Geschäftsprozesse in Low-Code Plattformen mit einem SAP-ECC Backend

Vergleich zweier Plattformen auf Basis eines DMS-Prozesses

Daniel Degenhardt
Hochschule Fulda
Leipziger Str. 123
36037 Fulda
Daniel.Degenhardt@cs.hs-fulda.de

Norbert Ketterer
Hochschule Fulda
Leipziger Str. 123
36037 Fulda
Norbert.Ketterer@cs.hs-fulda.de

ABSTRACT

In dieser Arbeit wird untersucht, in wieweit sich Prozesse des Dokumentenmanagements in zwei Low-Code Plattformen abbilden lassen. Die eigentlichen Funktionen des Dokumentenmanagements werden dabei über einen Backend in Form eines SAP-ERP Systems mittels SOAP- und REST-Services implementiert. Insbesondere soll die Art der technischen Anbindung zwischen den jeweiligen Plattformen und dem Backend untersucht werden; speziell das Zusammenspiel zwischen Funktionen in Front- und Backend und die Verwendbarkeit der verschiedenen Services. Ein weiterer Aspekt der Fragestellung ist auch, wie die Low-Code Plattformen die Entwicklung der Funktionen des Dokumentenmanagements und deren Einbettung in den Geschäftsprozess unterstützen. Diese Frage soll durch einen Prototypen validiert werden. Als Untersuchungsgrundlage dienen die beiden Low-Code Plattformen „Bizagi“ und „Mendix“. Es wird in der Untersuchung ein Prozess des Dokumentenmanagements betrachtet, da hier die Parametrisierung der Aufrufe zum Backend besonders komplex ist, denn es sind statt der sonst üblichen, durch einfache Variablen oder Listen darstellbaren Parameter (z.B. Werke, Materialnummern und Kunden), die Identifikation der Dokumente aber auch die Originale der Dokumente zu übertragen.

Keywords

Dokumentenmanagement, SAP, Low-Code, Bizagi, Mendix, Digitalisierung Geschäftsprozesse, REST, SOAP

EINLEITUNG

Für einen kontinuierlichen unternehmerischen Erfolg ist die Nachvollziehbarkeit aller Information, die im Zuge der Durchführung der Geschäftsprozesse anfällt oder von diesen als Referenz benötigt wird, unabdingbar. In diesem Zusammenhang definiert sich ein Dokument als Trägereinheit von inhaltlich zusammengehörender Information, die sich in unterschiedlichen Strukturierungsgraden ausdrückt und als Nachweis oder Beleg für betriebliche Vorgänge dient [1] (S. 27f). Angesichts der stetig wachsenden Komplexität von betriebswirtschaftlichen Prozessen sind Unternehmen dazu veranlasst, sich mit dem effizienten Management ihrer Dokumente zu beschäftigen. Beispiele von Prozessen sind etwa PLM-Prozesse, in denen eine Dokumentation zum Zweck einer stetigen Produktqualität oder aufgrund von gesetzlichen Vorschriften wie Aufbewahrungsfristen für bestimmte Geschäftsdokumente existiert.

Gemäß Götzer behandelt das Dokumentenmanagement primär Verwaltungsfunktionen der Dokumente [1] (S. 15 ff), nicht die Inhalte der Dokumente selbst. Als Teilgebiet des Informationsmanagements zeichnet sich das Dokumentenmanagement durch eine starke Wechselwirkung mit anderen unternehmerischen Prozessen aus und erstreckt sich damit in Form von unterstützenden Prozessen über die gesamte Wertschöpfungskette des Unternehmens. Im Zuge der Einführung von Dokumenten-Management-Organisationen gilt es von den Fachabteilungen zu ermitteln, was für Dokumente auf welche Art und Weise verwaltet werden müssen und in welchem Zusammenhang diese mit den einzelnen Geschäftsprozessen stehen; danach kann festgelegt werden, wie dies durch den Einsatz eines Informationssystems unterstützt werden soll.

In dieser Arbeit werden die Dienste des Dokumentenmanagementsystems (genauer CA-DMS/ Cross - Application Components - DMS) des ERP-Systems in Services gebündelt, die von parallel zum ERP-System laufenden Applikationen genutzt werden können, die mit Hilfe der Low-Code Plattformen implementiert werden. Dieses Szenario betrifft also die Digitalisierung eines speziellen, einfachen Geschäftsprozesses mit Hilfe einer Low-Code Plattform, wobei dieser Prozess trotzdem Services des ERP-Backends nutzen soll, um auf relevante Information Zugriff zu haben. Innerhalb des Dokumentenmanagements stellen sich hierbei besondere Herausforderungen - speziell die Verwaltung der Originale.

PROBLEMDEFINITION

Prozesse im Dokumentenmanagement

Die betriebswirtschaftlichen Prozesse des Dokumentenmanagements lassen sich in zwei Ebenen, nämlich **Metaprozesse** und **operative Prozesse** untergliedern. Die Metaprozesse definieren hierbei den Handlungsrahmen, in dem sich die operativen Prozesse ausprägen können. Dieser ist

stark an die Verfügbarkeit von Funktionen des Dokumentenmanagements gekoppelt, etwa

- der Festlegung der Dokumentenarten, der
- Definition der Verknüpfungslogik mit Objekten des Anwendungssystems, den verwendeten
- Statusnetzen und dem Bezug zum Klassensystem.

Abbildung 1: Dokumentenlebenszyklus nach Götzer [1], Kapitel 1

Dokumentenverwaltung im ERP-System

Die Architektur eines Dokumenten-Management-Systems (kurz „DMS“) besteht aus einer Anzahl von zueinander abhängigen Komponenten (Abbildung 2), die sich wiederum in eine Menge von Subkomponenten und -funktionen untergliedern lassen. Die Komponente „Eingang“ und „Klassifizierung“ muss gewährleisten, dass einerseits nur Dokumente dem System zugeführt werden können, die durch die Metaprozesse definiert sind, und dass diese darüber hinaus nach den Vorgaben der Dokumentenart ausreichend indiziert werden. Die Komponente „Ablage“ und „Archivierung“ stellt die Schnittstelle zum Transport von Nutzinhalt zwischen den Massenspeichersystemen (wie Dateisystemen, in der Abbildung als RAID- und File-System dargestellt) und dem DMS. Zur Eingrenzung des Ergebnisraums von Suchanfragen und zur Darstellung von einzelnen Dokumenten werden Funktionalitäten der „Recherche“ und der „Anzeigekomponente“ genutzt. Werden die Dokumenten im DMS auch physisch abgelegt und nicht nur verwaltet (also nur „Metainformation“ abgelegt), werden sie in besonderen Speicherstrukturen gespeichert. Dies sind speziell strukturierte Datenbanktabellen oder auch Dateisysteme.

Ein Beispiel eines Flusses der Dokumenteninformation durch das DMS zeigt Abbildung 3; das Dokument wird in Nutz- und Verwaltungsinformation aufgeteilt und in den verschiedenen Komponenten des DMS abgelegt. Dieser Fluss setzt den Dokumentenlebenszyklus nach Abbildung 1 um. Es ist im Rahmen der Fragestellung zu untersuchen, welche Funktionen im DMS des ERP-Systems genutzt werden können, um diesen Fluss zu implementieren und wie diese durch Services in den Low-Code Plattformen nutzbar sind.

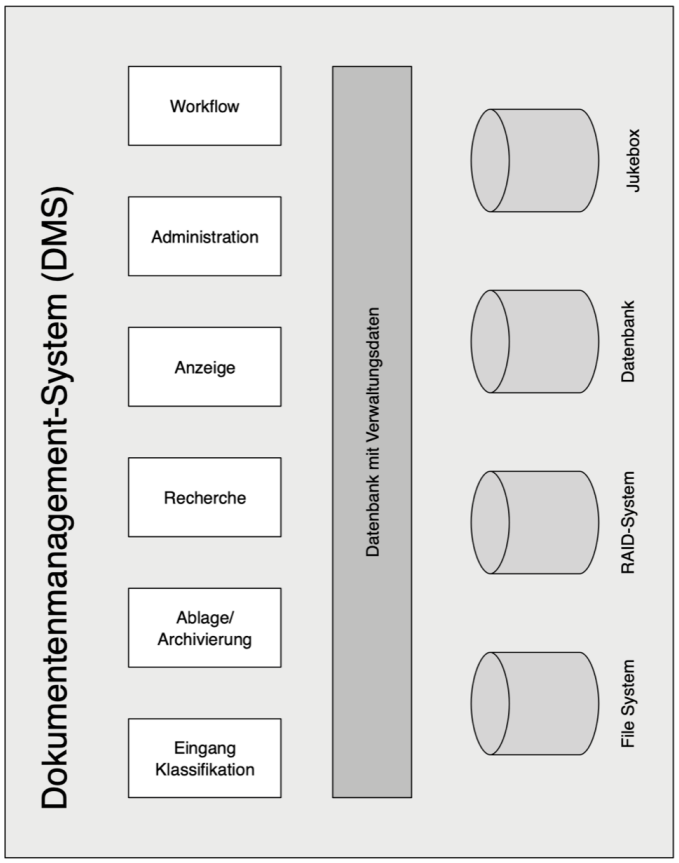

Abbildung 2: Architektur eines DMS - nach [2](S. 29)

Implementierung von Backend-Services im ERP-System

Um die Funktionen des Dokumentenmanagements durch die Low-Code Plattform ansprechen zu können, sind diese im ERP-System zu identifizieren, zu gruppieren und als Service zu veröffentlichen, dass sie sinnvoll durch die Low-Code Plattform angesprochen werden können. Eine wesentliche Frage wird hierbei sein, wie die Verwendung von aktuellen Kommunikationsmitteln, wie SOAP und REST die Implementierung im Backend und die Einbindung in die Low-Code Plattform beeinflussen.

Softwareentwicklung in den Low-Code Plattformen

Das Dokumentenmanagement ist eine Teilfunktion, die innerhalb eines digitalisierten Gesamtprozesses notwendig sein kann und im Falle dieser Untersuchung durch ein zentrales Backend bereitgestellt werden soll. Hierzu sind in der jeweiligen Low-Code Plattform die Geschäftsprozesse gemäß der individuellen Methodik der Plattform abzubilden, um dort dann die Funktionen des Dokumentenmanagements, die durch das Backend bereitgestellt werden, aufrufen zu können.

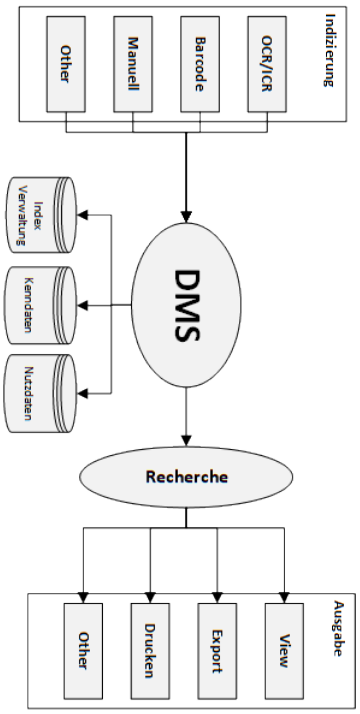

Abbildung 3: Dokumentenfluss innerhalb eines DMS - nach [1] (S. 164)

STAND DER TECHNIK

Architektur von SAP-DMS

Das Backend, welche als ERP-System in dieser Untersuchung die Funktionen des CA-DMS zur Verfügung stellt, wird zusammen mit der NetWeaver-Plattform ausgeliefert und bietet aufgrund der vollständigen Integration des CA-DMS in das ERP-System einen direkter Bezug zwischen den Dokumenten und den operativen Geschäftsprozessen; i.d.R. ist eine direkte Verknüpfung von Dokumenteninfosätzen und den durch die Geschäftsprozesse erzeugten Objekte möglich (etwa Kunden, Materialien, Aufträge). In diesem Kontext kann dann ein Dokument jegliche Form von speicherbarer Information darstellen, wie technische Zeichnungen, Grafiken, Programme oder einfach Textdokumente [3], die als Originale abgelegt werden können, die dann mit Dokumenteninfosätzen verknüpft sind. Diese Infosätze können selber wieder miteinander verknüpft oder auch klassifiziert werden (neben der Originalverknüpfung und Geschäftsobjektverknüpfung). Die Definition eines Dokuments beschränkt sich also nicht nur auf Dokumente in Papierform, sondern erweitert den Begriff auf beliebige Informationskonstellationen.

Bei Dokumenteninfosätze handelt es sich um Objekte, welche die Ausprägungen von Metadaten eines einzelnen Dokuments innerhalb der DMS-Datenbank abspeichern und somit eine Repräsentation des Nutzdokuments darstellen. Weiterhin definieren Dokumenteninfosätze diverse Schnittstellen zu internen Geschäftsobjekten (Materialien, Kunden, Lieferanten) und externen Diensten, über welche die Nutzdokumente in das Dateisystem abgelegt werden.

Zur Verwaltung von Dokumenten, die primär außerhalb des ERP-Systems erzeugt werden, eignet sich insbesondere der sogenannte **Knowledge Provider (KPRO)** als Vermittler zwischen dem DMS- und dem Dateisystem. Dieser greift auf Funktionen zurück, durch die eine bidirektionale Kommunikation mit dem ERP-System gewährleistet werden kann. Wie sich der Abbildung 4 entnehmen lässt, kommunizieren die einzelnen Applikationen mit der für sie zuständigen Komponente innerhalb des KPROs, der wiederum die Kommunikation mit dem Content-Server übernimmt.

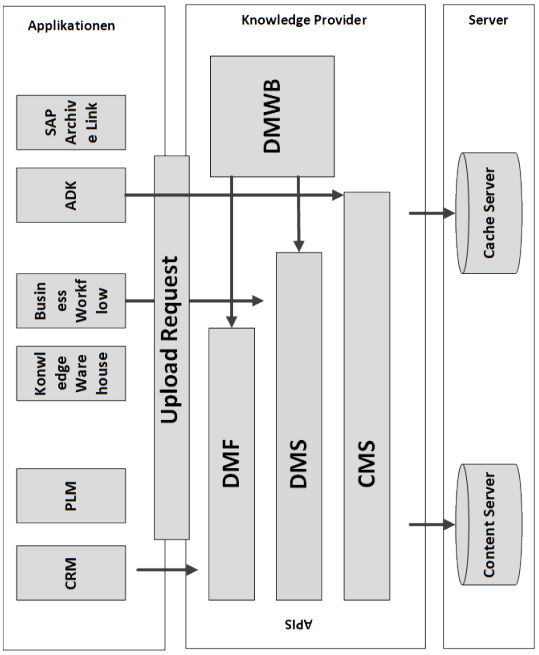

Abbildung 4: Architekturkomponenten SAP-DMS - nach [4]

Die Anlage eines Dokumenteninfosatzes erfordert intern im DMS als ersten Schritt das Definieren eines Primärschlüssels. Das System validiert die Einstellung der Dokumentenart und sichert damit zu, dass nur Dokumenteninfosätze mit einer gültigen Dokumentenart unter den speziellen Regeln der Dokumentenart angelegt werden können; etwa einem speziellem Statusnetz. Anschließend kann der Dokumenteninfosatz nach den zusätzlichen Kriterien der Dokumentenart genauer spezifiziert werden; etwa dem Einchecken der Originale, der Verknüpfung weiterer Objekte der Geschäftsprozesse, Klassifikationen, Hierarchisierungen, der Verwaltung der Status und von Signaturen - ein Beispiel der Anlage zeigt Abbildung 5.

Abbildung 5: Anlage eines Dokumentinfosatzes in der SAP-GUI

Es existiert ein OASIS-Standard auf Basis des SOAP-Protokolls (CMIS), der die Interoperabilität zwischen Enterprise-Content-Management-Systemen vereinheitlicht. Er wird zwar von einer der verwendeten Low-Code Plattformen unterstützt, aber nicht von dem ERP-System, weshalb er nicht weiter betrachtet werden soll.

Service-orientierte Architekturen

Die Geschäftsprozesse der Fachabteilungen lassen sich in einzelne Aktivitäten, die elementare Aufgabenblöcke repräsentieren, zerlegen. Es sollen deshalb Services erzeugt werden, die die Aktivitäten implementieren, soweit sie das ERP-System als Backend betreffen. Auf Basis der bestehenden Systemlandschaft lassen sich somit Services generieren, die ein Abbild der Aufgabenblöcke repräsentieren; einzelne Services können dann über die Low-Code Plattformen zu individuellen Applikationen innerhalb oder über Unternehmensgrenzen hinweg kombiniert werden.

SOAP und REST Web-Services

Ein Ansatz zur Realisierung einer Service Orientierten Architektur ist die Bereitstellung der zuvor beschriebenen „Aufgabenblöcke“ in Form von Web-Services. Diese bilden hierbei eine Abstraktionsschicht, die es ermöglicht, Funktionalitäten der eigenen Systemlandschaft unabhängig von der zugrundeliegenden Technologie plattformübergreifend durch den Einsatz des SOAP-Protokolls bereitzustellen, wie dies in Abbildung 6 skizziert ist; die Beschreibung der Schnittstelle erfolgt dabei durch die WSDL [5] (S.61).

Abbildung 6: Abstraktionsschicht von Web-Services

Eine WSDL-Datei stellt einen in XML spezifizierten Vertrag zwischen den Web-Service-Akteuren dar und beschreibt Anforderungen, die ein Dienstnutzer erfüllen muss, damit er eine bestimmte Funktion des Diensteanbieters in Anspruch nehmen kann.

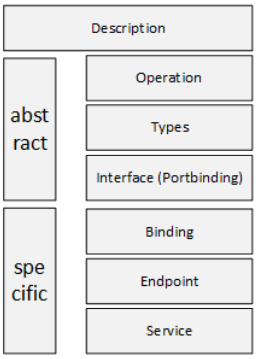

Abbildung 7: Aufbau einer WSDL-Datei nach Melzer, plus Types-Komponente [6] (S. 116)

Die WSDL-Datei kann somit als Metadaten-Dokument angesehen werden. Innerhalb der WSDL-Datei wird der Web-Service aus zwei Ebenen beschrieben, die sich in die Schnittstellenbeschreibung und die Implementierungsbeschreibung aufgliedern (Abbildung 7)). Die **abstrakte Ebene** beschreibt, was der Dienst anbietet und wie dieser implementiert ist; die **konkrete Ebene** beschreibt, wie der Dienst aufgerufen werden kann.

Ein weiterer relevanter Ansatz zur Realisierung von Web-Services, setzt auf dem REST- Architekturstil nach Roy Fielding auf. Dieser ermöglicht eine Kommunikation zwischen Diensten über das Internet hinweg, ohne die Notwendigkeit der SOAP- Abstraktionsschicht.

REST zielt darauf ab, lose gekoppelte, skalierbare und zugleich verteilte Systeme zu erstellen [7] (S. 16 f). Hierbei kommt kein spezielles Nachrichten-Austausch-Protokoll zum Übermitteln von Nutzdaten wie etwa SOAP zum Einsatz. Der Austausch von Nutzinhalten beschränkt sich auf das HTTP/S Protokoll. Der Dienstanbieter übermittelt die Nutzdaten eingebettet in der Zeichenkette des URL an den Dienst. Die Idee des REST-Ansatzes besagt, dass jede Art von beschreibbaren Daten eine Ressource darstellt, die durch eine URI eindeutig identifizierbar ist [7] (S. 7 ff). Ressourcen können durch die HTTP-Methoden (GET,POST,PUT,DELETE) im Sinne der CRUD-Operationen modifiziert werden. Im Gegensatz zum SOAP-Ansatz, der sich ausschließlich auf XML festlegt, kann eine Ressource im REST-Ansatz neben XML weitere Formate wie JSON, TXT, HTML usw. annehmen. Um mit einem REST basierten Service kommunizieren zu können, bedarf es keinem Service-Vertrag, wie es bei den SOAP-Services durch die WSDL verlangt wird. Die einfache Kenntnis des Service Endpunkts in Verbindung mit Authentifizierungsaspekten reicht aus, um eine Kommunikation zu ermöglichen.

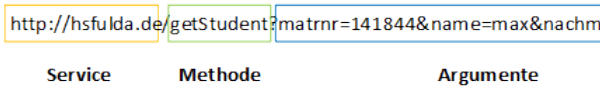

Abbildung 8: Aufruf eines fiktiven REST-Service

Der REST-Ansatz unterliegt keinerlei Standardisierung, wie es bei SOAP-Web-Services der Fall ist. Dies lässt sich darauf zurückführen, da es sich bei REST lediglich um ein Architekturparadigma handelt. Der Umstand bietet somit Interpretationsspielraum für die Ausgestaltung und Implementierung von Web-Services auf Basis der REST-Idee. Um diese Services dennoch nach qualitativen Kriterien beurteilen zu können, hat Richardson ein Reifegradmodell entwickelt, das eine Orientierungshilfe für die Beurteilung von REST-Services darstellt (Abbildung 9.

Auf Level 2 werden die HTTP-Operationen im Sinne der CRUD-Operationen genutzt, über welche die URI modifiziert wird. Eine Ressourcen-Repräsentation findet ausschließlich durch GET-Befehle und eine Ressourcen-Änderungen ausschließlich durch POST- Befehle statt - die CRUD Operationen werden auf Level 1 nicht verwendet. Auf Level 3 stellt der Dienstanbieter dem Dienstnutzer einzelne Funktionen und Information als „Hyper-Media-Control“ bereit. Der Client kann dadurch mit allen weiteren dem Dienst zugeordneten Ressourcen kommunizieren, ohne initial deren Lokalisation zu kennen. Dies lässt die Möglichkeit offen, einzelne andere Dienste/Ressourcen abzuändern ohne dass dadurch ein Problem beim Client auftritt. Der Client ist somit stets

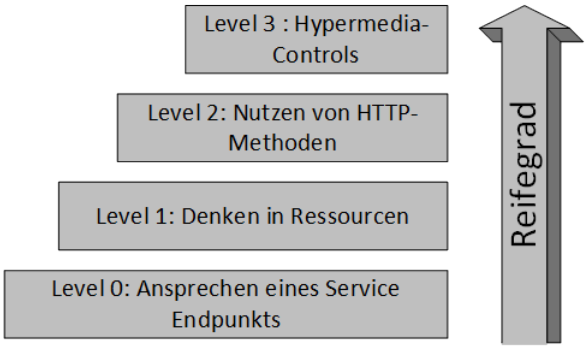

Abbildung 9: Richardson Maturity Model [8]

über den aktuellsten Ressourcenzustand informiert. Rest- und SOAP-basierte Services werden in Tabelle 1 kurz gegenübergestellt.

SOAP-Web-Services	REST-Web-Services
Erfordert eine client-seitige SOAP-Libary	Keine Library erforderlich
Alle Aktionen via HTTP-POST	Nutz CRUD-Operationen des HTTP-Protokolls
Kann zustandsbehaftet oder zustandslos sein	Ist zustandslos
N Ressourcen für N Aktionen	Eine Ressource für N Aktionen
Unterstützt nur XML	Unterstützt XML, JSON, TXT usw.
WSDL beschreibt sehr genau wie der Dienst aufgebaut ist	Keine Vertragsvereinbarungen notwendig

Table 1: SOAP vs. REST

Web-Services innerhalb von NetWeaver

In der NetWeaver-Plattform des Prototypen existiert ein Modul, welches das Empfangen und Versenden von HTTP-Nachrichten gewährleistet. Die technische Grundlage zur internetbasierten Kommunikation bildet der Internet-Communication-Manager und das Internet-Communication-Framework. Die beiden Dienste sind in der Lage, unterschiedliche Protokolle wie SMTP, HTTP/S zu verarbeiten. Unabhängig von dem gewählten Web-Service-Ansatz (SOAP/REST) ist stets die Kombination der beiden Dienste für die Nachrichtenverarbeitung zuständig. Eingehende HTTP-Anfragen können durch den ICM modifiziert und weitergeleitet werden. Nach der initialen Annahme der Nachricht wird diese an das ICF weitergereicht. Dieses entpackt die HTTP-Nachricht und reicht den Aufruf an die zuständigen Programme also die eigentlichen Dienste weiter ([9], S.958).

SOAP-Services können direkt erstellt werden; insbesondere mit Hilfe des eingebauten Web-Service-Wizards, der es ermöglicht, innerhalb von neun Schritten einen Funktionsbaustein web-fähig zu machen. Es muss hierzu im Wesentlichen der Service-Name und der Endpunkt, der die Geschäftslogik beinhaltet, angewählt sowie der Service konfiguriert werden (Zertifikate, Autorisierung). Der erstellte Service kann anschließend im Repository-Browser unter dem Punkt der Enterprise-Services gefunden werden. Damit der Service von außerhalb des NetWeaver-basierten Systems angesprochen werden kann, muss für diesen noch ein konkreter Eintrag im ICF hinterlegt werden. Die Kommunikation zwischen einem Dienstaufrufer und Dienstanbieter zeigt Abbildung 10: Eine eingehende Anfrage wird durch das Internet-Communication-Framework an die SOAP-Runtime weitergeleitet. Diese wird die SOAP-Nachricht interpretieren und den Nutzinhalt an den Funktionsbaustein weiterreichen, der daraufhin die Dienstleistung durchführt. Das Ergebnis des Aufrufs wird wiederum durch die Runtime in eine SOAP-Nachricht eingebettet und durch die Dienste des ICF und ICM in Form einer HTTP-Nachricht an den Dienstaufrufer zurückgesendet.

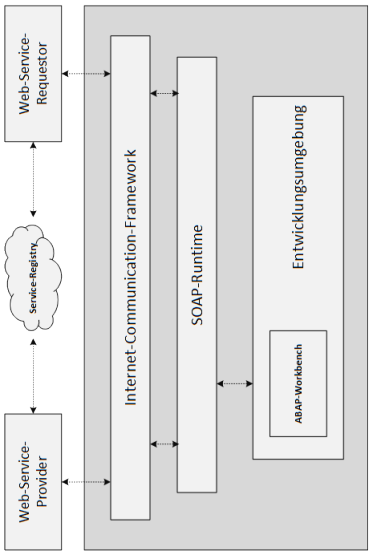

Abbildung 10: Schema der SOAP-basierten Nachrichtenverarbeitung [9] (S. 978)

Ein REST-basierter Service ist dagegen nicht so direkt in der verwendeten NetWeaver-Version aus einer Funktion erzeugbar, wie im Falle eines SOAP-Service. Deshalb wurde für die Untersuchung prototypisch eine ABAP-Klasse programmiert, die das Interface „IF_HTTP_EXTENSION“ implementiert. Das Interface verfügt ausschließlich über eine Methode „Handle_Request“, die gewährleistet, dass eine vom ICF übertragene HTTP-Nachricht durch die Klasse behandelt wird. Durch dieses Interface wird das ABAP-Programm in die Lage versetzt, HTTP-Nachrichten zu verarbeiten. Im ICF-Baum kann dann ein Service angelegt werden, der über eine URL ansprechbar ist. Dem Service wird dabei die ABAP-Klasse als Handler zugewiesen. Zur Laufzeit wird der ICF-Baum per Top-Down-Ansatz nach dem gewünschten Dienst abgesucht und es erfolgt eine dynamische Zuweisung der Handler-Klasse und die Übertragung der Nachricht an diese.

Low-Code Plattformen und Software Engineering

Durch die Zuhilfenahme von Low-Code Plattformen sollen Fachabteilungen befähigt werden, insbesondere für einfache Prozesse eigene Applikationen zur Unterstützung bei den anfallenden Geschäftsprozessinstanzen zu implementieren. Die hierfür notwendige Komplexitätsreduktion wird durch die Bereitstellung von Modellierungswerkzeugen gewährleistet.

Die Plattformen ermöglichen es den Unternehmen, schnell durch Modellierung Software zu erstellen, können aber auch angebundene Geschäftsanwendungen (wie eine Landschaft von OLTP-Systemen) mit einem einheitlichen User-Interface belegen und diese zusätzlich um Spezialfunktionen ergänzen.

Ausgehend von einem fachlichen Gegenstandsbereich werden Elemente und Beziehungen beschrieben, die durch eine formale Notation (z.B. UML- oder BPMN) spezifiziert werden. In mehreren Schritten werden die erzeugten Modelle anschließend auf eine darunter liegende Ebene transformiert und letztlich von den Plattformen in Quellcode übersetzt (in Anlehnung an [10]).

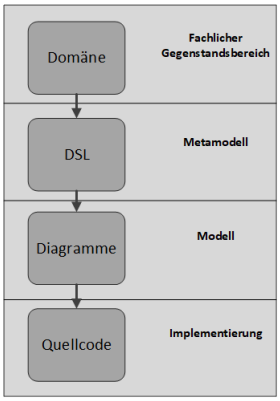

Abbildung 11: Vorgehensweise der modellbasierten Software-Entwicklung [10] (S.19)

Typisch gegenüber der herkömmlichen Software-Entwicklung ist, dass die Plattformen bereits Lösungen für Anforderungen mitliefern, die in vielen Software-Projekten eine Rolle spielen. So gibt es oftmals eigene Module für die Benutzerverwaltung, welche durch die Anbieter stetig gewartet und verbessert werden. Darüber hinaus müssen sich Anwender nicht mit der Installation der mitunter komplexen Infrastrukturen für Entwicklung, Deployment und Wartung befassen. Die Verwendung dieser Angebote resultiert in schnelleren Entwicklungszyklen und bringt Kostenvorteile für Nutzer der Plattformen. Die Plattformen bestehen aus vorgefertigten Komponenten, die eine klar definierte Schnittstelle besitzen und aus denen die Applikationen zusammengesetzt werden. Soll eine Komponente in die eigene Applikation eingebunden werden, ist es nicht erforderlich, ein Verständnis für die zugrundeliegende Logik der Komponente zu besitzen. Es reicht aus, wenn die Schnittstellenanforderungen zur Erbringung der Dienstleistung erfüllt werden. Die Komponenten übernehmen beispielsweise Aufgaben zur Integration der Bestandteile der Applikation im Sinne eines Service-Busses, der Datenbankanbindung, der Benutzerschnittstelle, der Integration mit Um-Systemen aber auch Spezialaufgaben, wie das Bereitstellen eines Mail-Clients.

Low-Code Plattform Bizagi

Bizagi ist ein Business-Process-Management-System, das es Anwendern ermöglicht, direkt auf Basis von **Geschäftsprozessen** Applikationen zu erstellen. Hierbei repräsentieren diese digitale Abbilder der Geschäftsprozesse in BPMN und zielen darauf ab, die Zusammenarbeit von menschlichen und maschinellen Akteuren zu organisieren. Innerhalb des Bizagi-Modelers besteht die Möglichkeit, Geschäftsprozesse zu modellieren. Das Bizagi-Studio ermöglicht ebenfalls eine Prozessmodellierung (mit Einschränkungen, z.B. bei der Verwendung von Pools/ Messages), aber dafür dann die Definition der Datenmodellebene und der Front-End-Ebene sowie die Anreicherung der Geschäftsprozesse um Geschäftsregeln und technische Komponenten. Der Aufbau der Plattform ist analog zur Architektur einer Geschäftsanwendung - besteht also aus Frontend, Geschäftsregeln und Datenbankanbindung (Abbildung 12).

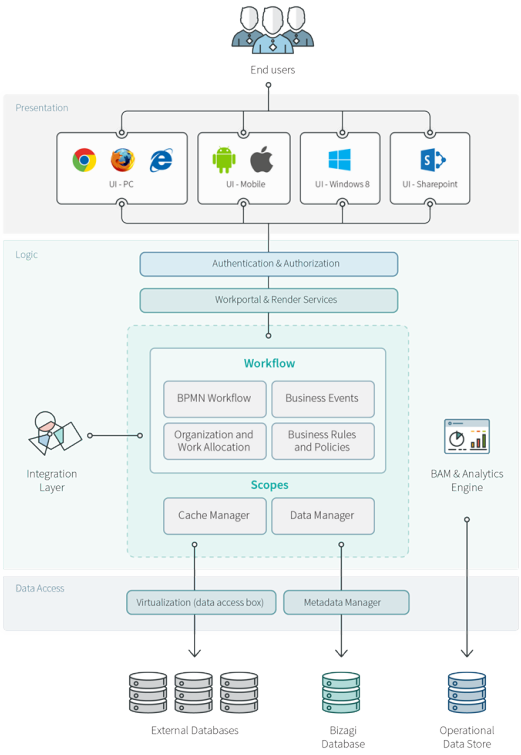

Abbildung 12: Architekturmodell von Bizagi [11]

Das Vorgehensmodell von Bizagi besteht aus der iterativen Anwendung der folgenden Schritte:

1. Erstellung eines **Geschäftsprozessmodells**,
2. Erstellung eines **Datenmodells**,
3. Erstellung des **Front-End-Modells**,
4. Erstellung von **Geschäftsregeln**.

Das Geschäftsprozessmodell wird mit Hilfe von BPMN und das Datenmodell wird mit Hilfe eines E-R-Diagramms dargestellt. Somit entsprechen diese beiden Schritte dem Standardvorgehen innerhalb einer prozessbezogenen Systementwicklung. Die Gestaltung des Front-Ends und der Geschäftsregeln ist etwas toolspezifischer, weshalb hier nur auf die letzten beiden Schritte eingegangen werden soll.

Die Komponenten der FORMS-Umgebung erstellen das Modell für die Präsentationsebene und verknüpfen dieses mit der Geschäftsebene und dem Datenmodell. Dabei stellen Geschäftsprozessaktivitäten des Typs „User-Task“ das In-

terface zu den Anwendern dar und können mit Front-End-Elementen angereichert werden.

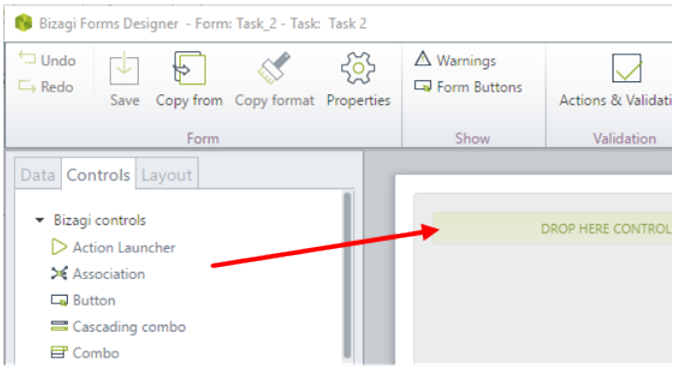

Abbildung 13: Standardkomponenten für das Bizagi-Front-End

Ein wesentlicher Teil der Frontendgestaltung ist das Abbilden der Attribute des Datenmodells auf die Front-End-Komponenten. Hier kann der Designer auf einen Pool an Front-End-Komponenten zurückgreifen, wie Buttons, Listen oder Druckausgaben (Abbildung 13) aber auch externe Komponenten, etwa Spracheingabe oder über eine vordefinierte API auf einen Amazon Cloud Connector. Auch kann die Datenvalidität geprüft werden, etwa indem bestimmte Eingabefelder als Pflichtfelder für Endanwender deklariert werden oder Validierungsregeln explizit erstellt werden (Abbildung 14).

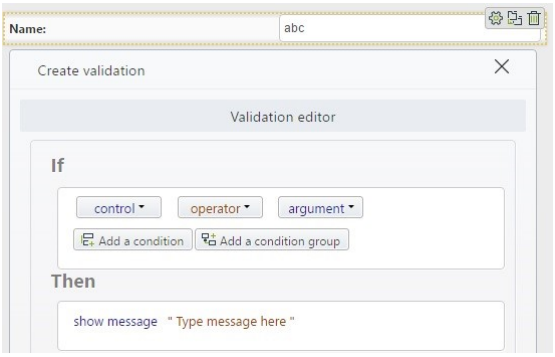

Abbildung 14: Regeln zur Datenvalidität

Zur Steuerung der Applikation nach organisatorischen Vorgaben bietet Bizagi zwei Ansätze, um den Sequenzfluss geschäftsprozesskonform zu implementieren:

- **Einfache Businessregeln** - einfache Validierungsregeln, welche Variablen, die zur Laufzeit mit Daten befüllt sind, mittels Gateways auswerten und auf Basis der Ergebnisse den BPMN-Sequenzfluss lenken; Definition Analog zu Abbildung 14.
- **Activity-Actions** - graphisches Programmiertool, um komplexere Verarbeitungslogiken abzubilden, wobei diese auch an Ereignisse und an Elemente der Eingabemaske gehängt werden können (Abbildung 15).

Externe SOAP- und REST-Services können eingebunden werden, wobei im Falle von SOAP-Services die Spezifikationen der WSDL-Datei in Bizagi-Aktionen und -Funktionen umgewandelt werden kann.

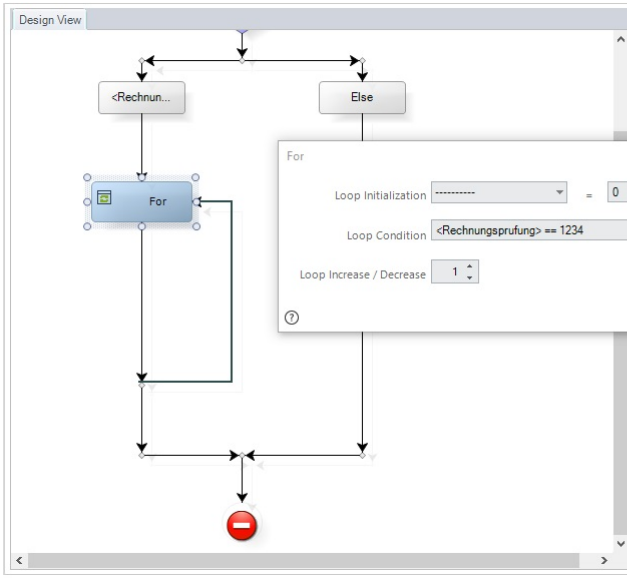

Abbildung 15: Grafische Modellierung von Programmierlogiken

Low-Code Plattform Mendix

Der Fokus von Mendix liegt auf der Erstellung individueller Applikationen, die im eigenen Unternehmen und darüber hinaus zur Unterstützung der Geschäftsprozesse eingesetzt werden können. Im Gegensatz zu Bizagi implementiert Mendix aber nicht direkt den Geschäftsprozess, sondern Ausgangspunkt ist eine User-Story, die potentiell in verschiedene Applikationen abgebildet wird.

Das Vorgehensmodell von Mendix basiert auf den folgenden Schritten:

1. Erstellung der **User-Stories**,
2. Umsetzung der User-Stories in ein **Front-End-Modell**,
3. Erzeugung eines **Datenmodells** auf Basis des Front-End-Modells,
4. Festlegen der **Applikationslogik** durch das Business-Modell,
5. **Deployment** und **kontinuierliche Anpassung** der Applikation.

Die Grundarchitektur von Mendix besteht wieder aus einem 3-Schichtenmodell mit dem User-Interface, der Applikationsebene sowie der Datenbankschnittstelle (Abbildung 16).

Die Möglichkeiten der Gestaltung der Nutzeroberfläche reichen hier von der Gestaltung von Navigationsmenüs über farbliche Hervorhebungen bis hin zu Ausgestaltung einzelner Front-End-Elemente wie Buttons, Listen, Views und Widgets.

Auf Basis des UI-Modells wird eine Datenrepräsentation generiert, welche die Front-End-Elemente in Form von Datenobjekten widerspiegelt. Entitäten werden in Mendix als Objekte im Sinne der objektorientierten Programmierung

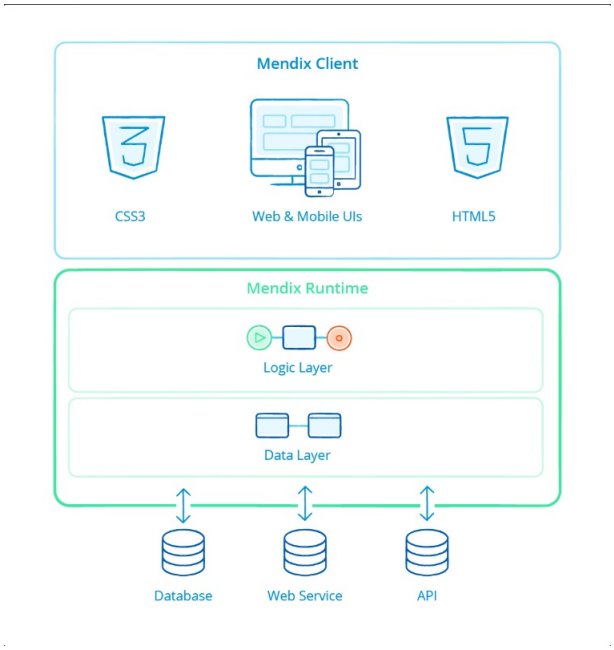

Abbildung 16: Architekturmodell von Mendix [12]

verstanden, die sich durch das Modellierungsframework der Datenmodellebene definieren lassen. Dabei können die Attribute sowie die Entitätsbeziehungen definiert werden.

Der Rahmen zur Formulierung der zeitlichen und logischen Abfolge der Schritte der Applikation wird in Mendix durch Elemente der eigenen Modellierungssprache definiert. Die Elemente ähneln stark einer BPMN-Modellierung, wobei in Mendix (im Unterschied zu Bizagi) der Ausgangspunkt nicht der Geschäftsprozess darstellt, sondern die durch die User-Stories definierten Applikationen, welche nach einem objektorientierten Programmiermodell umgesetzt werden.

Das Ausformulieren von Programmlogik kann entweder in einem Microflow [13] oder einem Nanoflow [14] modelliert werden. Nanoflows können nur Elemente verwenden, die sich offline ohne JAVA ausführen lassen. Daher eignen sich diese insbesondere für die Modellierung von Logiken auf dem UI-Layer. Für komplexere Geschäftslogiken empfiehlt sich der Einsatz von Microflows, da diese auf den vollen Sprachumfang zugreifen können. Innerhalb der Flows liegt der Fokus dann, gemäß dem objektorientierten Programmierparadigma von Mendix, auf dem Erzeugen und Verarbeiten von Objekten, die innerhalb des Datenmodells definiert werden. Tabelle 2 zeigt tabellarisch den Unterschied zwischen beiden Flow-Modellen.

Zur Verbindung von Mendix mit weiteren externen Diensten, existieren eine Reihe von Konnektoren; zusätzlich zu den Web-Service-Konnektoren, welche die Einbindung von eigenen SOAP- und REST-Services ermöglichen, besteht die Möglichkeit, spezielle Protokolle wie MQTT oder ODATA anzusprechen.

	Mircoflow	**Nanoflow**
Laufumgebung	Server	Client (browser/-device)
Sprache	Java / Scala	Java Script
Offline Mobile Apps	Werden nicht unterstützt	Werden unterstützt

Table 2: Micro- vs. Nanoflow

Aufbau des Prototypen

Implementierung der Backend-Services

Die Backend-Services werden mit Standardbausteinen des ERP-Systems implementiert - egal ob der Service später als SOAP oder REST mit der Low-Code Plattform verbunden wird. Hierzu werden teils gleiche Funktionsbausteine durch Wrapper-Funktionen umschlossen, um die Komplexität der Schnittstellen zu reduzieren und durchgehend RFC-fähig zu machen.

Die wesentlichen Funktionsbausteine in der verwendeten Version des ERP-Systems sind:

- **CVAPI_DOC_GETDETAIL** - liefert alle betriebswirtschaftlichen und technischen Details zu einem Dokumenteninfosatz zurück.
- **SDOK_PHIO_GET_URL_FOR_GET** - liefert eine URL, die einen externen Methodenaufruf auf dem Content-Server durchführt.
- **SDOK_PHIO_LOAD_CONTENT** - konvertiert ein Original-File in eine Binärdarstellung (zur Ablage im Contentbereich des DMS).
- **CMS_BINARY_TO_XSTRING** - konvertiert einen Binärstream in einen XString.
- **SSFC_BASE64_ENCODE** - kodiert einen XString in einen BASE64-String.
- **SSFC_BASE6_DECODE** - dekodiert einen BASE64-String in einen XString.
- **BAPI_DOCUMENT_CHANGE2** - Standard-API zum Abändern von Dokumenteninfosätzen.
- **BAPI_DOCUMENT_CREATE2** - Standard-API, der zum Erzeugen von Dokumenteninfosätzen eingesetzt wird.

SOAP Ansatz

Da eine Implementierung von SOAP-fähigen Web-Services in der verwendeten NetWeaver-Version direkt „out of the Box“ möglich ist, müssen die obigen Bausteine nur in neue Funktionen zusammengestellt werden, für die dann ein neuer SOAP-basierter Web-Service erzeugt wird, um in die entsprechende Funktion der Low-Code Plattform als Web-Service eingefügt werden zu können. Im Wesentlichen handelt es sich dann um Funktionen zum Erzeugen, Ändern und Lesen von Dokumenten, wobei hier wichtig ist, die Möglichkeit zu besitzen, auch eine Originaldatei zu verarbeiten.

REST Ansatz

Es müssen zunächst auf Basis des ICF die REST-Services erzeugt werden, da die verwendete NetWeaver-Version des ERP-Systems die Erzeugung noch nicht direkt „out of the Box“ unterstützt. Beispielhaft wird ein REST-basierter Service zum Anzeigen von Dokumenten genauer betrachtet.

Die Implementierung der Klasse Z_SHOW_Document folgt den im Stand der Technik zu den Rest-Services vorgestellten Kriterien und bietet die Möglichkeit, mittels einer URL auf Originale in Dokumenteninfosätze (Ressource) von außerhalb des SAP-Systems zuzugreifen. Einem aufrufenden REST-Client wird durch den Web-Service Z_SHOW mitgeteilt, über welche URI er auf das Originaldokument des angefragten Infosatzes, das innerhalb des Content-Servers liegt, zugreifen kann (Abbildung 17). Hierbei geht die Spezifikation der Dienstleistung innerhalb der Klasse Z_SHOW_-Document dem Erzeugen des Services voraus. Die Klasse bindet das in im Stand der Technik erwähnte Interface ein und verpflichtet sich damit, die Methode Handle_Request zu implementieren.

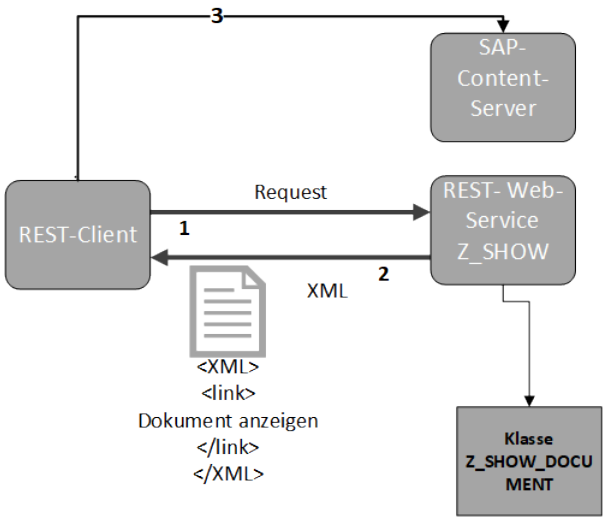

Abbildung 17: Diagramm des HATEOAS-REST-Services

Damit ein Client überhaupt auf die Dienstleistungen der Klasse zugreifen kann, ist es notwendig einen Endpunkt anzulegen, über den diese ansprechbar sind. Hierzu kann innerhalb des ICF-Service-Baums , der sich über die Transaktion **SICF** aufrufen lässt, ein Service-Eintrag definiert werden. Dem Service Z_SHOW wird hierbei die Klasse mit der Dienstleistung zugewiesen. Dieses Vorgehen lässt sich analog zu der im Stand der Technik beschrieben Methodik verstehen.

Dem REST-Client steht somit ein zentraler Zugriffspunkt zur Verfügung, über den er sich Zugang zu den Funktionalitäten des SAP-Dokumenten-Managements verschaffen kann. Es ist durchaus denkbar, die Klasse mit weiteren Funktionalitäten und einer Differenzierung zwischen einzelnen Methoden zu erweitern.

Implementierung der Low-Code Anwendungen

Implementierung in Bizagi

Da die Anwendungsentwicklung unter dieser Plattform auf der Definition eines Geschäftsprozesses basiert, wird hier ein Beispielprozess in BPMN zugrunde gelegt. Der wesentliche Aspekt dieses Prozesses sind dabei nur die Aufrufe des Backends im Rahmen des Dokumentenmanagements - etwa durch eine Anlage eines Infosatzes im DMS des ERP-Systems. Den relevanten Ausschnitt eines Prozesses aus Bizagi zeigt Abbildung 18; die Aktivität „Infosatz an SAP übermitteln“ beinhaltet den Aufruf in das Backend.

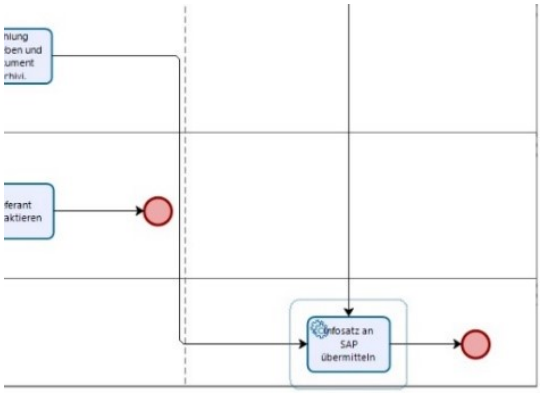

Abbildung 18: Ausschnitt Prozessmodells - Prozess „Rechnungsprüfung“

Auf die Darstellung des Datenmodells soll hier verzichtet werden - im Wesentlichen muss das Datenmodell im Rahmen des Prototypen den Prozessfluss innerhalb der Applikation nachvollziehbar machen und einen Aufruf der Web-Service-Schnittstelle gewährleisten.

Zur Sicherstellung der Datenvalidität des Dokumenteninfosatzes sind Regeln zu hinterlegen, welche die korrekte Bedienung der Schnittstelle zum Backend zur Laufzeit gewährleisten. Dies kann bereits bei der Definition der Forms des Frontend geschehen. Eine beispielhafte Regel ist, dass der Dokumententyp maximal einen Char-Wert der Länge 3 annehmen darf (Abbildung 19 - Maske 2). Sollte ein Mitarbeiter darüber hinaus einen Dokumententyp anlegen, der nicht vom Typ DRW ist, soll das Programm diesen automatisch in DRW umwandeln (Abbildung 19 - Maske 3).

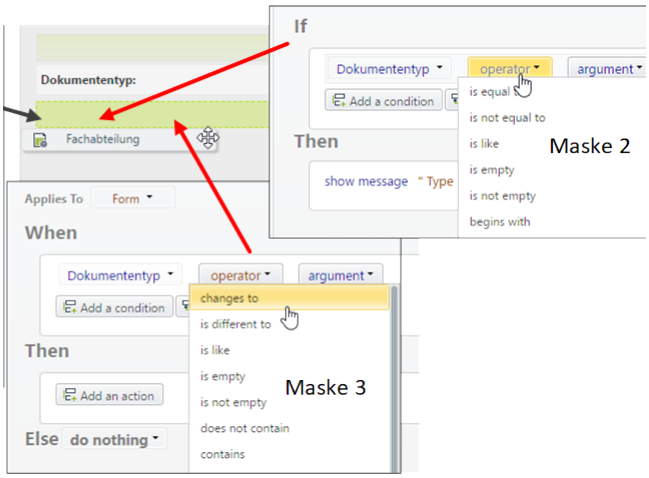

Abbildung 19: Design einer Form mit Regeln

Eine Form ist genau einer User-Task (Abbildung 20) zugeordnet; Anwender können in dem beispielhaft dargestellten Prozess-schritt entscheiden, ob eine Rechnung den Status **„ok“** oder **„nicht ok“** erhält. Der Parameter wird nach dem

Verlassen der Form durch das Expression-Gateway aus Abbildung 21 ausgewertet. Unter diesem Prozessschritt kann hierarchisch der Prozess gehängt werden, der das DMS im Backend aufruft.

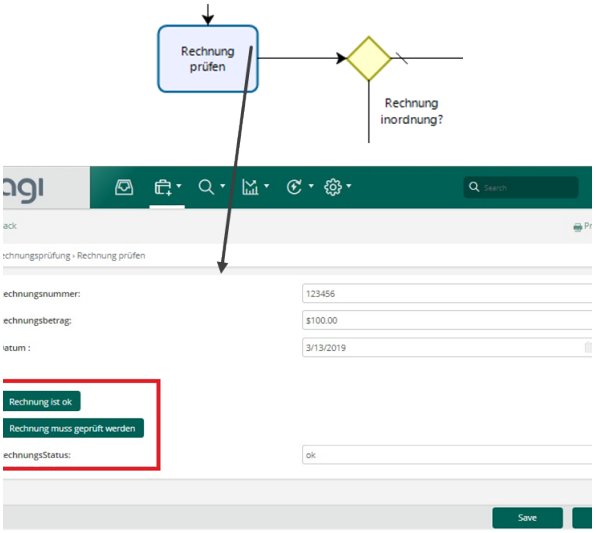

Abbildung 20: Gegenüberstellung Form und Task

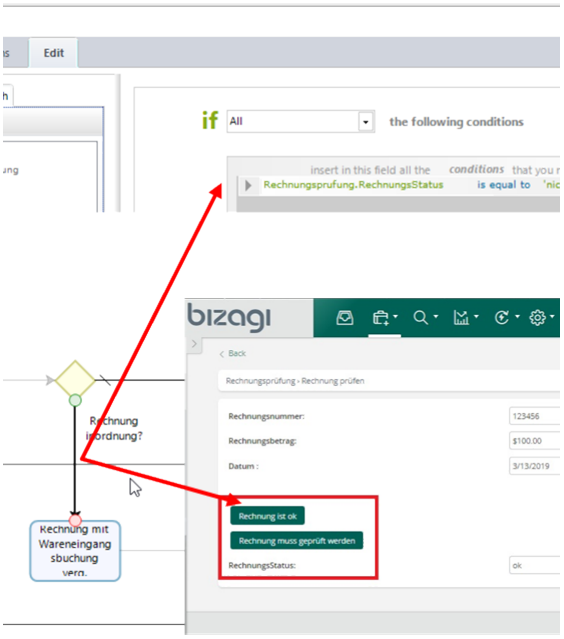

Abbildung 21: Expressions zum Steuern des Prozessflusses

Sobald der Prozess die Schritte der Datenerfassung und der Dokumentenprüfung durchlaufen hat, kann der Dokumenteninfosatz gemeinsam mit dem Originaldokument mittels des Web-Service-Aufrufs innerhalb des DMS des Backends archiviert werden. Durch den Aufruf der Service-URL interpretiert Bizagi die WSDL-Datei und visualisiert diese in Form einer Entität (Abbildung 22). Die Abbildung zwischen den Bizagi-Daten und der Schnittstelle ist intuitiv nachvollziehbar und wird durch den Web-Service-Wizard unterstützt. Innerhalb des Wizards kann neben der Abbildung auch eine Transformation und Berechnung der Bizagi-Daten mit anschließender Weiterleitung an die Schnittstelle erfolgen.

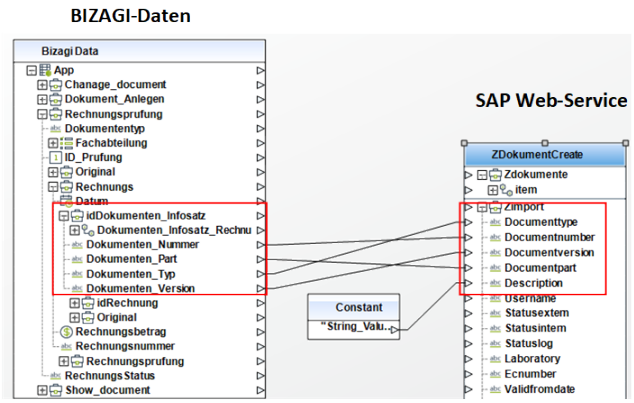

Abbildung 22: Abbildung der Bizagi-Daten auf Web-Service-Schnittstelle

Zur Einbindung des REST-Service kann auf den zuvor beschriebenen Service Z_SHOW_DOC zugegriffen werden. Der HTTP-Link zum Original-File soll anschließend durch eine geeignete Komponente den Nutzinhalt des Dokuments am Applikations-Front-End darstellen.

Hinsichtlich der Anbindung des zugrundeliegenden DMS des ERP-Systems gestaltet sich die Einbindung von RESTful-Services in Bizagi schwieriger, da es initial keinen Austausch von Schnittstelleninformation zwischen den beiden Systemen gibt, wie er beim SOAP-Service mit Hilfe der WSDL-Datei geschieht. Der syntaktisch richtige Aufruf der Services muss in diesem Fall durch den Entwickler selbst zugesichert werden. Dieser muss definieren, in was für einem Format die Antwort des REST-Services erfolgt, damit diese durch Bizagi verarbeitet werden kann. Neben der Content-Formulierung muss der Entwickler die Service-URL so gestalten, dass ein Aufruf mit dynamischen Daten von Bizagi aus möglich ist. Dies wird über die Definition von Parametern erreicht (Abbildung 23 - oberer Teil). Die Bizagi-Daten werden daraufhin auf die Parameter abgebildet, aus denen die Service-URI zur Laufzeit generiert wird (Abbildung 23 - unterer Teil).

Wenn der Service konfiguriert wird, können nach dem Programmstart die Daten eines Dokumenteninfosatzes an den REST-Service geleitet werden. Nachdem die Antwort mit dem HTTP-Link auf das Dokument in Bizagi ankommt, wird diese auf der Ausgangsform durch eine PDF-Komponente angezeigt. Damit sind beide Schnittstellenkonzepte zum Backend validiert.

Zu beachten ist, dass es innerhalb der Studie nicht gelungen ist, aus Bizagi heraus eine Originaldatei in das Backend zu übertragen - es konnten lediglich die Dokumenteninfosätze verarbeitet werden sowie das Original des Backends in der Low-Code Plattform angezeigt werden. Ein entsprechender Geschäftsprozess würde dann erfordern, das Original direkt im Backend einzuchecken. Eine Verarbeitung der Originaldateien (also eine Senden von der Low-Code Plattform ins Backend) erfordert bei den verwendeten Backend-Services

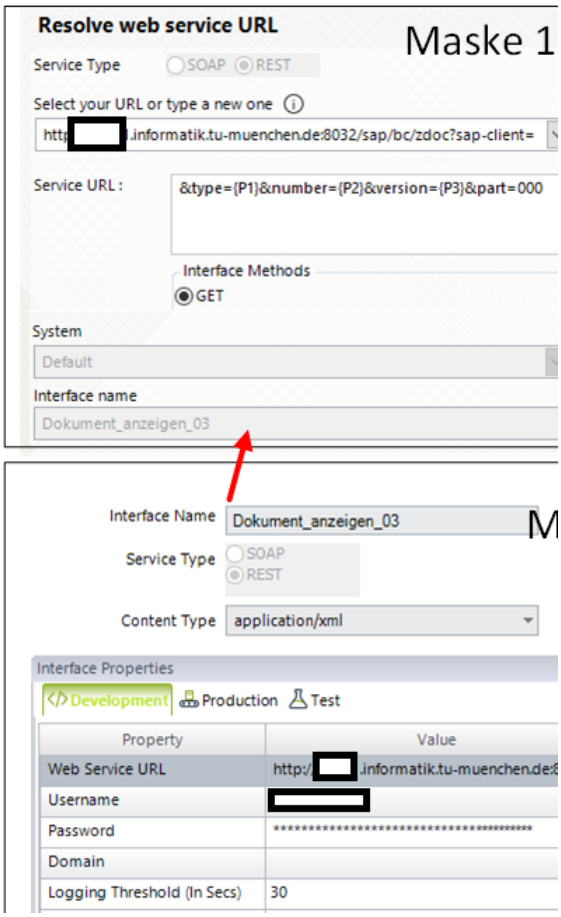

Abbildung 23: Konfiguration des REST-Services

eine Umsetzung der Originaldatei (also z.B. eines PDF) in eine XML-Datei (SOAP) bzw. einen URI (REST), was eine Einbindung eines entsprechenden Codings zur Konvertierung in Base64 erfordert. Für diese Aufgabe existiert zwar in Bizagi eine Funktion zur Enkodierung nach Base64, die als Expression in eine Business-Rule gehängt werden kann ([15]), dann trat aber in dem Zusammenhang ein technisches Problem mit dem HTTP-Request auf. Zur Anzeige des Originals konnte dagegen eine einfach eine existierende Routine in Bizagi genutzt werden.

Implementierung in Mendix

Wie weiter oben dargestellt, basiert die Entwicklung in Mendix auf User-Stories, die Anwendungsfälle definieren. Es werden daher für die Anwendungsfälle die einzelne Funktionalitäten innerhalb von einzelnen Applikationen abgebildet; die Applikationen können sowohl im Webbrowser aber auch direkt als App (apk) in Android-basierten Geräten ausführbar sein. Der Fokus der Untersuchung liegt in der Anbindung der Services des Backends und deren Einbettung in Mendix' Microflows; auch wenn das Vorgehensmodell eigentlich zuerst eine Gestaltung des Frontends vorsieht.

Aufgrund des objektorientierten Paradigmas in Mendix müssen für die zu verwaltenden Dokumente zuerst über die verwendeten Microflows korrespondierende Objekte erzeugt werden. Konkret kann dies wie folgt gelöst werden:

1. Definition eines Microflows, der ein Input-Dokument erzeugt,
2. Definition eines Microflows, der ein Zusatzdatenobjekt erstellt und eine Referenz dieses Objekts an das erzeugte Input-Dokument bindet,
3. Definition eines Microflows, der die erzeugten Objekte an einen Data-View bindet - das Data-View stellt die Schnittstelle zwischen den Datenobjekten und dem Front-End dar.

Abbildung 24 skizziert das Erstellen eines Zusatzdatenobjekts und die Erzeugung einer Referenz auf ein Input-Dokument, das über eine Parameter-Schnittstelle dem Microflow durch den Data-View übergeben wird. Es werden alle beteiligten Microflows gezeigt mit deren Bindung an das Frontend gezeigt.

Da die Dokumentendaten an das Backend übertragen werden sollen, kann ein Microflow gebaut werden, der beispielsweise an einen Front-End-Button gebunden wird und zur Laufzeit durch eine User-Aktion („Betätigen des Buttons") ausgeführt wird. Die Konfiguration des REST-Services ähnelt hierbei der von Bizagi und wird deshalb nicht weiter dargestellt.

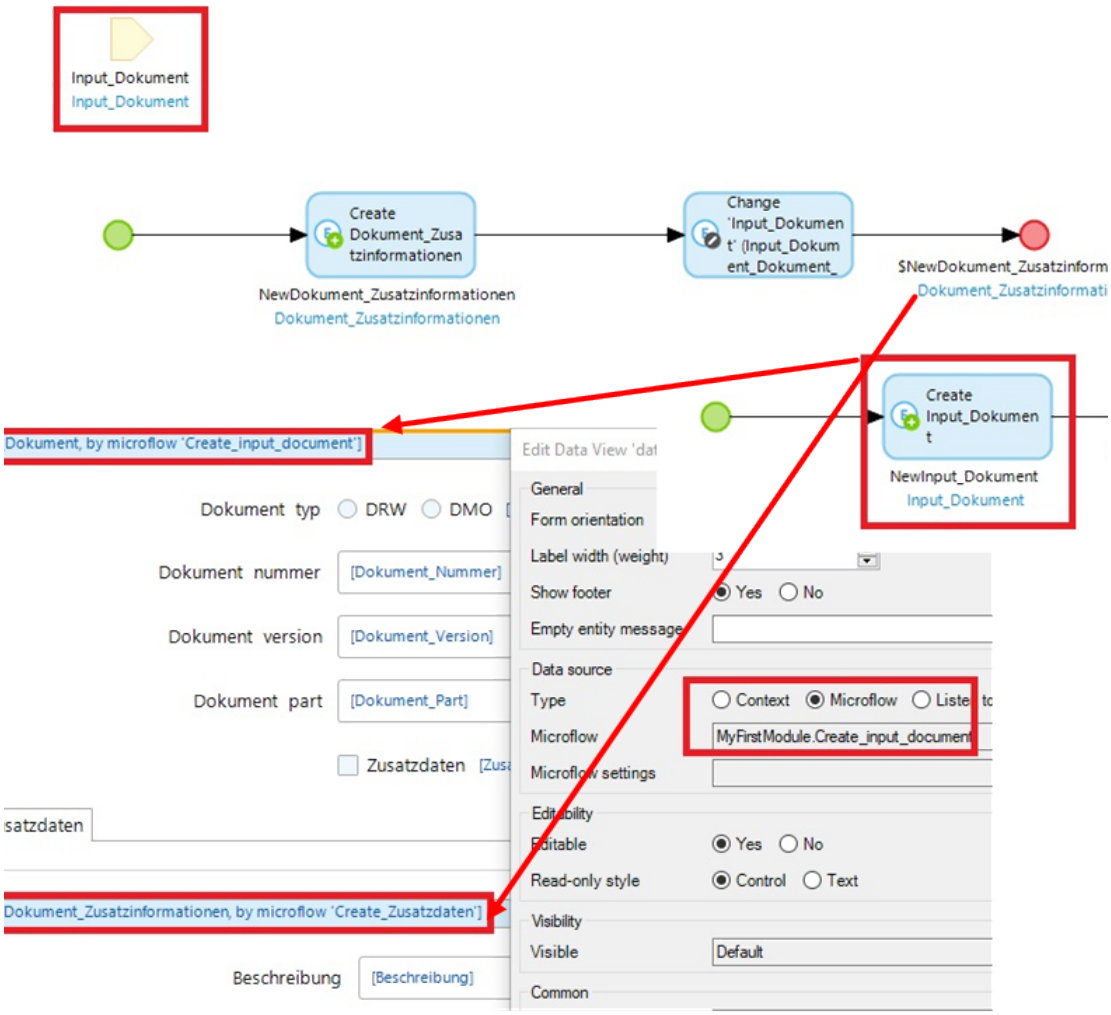

Abbildung 24: Microflows zur Erzeugung der Objekte und Bindung an Frontend

Die wesentliche in der Untersuchung erkannte Komplexität bei Verwendung eines WEB-Services entsteht dann, wenn in Mendix der Web-Service-Aufruf mit der entsprechenden Abbildung der Felder direkt auf Basis der in der NetWeaver-Plattform erzeugten WSDL-Datei erzeugt werden soll. Als Beispiel diene der folgende Ausschnitt der WSDL-Datei die in der NetWeaver-Plattform erzeugt wird:

```
<xsd:import namespace="urn:sap-com:
document:sap:rfc:functions"/>
<xsd:complexType name="DocDraw2">
<xsd:sequence>
    <xsd:element name="Documenttype" type="n0:char3"/>
    <xsd:element name="Documentnumber" type="n0:char25"/>
    <xsd:element name="Documentversion" type="n0:char2"/>
    <xsd:element name="Documentpart" type="n0:char3"/>
    <xsd:element name="Description" type="n0:char40"/>
  .....
</xsd:sequence>
</xsd:complexType>
```

Mendix interpretiert die Aussage der WSDL-Datei und gewährleistet eine Datenvalidität gegenüber der Schnittstelle. Es hat sich jedoch ergeben, dass es NetWeaver-seitig nicht notwendig ist, **alle Attribute** der Entität zu übertragen, um die hinter dem Parameter stehende Tabelle zu ändern bzw. Einträge in dieser zu erzeugen; die Parameter werden auch im Default der WSDL nicht durch NetWeaver erzeugt. Das Problem, das sich nun ergibt, ist, dass Mendix einem Service-Aufruf nur dann zustimmt, wenn die WSDL-Kriterien erfüllt werden. Das NetWeaver eine „unvollständige“ Nachricht im Sinne der WSDL verarbeiten kann, spielt hierbei keine Rolle. Im Fall der Untersuchung war es deshalb notwendig, die WSDL nach der Erzeugung manuell abzuändern und um ein weiteres Attribut aus dem Standard zu ergänzen. Wird einem Element der Zusatz nillable = „true“ hinzugefügt, ist es möglich, ein leeres Typelement beim Aufruf des Services zu übertragen. Die WSDL wird also manuell um den nillable-Parameter ergänzt; in Bizagi sind die Prüfungen offensichtlich nicht so streng, wie in Mendix und die WSDL kann direkt verarbeitet werden. Mit dem WSDL-Zusatz entsteht die Möglichkeit, einzelne Elemente der SOAP-Nachricht leer zu lassen.

```
<xsd:import namespace="urn:sap-com:
document:sap:rfc:functions"/>
<xsd:complexType name="DocDraw2">
<xsd:sequence>
    <xsd:element name="Documenttype" nillable="true"
    type="n0:char3"/>
    <xsd:element name="Documentnumber" nillable="true"
    type="n0:char25"/>
    <xsd:element name="Documentversion" nillable="true"
    type="n0:char2"/>
    <xsd:element name="Documentpart" nillable="true"
    type="n0:char3"/>
    <xsd:element name="Description" nillable="true"
    type="n0:char40"/>
  .....
</xsd:sequence>
</xsd:complexType>
```

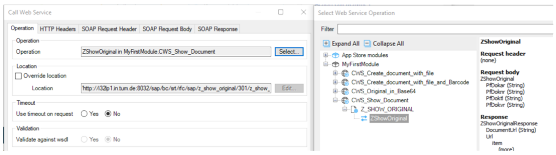

Abbildung 25: SOAP-Service-Aufruf in Mendix

Die nun manipulierte WSDL-Datei kann als Basis für das Datenmodell in Mendix genutzt werden und ermöglicht dem anschließend zu erstellenden Export-Mapping einen flexiblen Aufruf der Web-Service-Schnittstelle. Diese übersendet nach der Abhandlung der Dienstleistung das Ergebnis des Programmlaufs in Form einer SOAP-Response; Abbildung 25 zeigt den Aufruf. Es wurde prototypisch eine Applikation erstellt, mit der Dokumente über Mendix und die entsprechenden Services angelegt werden können (Beispiel der Android-App in Abbildung 26). Es ist zu beachten, dass in Mendix direkt Android-Applikationen erzeugt werden können - hier kann beispielsweise ein Widget aus dem Mendix-Store verwendet werden, mit dem ein Barcode gescannt werden kann; etwa um den Dokumenteninfosatz direkt mit einem Kundenstamm verknüpfen zu können.

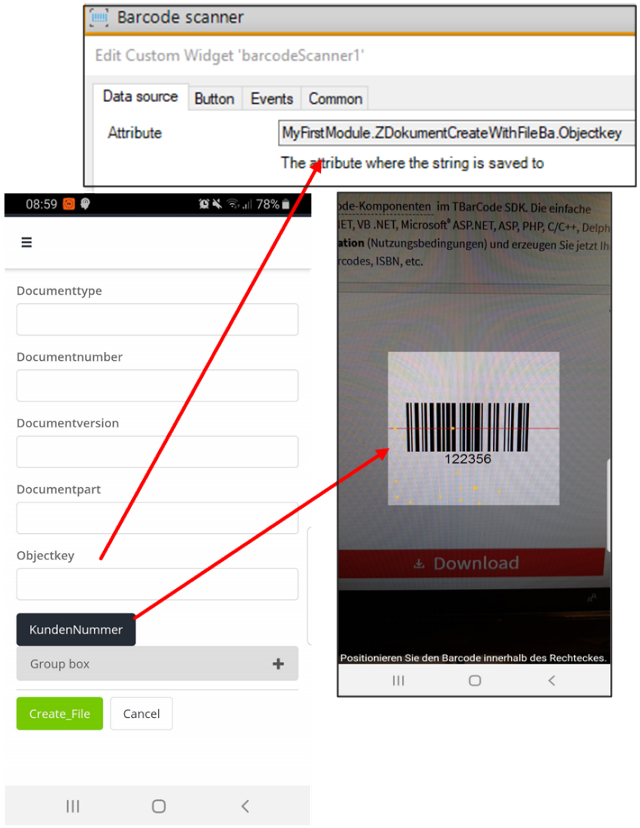

Abbildung 26: Beispiel der Android-Applikation mit Barcode-Scanner

Die Übertragung der Originaldatei von dem Frontend zum Backend in Mendix erfordert eine Kodierung der Datei in die Nachricht. Es kann eine Base64-Enkodierung verwendet werden, wofür Mendix (wie Bizagi) eine Standardfunktion besitzt. Im SAP-Backend existieren ebenfalls Standard-Routinen für die Dekodierung (Funktion „SSFC_BASE64_DECODE“).

RESÜMEE UND AUSBLICK

Mendix erweist sich als Plattform, die über eine Vielzahl an Werkzeugen verfügt, deren Funktionalitätsumfang es ermöglicht, individuellste Probleme zu lösen. Dies zeigt sich insbesondere an der Möglichkeit, trotz der eigentlich offiziell nicht zugreifbaren Front-End-Services von NetWeaver, Originale zwischen Mendix und der Schnittstelle beliebig austauschen zu können. Zu beachten ist jedoch, dass der Mendix-Entwicklungszyklus mit der Definition von User-Stories beginnt, die anschließend in ein Front-End-Modell übersetzt

werden. Erst im Anschluss folgt die Definition des Datenmodells und der Geschäftslogik. Es besteht also die Notwendigkeit, dass bereits im UI-Modell die Prozesslogik in Form von Masken zu berücksichtigten ist. Im Gegensatz zu Bizagi, welches direkt den Geschäftsprozess implementiert, führt eine Verwendung von Mendix somit zu einer anderen Logik der Einbindung der Backend-Services: der Service wird nicht direkt in den Geschäftsprozess eingebunden, sondern durch eine eigene Applikation implementiert, die auf einer User-Story basiert und wiederum in einem Geschäftsprozess verwendet werden kann, welcher aber nicht als solcher abgebildet wird. Somit ist in Mendix im Wesentlichen ein neues Frontend der Backend-Services implementierbar. Das Frontend kann dann von den Nutzern direkt in Prozessen aufgerufen werden. In Bizagi wird dagegen der Service aus dem Geschäftsprozess direkt aufgerufen, da die Applikation direkt auf dem Geschäftsprozess basiert. Dies bedeutet jedoch auch, dass für verschiedene Prozessvarianten potentiell verschiedene Applikationen in Bizagi entwickelt werden müssen, die dann den selben Service aufrufen. Die Behandlung von SOAP-Services hat sich in Mendix als aufwändiger erwiesen, REST-Services waren in der Komplexität vergleichbar. Die Einbindbarkeit komplexer individueller Routinen schien den Autoren im Untersuchungszeitraum aufgrund einer Java-Umgebung in Mendix etwas einfacher zu bewerkstelligen, als in Bizagi - dies hatte eine unmittelbare Auswirkung auf den Transfer der Originaldateien von den Low-Code Plattformen zum Backend.

Weitere Untersuchungsgegenstände könnten nun ODATA-Services, in der speziellen hier betrachteten Domäne ebenso die Verwendung des CMIS-Standards sowie weitere Low-Code Plattformen sein. Die Möglichkeiten, komplexeren Programmcode - etwa Aufrufe umfangreicher Libraries einzubinden oder auch komplexe Algorithmen - sollte speziell für Bizagi ebenfalls weiter validiert werden, da das Programmiermodell den Geschäftsprozess in den Vordergrund stellt.

LITERATUR

[1] Klaus Goetzer. *Dokumenten-Management.* Dpunkt.verlag, 2008.

[2] Rinaldo Heck. *Geschäftsprozessorientiertes Dokumentenmanagement.* GalileoPress, 2009.

[3] SAP. *Definition von Dokumenten.* https://help.sap.com/doc/saphelp_erp60_sp/6.0/de-DE/79/22bf53d25ab64ce10000000a174cb4/content.htm?no_cache=true, 2019. Aufgerufen im Internet am 20.03.2019.

[4] Pragya Pande. *Architekturkomponenten SAP DMS.* https://archive.sap.com/documents/docs/DOC-54583, 2019. Aufgerufen im Internet am 20.03.2019.

[5] Johann Wagner and Kurt Schwarzenbacher. *Föderative Unternehmensprozesse: Technologien, Standards und Perspektiven für vernetzte Systeme.* John Wiley & Sons, 2007.

[6] Ingo Melzer. *Service-orientierte Architekturen mit Web Services, 2.* Spektrum, 2007.

[7] Jim Webber, Savas Parastatidis, and Ian Robinson. *REST in practice: Hypermedia and systems architecture.* O'Reilly Media, Inc., 2010.

[8] Martin Fowler. *Richardson-Maturity-Model.* https://martinfowler.com/articles/richardson-MaturityModel.html, 2019. Aufgerufen im Internet am 20.03.2019.

[9] Horst Keller. *ABAP Objects.* GalileoPress, 2015.

[10] Gerhard Rempp, Mark Akermann, Martin Löffler, and Jens Lehmann. *Model Driven SOA : Anwendungsorientierte Methodik und Vorgehen in der Praxis.* Xpert.press, Berlin, Heidelberg, 2011.

[11] BIZAGI. *BIZAGI Architektur.* http://help.bizagi.com/bpm-suite/en/index.html?architecture.htm, 2019. Aufgerufen im Internet am 20.03.2019.

[12] Mendix. *Mendix Architektur.* https://www.mendix.com/evaluation-guide/enterprise-capabilities/architecture-intro, 2019. Aufgerufen im Internet am 20.03.2019.

[13] Mendix. *Mendix BPMN.* https://docs.mendix.com/refguide/microflows, 2019. Aufgerufen im Internet am 20.03.2019.

[14] Mendix. *Mendix Refguide.* https://docs.mendix.com/refguide/nanoflows, 2019. Aufgerufen im Internet am 20.03.2019.

[15] Bizagi. Convert Files to base64. http://help.bizagi.com/bpm-suite/en/-index.html?tobase64.htm. Zuletzt eingesehen am 12.10.2018.