hosted by Hamburg University Press

apropos
[Perspektiven auf die Romania]

Sprache∫Literatur∫Kultur∫Geschichte∫Ideen∫Politik∫Gesellschaft

Eine FAIRe Anwendungssoftware für textbasierte Forschungsdaten
Das UV2 Annotationstool

Anja Weingart & Georg A. Kaiser

apropos [Perspektiven auf die Romania]
hosted by Hamburg University Press
2022, 9
pp. 240-253
ISSN: 2627-3446

Online
https://journals.sub.uni-hamburg.de/apropos/article/view/1900

Zitierweise

Weingart, Anja & Georg A. Kaiser. 2022. „Eine FAIRe Anwendungssoftware für textbasierte Forschungsdaten. Das UV2 Annotationstool.“ *apropos [Perspektiven auf die Romania]* 9/2022, 240-253.
doi: https://doi.org/10.15460/apropos.9.1900

2022, n°9
pp. 240-253
doi: https://doi.org/10.15460/apropos.9.1900

Anja Weingart & Georg A. Kaiser

Eine FAIRe Anwendungssoftware für textbasierte Forschungsdaten

Das UV2 Annotationstool

Anja Weingart
ist promovierte Romanistin und arbeitet derzeit als Softwareentwicklerin beim Projektträger des Deutschen Zentrums für Luft- und Raumfahrt (DLR-PT).
Anja.Weingart@uni-konstanz.de

Georg A. Kaiser
ist Professor für Romanistische Sprachwissenschaft am Fachbereich Linguistik der Universität Konstanz.
Georg.Kaiser@uni-konstanz.de

Keywords

FAIR-Prinzipien – Forschungssoftware – webbasiertes Annotationstool

Als Richtlinien für den Umgang mit digitalen Forschungsdaten definieren Wilkinson et al. (2016) die FAIR-Prinzipien *findable* (auffindbar), *accessible* (zugänglich), *interoperable* (interoperabel) und *reusable* (wiederverwendbar). Diese vier Prinzipien gelten für alle Arten von digitalen Objekten und damit auch für Softwareanwendungen. Im Unterschied zu Forschungsdaten liegt Software jedoch in zwei Formen vor: als maschinenlesbares, ausführbares Artefakt und als menschenlesbarer Quellcode mit Verknüpfungen zu anderen Softwarebibliotheken. Diese Besonderheiten erfordern eine Anpassung der einzelnen Prinzipien aus Wilkinson et al. (2016), die in Chue Hong et al. (2021) als Ergebnis der Arbeit der *FAIR for Research Software working group* (FAIR4RS) vorliegen. In diesem Beitrag werden zum einen die FAIR Prinzipien nach Chue Hong et al. (2021) für Forschungssoftware vorgestellt. Hierbei handelt es sich um etablierte Richtlinien, die auch von der DFG in ihren Leitfäden zum Umgang mit Forschungsdaten bzw. in Bezug auf Forschungssoftware erwähnt werden. Zum anderen wird die Umsetzung dieser Prinzipien am Beispiel der freien, webbasierten Anwendungssoftware, dem UV2-Annotationstool, illustriert. Dieses Annotationstool wurde zur Untersuchung von Verbzweiteffekten in den romanischen und anderen Sprachen entwickelt und steht grundsätzlich zur Analyse anderer sprachlicher Phänomene zur Verfügung.

In unserem Beitrag möchten wir zeigen, wie diese vier Prinzipien bei der Entwicklung einer Softwareanwendung umgesetzt werden können. Der Schwerpunkt liegt hierbei auf den Prinzipien *reusable* und *interoperable.* Die Konzepte der Wiederverwendbarkeit und Interoperabilität von Software und Softwareelementen beschäftigen die Softwaretechnik seit Jahrzehnten und es existieren hierzu zahlreiche Ansätze und Modelle, die eine Modularisierung von Softwareelementen verschiedener Komplexität und die Architektur des Softwaresystems betreffen (cf. McIlroy 1969, Meyer 1997, Brügge & Dutoit 2013, Richards & Ford 2020). Im vorliegenden Beitrag wird die konkrete Umsetzung der im UV2-Annotationstool implementierten Konzepte dargestellt und im Hinblick auf die FAIR-Prinzipien bewertet. Eine umfassendere Bewertung existierender Konzepte und Verfahren sowie Implementierungsvorschläge ist im von der FAIR4RS Working Group geplanten Leitfaden zur Umsetzung der FAIR-Prinzipien für Forschungssoftware zu erwarten.

Zunächst wird das UV2-Annotationstool hinsichtlich der Entwicklungsstrategie, des Aufbaus des Quellcodes und der Art des ausführbaren Artefakts vorgestellt. Es folgt eine Einführung der FAIR-Prinzipien für Forschungssoftware und eine Einordnung, wie die dort formulierten Anforderungen bei der Implementierung des UV2-Annotationstools umgesetzt werden. Ein Blick hinter die Kulissen der Softwaretechnik ist sicher auch für die traditionelle Romanistik lohnend, weil diese Techniken und Werkzeuge auch für romanistische Forschung genutzt werden können. Uns bekannte Projekte, die diese Techniken bereits verwenden, sind VerbaAlpina (cf. Krefeld & Lücke (2020)) und die historischen Projekte, die Eckhart et al. (2021) vorstellen.

1. Das UV2-Annotationstool

Das UV2-Annotationstool wurde im Rahmen des DFG Projekts '*Uncovering verb-second effects. An interface-based typology* (UV2)' entwickelt (siehe Fußnote 1). Das Projekt untersucht syntaktische und pragmatische Bedingungen für so genannte Verbzweiteffekte in den romanischen Sprachen im Vergleich zu typologisch anderen Sprachen, wie z. B. Baskisch. Grundlage dieser Untersuchung sind umfangreiche Paralleltextkorpora basierend auf Bibelübersetzungen sowie andere in viele Sprachen übersetzte Texte, u. a. *Astérix*, *Le petit Nicolas, Commissario Montalbano*, *Sherlock Holmes*). Zu Beginn des Projekts lagen die Paralleltexte bereits in einem tabellenförmigen Format vor und waren teilweise für Kriterien wie Satztyp, Art der präverbalen Konstituente und Verbposition annotiert. Die Arbeit mit Office-Programmen (Textverarbeitung oder Tabellenkalkulation) verleitet jedoch zu einer „visuell orientierten" Datenstrukturierung und Annotation, also dem Einsatz von Formatierungen wie Farben oder Schriftstilen, weil die Datenspeicherung und die Datendarstellung nicht voneinander getrennt sind (zum häufigen Einsatz dieser Formate und Programme in der romanistischen Forschung siehe die Studie der AG Digitale Romanistik 2015). Die Konsequenz ist, dass so strukturierte Daten nicht direkt maschinenlesbar, also nicht in andere Formate

überführbar und damit nicht mit anderen Werkzeugen bearbeitbar sind.[1] Sie erfüllen nicht die FAIR-Prinzipien der Interoperabilität und der Wiederverwendbarkeit. Für die Forschungspraxis ist es jedoch viel problematischer, dass auf der Grundlage so strukturierter Daten keine komplexen Suchanfragen möglich sind, die Daten also nicht mehr systematisch und mit allen Vorteilen eines digitalen Korpus analysiert werden können.

Mit Hilfe des UV2-Annotationstools werden die Paralleltexte in eine relationale Datenbank übertragen, die online zugänglich sein wird. Die Webanwendung ermöglicht eine kollaborative Bearbeitung und Analyse der Daten, aber auch eine einschränkbare Veröffentlichung. Es werden Funktionalitäten für die Speicherung, Bearbeitung, Annotation und Alignierung von textbasierten Sprachdaten sowie eine Verwaltung von Benutzergruppen und den bibliographischen Angaben der Texte zur Verfügung gestellt. Auf konzeptioneller Ebene besteht die Anwendung aus vier Bereichen:

(1) a. Bereich Datenspeicherung mit Import- und Exportfunktionalitäten,

b. Bereich Sprachdaten mit Bearbeitung, Alignierung und Annotation,

c. Bereich Nutzerverwaltung mit Zugangs- und Rechteverwaltung und

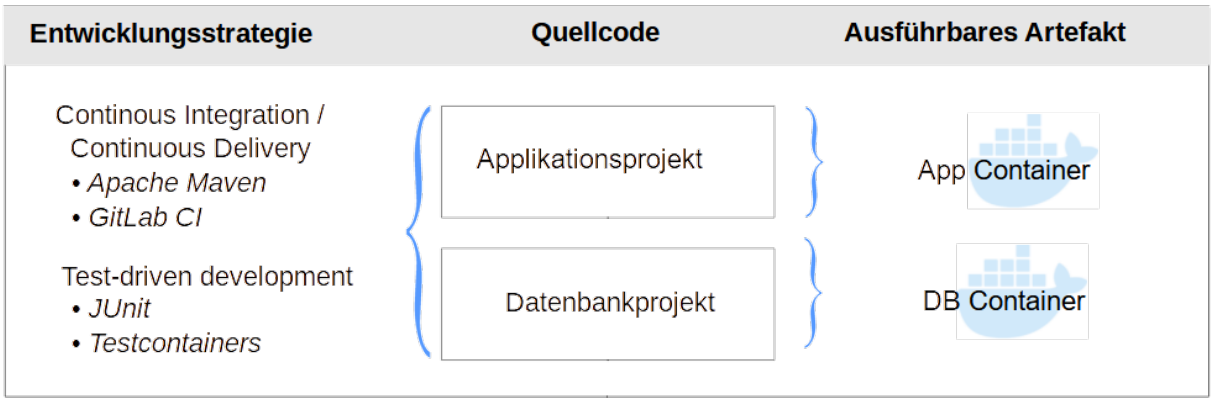
d. Bereich Bibliographie mit der Verwaltung bibliographischer Angaben und Kürzeln.

Damit die Anwendung auch für andere Forschungsprojekte nutzbar und an neue Anforderungen anpassbar ist, sollen die Bereiche möglichst unabhängig voneinander sein und getrennt wiederverwendet werden können. Auf dieses Entwicklungsziel wird bei der Betrachtung der FAIR-Prinzipien noch detailliert eingegangen werden. Im Folgenden wird das UV2-Annotationstool hinsichtlich der Entwicklungsstrategie, des Aufbaus des Quellcodes und der Art des ausführbaren Artefakts vorgestellt. Diese drei Perspektiven sind in Abbildung 1 schematisch aufgeführt. Vorab sei erwähnt, dass bei der Entwicklung nur freie Software, etablierte Verfahren der Softwaretechnik und gängige Entwicklungswerkzeuge eingesetzt werden.

[1] Als Beispiel für „visuelle Annotationen“ sei der Satz in (i) gegeben, der im Asterix Paralleltextkorpus folgende Formatierungen enthält. W-Fragen haben eine rote Textfarbe, das Fragewort ist unterstrichen und das Subjekt ist fett markiert.

(i) "Wo *bleibt* **das Wunder**?" Asterix und die Goten (AST03-08:07a2)

Diese Formatierungen können zwar maschinenlesbar gemacht werden, indem das Tabellendokument als .xml (siehe Bray et. al. 2008) gespeichert oder entpackt wird, aber ihre intendierte Bedeutung ist nicht kodiert. Es bedarf zusätzlicher Transformationsschritte um einen Text, der eine bestimmte Menge an Stilattributen hat, mit linguistischen Annotationen zu versehen. So müsste zum Beispiel der Phrase *das Wunder*, weil sie das Attribut font-weight="bold" hat, eine Annotation für Subjekt zugewiesen werden. In diesem Sinne sind diese Dokumente nicht direkt maschinenlesbar. Es spricht jedoch nichts dagegen, einen Text, der mit bestimmten linguistischen Annotationen ausgezeichnet ist, farbig oder durch Schriftstile darzustellen.



1 | Drei Perspektiven auf eine Softwareanwendung

1.1. Entwicklungsstrategie und ausführbares Artefakt

Die Entwicklungsmethode des Continuous Integration und Continuous Delivery hat zum Ziel, dass zu jeder Zeit, also trotz kontinuierlicher Veränderung des Quellcodes, eine lauffähige Softwareanwendung zur Verfügung steht (cf. Lester 2018). Dies bedeutet, dass mit jeder Änderung der Build- und Testprozess[2] automatisiert ausgeführt werden soll. Für die Automatisierung dieser Prozesse werden für das UV2-Annotationstool das Buildmanagement Apache Maven[3] und die Versionsverwaltung Git[4] sowie GitLab CI wie folgt genutzt. Der Quellcode der Anwendung besteht aus zwei Teilprojekten, dem Applikationsprojekt und dem Datenbankprojekt. Zusätzlich wird eine Entwicklerdokumentation erstellt, auf die hier jedoch nicht näher eingegangen wird. Alle Projekte nutzen Apache Maven um Abhängigkeiten zu verwalten und den Build- und Testprozess zu automatisieren. Alle Projekte werden mit der Versionsverwaltung Git verwaltet. Die Projektdateien werden in einem lokalen Quellcoderepositorium gespeichert, das mit einem externen Quellcoderepositorium synchronisiert werden kann. Für das UV2-Annotationstool wird der Anbieter Gitlab genutzt. Jedes Git-Projekt besitzt eine URL und ist damit eindeutig identifizierbar. Der Zugang auf das Projektrepositorium kann öffentlich gemacht werden oder für ausgewählte Nutzer und Nutzerinnen beschränkt sein. Gitlab ermöglicht es, dass bei jeder Änderung des Quellcodes der Build- und Testprozess automatisiert ausgeführt werden kann. Bei erfolgreichem Abschluss dieser Prozesse wird das ausführbare Artefakt in Form von Docker[5]

[2] Zur Erläuterung dieser Begriffe sei hier kurz angemerkt, dass der Quellcode einer Software nicht nur aus eigenem, sondern größtenteils aus bereits existierendem Programmcode besteht. Letzterer wird importiert und die Verweise auf die entsprechenden Bibliotheken (Abhängigkeiten) müssen verwaltet werden. Während des Buildprozesses werden die Codeteile zusammengeführt, kompiliert (d.h. in ausführbaren Code umgewandelt) und es wird ein ausführbares Objekt erzeugt. Hierzu gehört auch das Ausführen von Programmtests. Diese überprüfen zum Beispiel, ob einzelne Funktionalitäten die gewünschten Ergebnisse liefern.

[3] Apache Maven ist ein freies Buildmanagementtool, das in The Apache Software Foundation (2022) dokumentiert ist.

[4] Git ist ein Open Source Versionsverwaltungssystem (cf. Chacon & Straub 2014). Die bekanntesten Dienstleister, die Git als Clouddienst anbieten, sind gitlab.com oder github.com.

[5] Mit Hilfe von Docker, einer teilweise freien Software, können Anwendungen in sogenannte Container verpackt werden. Auf diese Weise ist die Anwendung vom Betriebssystem isoliert, kann aber trotzdem dessen Ressourcen nutzen.

Containern für das Datenbankprojekt und die Applikation ausgeliefert. Beide Container stehen unter einer eigenen URL zum download zur Verfügung. Auf diese Weise ist das UV2-Annotationstool zu jeder Zeit als lauffähiges Programm erhältlich.

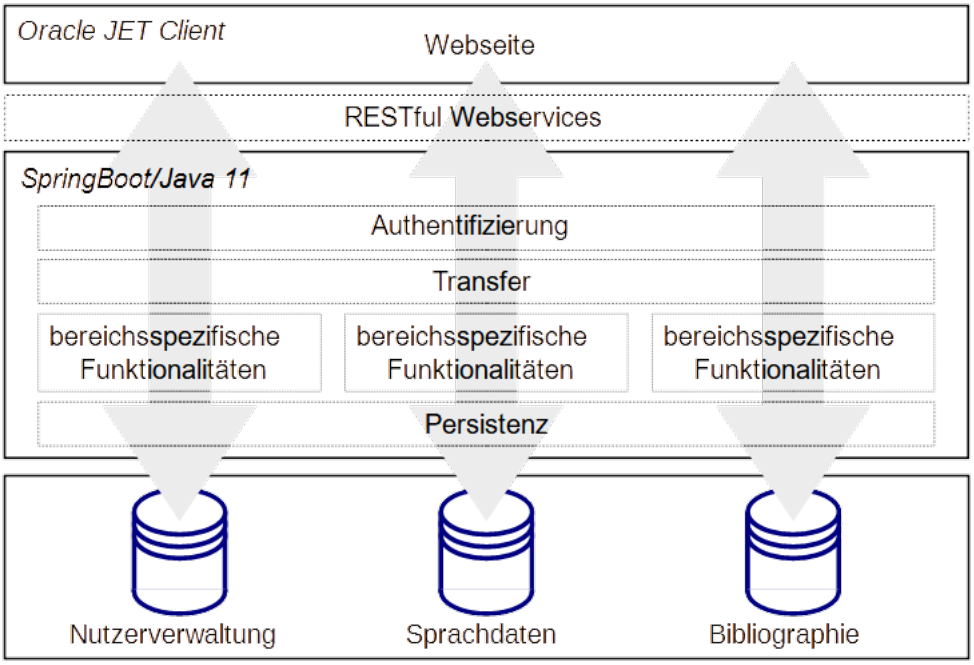
Die Strategie des *test-driven developments* (cf. Beck 2003) kann auf folgende Weise umgesetzt werden: Bevor eine neue Funktionalität implementiert wird, wird für sie ein Test geschrieben. Hierfür wird das Testframework JUnit genutzt (cf. Junit-team 2015-2022). Ein Test überprüft, ob das Programm bei erwarteten Eingaben eine erwartete Ausgabe erzeugt. Mit Hilfe des Tests kann überprüft werden, an welcher Stelle der Programmablauf fehlerhaft oder unvollständig ist. Diese Methode liefert einen Quellcode, dessen Funktionalitäten durch Tests überprüfbar und nachvollziehbar sind. Die Testinfrastruktur des UV2-Annotationstools erlaubt es, die Tests für den Quellcode und für die Container mit Hilfe von Testcontainers (cf. North & Egorov et al. 2015-2022) sowohl lokal, als auch automatisiert auszuführen.

1.2. Struktur des Quellcodes

Im Folgenden wird die Struktur des Quellcodes, insbesondere des Applikationsprojekts detaillierter betrachtet. Das UV2-Annotationstool kombiniert Aspekte des 3-Schichtenmodells[6] und der serviceorientierten Architektur[7] (cf. Richards & Ford 2020). Die Abbildung 2 zeigt eine schematische Darstellung des Aufbaus des Quellcodes.

[6] Im 3-Schichtenmodel wird eine Softwareanwendung in eine Präsentationsschicht, eine Logikschicht und eine Datenhaltungsschicht eingeteilt. In Abbildung 2 sind dies die horizontalen Kästen für den Client, die SpringBoot Applikation und die Datenbanken.

[7] Ein zentraler Gedanke der serviceorientierten Architektur ist die Wiederverwendbarkeit von Softwarekomponenten. In Abbildung 2 sind zum Beispiel die drei Schichten der Bibliographieverwaltung eine wiederverwendbare Komponente. Aber auch die Funktionalität für Authentifizierung und Transfer sind als wiederverwendbare Komponente implementiert.



2 | Architektur des UV2-Annotationstools

Das Applikationsprojekt besteht aus zwei Teilprojekten, dem Client und der SpringBoot/Java11[8] Anwendung, dem sogenannten Backend. Im Clientprojekt werden die Webseite und die Funktionalitäten zur Nutzerinteraktion mit HTML5 (cf. Hickson et al. 2022) und TypeScript (cf. Microsoft 2012-2022) erstellt. In der Backend-Anwendung werden HTTP-Anfragen verarbeitet (Transfer) und Funktionalitäten zu den in (1) genannten Bereichen Nutzerverwaltung, Sprachdaten und Bibliographie ausgeführt sowie die Kommunikation mit der Datenbank geregelt (Persistenz). Der Datenaustausch zwischen Client und Applikation erfolgt über RESTful Webservices (cf. Fielding 2000), indem der Client Anfragen mit dem HTTP Protokoll an Ressourcenadressen, wie zum Beispiel <www.uni-konstanz.de/ling/uv2/alignments>, stellt und diejenigen Daten als Antwort erhält, die unter dieser Adresse zur Verfügung gestellt werden, in unserem Fall die Paralleltexte. Die einzige Verbindung zwischen Client und Backend-Anwendung besteht somit in dem Wissen um die Ressourcenadressen. Auf diese Weise kann bei der Nachnutzung des UV2-Annotationstools auch leichter ein anderer Client verwendet werden.

Im Datenbankprojekt wird für jeden der drei Bereiche – Nutzerverwaltung, Sprachdaten, Bibliographie – eine relationale Datenbank zur Verfügung gestellt. Durch eine entsprechende Abstraktion der Funktionalitäten in der Backend-Anwendung können die drei Bereiche unabhängig voneinander verwendet werden. So ist zum Beispiel denkbar, dass die Webanwendung mit Nutzerverwaltung und Bibliographie wiederverwendet wird, jedoch der Bereich Sprachdaten nicht genutzt wird oder verändert werden soll. Auch kann das Datenbankprojekt vollkommen

[8] Spring Boot Framework ist in The Spring Team (2022) beschrieben und die Programmiersprache Java ist in Oracle (2021) dokumentiert.

unabhängig von der Webapplikation genutzt werden, zum Beispiel um Datenbanken aus Tabellendokumenten zu erstellen und, wenn gewünscht, in einem Docker Container auszuliefern. Die Datenbanken und der Container können wiederum unabhängig vom UV2-Annotationstool mit anderen Anwendungen verwendet werden.

2. Die FAIR-Prinzipien für Forschungssoftware

Die Diskussion der FAIR-Prinzipien für Forschungssoftware beginnt mit den Prinzipien der Wiederverwendbarkeit (*reusable*) und der Interoperabilität (*interoperable*), weil beide zentral für die Entwicklung des UV2-Annotationstools sind. Anschließend werden die Prinzipien der Auffindbarkeit (*findable*) und der Zugänglichkeit (*accessible*) diskutiert. Letztere sind vor allem für die Veröffentlichung von Softwareanwendungen relevant.

2.1. Wiederverwendbarkeit

Das Prinzip der Wiederverwendbarkeit (s. (2) unten) expliziert die Zweigestaltigkeit von Softwareanwendungen: Software ist nutzbar im Sinne eines ausführbaren Artefakts und wiederverwendbar. Wiederverwendbarkeit betrifft vor allem, aber nicht ausschließlich, den Quellcode und beinhaltet Kriterien wie die Verständlichkeit und die Anforderungen, dass Softwareelemente verändert, erweitert und in andere Softwareanwendungen integriert werden können. Wie in Abschnitt 1.2. zur Softwarearchitektur dargelegt, ist die Wiederverwendung einzelner Softwarekomponenten des UV2-Annotationstools, aber auch die Wiederverwendung der ausführbaren Artefakte möglich. Die Verwendung von Maven erleichtert zusätzlich die Lesbarkeit und Verständlichkeit des Quellcodes.

(2) R. Reusable

The software is both usable (it can be executed) and reusable (it can be understood, modified, built upon, or incorporated into other software).

R1. Software is described with a plurality of accurate and relevant attributes.

R1.1. Software must have a clear and accessible license.

R1.2. Software is associated with detailed provenance.

R2. Software includes qualified references to other software.

R3. Software meets domain-relevant community standards.

(Chue Hong et al. 2021, 13-14, Hervorhebung im Original)

Das Prinzip der Wiederverwendbarkeit enthält die drei Unterprinzipien R1-R3, die technische, rechtliche und organisatorische Aspekte betreffen. Das Kriterium R1 besagt allgemein, dass eine Softwareanwendung mit Metadaten ausgezeichnet werden soll, die sowohl technische als auch forschungsrelevante Details beschreiben. Dies kann als Teil des Quellcodes oder in Form einer Dokumentation geschehen. Explizit wird die Softwarelizenz genannt (R 1.1), weil die Art der Lizenz entscheidend ist, inwieweit und mit welchen Bedingungen der Quellcode wiederverwendet werden darf. Das UV2-Annotationstool wird unter der European Public

License 2.0 veröffentlicht, einer Open Source Lizenz mit einer starken *copy-left* Klausel. Die Software darf daher ganz oder in Teilen wiederverwendet werden, jedoch mit der Einschränkung, dass die neu entstandenen Anwendungen auch unter einer kompatiblen Open Source Lizenz veröffentlicht werden müssen. Des Weiteren werden für das UV2-Annotationstools nur freie und Open Source Tools und Software verwendet. Auf diese Weise entstehen keine lizenzbedingten Einschränkungen für eine Wiederverwendung.

Das Kriterium R1.2 bezieht sich auf technische und organisatorische Informationen, wie zum Beispiel die Autorenschaft und die Entwicklungshistorie, aber auch auf die genutzten Tools und Techniken sowie die Designentscheidungen. Gerade bei größeren Open Source Projekten ist die Autorenschaft einzelner Softwareelemente häufig wesentlich komplexer, als bei Forschungsdaten, da mehrere Akteure die Software über den Entwicklungszeitraum ergänzen und verändern. Mit der Versionsverwaltung Git, die auch für das UV2 Annotationstool verwendet wird, lassen sich die Entwicklungshistorie und die genaue Autorenschaft einzelner Softwareelemente dokumentieren. Jede Änderung des Quellcodes, ein sogenannter *commit*, trägt einen Bezeichner und ist eindeutig einem Zeitpunkt und einem Entwickler zuzuordnen. Bei Softwareprojekten, die Maven verwenden, sind die Angaben zur Autorenschaft, zu verwendeten Tools und den in R2 genannten Abhängigkeiten zu anderen Softwarebibliotheken in der *project object model* Datei (pom.xml) aufgeführt. Diese Datei steuert die Automatisierung, ist aber auch mit einem Inhaltsverzeichnis zu vergleichen und gibt einen strukturierten Überblick zu den genannten Angaben.

Das Kriterium R3 ist dahingehend zu verstehen, dass Software dann wiederverwendbar ist, wenn die in der Forschungsrichtung üblichen Techniken (Programmiersprache, Teststandards), aber auch Dateiformate verwendet werden. In Bezug auf übliche Dateiformate können laut Umfrage der AG Digitale Romanistik von 2015 tabellenförmige Formate als *community standard* in der Romanistik bezeichnet werden. Das UV2 Annotationstool ermöglicht es Tabellendokumente als CSV[9] Dateien direkt zu importieren und zu exportieren. Bezüglich der Programmiersprachen oder der Teststandards gibt es unseres Wissens nach keine *community standards*, weshalb für das UV2-Annotationstool die weitverbreiteten Programmiersprachen Java 11, HTML5 und TypeScript verwendet werden.

2.2. Interoperabilität

Interoperabilität bezeichnet die Zusammenarbeit von unabhängig voneinander ausführbaren Systemen, wobei Chue Hong et al. (2021) hier Zusammenarbeit auf die Möglichkeit des Datenaustausches beschränken. Fragen zur Kompatibilität von Softwareelementen und Softwaresystemen sind im Geltungsbereich der Wiederverwendbarkeit angesiedelt.

[9] CSV steht für *Comma-Separated Values* und ist ein Dateiformat zur Darstellung von tabellenförmigen Daten (cf. Shafranovich 2005).

(3) I. Interoperable

The software interoperates with other software through exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs).

I1. Software reads, writes and exchanges data in a way that meets domain-relevant community standards.

I2. Software includes qualified references to other objects.

(Chue Hong et al. 2021, 12-13, Hervorhebung im Original)

Das UV2 Annotationstool hat zwei Schnittstellen, an denen Daten ausgetauscht werden. Dies sind zum einen die RESTful Webservices und die Schnittstelle zwischen Applikation und Datenbank. Zur Übertragung der Daten, zum Beispiel der Sprachdaten, zwischen Client und Applikation, wird das JSON Format genutzt. JSON ist ein Internetstandard zum Austausch hierarchisch strukturierter Daten (cf. Bray 2017). Die Applikation wandelt die JSON Daten in Java Einheiten um (und umgekehrt), so dass diese mittels der Spring Data JPA, der Schnittstelle zwischen Datenbank und Applikation, in der Datenbank gespeichert oder von dort gelesen werden können. Das Kriterium I1 nimmt Bezug auf sogenannte *community standards.* Für (halb-)automatisch annotierte Korpora existieren verschiedene Modelle und Dateiformate, die von globaleren Standards wie XML[10] oder TSV[11] abgeleitet sind. So zum Beispiel die XML-basierten Formate TEI[12] oder WebLicht TCF[13], die TSV Formate der CoNLL Familie oder WebAnno TSV[14]. Des Weiteren werden auch OWL oder graphenbasierte Formate wie Salt verwendet.[15]

Für das UV2-Annotationstool wurde aus den folgenden Gründen der Internetstandard JSON und eine relationale Datenbank gewählt: Zum einen ist eine Konvertierung in oben genannte Formate grundsätzlich möglich, die integriert werden kann. Zum anderen verfolgt das UV2-Annotationstool eine andere Zielsetzung als Werkzeuge zur Erstellung eines automatisch annotierten Korpus, das als Endergebnis zur Veröffentlichung vorliegt. Vielmehr dient das UV2-Annotationstool der Erstellung einer textbasierten Sprachdatensammlung, die vollständige Texte, aber auch einzelne Sätze enthält und je nach Forschungsinteresse (re-)annotiert und erweitert werden kann. Ziel des Tools ist es, die linguistische Analyse zu unterstützen, indem eine personalisierte, aber FAIRe Datenbasis erstellt, annotiert, durchsucht und veröffentlicht werden kann.

[10] XML steht für *Extensible Markup Language* und ist in Bray et. al. (2008) dokumentiert.
[11] Das TSV Format (Tab Separated Values) ist ein Datenaustauschformat für tabellenförmige Daten, deren Spalten mittels des tab-Zeichens (U+0009) getrennt werden (cf. Korpela 2005).
[12] TEI ist ein XML-basiertes Format zur Auszeichnung von Texten, das von der Text Encoding Initiative kuratiert wird (cf. TEI Consortium 2021).
[13] WebLicht TCF (Text Corpus Format) ist ein XML-basiertes Format, das in CLARIN-D/SfS-Universität Tübingen (2012) dokumentiert ist.
[14] Das WebAnno TSV Format ist in Eckart de Castilho et al. (2016) und The WebAnno Team (2021) beschrieben.
[15] Für eine Dokumentation von OWL siehe W3C OWL Working Group (2012) und für Salt Zipser & Romary (2010).

Das Kriterium I2 bezieht sich auf Datenobjekte, wie zum Beispiel Konfigurationsdateien, die zur Ausführung der Software nötig sind. Im Falle des UV2-Annotationstools benötigt der Applikationscontainer eine Parameterdatei, um sich mit dem Datenbankcontainer zu verbinden. Diese Dateien sollten mindestens mit einem Bezeichner, aber idealerweise einer auflösbaren Referenz verknüpft sein, wie zum Beispiel einer URL oder einem DOI.

2.3. Auffindbarkeit

Das Prinzip *findable* gibt vier Kriterien an, wie eine Software in Form von Quellcode oder als ausführbares Artefakt sowie dazugehörige Metadaten leicht gefunden werden kann.

(4) F. Findable

> *The software, and its associated metadata, should be easy to find for both humans and machines.*
>
> *F1. Software is assigned a globally unique and persistent identifier.*
>
> *F1.1. Different components of the software must be assigned distinct identifiers representing different levels of granularity.*
>
> *F1.2. Different versions of the same software must be assigned distinct identifiers.*
>
> *F2. Software is described with rich metadata.*
>
> *F3. Metadata clearly and explicitly include the identifier of the software they describe.*
>
> *F4. Metadata are FAIR and are searchable and indexable.*
>
> (Chue Hong et al. 2021, 9-11, Hervorhebung im Original)

Das wichtigste Kriterium ist sicher F1. Die Software sollte mit einem eindeutigen und persistenten Bezeichner versehen werden. Diese Bezeichner können Softwareelemente unterschiedlicher Komplexität (F1.1) und verschiedene Versionen einer Software (F1.2) kennzeichnen. Das Arbeitspapier der Research Data Alliance/ FORCE11 SSCID WG et al. (2020) gibt eine Übersicht, welche Bezeichner sich für Softwareelemente verschiedener Komplexität eignen. So wird zum Beispiel Wikidata für eine Auszeichnung auf Projektebene angeführt, weil dort eine ausführliche Beschreibung mit Metadaten und Verweisen vorhanden ist. Ein DOI bietet sich für die Identifikation von Softwareversionen und Projektordnern an. Ein softwarespezifischer Bezeichner ist der Software Heritage Persistent Identifier (SWHID), der auch mit Bezeichnern der Versionsverwaltung Git kompatibel ist (cf. Di Cosmo & Zacchiroli 2017, Software Heritage 2021). Dieser erlaubt eine Identifikation von Dateien, aber auch von *commits* und Programmcodeelementen. Das UV2-Annotationstool ist auf Projektebene (Quellcode und Container) über die Gitlab URL auffindbar. Eine URL ist zwar ein eindeutiger, aber kein persistenter Bezeichner, weshalb eine spätere Veröffentlichung mit einem DOI geplant ist. Die Kriterien F2-F4 beziehen sich auf Metadaten, die auch über Suchmaschinen gefunden werden sollen. Bezüglich ihrer Umsetzung gibt es keine Unterschiede

zwischen Softwareanwendungen und Forschungsdaten, sodass hier nicht näher auf diese Kriterien eingegangen wird.

2.4. Zugänglichkeit

Eine Softwareanwendung sollte nicht nur auffindbar, sondern auch zugänglich bzw. abrufbar sein. Zugänglichkeit meint hier in erster Linie technische Erreichbarkeit und nicht barrierefreie Nutzung der Softwareanwendung. Zugänglichkeit oder Abrufbarkeit ist garantiert, wenn ein standardisiertes Kommunikationsprotokoll verwendet wird, wie zum Beispiel HTTP. HTTP erfüllt die Kriterien A1.1 und A1.2.

(5) A. Accessible

The software, and its metadata, must be retrievable via standardized protocols.

A1. Software is retrievable by its identifier using a standardized communications protocol.

A1.1. The protocol is open, free, and universally implementable.

A1.2. The protocol allows for an authentication and authorization procedure, where necessary.

A2. Metadata are accessible, even when the software is no longer available.

(Chue Hong et al. 2021, 11-12, Hervorhebung im Original)

In Bezug auf das Problem des Langzeitbetriebs bzw. der Langzeitarchivierung von Softwareanwendungen ist das Kriterium A2 ein Kompromiss. So wird vorgeschlagen, dass die Metadaten, die eine Software beschreiben, zugänglich bleiben sollen, auch wenn die Software selbst nicht mehr abrufbar ist. Der Quellcode und die ausführbaren Dateien des UV2-Annotationstools sind über HTTP zugänglich. Das Tool kann als Webservice auf einem lokalen Rechner oder in einem Rechenzentrum betrieben werden.

3. Ausblick

Mit den FAIR-Prinzipien für Forschungssoftware geben Chue Hong et al. (2021) einen Leitfaden für die Entwicklung und Evaluation von Softwareanwendungen vor.

Wie im Beitrag aufgezeigt, werden die Prinzipien bei der Entwicklung des UV2-Annotationstools mit etablierten Verfahren der Softwaretechnik umgesetzt. Die Nutzung der Versionsverwaltung Git und eines Anbieters wie Gitlab oder Github ermöglicht es, schon zu Beginn der Entwicklung, die Prinzipien der Auffindbarkeit und Zugänglichkeit bzw. Abrufbarkeit umzusetzen. Wesentlich vielschichtiger ist die Umsetzung der Prinzipien der Interoperabilität und Wiederverwendbarkeit, weil hierbei technische, organisatorische und rechtliche Faktoren eine Rolle spielen. Um Wiederverwendbarkeit auf rechtlicher Ebene zu sichern, werden für das UV2-Annotationstool eine Open Source Lizenz gewählt und nur freie Software und offene Standards verwendet. Der Aufbau des Quellcodes ermöglicht eine Wiederverwendung auf der Ebene der Schichten (Client, Applikation, Datenbank)

sowie auf der Ebene der Bereiche Datenspeicherung, Sprachdaten, Nutzerverwaltung und der Bibliographie. Die Wiederverwendung auf der Ebene der Schichten wird durch die Implementierung der RESTful Webservices und die Verwendung der Spring Data JPA erleichtert. Die komponentenbasierte Architektur des Applikationsprojekts erlaubt die getrennte Wiederverwendung der oben genannten Bereiche. Hinsichtlich der zu verwendenden Datenformate und Programmiersprachen werden in den FAIR-Prinzipien geltende *community standards* erwähnt. Innerhalb der Romanistik können tabellenförmige Datenformate als Standard bezeichnet werden, der im UV2-Annotationstool berücksichtigt wird. Weitere fachspezifische Anforderungen an Forschungssoftware zu erörtern und zu formulieren, ist sicherlich ein längerer Prozess, den dieser Beitrag mit anstoßen möchte.

Bibliografie

AG DIGITALE ROMANISTIK. 2015. „Ergebnisse der Umfrage der AG Digitale Romanistik zur Langzeitarchivierung von digitalen Forschungsdaten für die Romanistik.“ *Mitteilungsheft des Deutschen Romanistenverbands e.V., Frühjahr 2015*, 36–40.
<http://deutscher-romanistenverband.de/wp-content/uploads/sites/14/Auswertung_Forschungsdaten-Umfrage.pdf>, 17.2.2022.

BECK, Kent. 2003. *Test-driven development: by example*. Boston: Addison-Wesley.

BRAY, Tim. 2017. „The JavaScript Object Notation (JSON) Data Interchange Format.“ STD 90, RFC 8259.
<https://doi.org/10.17487/RFC8259>, 17.2.2022.

BRAY, Tim, et al. (eds.). 2008. *Extensible Markup Language (XML) 1.0*. W3C Recommendation 26 November 2008.
<https://www.w3.org/TR/xml/>, 17.2.2022.

BRÜGGE, Bernd & Allen H. Dutoit. 2014. *Object-oriented software engineering using UML, Patterns, and Java.* Harlow, Essex: Pearson.

CHACON, Scott & Ben Straub. 2014. *Pro Git*. New York. Apress.

CHUE HONG, Neil P. et al. 2021. „FAIR Principles for Research Software (FAIR4RS Principles).“ Research Data Alliance.
<https://doi.org/10.15497/RDA00065>

CLARIN-D/SfS-Universität Tübingen. 2012. *WebLicht: Developer Manual.*
<https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/The_TCF_Format>, 17.2.2022.

DI COSMO, Roberto & Stefano Zacchiroli. 2017. „Software Heritage: Why and How to Preserve Software Source Code.“ *iPRES 2017: 14th International Conference on Digital Preservation*, Kyoto, Japan.
<https://hal.archives-ouvertes.fr/hal-01590958>, 24.9.2022

ECKHART Arnold et al. 2021. „Einfach FAIR. Geisteswissenschaftliches Arbeiten und nachhaltiges Publizieren von Forschungsdaten mit Git.“ Vortrag auf der Konferenz Forschungsdaten in den Geisteswissenschaften (FORGE 2021), Universität zu Köln, 08.-10.09.2021.

ECKART DE CASTILHO, R. et al. (2016): „A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures.“ *Proceedings of the LT4DH workshop at COLING 2016*, Osaka, Japan.
<https://aclanthology.org/W16-4011/>, 24.9.2022.

FIELDING, Roy Thomas. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. PhD Dissertation. University of California, Irvine.

HICKSON, Ian et al. 2022. *HTML Living Standard*.

<https://html.spec.whatwg.org/>, 17.2.2022.

JUNIT-TEAM. 2015-2022. *JUnit5*.
<https://junit.org/junit5/>, 17.2.2022.

KORPELA, Jukka. 2005. „Tab Separated Values (TSV): a format for tabular data exchange."
<https://jkorpela.fi/TSV.html>, 17.2.2022.

KREFELD, Thomas & Stephan Lücke. 2020. „54 Monate VerbaAlpina – auf dem Weg zur FAIRness." *Ladinia* XLIII, 139-156.

LESTER, Brent. 2018. *Continuous Integration Vs. Continuous Delivery Vs. Continuous Deployment: Distinguishing Terms and Understanding how their Implementation Methods and Tools Differ*. Sebastopol, CA: O'Reilly Media.

MCILROY, Malcolm Douglas. 1969. „Mass Produced Software Components." In *Software Engineering Concepts and Techniques*, ed. Naur, Peter, Brian Randell & Friedrich Ludwig Bauer, 88–98, Brussels: Scientific Affairs Division, NATO.

MEYER, Bertrand. 1997. *Object-oriented software construction*. Upper Saddle River, NJ: Prentice Hall PTR.

MICROSOFT 2012-2022. *TypeScript*.
<https://www.typescriptlang.org/>, 17.2.2022.

NORTH, Richard & Sergei Egorov et al. 2015-2022. *Testcontainers*.
<https://www.testcontainers.org/>, 17.2.2022.

ORACLE. 2021. *Java*.
<https://dev.java/>, 17.2.2022.

RESEARCH Data Alliance/FORCE11 Software Source Code Identification WG et al. 2020. „Use cases and identifier schemes for persistent software source code identification (V1.1)." Research Data Alliance.
<https://doi.org/10.15497/RDA00053>, 24.9.2022.

RICHARDS, Mark & Neal Ford. 2020. *Handbuch moderner Softwarearchitektur* (Übersetzung von Jorgen W. Lang). Heidelberg: dpunkt.

SHAFRANOVICH, Yakov. 2005. „Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC Editor, RFC 4180.
<https://doi.org/10.17487/RFC4180>, 17.2.2022.

SOFTWARE HERITAGE. 2021. *Development Documentation: SoftWare Heritage persistent IDentifiers (SWHIDs).*
<https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html>, 17.2.2022.

TEI CONSORTIUM. 2021. *TEI P5: Guidelines for Electronic Text Encoding and Interchange* (v4.3.0). Zenodo.
<https://doi.org/10.5281/zenodo.5347789>, 17.2.2022.

THE APACHE SOFTWARE FOUNDATION. 2022. *Apache Maven Project*.
<https://maven.apache.org/index.html>, 17.02.2022.

THE SPRING TEAM. 2022. *Spring Boot Framework*.
<https://spring.io/projects/spring-boot>, 17.2.2022.

THE WEBANNO TEAM. 2021. *WebAnno User Guide*, version 3.6.11.
<https://webanno.github.io/webanno/releases/3.6.11/docs/user-guide.html#sect_webannotsv>, 17.2.2022.

WILKINSON, Mark D. et al. 2016: „The FAIR Guiding Principles for scientific data management and stewardship." *Science Data* 3 (160018).
<https://doi.org/10.1038/sdata.2016.18>, 24.9.2022.

W3C OWL WORKING GROUP. 2012. *OWL 2 Web Ontology Language. Document Overview* (Second Edition). W3C Recommendation 11 December 2012.
<https://www.w3.org/TR/owl-overview/>, 17.2.2022.

ZIPSER, Florian & Laurant Romary. 2010. „A model oriented approach to the mapping of annotation formats using standards." *Proceedings of the*

Workshop on Language Resource and Language Technology Standards, LREC 2010. Malta.
<http://hal.archives-ouvertes.fr/inria-00527799/en/>, 17.2.2022.

Zusammenfassung

Der Beitrag stellt die FAIR-Prinzipien für Forschungssoftware nach Chue Hong et al. (2021) vor und diskutiert ihre Umsetzung am Beispiel des UV2-Annotationstools. Hierbei werden die umgesetzten Entwicklungsstrategien des Continuous Integration/Continuous Delivery und des test-driven development, die komponentenbasierte Architektur und die thematische Strukturierung des Quellcodes sowie die Art des ausführbaren Artefakts (Docker Container) vorgestellt und bezüglich der FAIR-Prinzipien bewertet.

Abstract

The article introduces the FAIR principles for research software, as proposed in Chue Hong et al. (2021), and discusses their realisation by means of the UV2 annotation tool. We examine the following aspects of the tool's implementation: the component-based architecture, the structuring of the source code, the type of the executable artefact (docker container), and the way the development strategies Continuous Integration/Continuous Delivery and test-driven development are realized.