Entwicklung Python Script zum Prüfen und Korrigieren von RDF Files

Leistungsnachweis 2

von

Andrés Baumeler

Telefon: 076 443 04 71, E-Mail: andres@baumeler.dev

Alte Riedikerstrasse 5c, 8610 Uster

Betreuer

Ismail Prada

Universität Zürich, Philosophische Fakultät /
Zentralbibliothek Zürich

CAS Datenmanagement und Informationstechnologien

Inhaltsverzeichnis

Kapitel 1

Einleitung

Banken sind gesetzlich verpflichtet gewisse Dokumente aus ihren Geschäftstätigkeiten aufzubewahren. Um die aufzubewahrenden Dokumente zentral zu verwalten werden digitale Archivsysteme eingesetzt. Ein solches System ermöglicht es alle in einer Bank produzierten Dokumente im Überblick zu behalten und den Lebenszyklus der Dokumente zu verwalten. Das Archivsystem ist in der Lage Dokumente und Metadaten aus einer Vielzahl an unterschiedlichen Quellen entgegenzunehmen. Während die akzeptierten Formate zwar klar definiert sind, ergeben sich beim Betrieb eines solchen Archivsystems immer wieder Herausforderungen mit Dokumenten welche die Anforderungen an den Archivierungsprozess nicht erfüllen. Im Rahmen dieser Arbeit wurde ein Python Script entwickelt, welches den Betreiber einer Archivlösung dabei unterstützen kann solche Dokumente zu prüfen.

Konkret behandelt das in dieser Arbeit entwickelte Python Script Files welche Metadaten im Ressource Description Format (RDF) enthalten. Die Files sind in der eXtensible Markup Language (XML) strukturiert. In dieser Arbeit wird nur das Archivsystem Hyper Suite 5 (HS5) der Firma IMTF[1] betrachtet.

[1] https://imtf.com/

1.1 Hintergrund

In einer Bank gibt es in der Regel eine Vielzahl an Systemen welche Aufbewahrungspflichtige Dokumente produzieren. Die Aufbewahrung der Dokumente wird zentral in einem Archivsystem durchgeführt. Dadurch ergibt sich der Bedarf für Schnittstellen von produzierenden Systemen zum Archivsystem. HS5 bietet dazu die Möglichkeit Dokumente über eine Filebasierte Schnittstelle als RDF und PDF Paar entgegenzunehmen. Das PDF enthält das eigentliche Dokument während das RDF File die Metadaten zum Dokument enthält. Der Import dieser File Paare erfolgt aus einem überwachten Ordner auf dem Filesystem des Archivservers. Sobald dort ein File Paar abgelegt wird, prüft das Archivsystem die Dokumente. Erfüllt das angelieferte File Paar die Anforderungen für die Archivierung wird das PDF archiviert. Bei der Archivierung werden die benötigten Metadaten aus dem RDF gelesen und in der Archiv Datenbank gespeichert. Das PDF wird auf eine Write Once Read Many (WORM) Storage geschrieben, um sicherzustellen, dass das Dokument selbst nicht mehr verändert werden kann. Das originale RDF File wird einige Tage nach erfolgreicher Archivierung des PDFs gelöscht.

Da die Dokumente auf eine WORM Storage Lösung geschrieben werden, ist ein Löschen vor dem Ablauf der Aufbewahrungsfrist nicht möglich. Aus diesem Grund müssen die gelieferten Metadaten nicht nur syntaktisch, sondern auch inhaltlich korrekt sein. Das Archivsystem führt aus diesem Grund eine Datenbank mit Stammdaten der Bank. Darin enthalten sind etwa Kunden- und Kontonummern. Vor der Archivierung werden die im RDF angelieferten Dokumente gegen diese Datenbank geprüft. Wird beispielsweise eine Kontonummer in den Metadaten angegeben, welche dem Archiv nicht bekannt ist, wird das Dokument nicht archiviert. Für Dokumente, welche die Anforderung an die Archivierung nicht erfüllen, wird das RDF und PDF Paar zusammen mit einem kurzen Fehlerbeschrieb in ein Verzeichnis geschrieben. Dieses sogenannte failed Verzeichnis wird überwacht und die dort abgelegten Dokumente werden regelmässig durch die für den Betrieb des Archivs verantwortliche Person geprüft.

1.2 Problemstellung

Die Prüfung von Dokumenten im failed Verzeichnis erfolgt von Hand und nimmt pro Dokument einiges an Zeit in Anspruch. Die Dokumente müssen auf der Commandline betrachtet werden da keine grafische Benutzeroberfläche verfügbar ist. Bei der Prüfung der fehlgeschlagenen Dokumente wird zunächst geprüft, ob das angelieferte RDF korrekt formatiert ist (stimmt das Encoding?, gibt es offene XML Elemente?). Anschliessend wird der Fehlerbeschrieb geprüft. Dieses Textfile wird vom Archivsystem geschrieben und enthält Hinweise darauf wieso die Archivierung nicht möglich war.

Die Anlieferung von Dokumenten kann entweder als einzel Dokumente oder als Massenverarbeitung erfolgen. Bei der Massenverarbeitung wird ein sogenanntes Job-File geliefert welches die Metadaten für bis zu 10'000 PDF Dateien beschreibt. Bei der Archivierung eines Jobs

Wenn ein Dokument im failed Verzeichnis liegt, kann es dafür mehrere Gründe geben. Die im täglichen Betrieb an häufigsten auftretenden Probleme sind die folgenden:

- Metadaten sind nicht vollständig (z.B. keine Kontonummer angegeben)
- Metadaten sind im falschen Format (z.B. Kontonummer hat zu wenig Stellen)
- Metadaten waren dem Archiv zum Zeitpunkt der Prüfung nicht bekannt (z.B. Kontonummer ist nicht in Archiv Datenbank)
- Ein PDF welches im RDF erwähnt wird, wurde nicht geliefert.
- Metadaten sind nicht korrekt formatiert (z.B. RDF enthält nicht geschlossene Elemente)

Nicht vollständige Metadaten bzw. Metadaten im falschen Format können nicht automatisch geprüft werden und müssen weiterhin manuell abgeklärt werden. Grund hierfür ist, dass die Verantwortlichkeit über die Metadaten beim anliefernden System liegt. Hier ist eine persönliche Abstimmung der jeweiligen Applikationsverantwortlichen notwendig. Eine gewisse Automatisierung ist aber möglich, indem File Paare mit unvollständigen oder falsch

formatierten Metadaten in einen anderen Ordner verschoben werden, ohne dass die Files manuell ausgewählt und kopiert werden müssen. Unterstützung der manuellen Prüfung ist auch möglich indem der Archivierungstatus aller in einem Job angelieferten PDFs automatisch geprüft wird. Oft sind bei einem Job von 1000 Dokumenten nur 2-3 Dokumente mit fehlerhaften Metadaten dabei. Das System verschiebt aber den ganzen Job ins failed Verzeichnis und überlässt dem Betreiber das Prüfen, welche Dokumente konkret nicht archiviert werden konnten.

Der dritte Punkt (Metadaten nicht in Archiv Stammdaten) kann automatisiert werden. Aufgrund der asynchronen Verarbeitung der Dokumente und Stammdaten, kommt es immer wieder vor, dass Dokumente geliefert werden, bevor die Stammdaten dafür im Archiv sind. Das Archivsystem archiviert solche Dokumente nicht, da diese Dokumente ansonsten nicht mehr aufgefunden werden könnte. Wurde verifiziert, dass die Stammdaten in die Archivdatenbank geladen wurden, kann der Archivierungsprozess für das Dokument nochmals gestartet werden. Wird ein ganzer Job geprüft muss für eine erneute Archivierung der fehlgeschlagenen Dokumente ein neues Job RDF File erstellt werden (delta RDF). Dieser Vorang besteht heute durch manuelles Kopieren der Job Header Elemente und der Metadaten Elemente der betroffenen Dokumente in ein neues RDF File. Dieses zusammenbauen eines delta RDF könnte automatisiert werden. Die automatische Prüfung ob Metadaten in der Archivdatenbank bekannt sind, war der Hauptpunkt, welcher durch das Script gelöst werden sollte. Dieser Prozess nimmt heute viel Zeit in Anspruch, da diese Fehler einerseits relativ häufig vorkommen und die Prüfung sowie Behebung relativ aufwändig sind. Für die Prüfung der Metadaten müssen die RDF Files auf der Commandline betrachtet werden während die Metadaten per manuellem SQL-Query in einem 3. System geprüft werden.

Der vierte Punkt (PDF nicht geliefert obwohl im RDF erwähnt) kommt eher selten vor. Eine Prüfung ist aber technisch einfach möglich, deshalb wirde dieser Punkt auch aufgenommen. Die Prüfung ob alle PDFs vorhanden sind bietet eine zusätzliche Sichereheit beim Verschieben von RDF Files da so nur komplete Packete aus RDF und PDF verschoben werden.

Der fünfte Punkt (RDF nicht korrekt Formatiert) kommt in der Praxis

ebenfalls nicht so häufig vor, aber wenn so ein Fall auftritt, kann die manuelle Prüfung aufwändig sein. Da korrekte RDFs für die automatische Verarbeitung aber eine Voraussetzung sind, muss dieser Punkt auch geprüft werden.

Kapitel 2

Hauptteil

2.1 Lösungsansatz

Als Lösung für das beschriebene Problem sollte ein Script entwickelt werden, welches die im RDF File gelieferten Metadaten gegen die Archivdatenbank prüft. Als Unterstützung sollte das Script auch anzeigen, wenn die Syntax der Metadatenfiles nicht korrekt ist (kein gültiges XML). Als weitere Anforderung sollte das Script auch in der Lage sein einen ganzen Job zu prüfen und Aussagen über den Archivierungsstatus aller im Job gelieferten Dokumente geben. Sind Dokumente in einem Job noch nicht archiviert soll das Script automatisch ein sogenanntes Delta-RDF erstellen. Darin sind alle Informationen aus dem original Job enthalten aber nur die Metadaten für die PDFs welche nochmals archiviert werden sollen. Wenn ein File Paar oder ein Job durch das Script geprüft wurden können diese entweder nochmals an die Archivierung übgergeben werden oder für eine weiter manuelle Analyse in einen anderen Ordner auf dem Filesystem verschoben werden. Beim Verschieben ist zu beachten, dass immer alle im RDF referenzierten PDF Files mit verschoben werden. Das Script soll beim Verschieben automatisch prüfen ob alle notwendigen Files vorhanden sind und diese zusammen verschieben.

Zur Umsetzung wurde Python gewählt. Grund für die Wahl von Python ist, dass eine entsprechende Laufzeitumgebung auf dem Zielsystem bereits vorhanden ist und die Sprache für den Umgang mit Dokumenten im XML

Format sowie bei der Darstellung gegenüber Alternativen wie Shell Script einige Annehmlichkeiten bietet.

Das Script sollte ein Interface bieten welches auch von Benutzenden verwendet werden kann welche sich nicht täglich mit dieser Problematik befassen. Ein Commandline Script mit mehreren Flags und Optionen erschien mir dafür nicht geeignet. Deshalb habe ich mich entschieden, dass Script mit einem Terminal UI zu versehen. So können den Benutzenden Informationen ansprechend und effizient präsentiert werden. Für das Terminal UI wurde das Textual Framwork[1] verwendet. Dieses Framwork bietet bereits viele vorgefertigte Komponenten für ein Terminal UI an wie etwa Listen und Buttons.

Ziel war es ein Script zu entwickeln welches die folgenden Anforderungen erfüllt:

1. Automatisches Prüfen von Metadaten gegen Archiv Datenbank
2. Prüfen und Verschieben von RDF + PDF Files in die entsprechenden Ordner für die weitere Verarbeitung
3. Als terminal UI Applikation verwendbar.

Da das Script auf meinem privaten Gerät entwickelt wurde war ein Zugriff auf die Unternehmensinternen Entwicklungsysteme nicht möglich. Aus diesem Grund wurden die benötigten Tabellen der Archiv Datenbank in einer lokalen Postgress Datenbank nachgestellt. Die Tabellen wurden dann mit erfundenen Daten gefüllt. Damit die Test- und Entwicklungsumgebung weitgehendst automatisiert bereit gestellt werden kann wurden docker-compose und Shell Scripts angelegt.

2.2 Ergebnis

Das Ergebnis besteht aus einem Python Script welches über ein Terminal User Interface (TUI) verfügt. Dadurch können die fehlerhaften Metadatenfiles gleich neben den möglichen Aktionen angezeigt werden. Die Behandlung

[1] https://textual.textualize.io/

von fehlgeschlagenen Dokumenten ist somit auch für weniger erfahrene Benutzende möglich. Ein Arbeiten mit mehreren SSH Sessions und 3. Werkzeugen für den Datenbank Zugriff entfallen durch das TUI.

Das Python Script erfüllt die von mir gesetzten Anforderungen grösstenteils. Das Script kann RDF Files anzeigen und bietet die Möglichkeit, die in den RDF enthaltenen Metadaten gegen die Archiv Datenbank zu prüfen. Je nach Prüfergebniss stehen verschiedene Optionen offen: Ein RDF kann zusammen mit allen referenzierten PDF Files in den Input Order zur erneuten Verarbeitung oder in den Wait Ordner für eine weitere Prüfung verschoben werden.

Die Pfade und Datenbank Verbindungen welche das Script verwendet, können über eine externe Konfigurationsdatei (*config.ini*) konfiguriert werden. So lässt sich das Script leicht auf verschiedenen Umgebungen verwenden.

2.3 Dokumentation

Die aktuellste technische Dokumentation sowie eine Anleitung zum Aufsetzen einer Referenzumgebung ist in den Files *start.md* sowie *README.md* enthalten. Der Source Code für das Script sowie die zugehörige Dokumentation ist auf GitHub[2] verfügbar. Das Script benötigt mindestens Python 3.11 sowie die im requirements.txt festgehaltenen Pakete.

Das Script zeigt nach dem Start automatisch den Inhalt von textitstart.md an und bietet dem Nutzenden so eine Hilfestellung für die Verwendung an der Stelle wo diese benötigt wird.

[2]https://github.com/abaumeler/CAS_DMIT/tree/main/ln02

Kapitel 3

Schluss

3.1 Offene Punkte & Ausblick

Es gibt diverse Punkte, welche noch verbessert werden könnten oder Features welche noch implementiert werden könnten. Ein solches Feature wäre etwa die automatische Verarbeitung zu ermöglichen. Dazu müsste das Script erweitert werden so dass commandline Argumente gelesen werden können. Ebenfalls nützlich wäre es das Log welches in der Applikation angezeigt wird in ein eigenes File zu schreiben, damit auch bei einer automatisierten Verarbeitung im Nachhinein festgestellt werden kann was das Script genau gemacht hat. Weiter wäre es wohl sinnvoll automatisierte Unit-Tests einzubauen, um die Weiterentwicklung zu vereinfachen. Sollte die Codebasis noch weiter wachsen muss gegebenenfalls auch darüber nachgedacht werden den Code auf mehrere Files zu verteilen, um die Lesbarkeit und Wartbarkeit zu erhöhen.

Als nächsten Schritt werde ich das Script auf eine Unternehmensinterne Entwicklungsumgebung übertragen und für die Verwendung auf dem Zielsystem anpassen. Für die Verwendung auf dem Zielsystem muss der Code welcher mit der Archivdatenbank kommuniziert für die Zusammenarbeit mit einer Oracle Datenbank angepasst werden.

3.2 Fazit und Reflexion

Meine wichtigsten Anforderungen an das Script konnte ich realisieren. Gewisse Features wie etwa das Generieren eines Delta-RDF File konnten aus Zeitgründen nicht umgesetzt werden.

Das Projekt hat mir die Gelegenheit gegeben meine Python, XML und Shell Kenntnisse zu vertiefen sowie etwas Erfahrungen mit dem Bereitstellen von Entwicklungsinfrastruktur mittels Docker und Docker-Compose zu sammeln. Im Bereich Python konnte ich gerade in den Bereichen Filehandling, asynchrone Verarbeitung und TUI viel dazulernen. Der Einsatz eines Frameworks für das TUI hatte sehr viele Vorteile aber auch eine gewisse Lernkurve. Insgesammt hat sich der Einsatz aber gelohnt, da ich dadurch einiges über DOM und Eventhandling lernen konnte und als Ergebniss ein Script entstanden ist welches sich einfach einsetzen lässt.

Eine Herausforderung war zu Beginn des Projekts, dass die Entwicklung nicht auf der Umgebung statt finden konnte wo das Script später verwendet werden sollte. Aus diesem Grund mussten zuerst die benötigten Tabellen und Ordner Strukturen nachgestellt werden. Mittels Docker-Compose und einem Shell Script konnte hier aber eine angenehme Lösung gefunden werden. Eine weitere Herausforderung waren die Testdaten. Aus Gründen des Datenschutz mussten die Testdaten manuell erstellt werden was einiges an Zeit in Anspruch nahm. Weiter musste ein Shell Script geschrieben werden welches die Testdaten nach einem Testdurchlauf wieder in die korrekten Verzeichnisse verschiebt um die Umgebung so für weitere Tests bereit zu machen.

Aus dem Grund, dass ich das Projekt losgelöst vom täglichen Betrieb umsetzen konnte, war ich in der Lage meine Qualitätsansprüche zu erhöhen und Features einzubauen, welche ich nicht eingebaut hätte, wenn ich das Script auf Arbeitszeit entwickelt hätte. Ein TUI bringt zwar viele Vorteile und Annehmlichkeiten aber doch auch einiges an Komplexität in das Script. Dazu hätte mir im Betrieb die Zeit gefehlt und ich hätte das Script wohl als reines Commandline Script umgesetzt.

Abkürzungen

HS5	Hyper Suite 5
RDF	Ressource Description Format
WORM	Write Once Read Many
XML	eXtensible Markup Language
TUI	Terminal User Interface