Hochschule für Technik, Wirtschaft und Kultur Leipzig

Fakultät Informatik und Medien

Studiengang Medieninformatik B.Sc.

Automatisierung der Meldepflicht elektronischer Aufzeichnungssysteme: Integration des ERiC Elster Clients in Cloud Java Anwendungen

Bachelorarbeit

von

Paul Hublitz

geb. am 18. 07. 1997

in Melsungen

79639

Verantwortlicher Hochschullehrer: Prof. Dr. rer. nat. Thomas Riechert
Zweitgutachter: Dipl.-Inf Steven Schwarznau
Leipzig, November 2024 – Januar 2024

Zusammenfassung

Die vorliegende Arbeit untersucht die Integration des Elster Rich Clients (ERiC) in eine Cloud Java-Anwendung zur Automatisierung der gesetzlichen Meldepflichten nach §146a Abgabenordnung (AO). Es werden eine konkrete Klassenarchitektur, die Implementierung sowie die Erstellung der XML-Struktur gemäß XML Schema Definition (XSD) Dateien beschrieben. Des Weiteren werden die für die Umsetzung verwendeten Technologien erläutert. Herausforderungen ergaben sich aus den komplexen rechtlichen Rahmenbedingungen sowie der technischen Umsetzung der Speicherung des ERiC. Die Umsetzung erfolgte in einer verteilten Architektur, die lokale Kassensysteme mit einem zentralen Kundenserver und einer privaten Cloud-Server-Komponente verbindet. Die vollautomatische Generierung und Bereitstellung von XML-Dateien wurde zwar umgesetzt, wird aber aus haftungsrechtlichen Gründen nicht genutzt. Dennoch können mit der entwickelten Lösung XML-Daten effizient erzeugt und für die Übermittlung an die Finanzbehörden bereitgestellt werden.

Erklärung

Ich versichere wahrheitsgemäß, die Bachelorarbeit selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

.....
Paul Hublitz
Leipzig, den 31. Januar 2025

Inhaltsverzeichnis

Abbildungs- und Listingverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Die zunehmende Bürokratisierung stellt für Unternehmen eine große Belastung dar, es müssen Zeit und Ressourcen investiert werden, um diesen Anforderungen gerecht zu werden. Umso mehr sind Unternehmen bestrebt, möglichst viele Verwaltungsprozesse zu automatisieren und den Aufwand so gering wie möglich zu halten. Die Automatisierung bietet einen attraktiven Lösungsansatz, bei dem menschliche Eingriffe auf ein Minimum reduziert werden. Das Ziel der Meldepflicht ist es, alle relevanten Kassendaten in einem standardisierten Format an die Finanzbehörde zu übertragen. Damit soll die Transparenz und Nachvollziehbarkeit für die Finanzbehörden erhöht werden. Durch die Erfassung aller elektronischen Registrierkassen werden Manipulationsmöglichkeiten weiter eingeschränkt, da Finanzbehörden eine genaue Übersicht über die eingesetzten Kassen bekommen. Es wird ein wirksamer Rahmen geschaffen, der die Ordnungsmäßigkeit der Geschäftsaufzeichnungen sicherstellt und es trägt zur Steuergerechtigkeit bei. Die Daten sind digital im XML-Format entweder als manueller Upload über das Elster-Portal oder mit dem Elster Rich Client zu übermitteln.

Elektronische Kassen wurden im Infoblatt vom Finanzamt Saarland folgendermaßen definiert: [1]:

„Elektronische Aufzeichnungssysteme sind elektronische oder computergestützte Kassensysteme oder Registrierkassen. Die Systeme dienen der Erfassung von Verkäufen von Waren oder der Erbringung von Dienstleistungen und deren Abrechnung.“

Eine automatisierte Lösung zur Übermittlung steuerlich relevanter Daten bietet hier einen

vielversprechenden Ansatz. Die Automatisierung soll mit dem Elster Rich Client (ERiC) (vgl. Kap. 2) erfolgen, einem verifizierten Übertragungsmodul von Elster, das eine effiziente und standardisierte Automatisierung ermöglicht.

Durch die Integration von ERiC in eine cloudbasierte Java-Anwendung können wir manuelle Prozesse reduzieren. Zudem wird durch ERiC sichergestellt, dass alle gesetzlichen Anforderungen erfüllt sind, indem eine Validierung und Zertifizierung der Daten erfolgt. Nach erfolgreichem Abschluss des Prozesses werden die Daten an die jeweilige Steuerbehörde übermittelt. Diese Arbeit dokumentiert die Umsetzung der Integration von ERiC und geht auf die verschiedenen Schritte ein. Es wird aufgezeigt, wie Unternehmen durch die Automatisierung der gesetzlichen Meldepflicht im Steuerwesen nachhaltig entlastet werden können.

1.1 Rechtliche Grundlagen

Die rechtliche Grundlage für die Meldepflicht von elektronischen Kassensystemen basiert auf mehreren gesetzlichen Regelungen. Zentraler Ausgangspunkt ist das Gesetz zum Schutz vor Manipulationen an digitalen Grundaufzeichnungen vom 22. Dezember 2016 [2], auch Kassengesetz genannt. Dieses Gesetz führte zu einer Änderung der Abgabenordnung, insbesondere durch die Einführung des § 146a AO, der die grundlegenden Anforderungen an Aufzeichnungssysteme festlegt.[3] [4]

Die Kassensicherungsverordnung (KassenSichV) vom 26.09.2017 konkretisiert diese gesetzlichen Vorgaben und legt die technischen Details zur Umsetzung fest. Sie spezifiziert die Anforderungen an das Sicherheitsmodul, das Speichermedium und die digitale Schnittstelle des elektronischen Aufzeichnungssystems. Ergänzend legt die KassenSichV fest, dass alle elektronischen Kassensysteme beim zuständigen Finanzamt anzumelden sind. Jedes System

muss mit einer eindeutigen Seriennummer versehen und einzeln registriert werden. Die Meldung umfasst neben der Seriennummer auch die Übermittlung des Firmennamens, der Steuernummer, der Art der zertifizierten technischen Sicherheitseinrichtung (TSE) 1.2.1 sowie des Anschaffungsdatums. Auch die Außerbetriebnahme eines Systems ist der Finanzverwaltung zu melden. Alle elektronischen Aufzeichnungssysteme müssen bis spätestens 31. Juli 2025 beim zuständigen Finanzamt angemeldet werden.[5] [6]

Das Gesetz sieht vor, dass Verstöße gegen die Meldepflicht und die damit verbundenen technischen Anforderungen als Ordnungswidrigkeit geahndet werden können. Dies unterstreicht die Bedeutung der korrekten Umsetzung dieser Vorschriften sowohl für die Unternehmen als auch für die Kassenhersteller. [4]

1.2 Technische Anforderungen und Standards

Ein wichtiger Aspekt um das Thema richtig einzuordnen, ist die zertifizierte technische Sicherheitseinrichtung (TSE), die aus einem Speichermedium, einem Sicherheitsmodul und einer standardisierten digitalen Schnittstelle besteht.[6] Sie dient der Verifizierung und Signierung der Transaktion, so dass im Nachhinein keine Änderungen vorgenommen werden können. Die TSE muss spezifische Sicherheitsstandards erfüllen und vom Bundesamt für Sicherheit in der Informationstechnik (BSI) [7] zertifiziert werden.

Die Abbildung 1.1 zeigt den Ablauf der Kommunikation einer Registrierkasse mit einer technischen Sicherheitseinrichtung (TSE). Zuerst übermittelt der Benutzer die Anwendungsdaten wie Seriennummer und Modell des Aufzeichnungsgeräts an die TSE. Die TSE signiert daraufhin den Beleg mit wichtigen Informationen wie dem Zeitpunkt der Transaktion und dem Verkaufsbetrag. Diese signierten Daten werden anschließend im Speichermodul der TSE revisionssicher und fälschungssicher abgelegt.

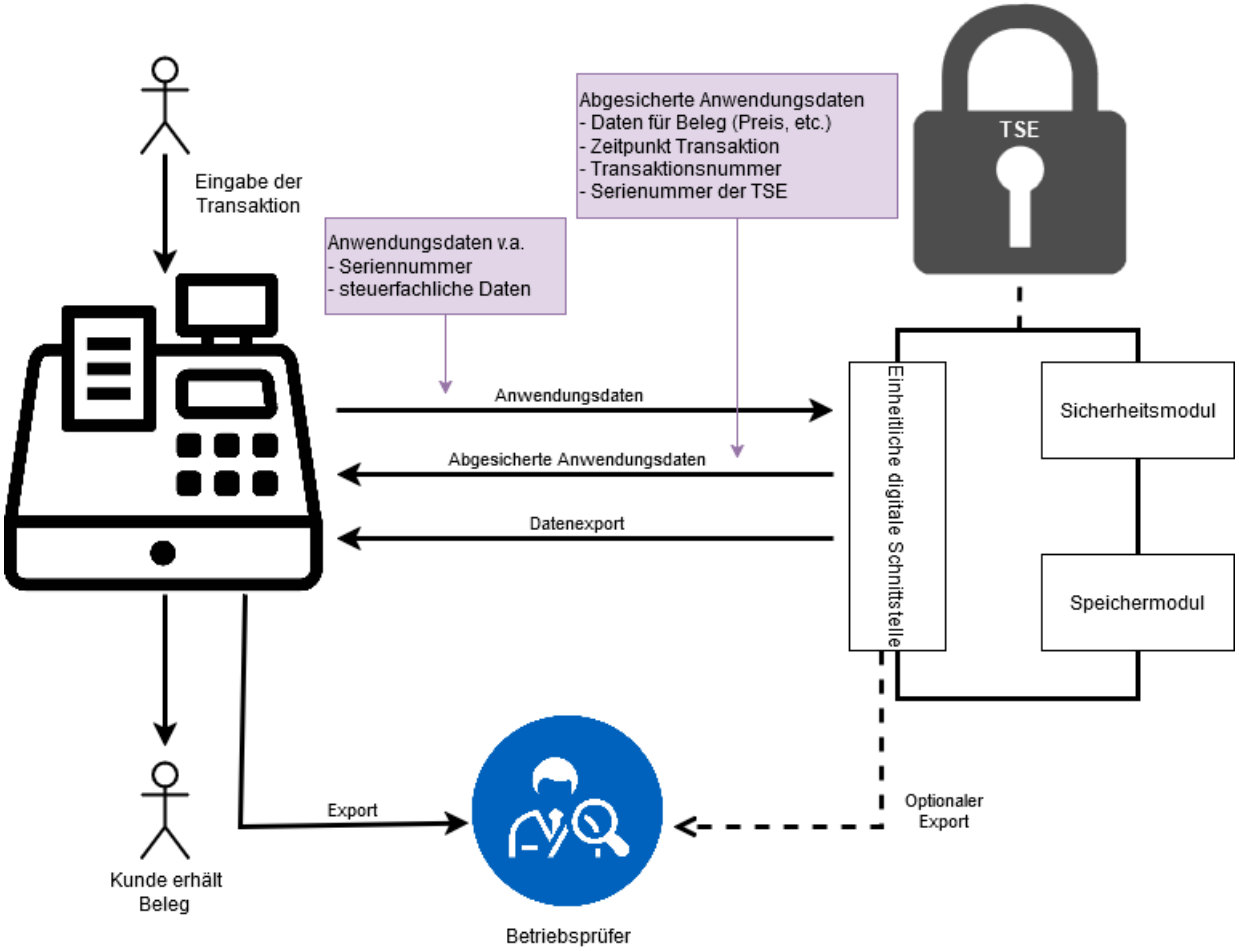

Abbildung 1.1: Funktion der TSE

Von dort können die gesicherten Anwendungsdaten später exportiert werden, zum Beispiel für die Datenübermittlung an die Finanzbehörden. Die TSE stellt somit sicher, dass alle relevanten Transaktionsinformationen korrekt erfasst und geschützt werden, ohne dass die Kasse selbst diese Sicherheitsfunktionen übernehmen muss. Die Trennung von Kassenfunktion und Datensicherung durch die TSE ist ein zentraler Bestandteil der gesetzlichen Anforderungen an die digitale Kassensicherung.

Die TSE gewährleistet bereits ein hohes Maß an Sicherheit, indem sie Transaktionen

signiert und somit Manipulationen verhindert. Die Mitteilungspflicht geht allerdings noch einen Schritt weiter. Sie verpflichtet die Unternehmen, alle eingesetzten elektronischen Aufzeichnungssysteme bei der Finanzbehörde zu registrieren und alle damit zusammenhängenden Informationen über das System zur Verfügung zu stellen. Ziel ist es, eine lückenlose Kontrolle über die eingesetzten Systeme zu ermöglichen, um die Integrität und Konsistenz der gespeicherten Transaktionsdaten sicherzustellen.

1.2.1 Varianten der TSE

Integrierte TSE im Kassensystem

Bei modernen Kassensystemen ist die TSE direkt in die Hardware des Kassensystems integriert. Sie funktioniert wie eine interne Komponente des Systems, wodurch der physische Anschluss entfällt. In diesem Fall wird die TSE in das Kassensystem eingebaut und übernimmt ihre Funktion durch eine digitale Schnittstelle. Alternativ kann die TSE auch in Form eines USB-Sticks oder einer SD-Karte verwendet werden, die an das Kassensystem angeschlossen werden.

Cloud-basierte TSE

Es gibt auch cloudbasierte Lösungen, bei denen die TSE nicht lokal auf einem USB-Stick oder in einem Kassensystem gespeichert wird, sondern in einer Cloud-Infrastruktur betrieben wird. Die Daten werden in diesem Fall online signiert und gespeichert, was eine flexible Lösung für Unternehmen darstellt, die mehrere Standorte oder mobile Kassensysteme betreiben.

1.3 Datenübertragung

Für die Datenübermittlung an die Finanzbehörden ist das standardisierte Datenformat DSFinV-K („Digitale Schnittstelle der Finanzverwaltung für Kassensysteme“) [8] verpflichtend. Das DSFinV-K definiert das Format für die Speicherung, den Export und die Übermittlung der Kassendaten, um eine einheitliche Struktur und maschinelle Überprüfbarkeit sicherzustellen. Im Rahmen des DSFinV-K-Exports werden auch die kryptografischen Signaturen der TSE für jede Transaktion übermittelt, die deren Unveränderbarkeit nachweisen.

1.4 Zielsetzung

Es wird ein Ansatz für die technische Integration des Elster Rich Clients (ERiC) in eine Cloud-basierte Java-Anwendung entwickelt und analysiert, der eine gesetzeskonforme Automatisierung der Meldepflicht für elektronische Aufzeichnungssysteme gemäß § 146a AO ermöglicht. Diese Implementierung soll eine automatisierte und fehlerfreie Datenübermittlung an die Finanzbehörden gewährleisten und den Verwaltungsaufwand reduzieren.

1.5 Aufbau der Arbeit

In dieser Arbeit werden zunächst Grundlagen der ERiC-Schnittstelle dargestellt. Anschließend erfolgt eine Analyse bestehender Systemarchitekturen und ein Vergleich relevanter Technologien, die für eine Cloud-basierte Integration geeignet sind. Darauf aufbauend wird ein Implementierungsansatz entwickelt und dessen technischer Ablauf detailliert erläutert. Es werden die Herausforderungen und Lösungen beschrieben, die mit der Integration

der Anwendung verbunden sind. Abschließend wird auf den tatsächlichen Einsatz der Implementierung und auf rechtliche Schwierigkeiten eingegangen.

2 ERiC-Schnittstelle

ERiC ist die zentrale Komponente für die Entwicklung von Elster-kompatibler Software, er ermöglicht die sichere und automatisierte Übermittlung steuerlicher relevanter Daten an die deutschen Finanzbehörden. ERiC kann in alle Softwarelösungen implementiert werden, die C-Bibliotheken unterstützen und ermöglicht so eine effiziente und breite Integration gesetzlicher Anforderungen direkt in bestehende Anwendungen. In Cloud-Umgebungen bietet der ERiC-Client eine skalierbare und flexible Lösung zur Umsetzung steuerlicher Meldepflichten. [9, S. 10]

2.1 Funktionsweise

In der Abbildung 2.1 wird die Funktionsweise des Systems beschrieben. Zu Beginn wird in unserer Anwendung eine Datei mit den relevanten Informationen, die die Eingabedaten enthält, im Elster XML-Format erstellt (Schritt 1). Die Software ruft anschließend den ERiC-Client auf (Schritt 2), der die eingehenden Daten auf Gültigkeit prüft (Schritt 3). Nach erfolgreicher Prüfung werden die Daten an den ELSTER-Annahmeserver über das Internet übermittelt (Schritt 4).

Der Annahmeserver antwortet mit einer Empfangsbestätigung (Schritt 5), die vom ERiC-Client auf Korrektheit geprüft wird (Schritt 6). Bei erfolgreicher Verifizierung generiert der ERiC-Client ein PDF-Dokument als Bestätigung für den Anwender (Schritt 7). Abschließend wird das Ergebnis an die Steuersoftware des Softwareherstellers zurückgemeldet (Schritt 8).

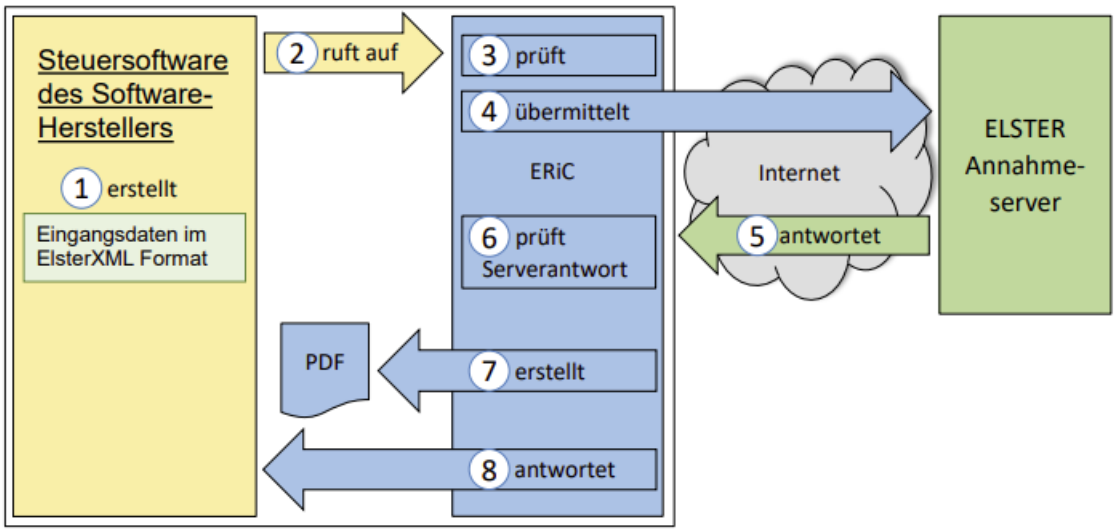

Abbildung 2.1: Kernfunktion des ERiC [9, S. 78]

Dieser Ablauf gewährleistet die Datenintegrität während der Übermittlung und gibt dem Anwender eine Bestätigung über die erfolgreiche Übermittlung seiner Daten.

2.2 Datenverarbeitung

ERiC arbeitet mit dem XML-Format, das so aufgebaut ist, wie in der Abbildung 2.2 veranschaulicht wird. Der Datensatz besteht aus einem **TransferHeader**. Diese enthält wichtige Informationen wie Daten Art, Hersteller-ID und Datenlieferant, damit die Daten auf dem ELSTER-Annahmeserver dem richtigen Verfahren zugeordnet werden können. Der **NutzdatenHeader** enthält alle Informationen für die Verarbeitung der Nutzdaten, z. B. NutzdatenTicket und Empfänger. Weiterhin werden dort Rückgabemeldungen und Fehlermeldungen vom ELSTER-Annahmeserver abgelegt. In den **Nutzdaten** werden die eigentlichen Informationen der Kassensystems für die Meldepflicht abgelegt. [9, S. 79]

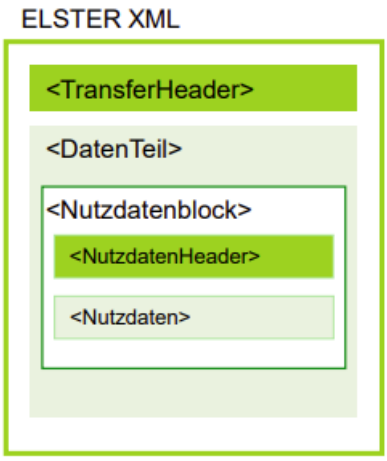

Abbildung 2.2: Datenaufbau

2.3 ERiC Release-Zyklen

Für ERiC sind pro Kalenderjahr zwei Releases vorgesehen. Es gibt ein technisches Release Anfang Mai. In diesem Release werden neue technische Standards und Schnittstellenänderungen implementiert. Mitte November ist die Auslieferung des Hauptreleases, dort ist der Hauptbestand Jahresfortschreibung, z. B. neue Veranlagungs-/Anmeldezeiträume. Von Elster wird empfohlen, das Mai-Release frühzeitig in die Anwendung zu integrieren, um Engpässe bei der Implementierung des November-Releases zu vermeiden, da eine Mindestversionserhöhung jedes Jahr im April stattfindet. [9, S. 20]

2.4 Technische Spezifikation

Der ERiC Client stellt spezifische Anforderungen an Hard- und Software, um eine zuverlässige Integration in Steuer-, Finanz- und Lohnbuchhaltungssysteme zu gewährleisten. Für den Betrieb sind 32- oder 64-Bit-Architekturen notwendig und die unterstützten Betriebssysteme umfassen Windows, Linux, macOS und AIX. Für Windows- und Linux-Systeme

wird eine IA-32-CPU mit SSE2-Erweiterung oder eine 64-Bit-Intel/AMD-Architektur benötigt. Auf Power-Systemen (AIX und Linux) werden ab Power8 (Little Endian) bzw. ab Power7 (Big Endian) unterstützt, während für macOS Universal Binaries für Intel und ARM64 zur Verfügung stehen.

Für die Installation der ERiC-Programmbibliotheken sind mindestens 5 GB freier Speicherplatz erforderlich (bei AIX und Linux Power 10 GB). Zusätzlich sind 5 GB für Dokumentation und Vordrucke notwendig. Unterstützt werden ausschließlich Betriebssystemversionen, für die Sicherheitsupdates bereitgestellt werden. Die Unterstützung für veraltete Systeme wird über Newsletter angekündigt und anschließend beendet.

Zur Einhaltung der Sicherheitsanforderungen erfolgt die Datenübertragung über SSL/TLS-Protokolle. [9, S. 21-28]

3 Analyse der Systemarchitektur

3.1 Überblick bestehender Architekturen

Die Integration von steuerlichen Schnittstellen wie des ERiC in IT-Infrastrukturen kann je nach spezifischen Anforderungen und Einsatzgebiet unterschiedliche Architekturansätze erfordern. Die gängigen Ansätze lassen sich in die drei folgenden Hauptkategorien unterteilen: Cloud-Architekturen, Hybrid-Cloud-Architekturen und On-Premise-Architekturen.

3.1.1 Cloud-Architektur

Die Cloud-Architektur ist ein Modell, das laut National Institute of Standards and Technology [10] als eine weitreichende Umgebung beschrieben wird, in der IT-Ressourcen als Dienste über das Internet bereitgestellt werden. Diese Architektur beruht auf fünf wesentlichen Merkmalen: bedarfsorientierter Selbstbedienung, breitem Netzwerkzugang, Ressourcen-Pooling, elastischer Skalierbarkeit und einem nach Verbrauch abgerechneten Service. In diesem Modell werden physische und virtuelle Ressourcen dynamisch zugewiesen und auf Abruf bereitgestellt, um die Flexibilität und Effizienz des IT-Betriebs zu maximieren. Cloud-basierte Architekturen können in 3 verschiedene Service-Modelle unterteilt werden, darunter Infrastructure as a Service (IaaS), Platform as a Service (PaaS) und Software as a Service (SaaS) (siehe Abbildung 3.1), die jeweils eine unterschiedliche Ebene der Abstraktion und Kontrolle bieten. Diese Modelle erlauben Unternehmen eine agile Anpassung an wechselnde Anforderungen und die Skalierung ihrer IT-Umgebung gemäß der jeweiligen Auslastung und Geschäftsanforderungen.

3.1.2 On-Premise-Architektur

Die On-Premise-Architektur bezeichnet ein IT-Bereitstellungsmodell, bei dem Hardware und Software vollständig innerhalb der eigenen Organisation betrieben werden. Im Gegensatz zu Cloud-basierten Lösungen, bei denen Ressourcen über das Internet von Drittanbietern bereitgestellt werden, verbleiben bei der On-Premise-Architektur alle Daten und Anwendungen im eigenen Rechenzentrum. Dies ermöglicht eine direkte Kontrolle über die IT-Infrastruktur, was insbesondere für Unternehmen mit strengen Datenschutzanforderungen oder spezifischen Compliance Vorgaben von Vorteil ist. Allerdings erfordert der Betrieb einer On-Premise-Architektur erhebliche Investitionen in Hardware, Software und qualifiziertes Personal für Wartung und Betrieb. Zudem ist die Skalierbarkeit oft eingeschränkt, da Erweiterungen der Infrastruktur zeit- und kostenintensiv sein können. Dennoch bietet dieses Modell für bestimmte Anwendungsfälle, insbesondere bei sensiblen Daten oder speziellen Leistungsanforderungen, eine geeignete Lösung. [11]

3.1.3 Hybrid-Cloud-Architektur

Laut der Definition des National Institute of Standards and Technology [10] benötigt eine Hybride-Cloud-Architektur zwei oder mehr unterschiedliche Cloud-Strukturen, darunter zählen privat, gemeinschaftlich oder öffentliche Clouds. Die verschiedenen Clouds werden verbunden durch standardisierte oder proprietäre Technologie. Das Verbinden einer On-Premise-Architektur mit einer Cloud-Architektur führt also nicht automatisch zu einer hybriden Cloud-Architektur. Weitere Punkte, die für eine Hybrid-Cloud sprechen [12]:

- Verbindung von mehreren Computern über ein Netzwerk
- Konsolidieren von Ressourcen in einem Pool
- Verschieben des Workloads zwischen den Anwendungen

- Skalierung und schnelle Bereitstellung von neuen Ressourcen
- Integration eines einheitlichen Verwaltungstools

Durch diese Aspekte hat sie gewisse Vorteile gegenüber anderen Technologien. Es besteht eine gute Kosteneffizienz, da Unternehmen den Kauf und die Wartung von Server-Hardware vermeiden können. Geschäftskritische Daten können sicher in der Private-Cloud aufbewahrt werden, während die Rechenleistung der öffentlichen Cloud für komplexe Aufgaben verwendet wird. Sie kann jederzeit und überall genutzt werden, was eine globale Verfügbarkeit garantiert.[13]

3.2 Vergleich der Architekturen

Im Vergleich der verschiedenen Architekturen weist die Cloud eine hohe Agilität und Kosteneffizienz auf, da Ressourcen-Pooling und elastische Skalierbarkeit sowie eine flexible Anpassung an schwankende Anforderungen möglich sind. Durch eine bedarfsgerechte Nutzung von Ressourcen und die Verfügbarkeit der verschiedenen Servicemodelle wie IaaS, PaaS und SaaS können Unternehmen eine angepasste und effiziente IT-Landschaft schaffen. Die On-Premise Variante hingegen garantiert maximale Kontrolle, da sämtliche Hardware und Software lokal betrieben wird. Das ist von Vorteil, wenn Unternehmen mit sensiblen Daten arbeiten oder strengen Datenschutzrichtlinien unterliegen. Dieses Modell erfordert allerdings hohe Anfangsinvestitionen und einen großen Aufwand für Wartung, Betrieb und qualifiziertes Personal. Die feste Infrastruktur begrenzt zudem die Skalierbarkeit und kann aufgrund der kostenintensiven Erweiterung zu hohen laufenden Kosten führen. Eine Hybrid-Cloud-Architektur kombiniert die Vorteile beider Ansätze, indem sie es ermöglicht, sensible Daten und Anwendungen lokal in einer On-Premise privaten Cloud zu betreiben, während weniger kritische Workloads in die öffentliche Cloud ausgelagert werden. Diese Struktur

schafft eine optimierte Balance zwischen Flexibilität, Kosten und Kontrolle. Allerdings erfordert sie eine sorgfältige Planung und eine gut durchdachte Integrationsstrategie, da ihre Implementierung oft komplex ist. Eine Herausforderung besteht in der effizienten Aufteilung des Workloads über die verschiedenen Plattformen hinweg. Bei Missallokationen kann dies zu erheblichen Mehrkosten führen. [14]

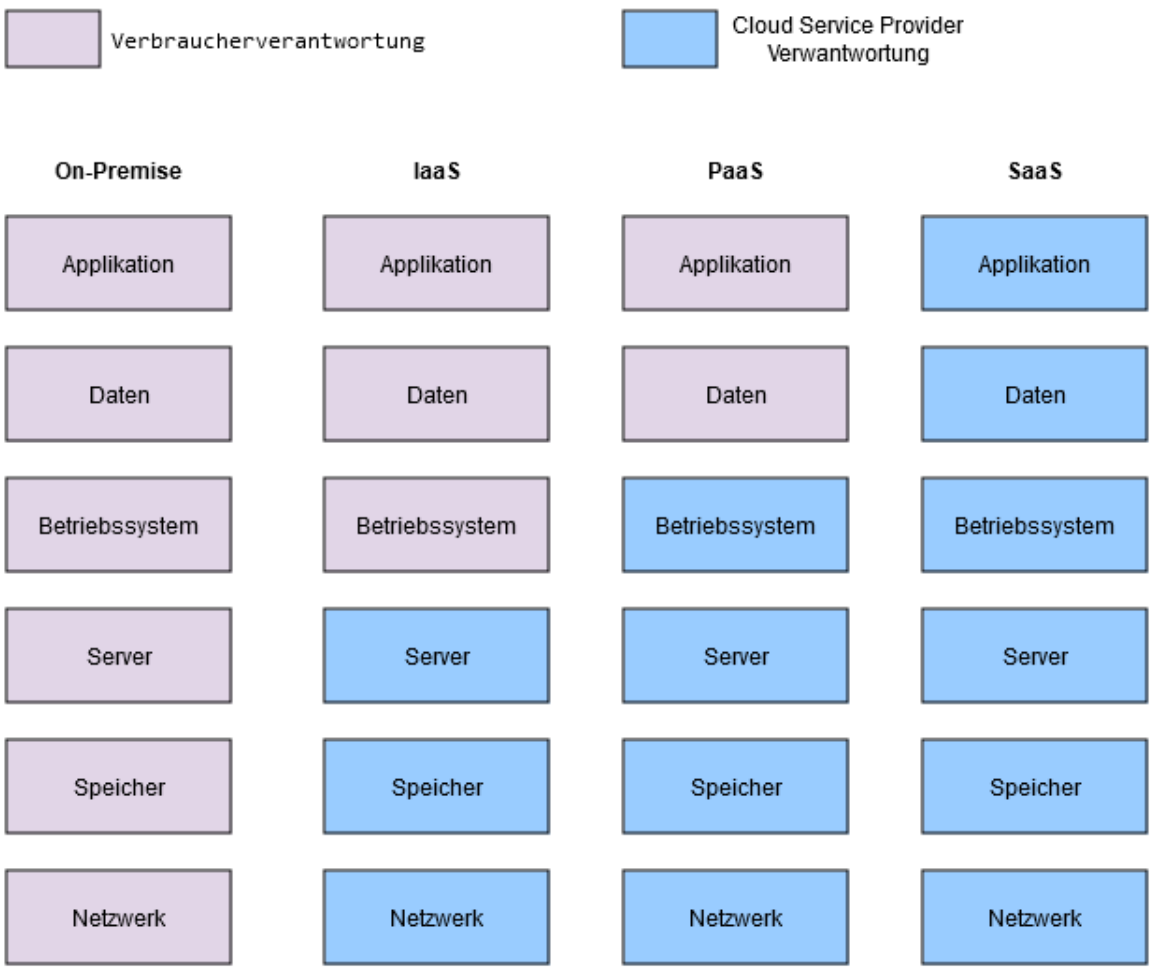

Abbildung 3.1: Verantwortlichkeiten der Umgebungen nach NIST Spezifikation[10]

3.3 Bestehende Architektur der Anwendung

Die zugrunde liegende Architektur 3.2, in die wir ERiC integrieren wollen, basiert auf einer hybriden Struktur, da sie lokale On-Premise-Komponenten mit einer Private-Cloud-Umgebung kombiniert.

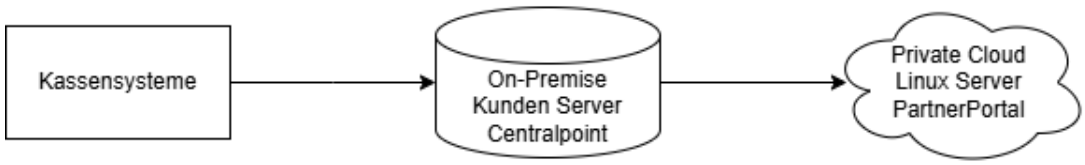

Abbildung 3.2: Systemarchitektur

Der On-Premise-Teil umfasst die Kassensysteme und den Centralpoint-Kundenserver. Die Kassensysteme laufen lokal in den Einzelhandelsgeschäften und arbeiten unabhängig von einer Cloud-Infrastruktur. Auch bei einer Unterbrechung der Internetverbindung bleibt die Funktionalität gewährleistet, da die Daten zunächst lokal verarbeitet und gespeichert werden.

Der Centralpoint-Kundenserver ist ebenfalls On-Premise und dient als zentrale Schnittstelle zwischen den Kassensystemen und der übergeordneten Infrastruktur. Die Kassensysteme senden ihre Daten an diesen Server, der sie für die weitere Verarbeitung aufbereitet.

Die Private Cloud wird durch den Linux-Server repräsentiert, auf dem das PartnerPortal betrieben wird. Dieser Server übernimmt die zentrale Datenverarbeitung und Speicherung der relevanten Informationen.

Nach der Definition des National Institute of Standards and Technology [10] ist eine Hybrid-Cloud-Architektur durch die Integration von mindestens zwei unterschiedlichen

Cloud-Umgebungen gekennzeichnet. Da unser System primär aus lokal betriebenen Kassensystemen und einem zentralen Linux-Server für die Datenverarbeitung besteht, erfüllt es nicht vollständig die Anforderungen einer klassischen Hybrid-Cloud-Struktur.

Trotzdem weist das System Eigenschaften einer hybriden Infrastruktur auf, da es den Workload zwischen Anwendungen verschiebt, mehrere Computer über ein Netzwerk verbindet und Ressourcen in einem gemeinsamen Pool konsolidiert. Die Kassensysteme und der Centralpoint-Server übernehmen die lokale Verarbeitung, was eine hohe Ausfallsicherheit und Unabhängigkeit ermöglicht. Gleichzeitig bietet der Linux-Server eine zentrale Umgebung für Datenverwaltung und Skalierbarkeit, was zu einer effizienteren Ressourcennutzung führt.

4 Integrationsansatz

Im Folgenden wird ein Implementierungskonzept für die Integration von ERiC in eine Cloud-Java-Anwendung vorgestellt. Ziel ist es, ERiC so zu integrieren, dass er effizient und skalierbar genutzt werden kann und sich nahtlos in die bestehende Struktur mit lokalen Kassensystemen, zentralem Kundenserver und privater Cloud einfügt.

4.1 Zentrale Verwaltung der ERiC-Dateien

Die Implementierung des ERiC erfordert eine zentrale Verwaltung der benötigten Dateien. Dies ist aus verschiedenen Gründen vorteilhaft:

1. **Effiziente Aktualisierung:** Eine dezentrale Installation des ERiC auf einzelnen Geräten ist nicht praktikabel, da ERiC jährliche Aktualisierungen erfordern würde, die aufwendig und fehleranfällig sein können. Ohne zentrale Verwaltung müsste jedes Gerät, wie etwa jede Kasse, separat aktualisiert werden. Durch eine zentrale Softwareverteilung hingegen können Updates effizient, konsistent und zeitgleich für alle Geräte durchgeführt werden [15, S. 8].

2. **Komplexitätsreduktion:** Kassensysteme unterscheiden sich häufig in Modellen und Softwareversionen. Eine direkte Implementierung des ERiC auf jedem Gerät könnte aufgrund dieser Unterschiede zu Kompatibilitätsproblemen führen. Durch ein zentrales Management können Softwarepakete so konfiguriert werden, dass sie konsistent auf verschiedenen Geräten laufen und so die Komplexität reduzieren. [15, S. 8].

Durch die zentrale Verwaltung der ERiC-Client-Dateien wird sowohl die Effizienz gesteigert als auch die Fehleranfälligkeit bei der Softwareverteilung minimiert.

4.2 Auslagerung des ERiC

Die ERiC-Dateien sollten aus folgenden Gründen nicht in das Projekt integriert, sondern in einen externen Speicher ausgelagert werden:

1. **Unabhängigkeit und Modularität:** Durch die Entkopplung von ERiC und der Hauptanwendung können Updates von ERiC und unserer Anwendung unabhängig voneinander durchgeführt werden. Ein Update von ERiC, das regelmäßig erforderlich ist, zieht nicht zwangsläufig ein Update der gesamten Anwendung nach sich. Dies spart Zeit, minimiert das Fehlerrisiko und ermöglicht eine flexiblere Wartung.

2. **Deployment-Paket:** Ein weiteres Argument für die externe Speicherung des ERiC ist die Reduzierung der Größe des Deployment-Pakets. Integriert man den ERiC-Client im Projekt, vergrößert sich das Paket signifikant, was zu längeren Upload-Zeiten auf dem Server und einem verlängerten Verteilungsprozess führt. Dies erhöht die Downtime und steigert die Nutzung von Speicherplatz und Bandbreite. Besonders bei regelmäßigen Updates könnte dies problematisch werden.

3. **Plattformflexibilität:** Um sicherzustellen, dass die Gesamtanwendung plattformübergreifend flexibel bleibt, ist ERiC in verschiedenen Versionen für unterschiedliche Betriebssysteme verfügbar. Die externe Speicherung ermöglicht es, das verwendete Betriebssystem zu ermitteln und die passende ERiC-Version dynamisch nachzuladen. Dieser Ansatz gewährleistet, dass die Hauptanwendung unabhängig vom Betriebssystem bleibt und keine betriebssystemspezifischen Anpassungen erfordert.

4.3 Begründung der gewählten Architektur

Basierend auf den Informationen in den Abschnitten 3.3, 4.1 und 4.2 lässt sich die Wahl der Integration von ERiC in die bestehende Architektur wie folgt begründen:

Die Entscheidung, die ERiC-Dateien in einer separaten privaten Cloud zu speichern, vereinfacht die Entwicklung, verbessert die Wartbarkeit und gewährleistet die Unterstützung verschiedener Plattformen. Die in der Abbildung 4.1 dargestellte Architektur berücksichtigt alle oben genannten Überlegungen und setzt sie um. Die jährlichen Updates können direkt im separaten Objektspeicher durchgeführt werden und bleiben vollständig von der Anwendung getrennt. ERiC wird aus einem zentralen Speicherort abgerufen, was die Softwareverteilung erheblich vereinfacht und gleichzeitig die Größe des Deployment-Pakets klein hält. Dadurch werden Upload-Zeiten reduziert und der Verteilungsprozess beschleunigt, was eine effiziente Bereitstellung der Anwendung gewährleistet.

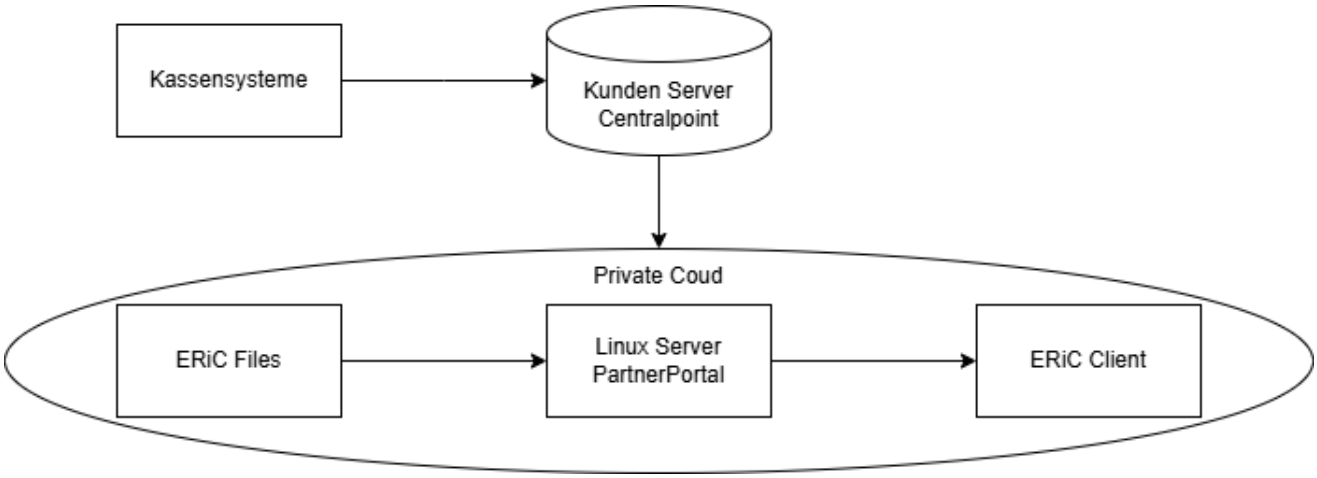

Abbildung 4.1: Systemarchitektur mit ERiC-Integration

Die Erweiterung der bestehenden Architektur um ERiC fügt sich nahtlos in das bestehende System ein, ohne dass Änderungen an der bestehenden Infrastruktur erforderlich sind.

Dadurch bleibt die Stabilität der Architektur erhalten, während die Integration flexibel und effizient gestaltet wird.

4.4 Technologische Grundlage der Implementierung

Im Folgenden werden die zentralen Technologien vorgestellt, die für die Implementierung der Anwendung erforderlich sind. Diese Technologien bilden die Grundlage für die Entwicklung und den Betrieb der Software. Der Schwerpunkt liegt dabei auf Frameworks, Programmiersprachen und Schnittstellen, die den Anforderungen der Projektumgebung entsprechen.

4.4.1 Java Platform, Enterprise Edition (Java EE)

Java EE ist ein Open-Source-Standard für Unternehmenssoftware, der eine Reihe von Spezifikationen für die Entwicklung skalierbarer, robuster und sicherer serverseitiger Java-Anwendungen bietet. Es baut auf der Java Standard Edition (Java SE) auf und erweitert diese um Komponenten für die Entwicklung von Webservices, Komponentenentwicklung, Management und Cloudkommunikations-APIs.[16]

4.4.2 Java Native Acces (JNA)

JNA ermöglicht Java-Programmen den Zugriff auf native Bibliotheken, ohne dass dafür spezieller Code für das Java Native Interface (JNI) geschrieben werden muss. Dies erleichtert die Integration von Bibliotheken, die in anderen Programmiersprachen geschrieben wurden, und erweitert somit die Funktionalität von Java-Anwendungen durch die Nutzung vorhandener nativer Bibliotheken. [17]

4.4.3 Java Architecture for XML Binding (JAXB)

JAXB ist ein Framework, das die Umwandlung zwischen XML-Dokumenten und Java-Objekten ermöglicht. Es bietet Entwicklern die Möglichkeit, XML-Inhalte in Java-Objekte zu überführen (Unmarshalling) und umgekehrt Java-Objekte in XML-Daten zu konvertieren (Marshalling). Dies erleichtert die Verarbeitung von XML-Daten in Java-Anwendungen erheblich, da es eine direkte Bindung zwischen XML-Elementen und Java-Klassen ermöglicht. JAXB nutzt dabei Annotations, um die Zuordnung zwischen XML-Strukturen und Java-Klassen zu definieren, was die Entwicklung und Wartung von Anwendungen vereinfacht. [18]

4.4.4 Jakarta Persistence API (JPA)

Die Jakarta Persistence API (JPA) ist ein Standard für die Persistenz von Java-Objekten in relationalen Datenbanken. Sie erleichtert grundlegende Datenbankoperationen wie das Erstellen, Lesen, Aktualisieren und Löschen (CRUD) durch die Definition von Entitäten und deren Zuordnung zu Datenbanktabellen mittels Annotationen. JPA unterstützt zudem flexible Abfragen über die Jakarta Persistence Query Language (JPQL) und bietet Mechanismen wie Lazy Loading für effiziente Datenzugriffe. Häufige Implementierungen wie Hibernate gewährleisten breite Anwendbarkeit und Unterstützung. [19]

4.4.5 MariaDB

MariaDB ist ein Open-Source-Fork von MySQL und wurde entwickelt, um die Datenbanksoftware nach der Übernahme von MySQL durch Oracle offen zu halten. MariaDB ist vollständig kompatibel zu MySQL und unterstützt dessen Protokoll und APIs. Dadurch können MySQL-Anwendungen ohne oder mit minimalen Anpassungen auch mit MariaDB

betrieben werden. MariaDB bietet eine Reihe von Erweiterungen, die die Performance und Sicherheit verbessern, sowie Unterstützung für parallele Abfragen.[20]

4.4.6 Java WildFly

WildFly ist ein modularer und leichtgewichtiger Open Source Application Server, der die neuesten Enterprise Java Standards von Jakarta EE und Eclipse MicroProfile implementiert. Er bietet eine flexible und leistungsfähige Plattform für die Entwicklung und den Betrieb von Java-Anwendungen und ermöglicht durch seine modulare Architektur die bedarfsgerechte Bereitstellung von Diensten.[21]

5 Umsetzung der Implementierung

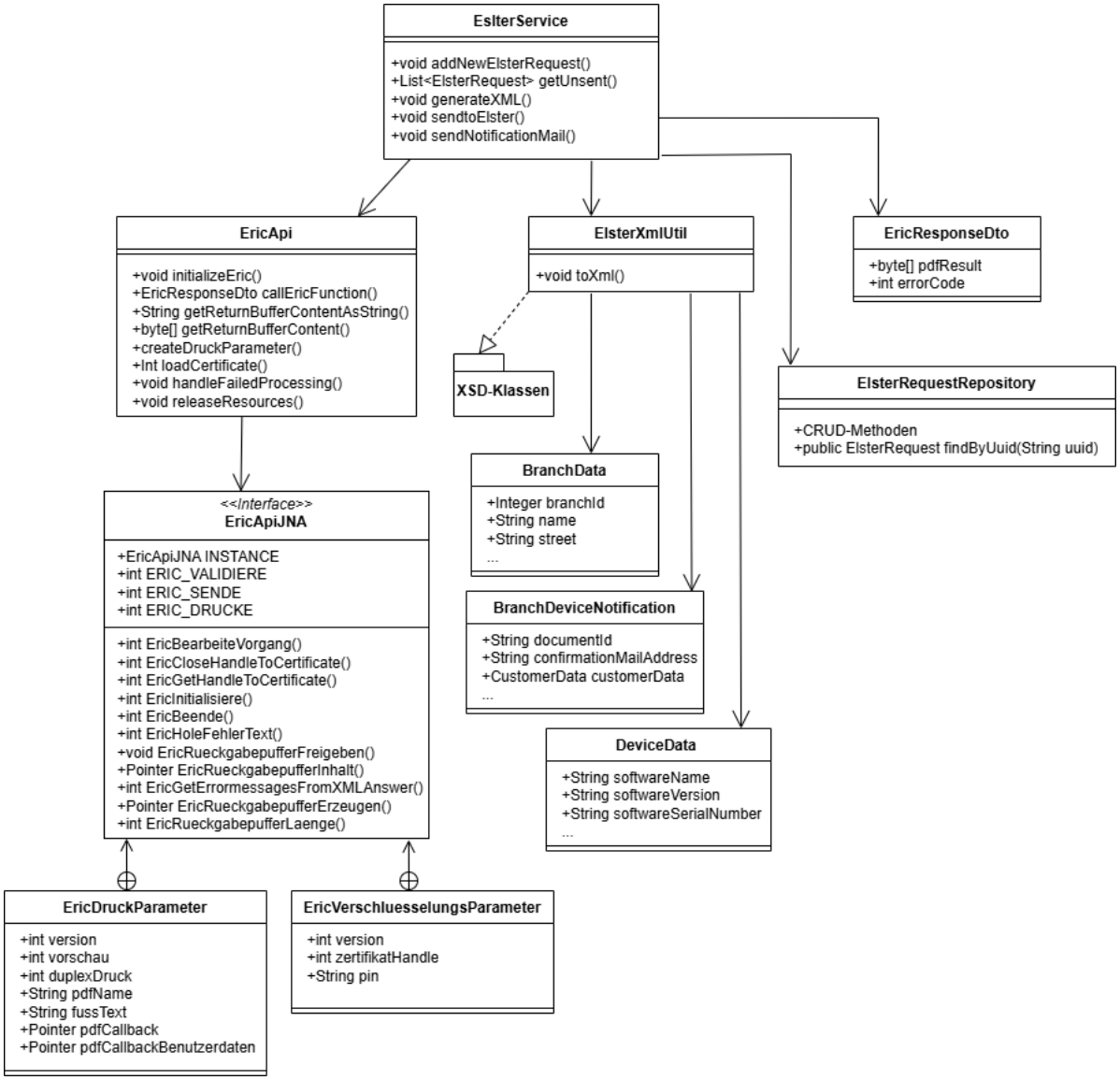

Abbildung 5.1: Klassendiagramm der ERiC-Integration

Die Klassenarchitektur ist so gestaltet, dass sie eine klare Trennung der Verantwortlichkeiten und eine modulare Struktur ermöglicht. Jede Klasse hat eine spezifische Aufgabe, wodurch die Wartbarkeit und Erweiterbarkeit der Anwendung sichergestellt werden. Die Abbildung 5.1 zeigt die zentralen Klassen und deren Interaktionen.

5.1 Beschreibung der zentralen Klassen

5.1.1 EricApi

Die Klasse `EricApi` bildet die zentrale Schnittstelle zwischen der Anwendung und der nativen ERiC-Bibliothek. Sie abstrahiert die komplexen Details der Interaktion und stellt eine leicht zugängliche API zur Verfügung. Folgende Funktionalitäten wurden in der Klasse implementiert:

Speicher allokation

Dies beinhaltet die Verwaltung der nativen Speicherbereiche, die ERiC benötigt. Für die Verarbeitung und Rückgabe von Ergebnissen wird der Speicher explizit mit der Methode EricRueckgabepufferErzeugen allokiert, die Puffer für die Rückgabe von Verarbeitungsdaten und Serverantworten bereitstellt. Das folgende Codebeispiel 5.1 erzeugt zwei Puffer:

Listing 5.1: Puffererzeugung mit ERiC

```
Pointer ericResponseBuffer = EricApiJNA.INSTANCE.
    ↪ EricRueckgabepufferErzeugen();
Pointer serverResponseBuffer = EricApiJNA.INSTANCE.
    ↪ EricRueckgabepufferErzeugen();
```

Der ericResponseBuffer dient der Speicherung von Verarbeitungsdaten wie Validierungen oder Ergebnisrückgaben, während der serverResponseBuffer Antworten des ELSTER-Servers wie Fehlermeldungen oder Bestätigungen enthält. Da der Speicher manuell allokiert wird, muss er nach Gebrauch explizit freigegeben werden. Die konsistente Freigabe der Puffer im Block finally stellt sicher, dass der Speicher unabhängig vom Verarbeitungsstatus korrekt freigegeben wird und Speicherlecks vermieden werden. Dies geschieht mit der Methode EricRueckgabepufferFreigeben, wie im Listing 5.2 gezeigt wird.

Listing 5.2: Puffer Freigabe

```
 finally {
    EricApiJNA.INSTANCE.EricRueckgabepufferFreigeben(ericResponse);
    EricApiJNA.INSTANCE.EricRueckgabepufferFreigeben(serverResponse);
}
```

Die im Puffer gespeicherten Daten werden mit der Methode EricRueckgabepufferInhalt 5.3 ausgelesen und können zur weiteren Verarbeitung oder Analyse wie folgt in ein Java-Array konvertiert werden. Zuerst wird die Länge des Rückgabepuffers ermittelt. Anschließend wird ein Pointer (dataPointer) erzeugt, der den Inhalt des Rückgabepuffers referenziert. Schließlich wird mit der Methode getByteArray der Inhalt des Pointers als Byte-Array extrahiert.

Listing 5.3: Auslesen der Pufferinhalte

```
try {
    int dataLength = EricApiJNA.INSTANCE.EricRueckgabepufferLaenge(buffer);
    Pointer dataPointer =EricApiJNA.INSTANCE.EricRueckgabepufferInhalt(
        ↪ buffer);
    dataPointer.getByteArray(0, dataLength);
}
```

Dynamischer Pfad

Die plattformübergreifende Nutzung des ERiC-Clients stellte eine Herausforderung dar. Um dies zu lösen, wurde ein dynamisches System implementiert, das das Betriebssystem erkennt und die Pfade zur Laufzeit anpasst. Die Methode initializeEric veranschaulicht im Listing 5.4 diesen Ansatz.

Listing 5.4: Dynamische Anpassung der Pfade

```
String os = System.getProperty("os.name").toLowerCase();
if (os.contains("win")) {
    tempPath = "D:/tmp/";
    libPath = tempPath + "eric/dll";
    bucketSubDir = "dll";
} else {
    tempPath = "/tmp/";
    libPath = tempPath + "eric/lib";
    bucketSubDir = "lib";
}
File libDir = new File(tempPath+"eric");
libDir.mkdirs();
```

Zuerst wird der Name des Betriebssystems mit der Eigenschaft os.name abgefragt. Anschließend wird der Pfad für das erkannte Betriebssystem gesetzt: Für Windows wird der Pfad D:/tmp/eric/dll und für Linux der Pfad /tmp/eric/lib verwendet. Um sicherzustellen, dass das Zielverzeichnis existiert, wird die Methode libDir.mkdirs() aufgerufen. Diese Methode versucht, das Verzeichnis anzulegen, falls es noch nicht existiert. Wenn das Verzeichnis bereits vorhanden ist, wird keine weitere Aktion ausgeführt. Die Existenz des Zielverzeichnis garantiert aber noch nicht das die Bibliothek auch vorhanden ist. Um zu prüfen,

ob die Bibliothek lokal vorhanden ist, wird eine Liste aller Dateien im Zielverzeichnis und eine Liste aller Dateien im Bucket der privaten Cloud erstellt. Wenn eine Datei im Zielverzeichnis nicht vorhanden ist, wird sie heruntergeladen. Der Indikator bucketSubDir gibt an, in welchem Unterordner der Cloud gesucht werden soll. Der konkrete Zugriff auf den Cloud-Speicher erfolgt über die Klasse `MinioProducer` 5.1.6 und wird weiter unten erläutert. Die folgende Schleife 5.5 durchläuft die Liste aller Cloudelemente und führt den Download durch.

Listing 5.5: Herunterladen fehlender Dateien aus der Cloud

```
for(String file : files) {
    File result = new File(libDir,file);
    if(result.exists()) {
        continue;
    }
byte[] content = minioProducer.getFile(LIB_BUCKET_NAME,file, true);
```

Anschließend wird der Pfad über System Properties mit der Eigenschaft jna.library.path gesetzt, so dass Java Native Access die Bibliothek an dieser Stelle suchen kann. Dieser Mechanismus stellt sicher, dass die Bibliotheken unabhängig von der Zielplattform immer verfügbar sind.

Fehlerauswertung

Ein Teil der Fehlerauswertung wird über die Methode `handleFailedProcessing` abgewickelt. Sie wird im `try`-Block in der `callEricFunction` ausgeführt. Es wird ein Speicherpuffer mit der Methode `EricRueckgabepufferErzeugen` 5.1 erzeugt. Dieser Puffer dient dazu, den Fehlertext von ERiC zu empfangen. Anschließend wird die Methode

`EricHoleFehlerText` 5.6 aufgerufen, um den vollständigen Fehlertext basierend auf dem Fehlercode zu laden. Der Text wird aus dem Puffer extrahiert und zur weiteren Analyse verwendet.

Listing 5.6: Fehlerbehandlung

```
EricApiJNA.INSTANCE.EricHoleFehlerText(result, buffer);
getReturnBufferContentAsString(buffer);
```

Zusätzlich wird das Logfile des ERiC ausgelesen 5.7, um weitere Informationen über den Fehler zu erhalten. Dazu wird das Logfile mit einem `BufferedReader` zeilenweise in der Konsole ausgegeben.

Listing 5.7: Ausgabe des Logfile

```
File logfile = new File(ericApi.getTempPath(), "eric.log");
BufferedReader reader = new BufferedReader(new FileReader(logfile));
String line = reader.readLine();
```

Initialisierung und Aufruf von ERiC

Die Initialisierung des ERiC erfolgt in 5.8 durch die Definition einer Int Variable und den Aufruf der Methode `EricInitialisiere`. Da es sich bei `EricApiJNA` um ein Interface handelt, werden alle Variablen und Methoden automatisch als `static` geladen und stehen zur Verfügung, ohne dass ein Objekt instanziiert werden muss. Die Methode benötigt zwei Parameter, einen für den Bibliothekspfad (libPath) und einen für den Pfad, an dem das Logfile abgelegt wird (tempPath).

Listing 5.8: Initialisierungsaufruf

```
int result = EricApiJNA.INSTANCE.EricInitialisiere(libPath, tempPath);
```

Nach erfolgreicher Initialisierung erfolgt der Hauptaufruf des ERiC über die Methode `EricBearbeiteVorgang` 5.9. Es müssen alle notwendigen Parameter übergeben werden, die unten aufgeführt sind.

Listing 5.9: ERiC Bearbeitungsaufruf

```
int result = EricApiJNA.INSTANCE.EricBearbeiteVorgang(xml, datenartVersion,
    bearbeitungsFlags, druckEinstellungen, verschluesselungsParameter,
    null, ericResponseBuffer, serverResponseBuffer);
```

Die Parameter haben folgende Bedeutung:

- **xml:** Aufbereitete Daten im XML-Format, die verarbeitet werden sollen.
- **datenartVersion:** Gibt die Art der Anfrage an. In unserem Fall: `Aufzeichnung146a_1`.
- **bearbeitungsFlags:** Bestimmt die auszuführenden Funktionen wie Validierung, Übermittlung oder Druck. Beispiele:
  - `ERIC_VALIDIERE`: Führt eine Validierung der XML-Daten durch.
  - `ERIC_SENDE`: Übermittelt die XML-Daten an die ELSTER-Schnittstelle.
  - `ERIC_DRUCKE`: Erzeugt ein PDF-Dokument.
- **druckEinstellungen:** Definiert die Druckparameter für das zurückgelieferte PDF, wie z. B.:
  - `duplexDruck`: Aktiviert den Duplexdruck.
  - `fussText`: Legt den Text im Fußbereich des Dokuments fest.
- **verschluesselungsParameter:** Übermittelt Verschlüsselungsdetails wie:

  - Das verwendete Zertifikat (`ZertifikatHandle`).
  - Die zugehörige PIN.
  - Die Version der Verschlüsselung.

- **ericResponseBuffer und serverResponseBuffer:** Puffer, die zur Speicherung von Verarbeitungsdaten und Serverantworten dienen. Die Allokation dieser Puffer wird in Abschnitt 5.1.1 beschrieben.

Verarbeitung des Zertifikats

Für eine erfolgreiche Übertragung mit ERiC benötigen wir ein gültiges Zertifikat von Elster, mit dem wir uns als berechtigter Übermittler authentifizieren. Das Laden des Zertifikates wird mit dem Aufruf von `EricGetHandleToCertificate` 5.10 durchgeführt:

Listing 5.10: Zertifikathandle öffnen und schließen

```
int resultHandle = EricApiJNA.INSTANCE.EricGetHandleToCertificate(
    ↪ certificateHandle, pinSupport, certificatePath);
int closeResult = EricApiJNA.INSTANCE.EricCloseHandleToCertificate(
    ↪ resultHandle);
```

Die Methode benötigt 3 Parameter: 1. das `certificateHandle`, in dem eine eindeutige Referenz auf unser Zertifikat gespeichert wird. 2. den `pinSupport`, dort kann bei Bedarf ein Pin zur weiteren Absicherung hinterlegt werden. Diese beiden Variablen werden als IntByReference gespeichert und mittels JNA an ERiC übergeben. Es handelt sich also um Zeiger auf Speicheradressen. 3. `certificatePath` der Pfad zum Zertifikat. Nach Abschluss der Übertragung wird die Methode `EricCloseHandleToCertificate` aufgerufen, die die Speicheradressen wieder freigibt und den Zertifikatshandel schließt.

5.1.2 EricApiJNA

Das Interface `EricApiJNA` 5.11 definiert die notwendigen Methoden für die direkte Interaktion mit der nativen ERiC-Bibliothek. Die Nutzung von Java Native Access 4.4.2 ermöglicht eine einfache Integration.

Listing 5.11: ERiC Interface

```
import com.sun.jna.Native; import com.sun.jna.Library;
public interface EricApiJNA extends Library {
EricApiJNA INSTANCE = Native.load("ericapi", EricApiJNA.class);
    int EricInitialisiere(String pluginPfad, String logPfad);
    int EricBeende();
    void EricRueckgabepufferFreigeben(Pointer rueckgabepuffer);
    (...)
```

Die Definition der Schnittstelle enthält eine Instanz, die über `Native.load` die Verbindung zur nativen Bibliothek herstellt. Diese Instanz, `INSTANCE` wird automatisch beim Laden der Klasse initialisiert. Die Schnittstelle enthält alle wesentlichen Funktionen wie Initialisierung, Pufferverwaltung und Fehlerbehandlung, die eine einfache und effiziente Bedienung des ERiC ermöglichen.

Hier werden auch die Parameter `EricVerschluesselungsParameter` 5.12 und `EricDruckParameter` 5.13 als statische interne Klassen innerhalb des Interface definiert. Sie dienen als Datenübergabe-Objekte, die alle notwendigen Konfigurationsparameter bündeln. Die Klassen erben von der Structure-Klasse. Structure gehört zu JNA und wird verwendet, um Datenstrukturen zu definieren, die mit C-APIs ausgetauscht werden. Diese Objekte können vor ihrer Verwendung instanziiert und ihre Attribute entsprechend

den gewünschten Einstellungen gesetzt werden. Anschließend werden sie an die Methode `EricBearbeiteVorgang` übergeben.

Listing 5.12: Verschlüsselungs Parameter

```
public static class EricVerschluesselungsParameter extends Structure {
    public int version;
    public int zertifikatHandle;
    public String pin;
}
```

Listing 5.13: Druck Parameter

```
public static class EricDruckParameter extends Structure {
    public int version;
    public int duplexDruck;
    public String pdfName;
    (...)
```

5.1.3 ElsterService

Die Klasse `ElsterService` koordiniert die Verarbeitung von ELSTER-Anfragen, indem sie verschiedene Funktionen wie die Generierung der XML-Daten, die Kommunikation mit ERiC und die Benachrichtigung der Benutzer integriert. Sie fungiert als Schnittstelle zwischen den internen Datenstrukturen der Anwendung und der ELSTER-Schnittstelle.

Eine Komponente ist die Methode `generateXML` 5.14, die die XML-Daten für eine Anfrage erzeugt. Dabei wird das in der Datenbank gespeicherte Anfrageobjekt deserialisiert und mit Hilfe der Klasse `ElsterXmlUtil` in ein XML-Dokument umgewandelt. Dieses Dokument wird dann in der Anfrage gespeichert.

Listing 5.14: DTO-Deserialisierung und XML-Erstellung

```
public void generateXML(ElsterRequest elsterRequest) {
    // Deserialisieren des DTO
    String dtoSerialized = new String(elsterRequest.getRequestDto(),
        ↪ StandardCharsets.UTF_8);
    BranchDeviceNotification branchDeviceNotification = objectMapper.
        ↪ readValue(dtoSerialized, BranchDeviceNotification.class);

    // XML-Generierung
    XmlMapper xmlMapper = new XmlMapper();
    byte[] xmlBytes = xmlMapper.writeValueAsString(
        ElsterXmlUtil.toXml(
            branchDeviceNotification.getBranchData(),
            branchDeviceNotification.getBranchData().getDeviceData(),
            branchDeviceNotification.getCustomerData()
        )
    ).getBytes(StandardCharsets.UTF_8);
    elsterRequest.setRequestXML(xmlBytes);
}
```

Nach der XML-Generierung kann die Methode `sendtoElster` 5.15 verwendet werden, um die Anfrage über ERiC an die ELSTER-Schnittstelle zu senden. Dort wird das XML-Dokument zusammen mit dem Zertifikat und der PIN verarbeitet und die Antwort im Anfrageobjekt gespeichert.

Listing 5.15: Übermittlung von ElsterRequest an ERiC

```
public void sendtoElster(ElsterRequest elsterRequest) {
    EricResponseDto ericResponseDto = ericApi.callEricFunction(new String(
        ↪ elsterRequest.getRequestXML(), "UTF-8"), DATENART_VERSION,
        ↪ certificatePath, pin);
    elsterRequest.setResponseCode(String.valueOf(ericResponseDto.
        ↪ getErrorCode()));
    elsterRequest.setSentAt(ZonedDateTime.now());
    elsterRequest.setResponsePDF(ericResponseDto.getPdfResult());
    elsterRequestRepository.save(elsterRequest);
}
```

Die Klasse `ElsterService` vereint somit mehrere Kernfunktionen wie die Aufbereitung der Daten, den Aufruf des ERiC und die Weiterverarbeitung der Serverantwort. Diese strukturierte Implementierung erleichtert die Verwaltung von ELSTER-Anfragen und stellt sicher, dass alle relevanten Schritte in einem durchgängigen Prozess integriert sind.

5.1.4 ElsterXmlUtil

Die Klasse `ElsterXmlUtil` 5.16 dient der Generierung und Verarbeitung von XML-Daten und gewährleistet, dass diese den ELSTER-Spezifikationen entsprechen. Sie verwendet die aus den XSD-Dateien 5.1.8 generierten Klassen, um sicherzustellen, dass die Struktur und der Inhalt der XML-Daten korrekt sind. Die zentrale Aufgabe dieser Klasse ist es, die generierten Klassen mit Daten aus der Anwendung zu befüllen und so die XML-Dokumente entsprechend den Vorgaben zu erstellen.

Listing 5.16: Befüllung der XML-Struktur

```
public class ElsterXmlUtil {
AdrInl379369584CType xmlData = new AdrInl379369584CType();
xmlData.setStrasse(branchDeviceNotification.getCustomerData().getStreet());
xmlData.setHausnummer(branchDeviceNotification.getCustomerData().
    ↪ getStreetNumber());
(...)
```

5.1.5 ElsterRequestRepository

Die `ElsterRequestRepository` 5.17 Klasse dient als Schnittstelle zur Datenbank und ermöglicht mittels JPA 4.4.4 den Zugriff auf die ELSTER-Anfragen. Standardmäßig werden alle CRUD-Methoden (**C**reate, **R**ead, **U**pdate, **D**elete) zur Verfügung gestellt. Mit diesen sind die nötigsten Instrumente gegeben, um die Datenbank konsistent zu halten. Falls notwendig, lassen sich individuelle Queries schreiben, um spezielle Operationen zu ermöglichen.

Listing 5.17: CRUD Operation save

```
elsterRequestRepository.save(elsterRequest);
```

5.1.6 MinioProducer

Diese Klasse integriert den MinIO-Objektspeicher in die Anwendung und sorgt dafür, dass die benötigten Ressourcen vom ERiC-Client geladen werden können. Es wird eine MinioClient-Instanz mit der Methode `builder` 5.18 erstellt, über die wir mittels URL und Logindaten auf unseren Objektspeicher zugreifen können.

Listing 5.18: Authentifizierung Minio

```
MinioClient minioClient = MinioClient.builder()
    .endpoint("https://example-minio-server.com")
    .findValueByName("Elster Rich Client")
    .credentials("username", "password")
    .build();
```

Der Abruf von ERiC erfolgt über die Methode getObject 5.19. Die Dateien werden direkt aus dem Bucket geladen und lokal zwischengespeichert. Der Bucket ist vergleichbar mit einer Ordnerstruktur, er gibt an, unter welchem Container die Dateien zu finden sind.

Listing 5.19: Dateiabruf vom MinIO-Bucket

```
byte[] ericFile = client.getObject(GetObjectArgs.builder()
.bucket("eric-bucket")
.object("path/to/eric-client.zip")
.build()).readAllBytes();
```

5.1.7 BranchData, BranchDeviceNotification, DeviceData

Diese Klassen stellen die Datenstrukturen dar, die für die Kommunikation zwischen der Anwendung und der ELSTER-Schnittstelle benötigt werden. Mit den Daten wird das XML in der ElsterXmlUtil 5.1.4 Klasse für die Übermittlung an Elster erstelle und befüllt.

- **BranchData:** Stellt Informationen über eine Filiale bereit, z. B. ID, Name und Adresse.
- **BranchDeviceNotification:** Enthält gerätespezifische Informationen und Kundendaten.

- **DeviceData:** Speichert Softwaredetails eines Geräts wie Name, Version und Seriennummer.

5.1.8 XSD-Klassen

Das Package `XSD-Klassen` umfasst die 41 Java-Klassen, die aus den XSD-Dateien der ELSTER-Schnittstelle mit JAXB 4.4.3 generiert wurden. Diese Klassen 5.20 definieren die Struktur der XML-Daten und stellen sicher, dass die erzeugten Dokumente den Spezifikationen entsprechen.

Listing 5.20: XSD-Klasse aus ERiC

```
public class AngabenAufzeichnungssystem355871649CType {
@XmlElement(name = "Art")
protected String art;
@XmlElement(name = "ArtSonstiges")
protected String artSonstiges;
(...)
```

5.2 Tests

Um die korrekte Funktionalität von ERiC sicherzustellen, wurden mehrere Tests implementiert, die unterschiedliche Aspekte der Integration abdecken. Diese Tests überprüfen die Verarbeitung der ELSTER-Anfragen, die Interaktion mit dem ERiC-Client sowie die korrekte Generierung und Speicherung der Daten.

5.2.1 EricApiIntegrationTest

Die Klasse `EricApiIntegrationTest` 5.21 überprüft die Funktionalität der `EricApi`, die als Schnittstelle für die direkte Kommunikation mit ERiC dient. Ein zentraler Bestandteil des

Tests ist die Verarbeitung von XML-Requests. Der Test `testCallEricFunction` simuliert, wie die `EricApi` einen XML-Request verarbeitet. Dabei wird ein Testzertifikat als Stream geladen und ein Test-XML verwendet, um sicherzustellen, dass ERiC und der Elster-Annahmeserver das erwartete Ergebnis liefert.

Nach der Verarbeitung prüft der Test, ob das zurückgegebene Objekt ungleich Null ist und der Fehlercode den Wert 0 hat, was eine erfolgreiche Verarbeitung bestätigt. Wir testen, ob ein PDF-Dokument erzeugt wurde, das länger als 100 Zeichen ist. PDF-Dokumente sind in der Regel sehr lang und eine kurze Version kann auf Inkonsistenzen hinweisen.

Listing 5.21: Funktionsaufruf und Ergebnisprüfung

```
public class EricApiIntegrationTest {
EricResponseDto result = ericApi.callEricFunction(xml, datenartVersion,
    ↪ certificate, pin);
Assert.assertNotNull(result);
Assert.assertEquals(0, result.getErrorCode());
Assert.assertTrue(result.getPdfResult().length > 100);
```

Zusätzlich werden die Logfiles in der Konsole ausgegeben 5.22, um sicherzustellen, dass es keine weiteren Fehler gibt, die durch den Test nicht abgebildet werden.

Listing 5.22: Lesen des Logfiles

```
File logfile = new File(ericApi.getTempPath(), "eric.log");
BufferedReader reader = new BufferedReader(new FileReader(logfile));
String line = reader.readLine();
```

5.2.2 ElsterServiceTest

Der Test `generateXMLAndSaveToDatabase` 5.23 überprüft die korrekte Generierung und Speicherung der XML-Daten für Elster in der Datenbank. Es wird zunächst in der Me-

thode `setUp` eine Testumgebung mit vorbereiteten Daten aufgebaut. Dabei werden die Objekte `BranchDeviceNotification`, `CustomerData` und `BranchData` initialisiert und mit Beispieldaten gefüllt, um realistische Eingabedaten für die Methoden des `ElsterService` zu simulieren. Mit den Daten wird anschließend ein neues `elsterRequest` Objekt erstellt, welches daraufhin an die Methode `generateXML` übergeben wird. Der Fokus liegt auf der Methode `generateXML`, deren Funktionalität durch `generateXMLAndSaveToDatabase` überprüft wird.

Die Erstellung des XML erfolgt durch die Klasse `ElsterXmlUtil` 5.1.4, die für die korrekte Struktur und Gültigkeit der XML-Daten verantwortlich ist. Nach der Generierung werden die XML-Daten in der Datenbank gespeichert. Anschließend ruft der Test die gespeicherten Daten über die Uuid ab, um sicherzustellen, dass die Generierungs-, Speicher- und Abrufprozesse korrekt funktionieren.

Listing 5.23: Erstellung und Abruf von XML-Daten

```
elsterService.generateXML(elsterRequest);
ElsterRequest retrievedRequest = elsterRequestRepository.findByUuid(
    ↪ elsterRequest.getUuid());
assertNotNull("ElsterRequest retrieved from database", retrievedRequest);
String generatedXml = new String(retrievedRequest.getRequestXML(),
    ↪ StandardCharsets.UTF_8);
System.out.println("Generated XML from database:\n" + generatedXml);
```

5.3 Probleme beim Zertifikatspfad

Bei der Verarbeitung des Zertifikats durch die Methode `EricGetHandleToCertificate` kam es zu Problemen, da der Pfad zum Zertifikat nicht korrekt ausgelesen wurde. Anstelle des erwarteten Pfades `D:\tmp\eric\test-softidnr-pse.pfx` wurden fehlerhafte und nicht

zusammenhängende Zeichen aus dem Speicherbereich angehängt. Dies führte zu einem fehlerhaften Pfad, wie die folgenden Ausgaben zeigen:

```
szPath = D:\tmp\eric\test-softidnr-pse.pfx6¦µ5
szPath = D:\tmp\eric\test-softidnr-pse.pfxter
szPath = D:\tmp\eric\test-softidnr-pse.pfxüµ5
szPath = D:\tmp\eric\test-softidnr-pse.pfx
szPath = D:\tmp\eric\test-softidnr-pse.pfxd
```

Dieses Verhalten deutete auf ein tiefgreifendes Problem in der Methode von ERiC hin, bei dem die Speicheradressen falsch verarbeitet und ungewollt überschrieben werden. Es führte auch bei identischen Anfragen zu einer hohen Fehlerrate. Deshalb kam die Vermutung auf, es könnte an ERiC liegen und nicht an der Implementierung in unserem Projekt. Schließlich konnte das Problem durch eine Änderung des Datentyps in der Methode gelöst werden. Ursprünglich wurde der Pfad als `byte[]` übergeben, nach der Umstellung auf `String` als Parameter wurde der Fehler behoben und der Pfad korrekt ausgelesen. Ein `String` ist zwar intern ebenfalls ein Byte-Array, jedoch mit einem wichtigen Unterschied: Ein `String` enthält implizit Informationen über seine Länge und wird in der Regel mit einem Null-Terminator (\0) abgeschlossen, der das Ende der Zeichenkette markiert. Bei der Übergabe als `byte[]` fehlte diese Abschlusskennung, sodass die Methode nicht erkennen konnte, wo der Pfad endet. Dadurch wurden zusätzliche Bytes aus dem Speicher gelesen, die nicht zum Pfad gehörten, was die fehlerhaften Zeichen verursachte.

Dies zeigt, wie wichtig die genaue Handhabung von Datentypen und Speicheradressen bei der Kommunikation mit externen Systemen ist. Dieser kleine Fehler hat viel Zeit

gekostet und hätte durch eine genauere Recherche in der Dokumentation vermieden werden können.

5.4 Ergebnis

In diesem Abschnitt analysieren wir die Kommunikation mit der ERiC-Bibliothek auf Basis des Logfiles. Wir beleuchten dabei die wesentlichen Schritte und deren Bedeutung, die bei einer Anfrage über ERiC durchgeführt werden.

5.4.1 Initialisierung der ERiC-Bibliothek

Der Prozess beginnt mit der Initialisierung der ERiC-Bibliothek. Hierbei werden die Kernbibliotheken geladen, die Zielarchitektur bestimmt und die notwendigen Plugins registriert. Dies stellt sicher, dass alle Funktionen der Bibliothek verfügbar sind.

```
INFO: ERiC-Instanz "000001C4627237F0" erzeugt
INFO: Zielarchitektur=x86-64
INFO: PluginManager: Suche nach Plugins im Verzeichnis:\ERiC-40.3.4.0\
   ↪ Windows-x86_64\dll
```

5.4.2 Laden des Zertifikats

Im nächsten Schritt wird ein Zertifikat aus dem angegebenen Pfad geladen. Dies beinhaltet die Initialisierung des eSigner-Tokens sowie das Hinzufügen des Zertifikats unter einem spezifischen Zertifikathandle.

```
INFO: ERiCCertManager: Zertifikat erzeugen zu /Test_Zertifikate/test-
   ↪ softidnr-pse.pfx
INFO: Portalzertifikat erkannt
INFO: Fuege Zertifikat unter Zertifikathandle 1 ein.
```

5.4.3 Erzeugen von Rückgabepuffern und Einleiten des Vorgangs

Anschließend werden die Rückgabepuffer allokiert. Diese Speicherbereiche dienen zur Aufnahme der Bearbeitungsergebnisse sowie der Serverantwort. Mit dem Aufruf von EricBearbeiteVorgang wird der Bearbeitungsprozess gestartet.

```
INFO: }}>>EricMtRueckgabepufferErzeugen<< result = 0000024A01E124D0
INFO: }}>>EricMtRueckgabepufferErzeugen<< result = 0000024A01E12750
INFO: EricRueckgabepufferHandle rueckgabeXmlPuffer Adresse 0000024A01E124D0
INFO: EricRueckgabepufferHandle serverantwortXmlPuffer Adresse 0000024
    ↪ A01E12750
```

5.4.4 Auslesen und Validieren der Eingabeparameter

Im nächsten Schritt werden die übergebenen Parameter von ERiC ausgelesen. Diese umfassen Informationen wie die Datenartversion, Bearbeitungsflags sowie die Kryptoparameter. Diese Parameter sind entscheidend, um den Verarbeitungsprozess korrekt zu initialisieren.

```
INFO: Datenartversion = Aufzeichnung146a_1
INFO: Eingangs-XML vorhanden. Laenge=4822
INFO: Bearbeitungsflags =  VALIDIERE SENDE DRUCKE
INFO: Druckparameter = version=4 vorschau=0 duplexDruck=0
```

Anschließend wird das Eingabe-XML validiert und einer Schemaprüfung unterzogen. Dies stellt sicher, dass die Daten formal korrekt und für die weitere Verarbeitung geeignet sind.

```
INFO: Fuehre Schemapruefung der Eingangs-XML-Daten durch...
INFO: validateXml: XML-Datei wurde ohne Fehler und Warnungen validiert
INFO: Schemapruefung erfolgreich
```

5.4.5 Freigeben der Ressourcen

Zum Abschluss des Prozesses werden die allokierten Speicherbereiche der Rückgabepuffer freigegeben. Zudem wird das verwendete Zertifikat-Handle geschlossen.

```
INFO: }} Ausstieg >>EricMtRueckgabepufferFreigeben<< rc = 0
INFO: {{ Einstieg >>EricMtRueckgabepufferFreigeben<<
INFO: Rueckgabepuffer-Handle = 0000024A01E12750
INFO: }} Ausstieg >>EricMtRueckgabepufferFreigeben<< rc = 0
INFO: {{ Einstieg >>EricMtCloseHandleToCertificate<<
INFO: }} Ausstieg >>EricMtCloseHandleToCertificate<< rc = 0
```

5.4.6 Analyse der Automatisierung

Der Prozess ist weitgehend automatisiert und dauert nur wenige Sekunden, da die meisten benötigten Daten bereits in den Kassen vorhanden sind und ausgelesen werden können. Nur wenige Angaben müssen vom Benutzer manuell ergänzt werden. Um die restlichen Details zu erhalten, wird auf der Kasse eine Maske angezeigt, in der diese ergänzt werden können. Anschließend erfolgt die vollautomatische Übertragung über unsere Architektur an die Finanzbehörde.

Ohne Automatisierung müssen die Daten manuell über ein Formular oder eine XML-Datei auf der Website der Finanzbehörde eingereicht werden. Die Erstellung des XML-Datei ist für den Normalverbraucher zu kompliziert. Bei beiden Varianten können die Kassendaten nicht automatisch ausgelesen werden und erfordern zeitaufwändige Recherchen, zudem besteht die Notwendigkeit eines separaten Computers mit Internetzugang.

In der Metastudie des Instituts der deutschen Wirtschaft Köln [22, S. 29] wurde eine signifikante Produktivitätssteigerung durch Automatisierung von bis zu 20 Prozent festgestellt.

Dieser Wert basiert auf der Auswertung verschiedener Studien und kann je nach Branche und Unternehmensgröße variieren. Übertragen auf die Automatisierung der Meldepflichten kann von einer mindestens ähnlichen Effizienzsteigerung ausgegangen werden.

6 Fazit

Die Integration des Elster Rich Clients (ERiC) in eine Cloud-Java-Anwendung stellt einen entscheidenden Schritt zur Automatisierung der gesetzlichen Meldepflicht für elektronische Aufzeichnungssysteme dar. Im Rahmen dieser Arbeit konnte die Umsetzung erfolgreich realisiert werden. Alle Kernfunktionen wie die Verarbeitung von XML-Daten, die Verwaltung von Zertifikaten sowie die Kommunikation mit der ELSTER-Schnittstelle wurden zuverlässig umgesetzt. Die gewählte Architektur und technologische Basis erlauben eine effiziente Skalierung und flexible Anpassung an zukünftige Anforderungen.

Die Entwicklung hat aber auch Herausforderungen aufgezeigt. Unter anderem zeigte die Problematik der Speicheradressverwaltung beim Zugriff auf das Zertifikat, wie kritisch eine genaue Implementierung und Fehleranalyse ist. Eine gründliche Dokumentation und ausführliche Tests sind erforderlich, um mögliche Fehlerquellen zu minimieren. Auch die Wahl eines geeigneten Speicherortes für die ERiC-Bibliotheken erwies sich als entscheidend. Nach eingehender Analyse wurde eine externe Cloud-Lösung implementiert, die alle identifizierten Probleme wie Plattformunabhängigkeit, zentrale Verfügbarkeit und einfache Aktualisierung der Bibliotheken erfolgreich löst. Diese Entscheidung gewährleistet nicht nur einen effizienten Umgang, sondern auch einen reibungslosen Betrieb und eine erhöhte Flexibilität der Infrastruktur.

Ein vollständiger Systemtest mit realen Steuerdaten steht allerdings noch aus. Dieser wird notwendig sein, um das Zusammenspiel aller Komponenten unter praxisnahen Bedingungen zu testen und die Robustheit der Implementierung sicherzustellen.

Insgesamt wurde das Ziel der Arbeit erreicht und die Ergebnisse bieten eine solide Grundlage für zukünftige Erweiterungen oder die Übertragung des Konzepts auf andere Systeme. Es bleibt jedoch zu diskutieren, wie die rechtlichen und organisatorischen Rahmenbedingungen eine vollständige Automatisierung beeinflussen. Der von ALVARA eingeschlagene Weg, die XML-Daten zu generieren und die Verantwortung für die Übermittlung an die Kunden zu übertragen, zeigt beispielhaft, wie bürokratische Hürden technische Innovationen einschränken können. Dies unterstreicht die Notwendigkeit, technische und regulatorische Anforderungen in Einklang zu bringen, um langfristig eine effiziente und rechtssichere Digitalisierung von Steuerprozessen zu ermöglichen. Letztlich führt diese Mehrarbeit zu einem erhöhten Aufwand, was die Produktivität eines Unternehmens beeinträchtigen kann.

6.1 Ausblick

Im Hinblick auf die rechtlichen Rahmenbedingungen ist die Integration des ERiC in unserem Unternehmen ALVARA Digital Solutions zu hinterfragen. Der Gesetzgeber schreibt vor, dass der Datenübermittler für die Richtigkeit und Vollständigkeit der Daten zu haften hat. Für ALVARA stellt das eine besondere Herausforderung dar, da in der bestehenden Unternehmensstruktur folgende Verantwortlichkeiten enthalten sind:

ALVARA verkauft Lizenzen für Kassensysteme an sogenannte Reseller und bietet Cloud-Services auf SaaS-Basis an. Die Reseller vermieten diese Kassensysteme an ihre Kunden weiter und übernehmen den direkten Support. ALVARA ist für die Betreuung der zugrunde liegenden IT-Infrastruktur verantwortlich. Da ALVARA jedoch keinen direkten Zugriff auf die Datenerfassung und deren Kontrolle beim Endkunden hat, fehlen die notwendigen Instrumente, um die Richtigkeit der Daten zu gewährleisten. Insbesondere gibt es keine

praktikable Möglichkeit, die Identität der Endkunden zu überprüfen (z.B. über die Steuernummer). Dies würde den Automatisierungseffekt durch einen erheblichen Mehraufwand zunichtemachen.

Vor diesem Hintergrund hat sich ALVARA entschieden, die vorgeschlagene Umsetzung nicht vollständig durchzuführen. Stattdessen beschränkt sich das Unternehmen darauf, die XML-Dateien zu erstellen und an die Reseller weiterzuleiten, damit diese die Einreichung bei Elster selbst vornehmen können. Diese Vorgehensweise beseitigt das Haftungsrisiko für ALVARA und verlagert die Verantwortung auf die Parteien, die direkten Einfluss auf die Datenqualität haben.

Auch wenn die Möglichkeit der automatisierten Einreichung technisch gegeben ist, macht dies nur Sinn, wenn die Kassensysteme vollständig vom Kassenbesitzer verwaltet werden, da dieser für die gemeldeten Daten verantwortlich ist.

Literatur

[1] Ministerium der Finanzen und für Wissenschaft Saarland. *Informationsblatt zu elektronischen Aufzeichnungssystemen*. Besucht am 19. Dezember 2024. 28.11.2019. URL: https://www.saarland.de/SharedDocs/Downloads/DE/mfe/Steuern_und_Finanz%C3%A4mter/TSE-Infoblatt.pdf.

[2] Bundesministerium der Finanzen. *Gesetz zum Schutz vor Manipulationen an digitalen Grundaufzeichnungen*. Besucht am 2. Januar 2025. 28.12.2016. URL: https://www.bundesfinanzministerium.de/Content/DE/Gesetzestexte/Gesetze_Gesetzesvorhaben/Abteilungen/Abteilung_IV/18_Legislaturperiode/Gesetze_Verordnungen/2016-12-28-Kassenmanipulationsschutzgestz/0-Gesetz.html.

[3] Bundesministerium der Justiz. *Abgabenordnung (AO) §146 Ordnungsvorschriften für die Buchführung und für Aufzeichnungen*. Besucht am 2. Januar 2025. 22.12.2016. URL: https://www.gesetze-im-internet.de/ao_1977/__146a.html.

[4] Bundesministerium der Justiz. *Kassensicherungsverordnung - KassenSichV*. Besucht am 4. November 2024. 22.12.2016. URL: https://www.gesetze-im-internet.de/kassensichv/.

[5] Bundesministerium der Finanzen. *Beginn der Mitteilungsverpflichtung nach § 146a Absatz 4 Abgabenordnung (AO)*. Besucht am 4. November 2024. 28.06.2024. URL: https://www.bundesfinanzministerium.de/Content/DE/Downloads/BMF_Schreiben/Weitere_Steuerthemen/Abgabenordnung/2024-06-28-mitteilungsverpflichtung-nach-AO.html.

[6] Bundesministerium der Justiz. *Kassensicherungsverordnung*. Besucht am 2. Januar 2025. 22.12.2016. URL: https://www.gesetze-im-internet.de/kassensichv/BJNR351500017.html.

[7] Besucht am 7. Januar 2025. URL: https://www.bsi.bund.de/DE/Home/home_node.html.

[8] Bundeszentralamt für Steuern. *Digitale Schnittstelle der Finanzverwaltung für Kassensysteme (DSFinV-K)*. Besucht am 4. November 2024. 1.12.2020. URL: https://www.bzst.de/DE/Unternehmen/Aussenpruefungen/DigitaleSchnittstelleFinV/digitaleschnittstellefinv_node.html.

[9] Bayerisches Landesamt für Steuern. *ERiC 40.3.4.0 Entwicklerhandbuch*. Besucht am 4. November 2024. 2024. URL: https://www.elster.de/elsterweb/entwickler/infoseite/download_eric_40.

[10] Timothy Grance Peter Mell. *The NIST Definition of Cloud Computing*. Besucht am 7. November 2024. September 2011. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

[11] Hirsch Uray Katzinger. *Beyond Desktop Computation: Challenges in Scaling a GPU Infrastructure*. Besucht am 8. November 2024. 11.10.2021. URL: https://arxiv.org/pdf/2110.05156.

[12] Ian Smalley Stephanie Susnjara. *Was ist Hybrid Cloud-Architektur?* Besucht am 8. November 2024. 12.01.2024. URL: https://www.ibm.com/de-de/topics/hybrid-cloud-architecture.

[13] P.Srinivasan M.P.Vaishnnave K.Suganya Devi. *A Survey on Cloud Computing and Hybrid Cloud*. Besucht am 3. Januar 2025. 2019. URL: https://www.ripublication.com/ijaer19/ijaerv14n2_13.pdf.

[14] Santonu Sarkar Sreekrishnan Venkateswaran. *Architectural Partitioning and Deployment Modeling on Hybrid Clouds*. Besucht am 7. Januar 2025. 2016. URL: https://arxiv.org/pdf/2205.04467.

[15] Ruben Wolf Michael Herfert Thomas Kunz. *Ratgeber für eine sichere zentrale Softwareverteilung*. Besucht am 26. November 2024. 31. August 2017. URL: https://www.sit.fraunhofer.de/fileadmin/dokumente/studien_und_technical_reports/ratgeber-fuer-eine-sichere-zentrale-softwareverteilung.pdf.

[16] *Java EE at a Glance*. Besucht am 27. November 2024. URL: https://www.oracle.com/java/technologies/java-ee-glance.html.

[17] *Java Native Access (JNA)*. Besucht am 28. November 2024. URL: https://github.com/java-native-access.

[18] *Guide to JAXB*. Besucht am 26. November 2024. URL: https://www.baeldung.com/jaxb.

[19] *Jakarta Persistence API*. Besucht am 9. Dezember 2024. URL: https://jakarta.ee/specifications/persistence/.

[20] *MariaDB*. Besucht am 9. Dezember 2024. URL: https://mariadb.org/.

[21] *https://www.wildfly.org/*. Besucht am 27. November 2024. URL: https://www.wildfly.org/.

[22] Barbara Röhl Demary Vera Engels. *Digitalisierung und Mittelstand*. Besucht am 22. Januar 2025. 2016. URL: https://www.econstor.eu/bitstream/10419/157156/1/IW-Analyse-109.pdf.