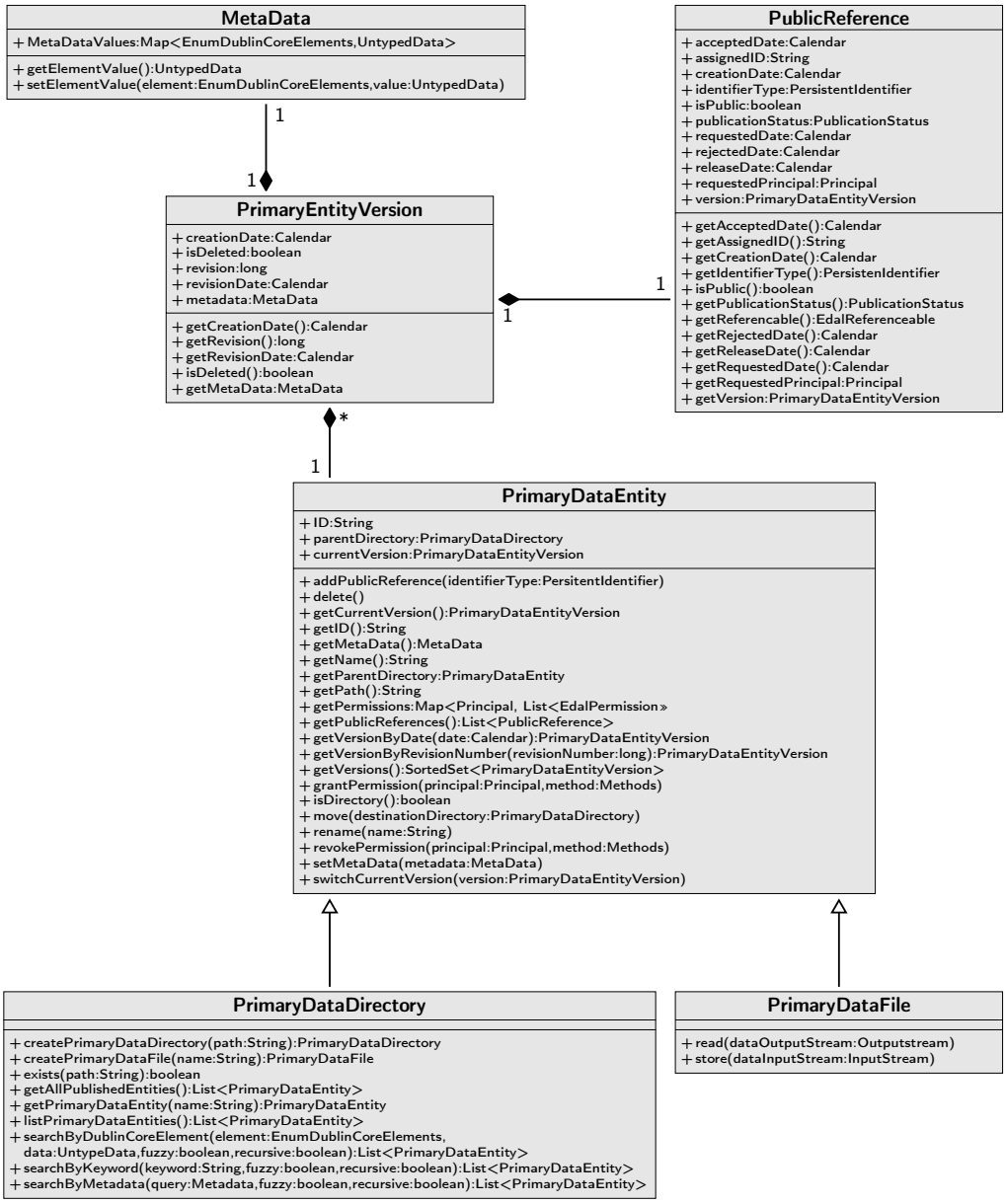

Abbildung A.1: UML-Klassendiagramm der eDAL-Datenstruktur

A.2 Aspect

```
public aspect PublicPermissionCheck {

 pointcut checkPublicMethods():
   execution(public * PrimaryDataEntity+.* (..))
   && !execution(public * PrimaryDataEntity+.getName(..))
   ...
   && !execution(public * PrimaryDataEntity+.toString(..));

  Object around() throws AccessControlException: checkPublicMethods(){
    checkPermission(thisJoinPoint);
    return proceed();
 }

  public void checkPermission(JoinPoint joinPoint) throws AccessControlException {

    Subject subject = DataManager.getSubject();
    ImplementationProvider ip = DataManager.getImplProv();
    PermissionProvider permissionProvider = ip.getPermissionProvider().newInstance();

    if (!permissionProvider.isRoot(subject)) {

       String uuid = useGetMethod(joinPoint,"getID");
       PrimaryDataEntityVersion version = useGetMethod(joinPoint,"getCurrentVersion");
       ...
       EdalPermission permission = new EdalPermission(uuid,version,actionClass,method);

       boolean checkedPermission = checkPerm(subject,permission);

       if(!checkedPermission){
         throw AccessControllException(...);
       }
    }
  }

 /** Method to call Getter methods from the Class of the JoinPoint */

 private Object useGetMethod(JoinPoint joinPoint, String methodName) {

   Method method = joinPoint.getThis().getMethod(methodName, new Class<?>[0]);
   Object object = method.invoke(joinPoint.getThis(),new Object[0]);
   return object;
 }

 /** Check if the permission for the {@link Subject} is set */

 private boolean checkPerm(Subject subject, EdalPermission edalPermission) {
    try {
      Subject.doAs(subject, new PrivilegedAction<Object>() {
        public Object run() {
          SecurityManager sm = System.getSecurityManager();
          sm.checkPermission(edalPermission);
        }
      });
    } catch (SecurityException e) { return false; }
    return true;
 }
}
```

Quelltext A.1: Aspekt zur Überprüfung der eDAL-Nutzerrechte

A.3 Datenbankschema

A.3.1 Datenstrukutur

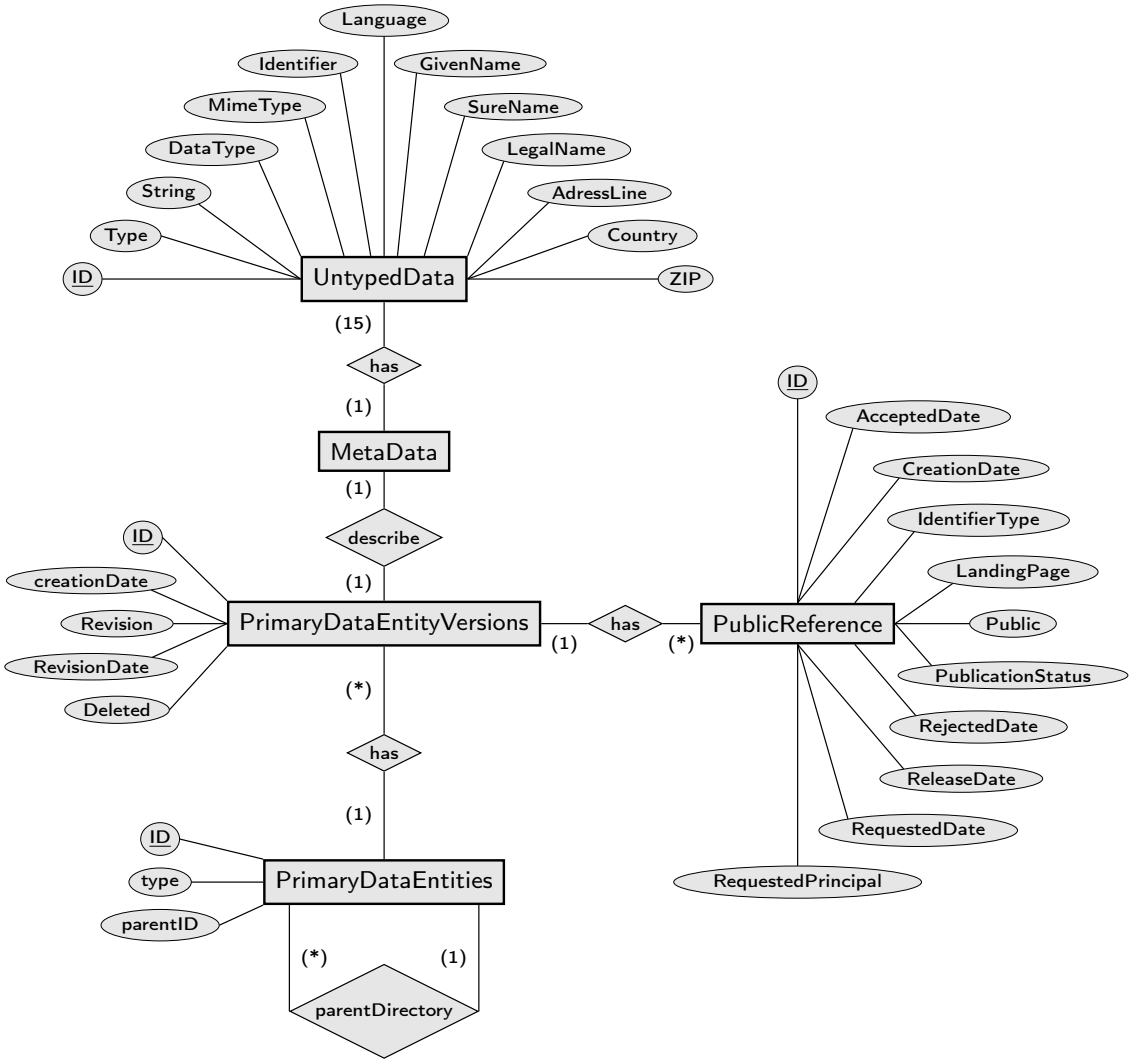

Abbildung A.2: Datenbankschema der eDAL-Datenstruktur

A.3.2 Sicherheitssystem

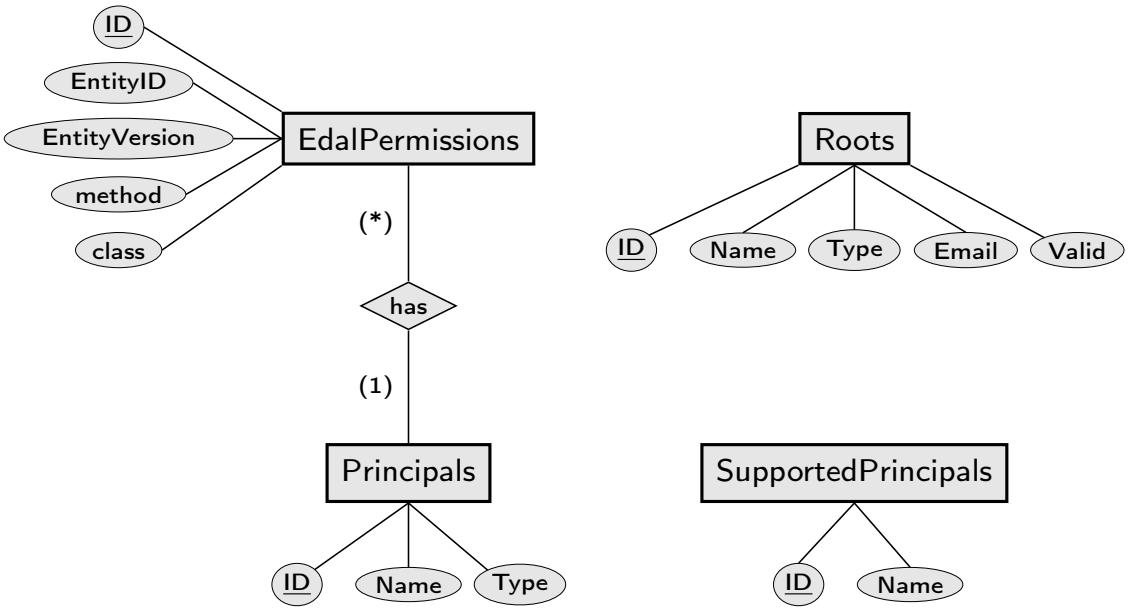

Abbildung A.3: Datenbankschema der Komponenten des eDAL-Sicherheitssystems

A.3.3 Begutachtungsprozess

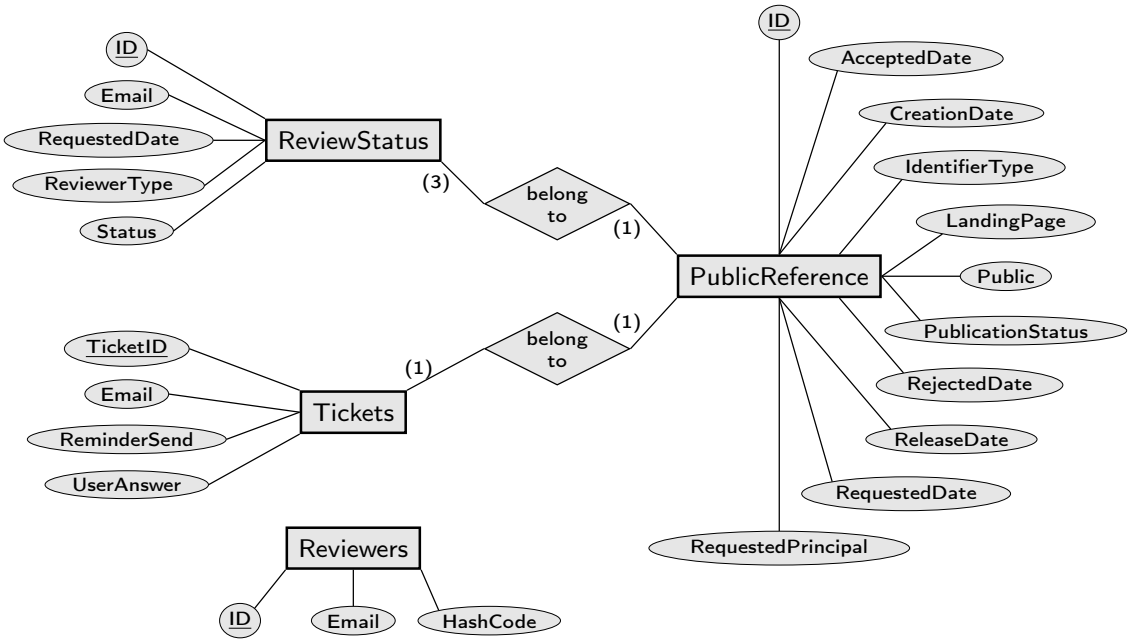

Abbildung A.4: Datenbankschema der Komponenten des eDAL-Begutachtungsprozesses

Literaturverzeichnis

Adams J, Nudurupati S, Gasparini N, Hobley D, Hutton E, Tucker G und Istanbulluoglu E. Landlab: Sustainable Software Development in Practice. In *The Second Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2), New Orleans, LA, USA*, volume 16, 2014, DOI: 10.6084/m9.figshare.1097629.v6.

Ahmed Z, Zeeshan S und Dandekar T. Developing sustainable software solutions for bioinformatics by the "Butterfly" paradigm. *F1000Research*, 3, 2014, DOI: 10.12688/f1000research.3681.2.

Alsheikh-Ali A. A, Qureshi W, Al-Mallah M. H und Ioannidis J. P. A. Public Availability of Published Research Data in High-Impact Journals. *PLOS ONE*, 6(9):1–4, 09 2011, DOI: 10.1371/journal.pone.0024357.

Anderson N, Tarczy-Hornoch P und Bumgarner R. On the persistence of supplementary resources in biomedical publications. *BMC Bioinformatics*, 7(1):260, 2006, DOI: 10.1186/1471-2105-7-260.

Apache Software Foundation. Apache Maven, 2018a. URL: `https://maven.apache.org/`. [Online; Stand Januar 2018].

Apache Software Foundation. Subversion, 2018b. URL: `https://subversion.apache.org/`. [Online; Stand Januar 2018].

Apweiler R, Bairoch A, Wu C. H, Barker W. C, Boeckmann B, Ferro S, Gasteiger E, Huang H, Lopez R, Magrane M, Martin M. J, Natale D. A, O'Donovan C, Redaschi N und Yeh L. L. UniProt: the Universal Protein knowledgebase. *Nucleic Acids Research*, 32(suppl 1):D115–D119, 2004, DOI: 10.1093/nar/gkh131.

Arend D. Konzeption und Implementierung einer Datenhaltungsinfrastruktur zur digitalen Langzeitarchivierung und dauerhaften Zitierbarkeit biologischer Primärdaten am Beispiel von "Next-Generation-Sequenzing"-Daten. Diplomarbeit, Martin-Luther-Universität Halle-Wittenberg, 2012.

Arend D, Lange M, Colmsee C, Flemming S, Chen J und Scholz U. The e!DAL JAVA-API: Store, share and cite primary data in life sciences. In *3rd International German/Russian Summer School on Integrative Biological Pathway Analysis and Simulation (IB-PAS), 18-21 June 2012, Bielefeld, Germany*, 2012a.

Arend D, Lange M, Colmsee C, Flemming S, Chen J und Scholz U. The e!DAL JAVA-API: Store, Share and Cite Primary Data in Life Sciences. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 4-7 October 2012, Philadelphia, USA*, pages 511–515, 2012b, DOI: 10.1109/BIBM.2012.6392737.

Arend D, Chen J, Hartmann A, Czauderna T, Scholz U, Schreiber F und Lange M. Sharing, Versioning and Annotating SBML Models using the e!DAL Data Repository API. In *4th Computational Modeling in Biology Network (COMBINE) Meeting, 16-20 September 2013, Paris, France*, 2013a, DOI: 10.4056/sigs.5279417.

Arend D, Lange M, Colmsee C, Flemming S, Chen J und Scholz U. The e!DAL JAVA-API: Store, Share and Cite Primary Data in Life Sciences. In *9th International Symposium on Integrative Bioinformatics (IB), 18-20 March 2013, Gatersleben, Germany*, 2013b.

Arend D, Colmsee C, Knüpffer H, Oppermann M, Scholz U, Schüler D, Weise S und Lange M. Data Management Experiences and Best Practices from the Perspective of a Plant Research Institute. In *10th International Conference, Data Integration in the Life Sciences (DILS), 17-18 July 2014, Lisbon, Portugal*, volume 8574, pages 41–49. Lecture Notes in Computer Science, 2014a, DOI: 10.1007/978-3-319-08590-6_4.

Arend D, Lange M, Chen J, Colmsee C, Flemming S, Hecht D und Scholz U. e!DAL - a framework to store, share and publish research data. *BMC Bioinformatics*, 15(1), 2014b, DOI: 10.1186/1471-2105-15-214.

Arend D, Junker A, Scholz U, Schüler D, Wylie J und Lange M. PGP repository: a plant phenomics and genomics data publication infrastructure. *Database*, 2016, 2016a, DOI: 10.1093/database/baw033.

Arend D, Lange M, Pape J.-M, Weigelt-Fischer K, Arana-Ceballos F, Mücke I, Klukas C, Altmann T, Scholz U und Junker A. Quantitative monitoring of Arabidopsis thaliana growth and development using high-throughput plant phenotyping. *Scientific Data*, 3 (160055), 2016b, DOI: 10.1038/sdata.2016.55.

Assante M, Candela L, Castelli D und Tani A. Are scientific data repositories coping with research data publishing? *Data Science Journal*, 15, 2016, DOI: 10.5334/dsj-2016-006.

Atlassian. Bitbucket, 2018. URL: `https://bitbucket.org/`. [Online; Stand Januar 2018].

Baker M. 1,500 scientists lift the lid on reproducibility. *Nature News*, 533(7604):452, 2016, DOI: 10.1038/533452a.

Balaji S und Murugaiyan M. S. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2 (1):26–30, 2012. ISSN: 2304-0777.

Ball C. A, Brazma A, Causton H, Chervitz S, Edgar R, Hingamp P, Matese J. C, Parkinson H, Quackenbush J, Ringwald M, Sansone S.-A, Sherlock G, Spellman P, Stoeckert C, Tateno Y, Taylor R, White J und Winegarden N. Submission of Microarray Data to Public Repositories. *PLOS Biology*, 2(9), 08 2004, DOI: 10.1371/journal.pbio.0020317.

Bardi A und Manghi P. Enhanced publications: Data models and information systems. *Liber Quarterly*, 23(4):240–273, 2014, DOI: 10.18352/lq.8445.

Barrett T, Wilhite S. E, Ledoux P, Evangelista C, Kim I. F, Tomashevsky M, Marshall K. A, Phillippy K. H, Sherman P. M, Holko M, Yefanov A, Lee H, Zhang N, Robertson C. L, Serova N, Davis S und Soboleva A. NCBI GEO: archive for functional genomics data sets—update. *Nucleic Acids Research*, 41(D1):D991–D995, 2013, DOI: 10.1093/nar/gks1193.

Bastow R und Leonelli S. Sustainable digital infrastructure. *EMBO reports*, 11(10):730–734, 2010, DOI: 10.1038/embor.2010.145.

Bauer C, King G und Gregory G. *Java Persistance with Hibernate*. Manning, 2015. ISBN: 978-1617290459.

Beck K. *Test-driven development: by example*. Addison-Wesley Professional, 2003. ISBN: 978-0321146533.

Begley C. G und Ellis L. M. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, 2012, DOI: 10.1038/483531a.

Bell G, Hey T und Szalay A. Beyond the data deluge. *Science*, 323(5919):1297–1298, 2009, DOI: 10.1126/science.1170411.

BMBF. Checkliste zur Erstellung eines Datenmanagementplans in der empirischen Bildungsforschung, 2018. URL: `http://wiki.bildungsserver.de/bilder/upload/checkliste_datenmanagement.pdf`. [Online; Stand Februar 2018].

Boehm B. W. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981.

Boehm B. W. Verifying and validating software requirements and design specifications. *IEEE software*, 1(1):75, 1984.

Boehm B. W. A spiral model of software development and enhancement. *Computer*, 21(5): 61–72, 1988, DOI: 10.1109/2.59.

Bolger M, Schwacke R, Gundlach H, Schmutzer T, Chen J, Arend D, Oppermann M, Weise S, Lange M, Fiorani F, Spannagl M, Scholz U, Mayer K und Usadel B. From plant genomes to phenotypes. *Journal of Biotechnology*, 261:46 – 52, 2017, DOI: 10.1016/j.jbiotec.2017.06.003.

Borgman C. L. The conundrum of sharing research data. *Journal of the American Society for Information Science and Technology*, 63(6):1059–1078, 2012. ISSN: 1532-2890, DOI: 10.1002/asi.22634.

Botstein D. It's the Data! *Molecular biology of the cell*, 21(1):4–6, 2010, DOI: 10.1091/mbc.E09-07-0575.

Boyd R. *OAuth 2.0*. O'Reilly Media, Inc, 2012. ISBN: 978-1-449-31160-5.

Brase J. Datacite and linked data. *JLIS. it*, 4(1), 2013, DOI: 10.4403/jlis.it-5493.

Brazma A, Krestyaninova M und Sarkans U. Standards for systems biology. *Nature Reviews Genetics*, 7(8):593, 2006, DOI: 10.1038/nrg1922.

Burke B. *RESTful Java with JaX-RS*. O'Reilly Media, Inc, 2009. ISBN: 978-0-596-15804-0.

Busch M, Gade K, Larson B, Lok P, Luckenbill S und Lin J. Earlybird: Real-Time Search at Twitter. In *IEEE 28th International Conference on Data Engineering*, pages 1360–1369, April 2012, DOI: 10.1109/ICDE.2012.149.

Buyya R, Calheiros R. N und Dastjerdi A. V. *Big Data: Principles and Paradigms*. Morgan Kaufmann, 2016. ISBN: 978-0128053942.

Cai L und Zhu Y. The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal*, 14, 2015, DOI: 10.5334/dsj-2015-002.

Cech T. R, Eddy S. R, Eisenberg D, Hersey K, Holtzman S. H, Poste G. H, Raikhel N. V, Scheller R. H, Singer D. B und Waltham M. C. Sharing Publication-Related Data and Materials: Responsibilities of Authorship in the Life Sciences. *Plant physiology*, 132(1): 19–24, 2003, DOI: 10.1104/pp.900068.

Chavan V und Penev L. The data paper: a mechanism to incentivize data publishing in biodiversity science. *BMC Bioinformatics*, 12(15):S2, Dec 2011, DOI: 10.1186/1471-2105-12-S15-S2.

Chen C. P und Zhang C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 275:314 – 347, 2014, DOI: 10.1016/j.ins.2014.01.015.

Chen D, Shi R, Pape J.-M, Neumann K, Arend D, Graner A, Chen M und Klukas C. Predicting plant biomass accumulation from image-derived parameters. *GigaScience*, 7 (2):1–13, 2018, DOI: 10.1093/gigascience/giy001.

Chen J, Chen Y, Du X, Li C, Lu J, Zhao S und Zhou X. Big data challenge: a data management perspective. *Frontiers of Computer Science*, 7(2):157–164, Apr 2013, DOI: 10.1007/s11704-013-3903-7.

CKAN Association. CKAN, 2018. URL: `https://ckan.org/`. [Online; Stand Mai 2018].

Clark T, Martin S und Liefeld T. Globally distributed object identification for biological knowledgebases. *Briefings in bioinformatics*, 5(1):59–70, 2004, DOI: 10.1093/bib/5.1.59.

Codd E. F. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970, DOI: 10.1007/978-3-642-48354-7_4.

Cook C. E, Bergman M. T, Finn R. D, Cochrane G, Birney E und Apweiler R. The European Bioinformatics Institute in 2016: Data growth and integration. *Nucleic acids research*, 44:D20–6, 2016, DOI: 10.1093/nar/gkv1352.

Cook C. E, Bergman M. T, Cochrane G, Apweiler R und Birney E. The European Bioinformatics Institute in 2017: data coordination and integration. *Nucleic Acids Research*, 46(D1):D21–D29, 2018, DOI: 10.1093/nar/gkx1154.

Coppens F, Wuyts N, Inzé D und Dhondt S. Unlocking the potential of plant phenotyping data through integration and data-driven approaches. *Current Opinion in Systems Biology*, 4:58 – 63, 2017. ISSN: 2452-3100, DOI: 10.1016/j.coisb.2017.07.002.

Corlan A. D. Medline trend: automated yearly statistics of PubMed results for any query, 2018. URL: `http://dan.corlan.net/medline-trend.html`. [Online; Stand Januar 2018].

Cragin M. H, Palmer C. L, Carlson J. R und Witt M. Data sharing, small science and institutional repositories. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 368(1926):4023–4038, 2010, DOI: 10.1098/rsta.2010.0165.

Creative Commons. Creative Commons Licenses, 2018. URL: https://creativecommons.org/. [Online; Stand Januar 2018].

Cruz J. A, Yin X, Liu X, Imran S. M, Morris D. D, Kramer D. M und Chen J. Multi-modality imagery database for plant phenotyping. *Machine Vision and Applications*, 27 (5):735–749, 2016, DOI: 10.1007/s00138-015-0734-6.

Ćwiek-Kupczyńska H, Altmann T, Arend D, Arnaud E, Chen D, Cornut G, Fiorani F, Frohmberg W, Junker A, Klukas C, Lange M, Mazurek C, Nafissi A, Neveu P, van Oeveren J, Pommier C, Poorter H, Rocca-Serra P, Sansone S.-A, Scholz U, van Schriek M, Seren Ü, Usadel B, Weise S, Kersey P und Krajewski P. Measures for interoperability of phenotypic data: minimum information requirements and formatting. *Plant Methods*, 12(1):44, Nov 2016. ISSN: 1746-4811, DOI: 10.1186/s13007-016-0144-4.

DataCite, 2018a. URL: https://www.datacite.org. [Online; Stand Februar 2018].

DataCite. Metadata Store Rest API, 2018b. URL: https://mds.datacite.org/. [Online; Stand Februar 2018].

DataCite. Metadata Schema, 2018c. URL: https://schema.datacite.org/. [Online; Stand Januar 2018].

DataCite. Metadata Store Search API, 2018d. URL: https://search.datacite.org/. [Online; Stand Februar 2018].

DCC Digital Curation Centre. Disciplinary metadata stadards, 2018. URL: http://www.dcc.ac.uk/drupal/resources/metadata-standards. [Online; Stand January 2018].

DFG Deutsche Forschungsgemeinschaft. Sicherung guter wissenschaftlicher Praxis. 2013, DOI: 10.1002/9783527679188.oth1.

Dhondt S, Wuyts N und Inzé D. Cell to whole-plant phenotyping: the best is yet to come. *Trends Plant Science*, 18(8):428–439, Aug 2013, DOI: 10.1016/j.tplants.2013.04.008.

Dice Holdings, Inc. SourceForge, 2018. URL: https://sourceforge.net/. [Online; Stand Januar 2018].

Diepenbroek M, Grobe H, Reinke M, Schindler U, Schlitzer R, Sieger R und Wefer G. PANGAEA - an information system for environmental sciences. *Computers & Geosciences*, 28(10):1201–1210, 2002, DOI: 10.1016/S0098-3004(02)00039-0.

Digital Science and Research Solutions Inc. Dimension AI, 2018. URL: https://app.dimensions.ai/discover/publication. [Online; Stand Mai 2018].

Drachen T, Ellegaard O, Larsen A und Dorch S. Sharing data increases citations. *Liber Quarterly*, 26(2), 2016, DOI: 10.18352/lq.10149.

Drago I, Mellia M, M Munafo M, Sperotto A, Sadre R und Pras A. Inside dropbox: understanding personal cloud storage services. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 481–494. ACM, 2012, DOI: 10.1145/2398776.2398827.

Dryad. Data from: Relatedness severely impacts accuracy of marker- assisted selection for disease resistance in hybrid wheat, 2007. URL: https://datadryad.org/resource/doi:10.5061/dryad.461nc. [Online; Stand Mai 2018].

Dryad. Dryad Data Repository, 2018. URL: http://datadryad.org/. [Online; Stand Januar 2018].

Dspace. Dspace, 2018. URL: http://www.dspace.org/. [Online; Stand Januar 2018].

DublinCore. DublinCore Schema, 2018. URL: http://dublincore.org/documents/dces/. [Online; Stand Januar 2018].

Durinx C, McEntyre J, Appel R, Apweiler R, Barlow M, Blomberg N, Cook C, Gasteiger E, Kim J, Lopez R, Redaschi N, Stockinger H, Teixeira D und Valencia A. Identifying ELIXIR Core Data Resources [version 2; referees: 2 approved]. *F1000Research*, 5(2422), 2017, DOI: 10.12688/f1000research.9656.2.

Dybå T und Dingsøyr T. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859, 2008, DOI: 10.1016/j.infsof.2008.01.006.

Eclipse Foundation. Jetty Webserver, 2017. URL: http://www.eclipse.org/jetty/. [Online; Stand Januar 2018].

Eclipse Foundation. Eclipse IDE, 2018. URL: https://www.eclipse.org/ide/. [Online; Stand Januar 2018].

eDAL. electronicDataArchiveLibrary, 2018. URL: https://bitbucket.org/ipk_bit_team/electronicdataarchivelibrary. [Online; Stand Mai 2018].

Edgar R, Domrachev M und Lash A. E. Gene expression omnibus: Ncbi gene expression and hybridization array data repository. *Nucleic Acids Research*, 30(1):207–210, 2002, DOI: 10.1093/nar/30.1.207.

ELIXIR. ELIXIR - A distributed infrastructure for life-science information, 2018. URL: https://www.elixir-europe.org/. [Online; Stand Mai 2018].

European Commission. Guidelines on FAIR Data Management in Horizon 2020, 2016. URL: http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf. [Online; Stand Februar 2018].

European Commission. Odin Project, 2018a. URL: https://odin-project.eu/. [Online; Stand Februar 2018].

European Commission. OpenAIRE, 2018b. URL: https://www.openaire.eu/. [Online; Stand Mai 2018].

European Commission. Thor Project, 2018c. URL: https://project-thor.eu/. [Online; Stand Februar 2018].

Fabre J, Dauzat M, Nègre V, Wuyts N, Tireau A, Gennari E, Neveu P, Tisné S, Massonnet C, Hummel I und Granier C. PHENOPSIS DB: an information system for Arabidopsis thaliana phenotypic data in an environmental context. *BMC Plant Biol*, 11:77, 2011, DOI: 10.1186/1471-2229-11-77.

Fahlgren N, Gehan M. A und Baxter I. Lights, camera, action: high-throughput plant phenotyping is ready for a close-up. *Current Opinion in Plant Biology*, 24:93–99, Apr 2015, DOI: 10.1016/j.pbi.2015.02.006.

FAIRSharing. FAIRSharing, 2018. URL: https://fairsharing.com/. [Online; Stand Januar 2018].

Fan J, Han F und Liu H. Challenges of Big Data analysis. *National Science Review*, 1(2): 293–314, 2014, DOI: 10.1093/nsr/nwt032.

Fecher B, Friesike S und Hebing M. What Drives Academic Data Sharing? *PLoS ONE*, 10 (2):1–25, 02 2015, DOI: 10.1371/journal.pone.0118053.

FEDORA. FEDORA, 2018. URL: http://fedorarepository.org/. [Online; Stand Januar 2018].

Field D, Sansone S, DeLong E. F, Sterk P, Friedberg I, Gaudet P, Lewis S, Kottmann R, Hirschman L, Garrity G, Cochrane G, Wooley J, Meyer F, Hunter S, White O, Bramlett B, Gregurick S, Lapp H, Orchard S, Rocca-Serra P, Ruttenberg A, Shah N, Taylor C und Thessen A. Meeting Report: BioSharing at ISMB 2010. *Standards in Genomic Sciences*, 3(3), 2010, DOI: 10.4056/sigs/1403501.

Fielding R. T. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

figshare. Phenotypes of transgenic Arabidopsis plants with different tagging constructs, 2007. URL: https://figshare.com/articles/Phenotypes_of_transgenic_Arabidopsis_plants_with_different_tagging_constructs/65122. [Online; Stand Mai 2018].

figshare. figshare, 2018. URL: https://figshare.com/. [Online; Stand Januar 2018].

Fiorani F und Schurr U. Future Scenarios for Plant Phenotyping. *Annu Rev Plant Biol*, 64: 267–291, 2013, DOI: 10.1146/annurev-arplant-050312-120137.

Flanagan D. *Java in a Nutshell*. O'Reilly Media, Inc, 2005. ISBN: 978-0596007737.

Fonticons Inc. Font Awesome 5, 2018. URL: https://fontawesome.com/license. [Online; Stand Mai 2018].

Freedman L. P, Cockburn I. M und Simcoe T. S. The Economics of Reproducibility in Preclinical Research. *PLOS Biology*, 13(6):1–9, 06 2015, DOI: 10.1371/journal.pbio.1002165.

FSU Friedrich-Schiller-Universität Jena. Richtlinien zur Sicherung guter wissenschaftlicher Praxis, 2006. URL: https://www.uni-jena.de/Sicherung_guter_wissenschaftlicher_Praxis-path-18,60,167.html. [Online; Stand Mai 2018].

Galperin M. Y, Fernández-Suárez X. M und Rigden D. J. The 24th annual *Nucleic Acids Research* database issue: a look back and upcoming changes. *Nucleic acids research*, 45 (D1):D1–D11, 2017, DOI: 10.1093/nar/gkw1188.

George B und Williams L. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337 – 342, 2004. ISSN: 0950-5849, DOI: 10.1016/j.infsof.2003.09.011.

GIT. GIT, 2018. URL: https://git-scm.com/. [Online; Stand Januar 2018].

GitHub Inc. GitHub, 2018. URL: https://github.com/. [Online; Stand Januar 2018].

Goodman A, Pepe A, Blocker A. W, Borgman C. L, Cranmer K, Crosas M, Di Stefano R, Gil Y, Groth P, Hedstrom M, Hogg D. W, Kashyap V, Mahabal A, Siemiginowska A und Slavkovic A. Ten Simple Rules for the Care and Feeding of Scientific Data. *PLOS Computational Biology*, 10(4):1–5, 04 2014, DOI: 10.1371/journal.pcbi.1003542.

Gradle Inc. Gradle, 2018. URL: https://gradle.org/. [Online; Stand Januar 2018].

Grainger T und Potter T. *Solr in Action*. Manning, 2014. ISBN: 978-1617291029.

Gregory K, Khalsa S. J, Michener W. K, Psomopoulos F. E, de Waard A und Wu M. Eleven quick tips for finding research data. *PLOS Computational Biology*, 14(4):1–7, 04 2018, DOI: 10.1371/journal.pcbi.1006038.

Haak L. L, Fenner M, Paglione L, Pentz E und Ratner H. ORCID: a system to uniquely identify researchers. *Learned Publishing*, 25(4):259–264, 2012, DOI: 10.1087/20120404.

Hammer-Lahav E. The OAuth 1.0 Protocol (RFC 5849). 2010, DOI: 10.17487/RFC5849.

Hardt D. The OAuth 2.0 Authorization Framework (RFC 6749). 2012, DOI: 10.17487/RFC6749.

Hatcher E, Gospodnetic O und McCandless M. *Lucene in Action*. Manning, 2010. ISBN: 978-1933988177.

Haug K, Salek R. M, Conesa P, Hastings J, de Matos P, Rijnbeek M, Mahendraker T, Williams M, Neumann S, Rocca-Serra P, Maguire E, González-Beltrán A, Sansone S.-A, Griffin J. L und Steinbeck C. MetaboLights—an open-access general-purpose repository for metabolomics studies and associated meta-data. *Nucleic Acids Research*, 41(D1): D781–D786, 2013, DOI: 10.1093/nar/gks1004.

Hey A. J, Tansley S und Tolle K. M. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009. ISBN: 978-0-9825442-0-4.

Hey T und Trefethen A. E. Cyberinfrastructure for e-Science. *Science*, 308(5723):817–821, 2005, DOI: 10.1126/science.1110410.

Houle D, Govindaraju D. R und Omholt S. Phenomics: the next challenge. *Nature Reviews Genetics*, 11(12):855–866, 2010, DOI: 10.1038/nrg2897.

International Organization for Standardization. ISO 26324:2012 Information and documentation - Digital object identifier system, 2012. URL: https://www.iso.org/standard/43506.html. [Online; Stand Februar 2018].

IQSS Institute for Quantitative Social Science (Harvard Universität). DataVerse, 2018. URL: https://dataverse.org/. [Online; Stand Mai 2018].

Janzen D und Saiedian H. Test-driven development concepts, taxonomy, and future direction. *Computer*, 38(9):43–50, 2005, DOI: 10.1109/MC.2005.314.

Jasny B. R, Chin G, Chong L und Vignieri S. Again, and Again, and Again .... *Science*, 334(6060):1225–1225, 2011, DOI: 10.1126/science.334.6060.1225.

JUnit. JUnit, 2017. URL: http://junit.org/. [Online; Stand Januar 2018].

Juty N, Ali R, Glont M, Keating S, Rodriguez N, Swat M, Wimalaratne S, Hermjakob H, Le Novère N, Laibe C und Chelliah V. Biomodels: Content, features, functionality, and use. *CPT: Pharmacometrics & Systems Pharmacology*, 4(2):55–68, 2015, DOI: 10.1002/psp4.3.

Kay A. C. The early history of Smalltalk. In *History of programming languages-II*, pages 511–598. ACM, 1996, DOI: 10.1145/234286.1057828.

Khondhu J, Capiluppi A und Stol K.-J. Is It All Lost? A Study of Inactive Open Source Projects. In *IFIP International Conference on Open Source Systems*, pages 61–79. Springer, 2013, DOI: 10.1007/978-3-642-38928-3_5.

Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier J.-M und Irwin J. Aspect-oriented programming. In *ECOOP'97 - Object-Oriented Programming: 11th European Conference, Finland*. Springer, 1997, DOI: 10.1007/BFb0053381.

Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J und Griswold W. G. An Overview of AspectJ. In *ECOOP 2001 - Object-Oriented Programming: 15th European Conference, Hungary*. Springer, 2001, DOI: 10.1007/3-540-45337-7_18.

Kılıç S, Sagitova D. M, Wolfish S, Bely B, Courtot M, Ciufo S, Tatusova T, O'Donovan C, Chibucos M. C, Martin M. J und Erill I. From data repositories to submission portals: rethinking the role of domain-specific databases in CollecTF. *Database*, 2016:baw055, 2016, DOI: 10.1093/database/baw055.

Klump J und Huber R. 20 Years of Persistent Identifiers – Which Systems are Here to Stay? *Data Science Journal*, 16, 2017, DOI: 10.5334/dsj-2017-009.

Klump J, Huber R und Diepenbroek M. DOI for geoscience data - how early practices shape present perceptions. *Earth Science Informatics*, 9(1):123–136, Mar 2016, DOI: 10.1007/s12145-015-0231-5.

Koboldt D. C, Steinberg K. M, Larson D. E, Wilson R. K und Mardis E. R. The Next-Generation Sequencing Revolution and Its Impact on Genomics. *Cell*, 155(1):27–38, Sep 2013, DOI: 10.1016/j.cell.2013.09.006.

Kodama Y, Shumway M und Leinonen R. The sequence read archive: explosive growth of sequencing data. *Nucleic Acids Research*, 40(D1):D54–D56, 2012, DOI: 10.1093/nar/gkr854.

Krajewski P, Chen D, Ćwiek H, van Dijk A. D, Fiorani F, Kersey P, Klukas C, Lange M, Markiewicz A, Nap J. P, van Oeveren J, Pommier C, Scholz U, van Schriek M, Usadel B und Weise S. Towards recommendations for metadata and data handling in plant phenotyping. *Journal of Experimental Botany*, 66(18):5417–5427, 2015, DOI: 10.1093/jxb/erv271.

Kratz J und Strasser C. Data publication consensus and controversies. *F1000Research*, 3 (94), 2014, DOI: 10.12688/f1000research.3979.3.

Lagoze C und Van de Sompel H. The Open Archives Initiative: Building a Low-barrier Interoperability Framework. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '01, pages 54–62, New York, NY, USA, 2001. ACM, DOI: 10.1145/379437.379449.

Lawrence B, Jones C, Matthews B, Pepler S und Callaghan S. Citation and Peer Review of Data: Moving Towards Formal Data Publication. *International Journal of Digital Curation*, 6(2):4–37, 2011, DOI: 10.2218/ijdc.v6i2.205.

Le Novère N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Snoep J. L und Hucka M. Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, 34(suppl 1):D689–D691, 2006, DOI: 10.1093/nar/gkj092.

Leibniz Gemeinschaft. Arbeitskreis Forschungsdaten, 2018. URL: `https://www.leibniz-gemeinschaft.de/ueber-uns/organisation/arbeitskreise/arbeitskreis-forschungsdaten/`. [Online; Stand Januar 2018].

Leinonen R, Sugawara H und Shumway M. The sequence read archive. *Nucleic Acids Research*, 2010, DOI: 10.1093/nar/gkq1019.

Leonelli S, Smirnoff N, Moore J, Cook C und Bastow R. Making open data work for plant scientists. *Journal of Experimental Botany*, 64(14):4109–4117, 2013, DOI: 10.1093/jxb/ert273.

Leonelli S, Davey R. P, Arnaud E, Parry G und Bastow R. Data management and best practice for plant science. *Nature Plants*, 3(17086), 2017, DOI: 10.1038/nplants.2017.86.

Leprevost F. d. V, Barbosa V. C, Francisco E. L, Perez-Riverol Y und Carvalho P. C. On best practices in the development of bioinformatics software. *Frontiers in Genetics*, 5 (199), 2014, DOI: 10.3389/fgene.2014.00199.

Lin J und Strasser C. Recommendations for the Role of Publishers in Access to Data. *PLoS Biol*, 12(10), 2014, DOI: 10.1371/journal.pbio.1001975.

Lobet G, Draye X und Périlleux C. An online database for plant image analysis software tools. *Plant Methods*, 9(1):38, 2013, DOI: 10.1186/1746-4811-9-38.

Loeliger J und McCullough M. *Version Control with Git: Powerful tools and techniques for collaborative software development.* O'Reilly Media, Inc., 2012. ISBN: 978-1-4499-31638-9.

Lu Z. PubMed and beyond: a survey of web tools for searching biomedical literature. *Database*, 2011, 2011, DOI: 10.1093/database/baq036.

Lynch C. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008, DOI: 10.1038/455028a.

Marcial L. H und Hemminger B. M. Scientific data repositories on the Web: An initial survey. *Journal of the American Society for Information Science and Technology*, 61(10): 2029–2048, 2010. ISSN: 1532-2890, DOI: 10.1002/asi.21339.

Mardis E. R. The impact of next-generation sequencing technology on genetics. *Trends in Genetics*, 24(3):133–141, Mar 2008, DOI: 10.1016/j.tig.2007.12.007.

Marx V. Biology: The big challenges of big data. *Nature*, 498, 2013, DOI: 10.1038/498255a.

Mattmann C und Zitting J. *Tika in action*. Manning, 2011. ISBN: 978-1935182856.

MaxMind Inc. GeoIP2 API, 2018a. URL: `https://github.com/maxmind/GeoIP2-java`. [Online; Stand Januar 2018].

MaxMind Inc. MaxMind, 2018b. URL: `https://www.maxmind.com`. [Online; Stand Januar 2018].

May M. Big Biological Impacts from Big Data. *Science*, 2014, DOI: 10.1126/science.opms.p1400086.

McNutt M. Journals unite for reproducibility. *Science*, 346(6210):679–679, 2014, DOI: 10.1126/science.aaa1724.

McNutt M. #IAmAResearchParasite. *Science*, 351(6277), 2016, DOI: 10.1126/science.aaf4701.

McNutt M, Lehnert K, Hanson B, Nosek B. A, Ellison A. M und King J. L. Liberating field science samples and data. *Science*, 351(6277):1024–1026, 2016, DOI: 10.1126/science.aad7048.

McQuilton P, Gonzalez-Beltran A, Rocca-Serra P, Thurston M, Lister A, Maguire E und Sansone S.-A. BioSharing: curated and crowd-sourced metadata standards, databases and data policies in the life sciences. *Database*, 2016:baw075, 2016, DOI: 10.1093/database/baw075.

Merelli I, Pérez-Sánchez H, Gesing S und D'Agostino D. Managing, Analysing, and Integrating Big Data in Medical Bioinformatics: Open Problems and Future Perspectives. *BioMed Research International*, 2014:1–13, 2014, DOI: 10.1155/2014/134023.

Metzker M. L. Sequencing technologies - the next generation. *Nature Reviews Genetics*, 11 (1):31–46, 2010, DOI: 10.1038/nrg2626.

Michener W, Vieglais D, Vision T, Kunze J, Cruse P und Janée G. DataONE: Data Observation Network for Earth — Preserving Data and Enabling Innovation in the Biological and Environmental Sciences. *D-Lib Magazine*, 17(1), 2011, DOI: 10.1045/january2011-michener.

Müller T. H2 Database Engine, 2018. URL: `http://www.h2database.com`. [Online; Stand Januar 2018].

MLU Martin-Luther-Universität Halle-Wittenberg. Gute wissenschaftliche Praxis, 2009. URL: http://www.verwaltung.uni-halle.de/KANZLER/ZGST/ABL/2009/09_05_02.pdf. [Online; Stand Mai 2018].

Mons B, Neylon C, Velterop J, Dumontier M, da Silva Santos L. O. B und Wilkinson M. D. Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud. *Information Services & Use*, 37(1):49–56, 2017, DOI: 10.3233/ISU-170824.

MPG Max-Planck-Gesellschaft. Regeln zur Sicherung guter wissenschaftlicher Praxis, 2009. URL: https://www.mpg.de/229457/Regeln_guter_wiss_Praxis__Volltext-Dokument_.pdf. [Online; Stand Mai 2018].

Mundt M. Der DOI (digital object identifier) ein verlagsorientiertes lndexierungswerkzeug auch anwendbar auf Datensätze? Technical report, 1998. DOI: 10.2312/GFZ.misc.370184.

Nature Editorial. Reality check on reproducibility. *Nature*, 553(7604), 2016, DOI: 10.1038/533437a.

Nature Genetics Editorial. Growing access to phenotype data. *Nature Genetics*, 47(2):99, Feb 2015a, DOI: 10.1038/ng.3213.

Nature Genetics Editorial. No impact without data access. *Nature Genetics*, 47(7):691, 2015b, DOI: 10.1038/ng.3351.

Nature Plants Editorial. Making the greatest impact. *Nature Plants*, 1, 2015, DOI: 10.1038/nplants.2015.32.

Nature Publishing Group. Scientific data, 2018a. URL: https://www.nature.com/sdata/. [Online; Stand Februar 2018].

Nature Publishing Group. Guide to Publication Policies of the Nature Journals, 2018b. URL: https://www.nature.com/authors/gta.pdf. [Online; Stand Februar 2018].

NCBI National Center for Biotechnology Information. PubMed, 2017. URL: http://www.ncbi.nlm.nih.gov/pubmed. [Online; Stand Februar 2018].

Nekrutenko A und Taylor J. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. *Nature Reviews Genetics*, 13(9):667, 2012, DOI: 10.1038/nrg3305.

Nelson B. Empty Archives. *Nature*, 461:160–163, 2009, DOI: 10.1038/461160a.

Neuroth H, Oßwald A, Scheffel R, Strathmann S und Huth K. *nestor-Handbuch: Eine kleine Enzyklopädie der digitalen Langzeitarchivierung*. SUB Göttingen, 2010. URL: https://nestor.sub.uni-goettingen.de/handbuch/nestor-handbuch_23.pdf.

Nosek B. A, Alter G, Banks G. C, Borsboom D, Bowman S. D, Breckler S. J, Buck S, Chambers C. D, Chin G, Christensen G, Contestabile M, Dafoe A, Eich E, Freese J, Glennerster R, Goroff D, Green D. P, Hesse B, Humphreys M, Ishiyama J, Karlan D, Kraut A, Lupia A, Mabry P, Madon T, Malhotra N, Mayo-Wilson E, McNutt M, Miguel E, Paluck E. L, Simonsohn U, Soderberg C, Spellman B. A, Turitto J, VandenBos G, Vazire S, Wagenmakers E. J, Wilson R und Yarkoni T. Promoting an open research culture. *Science*, 348(6242):1422–1425, 2015, DOI: 10.1126/science.aab2374.

Open Archives Initiative. The Open Archives Initiative Protocol for Metadata Harvesting v2, 2018. URL: http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm. [Online; Stand Februar 2018].

OpenHMS. RMI IO Utilites, 2018. URL: http://openhms.sourceforge.net/rmiio/. [Online; Stand Januar 2018].

Oracle. Java Authentication and Authorization Service (JAAS) Reference Guide, 2018a. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html. [Online; Stand Februar 2018].

Oracle. JAX-RS, 2018b. URL: https://github.com/jax-rs. [Online; Stand Februar 2018].

Oracle. Jersey API, 2018c. URL: https://jersey.github.io/. [Online; Stand Februar 2018].

Oracle. Java NIO.2, 2018d. URL: https://docs.oracle.com/javase/tutorial/essential/io/fileio.html. [Online; Stand Februar 2018].

Oracle. The Java Remote Method Invocation API (Java RMI), 2018e. URL: http://docs.oracle.com/javase/8/docs/technotes/guides/rmi/. [Online; Stand Februar 2018].

Oracle. VirtualBox, 2018f. URL: https://www.virtualbox.org/. [Online; Stand Februar 2018].

OVGU Otto-von-Guericke-Universität Magdeburg. Gute wissenschaftliche Praxis, 2018. URL: http://www.grs.ovgu.de/Promotion/Gute+wiss_+Praxis-p-438.html. [Online; Stand Mai 2018].

Oxford University Press. GigaScience, 2018. URL: https://academic.oup.com/gigascience. [Online; Stand Februar 2018].

Pampel H, Vierkant P, Scholze F, Bertelmann R, Kindling M, Klump J, Goebelbecker H.-J, Gundlach J, Schirmbacher P und Dierolf U. Making Research Data Repositories Visible: The re3data.org Registry. *PLOS ONE*, 8(11):1–10, 11 2013, DOI: 10.1371/journal.pone.0078080.

Payette S und Lagoze C. Flexible and extensible digital object and repository architecture (fedora). In *Research and Advanced Technology for Digital Libraries*, volume 1513 of *Lecture Notes in Computer Science*, pages 41–59. Springer Berlin Heidelberg, 1998, DOI: 10.1007/3-540-49653-X_4.

Peng R. D. Reproducible Research in Computational Science. *Science*, 334(6060):1226–1227, 2011, DOI: 10.1126/science.1213847.

Perez-Riverol Y, Gatto L, Wang R, Sachsenberg T, Uszkoreit J, Leprevost F. d. V, Fufezan C, Ternent T, Eglen S. J, Katz D. S, Pollard T. J, Konovalov A, Flight R. M, Blin K und Vizcaíno J. A. Ten Simple Rules for Taking Advantage of Git and GitHub. *PLOS Computational Biology*, 12(7):1–11, 07 2016, DOI: 10.1371/journal.pcbi.1004947.

Perez-Sanz F, Navarro P. J und Egea-Cortines M. Plant phenomics: an overview of image acquisition technologies and image data analysis algorithms. *GigaScience*, 6(11):1–18, 2017, DOI: 10.1093/gigascience/gix092.

Pilato C. M, Collins-Sussman B und Fitzpatrick B. W. *Version control with subversion*. O'Reilly Media, Inc., 2008. ISBN: 978-0-596-51033-6.

Pivotal Software. Spring Framework, 2018. URL: https://projects.spring.io/spring-framework/. [Online; Stand Januar 2018].

Piwowar H. A und Vision T. J. Data reuse and the open data citation advantage. *PeerJ*, 1: e175, 2013, DOI: 10.7717/peerj.175.

Piwowar H. A, Day R. S und Fridsma D. B. Sharing Detailed Research Data Is Associated with Increased Citation Rate. *PLoS ONE*, 2(3), 2007, DOI: 10.1371/journal.pone.0000308.

Piwowar H. A, Vision T. J und Whitlock M. C. Data archiving is a good investment. *Nature*, 473(285), 2011, DOI: 10.1038/473285a.

PLOS Public Library of Science. Data Availability Policy, 2018. URL: http://journals.plos.org/plosone/s/data-availability. [Online; Stand Februar 2018].

Pop M und Salzberg S. L. Use and mis-use of supplementary material in science publications. *BMC Bioinformatics*, 16(1):1–4, 2015, DOI: 10.1186/s12859-015-0668-z.

Priem J. Scholarship: Beyond the paper. *Nature*, 495(7442):437–440, 2013, DOI: 10.1038/495437a.

Prinz F, Schlange T und Asadullah K. Believe it or not: how much can we rely on published data on potential drug targets? *Nature reviews Drug discovery*, 10(9):712–712, 2011, DOI: 10.1038/nrd3439-c1.

Prlić A und Procter J. B. Ten Simple Rules for the Open Development of Scientific Software. *PLOS Computational Biology*, 8(12):1–3, 12 2012, DOI: 10.1371/journal.pcbi.1002802.

R Core Team. R, 2018. URL: https://git-scm.com/. [Online; Stand Januar 2018].

Ram K. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, 8(1):7, Feb 2013, DOI: 10.1186/1751-0473-8-7.

RDA Research Data Alliance, 2018a. URL: https://www.rd-alliance.org/. [Online; Stand Februar 2018].

RDA Research Data Alliance. Metadata Directory, 2018b. URL: http://rd-alliance.github.io/metadata-directory/standards/. [Online; Stand Januar 2018].

re3data. Plant genomics and phenomics research data repository, 2018.

Richardson L und Ruby S. *RESTful web services*. O'Reilly Media, Inc., 2008. ISBN: 978-0-596-52926-0.

Roche D. G, Kruuk L. E. B, Lanfear R und Binning S. A. Public Data Archiving in Ecology and Evolution: How Well Are We Doing? *PLoS Biology*, 13(11):1–12, 11 2015, DOI: 10.1371/journal.pbio.1002295.

Rodriguez-Tomé P, Stoehr P. J, Cameron G. N und Flores T. P. The European Bioinformatics Institute (EBI) databases. *Nucleic Acids Research*, 24(1):6–12, 1996, DOI: 10.1093/nar/24.1.6.

Roos D. S. Bioinformatics–Trying to Swim in a Sea of Data. *Science*, 291(5507):1260–1261, 2001, DOI: 10.1126/science.291.5507.1260.

Rousseau D, Chéné Y, Belin E, Semaan G, Trigui G, Boudehri K, Franconi F und Chapeau-Blondeau F. Multiscale imaging of plants: current approaches and challenges. *Plant Methods*, 11:6, 2015, DOI: 10.1186/s13007-015-0050-1.

Royce W. W. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338. IEEE Computer Society Press, 1987. URL: `http://dl.acm.org/citation.cfm?id=41765.41801`.

Rusch-Feja D. The Open Archives Initiative and the OAI Protocol for Metadata Harvesting: rapidly forming a new tier in the scholarly communication infrastructure. *Learned Publishing*, 15(3):179–186, 2002, DOI: 10.1087/095315102320140464.

Saake G, Sattler K.-U und Heuer A. *Datenbanken-Konzepte und Sprachen*. mitp Professional, 2013. ISBN: 978-3826694530.

Salek R. M, Haug K, Conesa P, Hastings J, Williams M, Mahendraker T, Maguire E, González-Beltrán A. N, Rocca-Serra P, Sansone S.-A und Steinbeck C. The metabolights repository: curation challenges in metabolomics. *Database*, 2013, 2013, DOI: 10.1093/database/bat029.

Sandve G. K, Nekrutenko A, Taylor J und Hovig E. Ten Simple Rules for Reproducible Computational Research. *PLOS Computational Biology*, 9(10):1–4, 10 2013, DOI: 10.1371/journal.pcbi.1003285.

Schmutzer T, Bolger M. E, Rudd S, Chen J, Gundlach H, Arend D, Oppermann M, Weise S, Lange M, Spannagl M, Usadel B, Mayer K. F und Scholz U. Bioinformatics in the plant genomic and phenomic domain: The German contribution to resources, services and perspectives. *Journal of Biotechnology*, 261:37 – 45, 2017, DOI: 10.1016/j.jbiotec.2017.07.006. Bioinformatics Solutions for Big Data Analysis in Life Sciences presented by the German Network for Bioinformatics Infrastructure.

Schneeberger K und Weigel D. Fast-forward genetics enabled by new sequencing technologies. *Trends in Plant Science*, 16(5):282–288, 2011, DOI: 10.1016/j.tplants.2011.02.006.

Schofield P. N, Eppig J, Huala E, De Angelis M. H, Harvey M, Davidson D, Weaver T, Brown S, Smedley D, Rosenthal N, Schughart K, Aidinis V, Tocchini-Valentini G und Hancock J. M. Sustaining the Data and Bioresource Commons. *Science*, 330(6004): 592–593, 2010, DOI: 10.1126/science.1191506.

Schuster S. C. Next-generation sequencing transforms today's biology. *Nature Methods*, 5 (1):16–18, Jan 2008, DOI: 10.1038/nmeth1156.

Scientific Data Editorial. Let referees see the data. *Scientific Data*, 2016(160033), 2016, DOI: 10.1038/sdata.2016.33.

Shvachko K, Kuang H, Radia S und Chansler R. The Hadoop Distributed File System. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. Ieee, 2010, DOI: 10.1109/MSST.2010.5496972.

Singh J. FigShare. *Journal of Pharmacology and Pharmacotherapeutics*, 2(2):138–139, 2011, DOI: 10.4103/0976-500X.81919.

Smith M, Barton M, Bass M, Branschofsky M, McClellan G, Stuve D, Tansley R und Walker J. H. Dspace: An open source dynamic digital repository. *D-Lib Magazine*, 9(1), 2003, DOI: 10.1045/january2003-smith.

Smith V. S. Data publication: towards a database of everything. *BMC Research Notes*, 2 (1):1–3, 2009, DOI: 10.1186/1756-0500-2-113.

Sonatype Inc. Maven Central Repository, 2018a. URL: https://search.maven.org/. [Online; Stand Juni 2018].

Sonatype Inc. eDAL dependecies in the Maven Central Repository, 2018b. URL: https://repo1.maven.org/maven2/de/ipk-gatersleben/. [Online; Stand Juni 2018].

Springer Nature. BMC Research Notes, 2018. URL: https://bmcresnotes.biomedcentral.com/. [Online; Stand Februar 2018].

Starr J und Gastl A. isCitedBy: A metadata scheme for DataCite. *D-Lib Magazine*, 17(1), 2011, DOI: 10.1045/january2011-starr.

Stephens Z. D, Lee S. Y, Faghri F, Campbell R. H, Zhai C, Efron M. J, Iyer R, Schatz M. C, Sinha S und Robinson G. E. Big Data: Astronomical or Genomical? *PLOS Biology*, 13(7):1–11, 07 2015, DOI: 10.1371/journal.pbio.1002195.

Stodden V. C. Reproducible Research - Addressing the Need for Data and Code Sharing in Computational Science. *Computing in science & engineering*, 12(5):8–12, 2010, DOI: 10.1109/MCSE.2010.113.

Szalay A und Gray J. 2020 Computing: Science in an exponential world. *Nature*, 440(7083): 413, 2006, DOI: 10.1038/440413a.

Tauch A und Al-Dilaimi A. Bioinformatics in Germany: toward a national-level infrastructure. *Briefings in Bioinformatics*, 2017, DOI: 10.1093/bib/bbx040.

Tellam R. L, Rushton P, Schuerman P, Pala I und Anane D. The primary reasons behind data sharing, its wider benefits and how to cope with the realities of commercial data. *BMC Genomics*, 16(1):1, 2015, DOI: 10.1186/s12864-015-1789-5.

Tenopir C, Allard S, Douglass K, Aydinoglu A. U, Wu L, Read E, Manoff M und Frame M. Data Sharing by Scientists: Practices and Perceptions. *PLoS ONE*, 6(6):1–21, 06 2011, DOI: 10.1371/journal.pone.0021101.

Terracotta Inc. Ehcache, 2018. URL: http://www.ehcache.org. [Online; Stand Januar 2018].

Terracotta Inc. Quartz Job Scheduling Library, 2018. URL: http://www.quartz-scheduler.org/. [Online; Stand Januar 2018].

Tester M und Langridge P. Breeding Technologies to Increase Crop Production in a Changing World. *Science*, 327(5967):818–822, Feb 2010, DOI: 10.1126/science.1183700.

TIOBE. TIOBE Index, 2018. URL: https://www.tiobe.com/tiobe-index/. [Online; Stand Februar 2018].

Treloar A und Harboe-Ree C. Data management and the curation continuum: how the Monash experience is informing repository relationships. *Proceedings of VALA*, 2008. URL: http://www.vala.org.au/vala2008-proceedings/vala2008-session-6-treloar.

Valdes A. M, Glass D und Spector T. D. Omics technologies and the study of human ageing. *Nature Reviews Genetics*, 14(9):601–607, 2013, DOI: 10.1038/nrg3553.

Van Noorden R. Data-sharing: Everything on display. *Nature*, 500(7461):243–245, 2013a, DOI: 10.1038/nj7461-243a.

Van Noorden R. Open access: The true cost of science publishing. *Nature*, 495(7442): 426–429, 2013b, DOI: 10.1038/495426a.

Vandewalle P. Code Sharing Is Associated with Research Impact in Image Processing. *Computing in Science & Engineering*, 14(4):42–47, 2012, DOI: 10.1109/MCSE.2012.63.

Vasilevsky N. A, Brush M. H, Paddock H, Ponting L, Tripathy S. J, LaRocca G. M und Haendel M. A. On the reproducibility of science: unique identification of research resources in the biomedical literature. *PeerJ*, 1:e148, September 2013. ISSN: 2167-8359, DOI: 10.7717/peerj.148.

Velocity. The Apache Velocity Project, 2017. URL: http://velocity.apache.org/. [Online; Stand Januar 2018].

Vihinen M. No more hidden solutions in bioinformatics. *Nature*, 521(261), 2015, DOI: 10.1038/521261a.

Vines T. H, Albert A. Y, Andrew R. L, Débarre F, Bock D. G, Franklin M. T, Gilbert K. J, Moore J.-S, Renaut S und Rennison D. J. The Availability of Research Data Declines Rapidly with Article Age. *Current Biology*, 24(1):94–97, 2014, DOI: 10.1016/j.cub.2013.11.014.

Vision T. J. Open data and the social contract of scientific publishing. *BioScience*, 60(5): 330–331, 2010, DOI: 10.1525/bio.2010.60.5.2.

Wallis J. C, Rolando E und Borgman C. L. If We Share Data, Will Anyone Use Them? Data Sharing and Reuse in the Long Tail of Science and Technology. *PLoS ONE*, 8(7): 1–17, 07 2013, DOI: 10.1371/journal.pone.0067332.

Walter A, Liebisch F und Hund A. Plant phenotyping: from bean weighing to image analysis. *Plant Methods*, 11:14, 2015, DOI: 10.1186/s13007-015-0056-8.

Weibel S. The Dublin Core: A Simple Content Description Model for Electronic Resources. *Bulletin of the American Society for Information Science and Technology*, 24(1):9–11, 1997. ISSN: 1550-8366, DOI: 10.1002/bult.70.

WGL Leibniz-Gemeinschaft. Empfehlungen der Leibniz-Gemeinschaft zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit Vorwürfen wissenschaftlichen Fehlverhaltens, 2015. URL: https://www.leibniz-gemeinschaft.de/fileadmin/user_upload/downloads/Presse/Posititionen/Leibniz-Gemeinschaft.Leitlinie_gute_wissenschaftlicher_Praxis.27.11.2015-1.pdf. [Online; Stand Mai 2018].

Wierling C, Herwig R und Lehrach H. Resources, standards and tools for systems biology. *Briefings in Functional Genomics*, 6(3):240–251, 2007, DOI: 10.1093/bfgp/elm027.

Wikipedia. List of biological databases, 2018. URL: https://en.wikipedia.org/wiki/List_of_biological_databases. [Online; Stand Mai 2018].

Wilkinson M. D, Dumontier M, Aalbersberg I. J, Appleton G, Axton M, Baak A, Blomberg N, Boiten J.-W, da Silva Santos L. B, Bourne P. E, Bouwman J, Brookes A. J, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo C. T, Finkers R, Gonzalez-Beltran A, Gray A. J. G, Groth P, Goble C, Grethe J. S, Heringa J, 't Hoen P. A. C, Hooft R, Kuhn T, Kok R, Kok J, Lusher S. J, Martone M. E, Mons A, Packer A. L, Persson B, Rocca-Serra P, Roos M, van Schaik R, Sansone S.-A, Schultes E, Sengstag T, Slater T, Strawn G, Swertz M. A, Thompson M, van der Lei J, van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft J, Katherine Zhao und Mons B. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3 (160018), 2016, DOI: 10.1038/sdata.2016.18.

Wimalaratne S. M, Juty N, Kunze J, Janée G, McMurry J. A, Beard N, Jimenez R, Grethe J. S, Hermjakob H, Martone M. E und Clark T. Uniform resolution of compact identifiers for biomedical data. *Scientific Data*, 5, May 2018, DOI: 10.1038/sdata.2018.29.

Wren J. D. 404 not found: the stability and persistence of URLs published in MEDLINE. *Bioinformatics*, 20(5):668–672, 2004, DOI: 10.1093/bioinformatics/btg465.

Zenodo. Measures for interoperability of phenotypic data: minimum information requirements and formatting, 2016. URL: https://zenodo.org/record/166403. [Online; Stand Mai 2018].

Zenodo. Zenodo, 2018. URL: https://zenodo.org/. [Online; Stand Januar 2018].

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 08.08.2018

Daniel Arend