OPEN ACCESS

Originalarbeit

Entwicklung eines Editors zur Erstellung und Bearbeitung Pflegerischer Informationsobjekte (PIOs) zur Pflegeüberleitung

Development of an editor for creating and editing PIOs (care information objects) for care transition

Matthias Regner[1]
Viktor Werlitz[1]
Lukas Kleybolte[1]
Elisabeth Veronika Mess[1]
Sabahudin Balic[1]
Lisa Daufratshofer[2]
Sabrina Tilmes[2]
Andreas Mahler[2]
Phillip Heidegger[1]
Claudia Reuter[1]
Alexandra Teynor[1]

1 Technische Hochschule Augsburg, Deutschland

2 Universitätsklinikum Augsburg, Deutschland

Abstract

Standardization and digitalization of the care transmission process in Germany can help to relieve the burden on nursing staff because manual work steps can be automated. The implementation and testing of the new PIO-ULB (care information object – nursing summary, German: **P**flegerisches **I**nformations**o**bjekt – **Ü**ber**l**eitungs**b**ogen, XML based FHIR bundle) in form of a PIO-ULB Editor should make the suitability of the format for a digital transition process verifiable at an early stage and should visualize its contents. The PIO-ULB Editor is a program that can be installed locally or tried out online. A clear user interface allows the users to enter care transmission data. A standardized PIO-XML file can be generated automatically and sent to receiving facilities. The PIO-ULB Editor also enables the import, display and editing of PIO-ULB files. The PIO-ULB Editor was designed and implemented as prototype software as part of the CARE REGIO research project. This article describes the development process as well as the software architecture of the editor and thus provides a basis for subsequent implementations. A lessons-learned chapter explicitly addresses potential improvements for subsequent PIO software.

Zusammenfassung

Eine Standardisierung und Digitalisierung des Pflegeüberleitungsprozesses in Deutschland kann zu einer Entlastung des Pflegefachpersonals beitragen, weil manuelle Arbeitsschritte automatisiert werden können. Die Umsetzung und Erprobung des neuen PIO-ULB (Pflegerisches Informationsobjekt – Überleitungsbogen, XML-basiertes FHIR-Bundle) in Form eines PIO-ULB Editors soll die Eignung des Formats für einen digitalen Überleitungsprozess frühzeitig überprüfbar machen und dessen Inhalte visualisieren. Der PIO-ULB Editor ist ein Programm, das sowohl lokal installiert als auch online ausprobiert werden kann. Eine übersichtliche Nutzeroberfläche erlaubt das Eingeben von Pflegeüberleitungsdaten. Per Knopfdruck kann eine standardisierte PIO-XML-Datei generiert werden, die an aufnehmende Einrichtungen versendet werden kann. Der PIO-ULB Editor ermöglicht ebenfalls das Importieren, Anzeigen und Editieren von PIO-ULB-Dateien. Der PIO-ULB Editor wurde im Rahmen des Forschungsprojektes CARE REGIO als prototypische Software konzipiert und implementiert. Dieser Artikel beschreibt den Entwicklungsprozess sowie die Softwarearchitektur des Editors und bietet somit eine Grundlage für Folgeimplementierungen. Ein Lessons-Learned-Kapitel geht explizit auf Verbesserungspotentiale für nachfolgende PIO-Software ein.

Schlüsselwörter: PIO-ULB Editor, medizinisches Informationsobjekt (MIO), pflegerisches Informationsobjekt (PIO), digitale Pflegeüberleitung, FHIR, HL7, Softwareentwicklung

1 Einleitung

Der Mangel an Pflegefachpersonal und die Zunahme pflegebedürftiger Personen in Deutschland [1] erfordern die effiziente Gestaltung von Pflegeprozessen. Eine digitale Prozessunterstützung scheint ein vielversprechender Ansatz zu sein, da aktuell viele Abläufe papierbasiert erfolgen. Ein Beispiel hierfür ist das Pflegeüberleitungsmanagement. Hier werden Pflegeüberleitungsberichte erstellt, um versorgungsbezogene Patientendaten zwischen relevanten Gesundheitseinrichtungen auszutauschen. Die Erstellung ist jedoch sehr zeitintensiv, da die Patientendaten händisch in das verwendete Pflegedokumentationssystem übertragen werden müssen, oftmals aus Papierausdrucken oder handschriftlichen Notizen. Weiterhin wird die Datenübermittlung unnötig verzögert, da die Pflegeüberleitungsberichte oft erst am Tag der Überleitung als Ausdruck mitgegeben werden [2]. Die aufnehmende Einrichtung kann sich daher nicht ausreichend auf die Patient:innen und Bewohner:innen vorbereiten. Gleichermaßen fehlt ein einheitlicher Formatstandard, welcher den Umfang und die Strukturierung der Informationen im Pflegeüberleitungsbericht vorgibt ([3], S. 117). Diese Problemstellungen belasten die Pflegefachpersonen administrativ. Erhebungen in Form von Beobachtungen in Bayern und einer deutschlandweiten Onlineumfrage [4] bestätigten diese Erkenntnisse und zeigen auf, dass in Deutschland eine Vielzahl an unterschiedlichen Pflegedokumentationssystemen verwendet wird. Dies erschwert eine flächendeckende Standardisierung des Pflegeüberleitungsprozesses zusätzlich.

Um das Kompatibilitätsproblem zu adressieren, hat die deutsche Bundesregierung 2021 das „Digitale-Versorgung-und-Pflege-Modernisierungs-Gesetz“ [5] erlassen und Aufträge zur Standardisierung papierbasierter Gesundheitsdokumente vergeben. Seit Januar 2023 sind Pflegeüberleitungsberichte als sogenanntes pflegerisches Informationsobjekt (PIO) sowohl standardisiert als auch digitalisiert. Ein PIO ist eine maschinenlesbare XML-Datei, welche von der mio42 GmbH in Berlin unter Nutzung des „HL7 – Fast Healthcare Interoperability Resources“-Standards (FHIR) spezifiziert wird.

Vor der Entwicklung von MIOs und PIOs gab es bereits Bemühungen der Hochschule Osnabrück und der Universitätsmedizin in Göttingen, den Pflegeüberleitungsprozess mit Hilfe eines Test-Netzwerkes und einem HL7-basierten ePflegebericht zu digitalisieren [6]. Erkenntnisse aus diesem Projekt flossen in die Neuentwicklung des PIO-Überleitungsbogens (PIO-ULB) ein.

Um den neuen PIO-ULB für Pflegeeinrichtungen nutzbar zu machen, hat die Technische Hochschule Augsburg im Rahmen des Pilotprojektes DigiPÜB (Teilprojekt des Verbundforschungsprojekts CARE REGIO) eine prototypische Implementierung eines PIO-ULB Editors vorangetrieben und im Dezember 2023 unter der Apache 2.0-Lizenz als Open-Source-Projekt veröffentlicht [7]. Weiterhin können alle Interessierten den PIO-ULB Editor auf unserer Website ausprobieren sowie selbst PIO-ULBs erstellen und exportieren [8].

Der Prototyp ermöglicht das Erstellen, Importieren, Editieren und Exportieren von PIO-ULB-Dateien und kann als visuelle Diskussionsgrundlage verwendet werden, sodass Nutzer:innen konstruktives Feedback bezüglich des PIO-ULB-Standards geben können. Die Entwicklung des zugehörigen User Interfaces wurde bereits in einem separaten Artikel veröffentlicht [9]. Dieser Artikel legt den Fokus auf die technische Entwicklung des Editors. In diesem Rahmen werden Grundlagen zum PIO-ULB erläutert, die Softwarearchitektur des Editors vorgestellt und komponentenweise im Detail beschrieben. Ein Lessons-Learned-Kapitel rundet den Artikel ab.

2 Grundlagen

2.1 Health Level 7 – Fast Healthcare Interoperability Resources (HL7 – FHIR)

Das HL7 FHIR-Datenformat ist ein international anerkannter und weit verbreiteter Standard zur maschinenlesbaren Abbildung von medizinischen Daten [10]. FHIR ist ein ressourcenbasiertes Konzept, wobei jede Ressource einen Informationsbaustein darstellt (z.B. Patient, Organisation, Arzt, Medikament, Pflegeplan, Allergie, usw.). Die Ressourcen können zu einem Paket (Bundle) inklusive Inhaltsverzeichnis (*Composition*) zusammengefasst werden, um Patienteninformationen abzubilden (siehe Abbildung 1). Dabei bleibt FHIR so flexibel, dass der internationale Standard leicht an spezifische Anwendungskontexte angepasst werden kann. Jeder Ressource ist eine statische ID zugeordnet. So haben zum Beispiel zwei Hausarzt-Ressourcen jeweils eine eigene ID, damit jede Ressource eindeutig identifizierbar ist. Innerhalb einer Ressource sind die Daten hierarchisch gegliedert und sehr feingranular aufgeteilt. Wie eine spezielle Ressource strukturiert ist, wird von dessen FHIR-Strukturdefinition eindeutig vorgeschrieben. FHIR-Bundles können als JSON- oder XML-Datei abgespeichert werden und sind daher maschinenlesbar.

2.2 PIO-Überleitungsbogen (PIO-ULB)

Im Rahmen der Bemühungen, papierbasierte Gesundheitsdokumente zu standardisieren, wurde die Kassenärztliche Bundesvereinigung (KBV) gesetzlich beauftragt, sogenannte medizinische und pflegerische Informationsobjekte (MIOs & PIOs) zu definieren. Der gesetzliche Auftrag wurde an die mio42 GmbH weitergegeben, welche eine vollständige Tochter der KBV ist. Diese Informationsobjekte sind FHIR-Bundles, die als XML-Datei abgespeichert werden und auf einen speziellen Anwendungskontext – also auf ein spezielles Gesundheitsdokument – zugeschnitten sind. MIOs und PIOs sollen zukünftig in der elektronischen Patientenakte abgelegt oder mithilfe des Dienstes „Kommunikation im Medizinwesen“ (KIM)

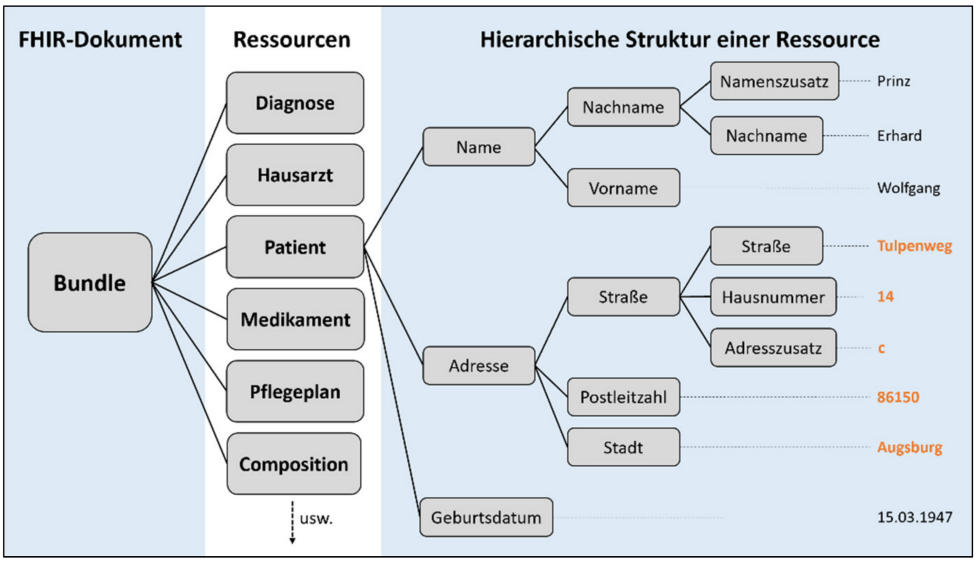

Abbildung 1: Schematische Darstellung eines FHIR-Bundles (Daten aus Abb. 2 orange markiert)

```
<address>
	<type value="postal"/>
	<use value="home"/>
	<text value="Tulpenweg 14 c, 86150 Augsburg, D"/>
	<line value="Tulpenweg 14 c">
		<extension url="http://hl7.org/fhir/StructureDefinition/iso21090-ADXP-streetName">
			<valueString value="Tulpenweg"/>
		</extension>
		<extension url="http://hl7.org/fhir/StructureDefinition/iso21090-ADXP-houseNumber">
			<valueString value="14"/>
		</extension>
		<extension url="http://hl7.org/fhir/StructureDefinition/iso21090-ADXP-additionalLocator">
			<valueString value="c"/>
		</extension>
	</line>
	<postalCode value="86150"/>
	<city value="Augsburg"/>
	<country value="D"/>
</address>
```

Abbildung 2: Darstellung der Patientenadresse im PIO-XML-Format (Daten aus Abb. 1 orange markiert)

versendet werden. Beides sind Dienste des deutschen Gesundheitsnetzwerks, der Telematikinfrastruktur (TI).

Im Januar 2023 wurde der PIO-Überleitungsbogen (PIO-ULB) veröffentlicht, welcher einen Pflegeüberleitungsbericht mit Hilfe von 90 FHIR-Ressourcen abbildet [11]. Dabei stellt jede Ressource einen Baustein dar (z.B. Angehörige, Pflegefachpersonen, Ärzte, Patient, Medikamente, Pflegeplan, Vitalparameter, Allergien, Pflegemaßnahmen, Diagnosen, usw.), welche je nach Überleitungsfall modular zusammengesetzt werden. Der PIO-ULB ist das erste und wichtigste Informationsobjekt für die Pflege, mit dem die Anbindung von Pflegeheimen an die TI vorangetrieben werden soll. Abbildung 2 zeigt einen Ausschnitt aus einem PIO-ULB im XML-Format, welcher eine Adresse darstellt. Eine Adresse ist keine eigene Ressource, sondern nur ein Teil einer Ressource. Die *„extensions“* sind ein von FHIR bereitgestelltes Werkzeug, um den internationalen Standard an nationale Gegebenheiten anzupassen.

3 Konzeption des PIO-ULB Editors

Folgendes Kapitel befasst sich mit der Softwarearchitektur des PIO-ULB Editors. Aufgrund des großen Umfangs der PIO-ULB Spezifikation wurde ein Sub-Set mit den wichtigsten Elementen ermittelt, welches als Teilmenge des PIO-ULB-Standards zu verstehen ist und im Folgenden als *PIO-Small* bezeichnet wird. Das Sub-Set beinhaltet nur Elemente, die von unseren Praxispartnern als relevant für den Überleitungsprozess eingestuft wurden. Als relevant gelten jene Elemente, welche in einer sechs Mal durchgeführten Befragung wiederholt von Pflegefachpersonen als relevant eingestuft wurden. So konnte der Umfang der Spezifikation auf zwei Drittel reduziert werden. Eine exakte Beschreibung des *PIO-Smalls* wird im Rahmen des PIO-ULB Editors bereitgestellt [12]. Der prototypische PIO-ULB Editor kann ausschließlich das *PIO-Small* vollumfänglich verarbeiten. Daten, die nicht im *PIO-Small* enthalten sind, können importiert und verein-

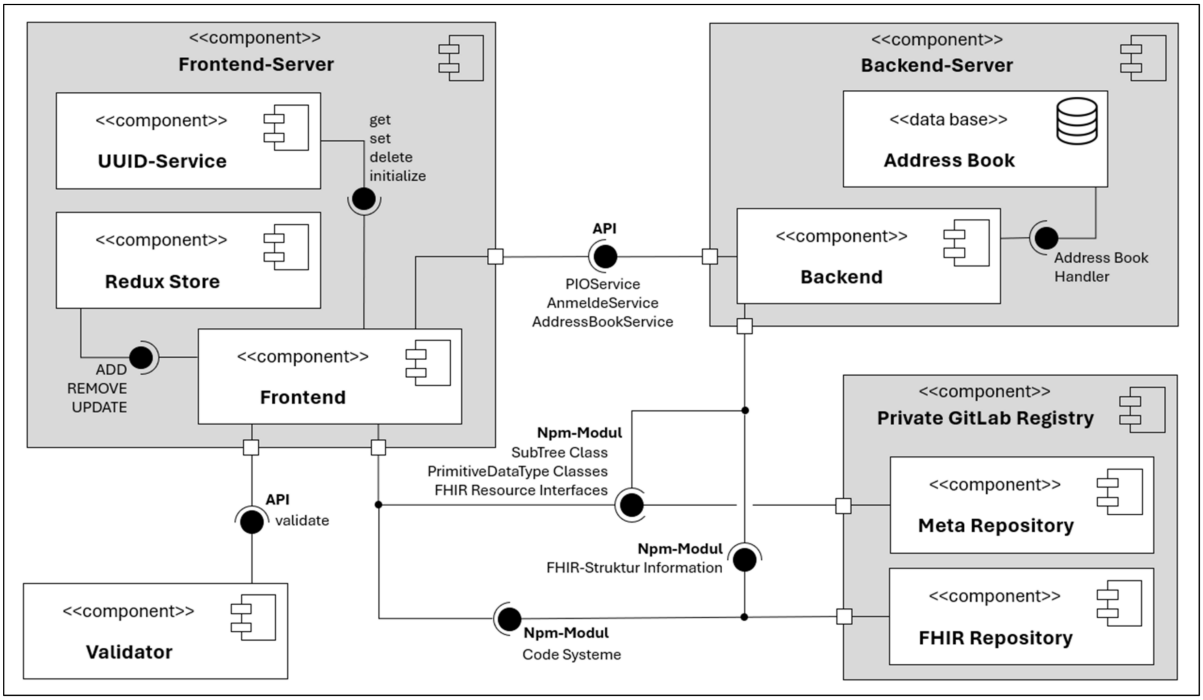

Abbildung 3: UML-Komponentendiagramm des PIO-ULB Editors

facht dargestellt werden, sind jedoch nicht editierbar oder exportierbar.

3.1 Softwarearchitektur des PIO-ULB Editors

Die einzelnen Bestandteile des PIO-ULB Editors und deren Schnittstellen sind im Komponentendiagramm (siehe Abbildung 3) dargestellt. Der Frontend-Server sowie der Backend-Server sind als Hauptkomponenten zu verstehen, die jeweils Dienste des Meta- und FHIR-Repositories beanspruchen. Während das FHIR-Repository Strukturinformationen über den PIO-ULB sowie verwendete Codesysteme bereitstellt, bündelt das Meta-Repository Klassen, Typen und Interfaces, die in beiden Hauptkomponenten benötigt werden. Hierfür wurden eigene Node-Package-Manager-Module implementiert, die in einer privaten GitLab Registry gehostet und dann eingebunden werden. Die Validator-Komponente kann ein XML-Dokument gegenüber FHIR-spezifischen Strukturdefinitionen offline validieren und wird nur vom Frontend angesprochen. Im Folgenden werden die Hauptkomponenten näher erläutert:

Der *Backend-Server* ermöglicht es mehreren Nutzer:innen gleichzeitig, PIOs zu bearbeiten, insofern jeder Nutzer und jede Nutzerin über ein eigenes Frontend angemeldet ist. Sobald die Nutzer:innen ihren Vor- und Nachnamen eingegeben haben, erstellt der „AnmeldeService“ eine neue Session. Die Eingabe eines Passwortes ist nicht nötig, da der Editor keine interne Nutzerverwaltung anbietet. Der Nutzername wird im PIO automatisch als Autor hinterlegt. Für jede Session werden alle weiteren Nutzereingaben zeitbegrenzt serverseitig gespeichert, sodass durch einen Absturz des Frontends keine Daten verloren gehen. Weiterhin stellt das Backend Logik zum Importieren und Exportieren von XML-Dateien bereit. Eine globale serverseitige Datenbank – das Adressbuch – kann FHIR-Organisationsressourcen permanent speichern, sodass Nutzer:innen nicht jedes Mal erneut Daten eines bekannten Pflegeheimes oder Krankenhauses eingeben müssen. Der *Frontend-Server* ist als React Webanwendung [13] implementiert und kann über den Browser aufgerufen werden. Die Benutzeroberfläche verwendet vorgefertigte Interface-Komponenten aus der „Ant Design“ Bibliothek [14] für die Dateneingabe. Thematisch zusammengehörende Eingabefelder sind zu einem *Ant Design Formular* zusammengefasst. Dies ermöglicht eine selbstständige Datenverarbeitung durch jedes *Formular* selbst (siehe Kapitel 3.2). Um jede FHIR-Ressource eindeutig zu identifizieren, wird ein „Universally Unique Identifier“ (UUID) verwendet. Die UUIDs von jeder zur Laufzeit erstellten FHIR-Ressource werden im Hintergrund vom UUID-Service gespeichert, damit eingegebene Daten stets der richtigen Ressource zugeordnet werden können. Der Redux Store wird als Zustandsspeicher verwendet. Hier sind Nutzer- und Navigationsinformationen sowie Daten, die über mehrere React-Komponenten synchron gehalten werden müssen, temporär abgelegt.

3.2 Funktionales Zusammenspiel der Hauptkomponenten (Frontend & Backend)

Im Frontend werden HTTP-Requests erzeugt, die das Backend entgegennimmt. Wenn sich ein Nutzer im Editor anmeldet, wird ein JSON Web Token (JWT) erstellt, welcher der eindeutigen Identifizierung der zugehörigen Session im Backend dient und bei jedem folgenden HTTP-

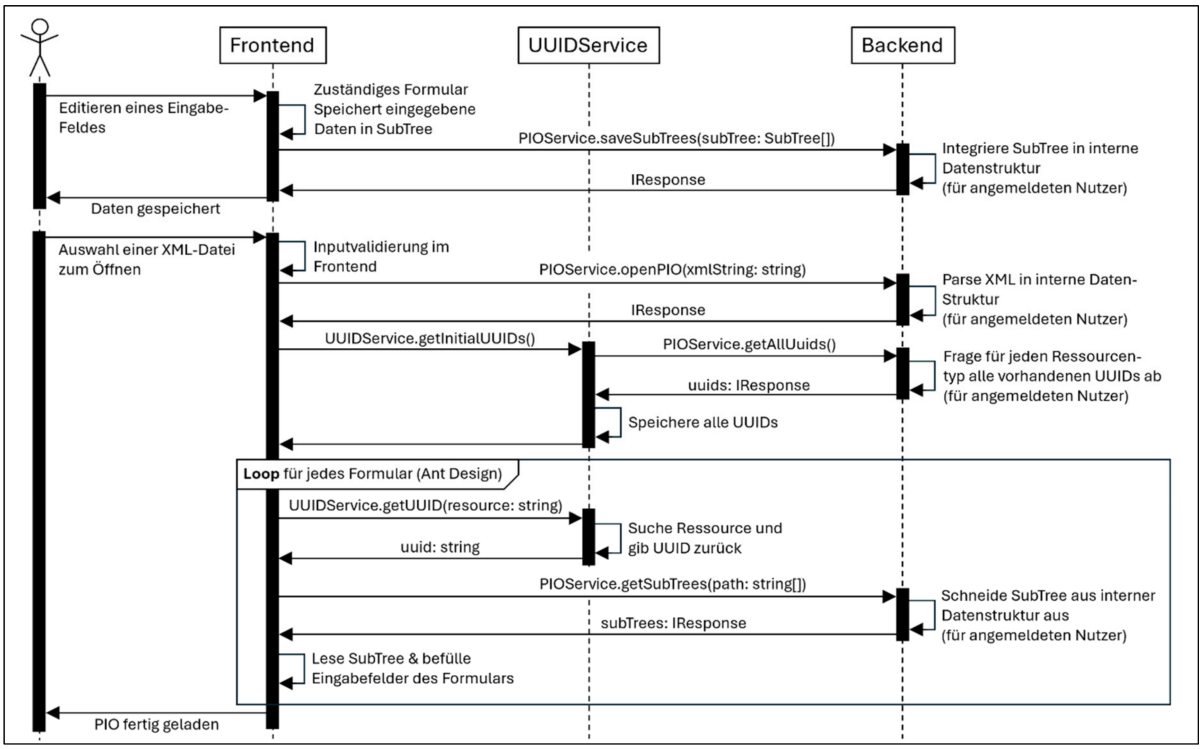

Abbildung 4: UML-Sequenzdiagramm für zwei ausgewählte Prozesse (1: Eingabefeld editieren, 2: XML-Datei öffnen)

Request dem Header angehängt wird. Der Austausch von PIO-Daten erfolgt über sogenannte SubTrees (siehe Kapitel 4.2). Die eigens implementierte SubTree-Klasse wird vom Meta-Repository bereitgestellt und repräsentiert einen beliebigen Ausschnitt aus der XML-Baumstruktur des PIO-ULBs. Jedes *Ant Design Formular* hält einen oder mehrere SubTrees bereit, um Werte aus den Eingabefeldern als SubTree an das Backend zu schicken.

Neben der Neuerstellung eines PIOs, dem XML-Export und dem Erzeugen von neuen FHIR-Ressourcen wurde sowohl die Nutzeraktion „Eingabefeld editieren“ als auch „Öffnen einer XML-Datei“ als besonders aussagekräftig erachtet und deshalb im Sequenzdiagramm abgebildet (siehe Abbildung 4). Beide Prozesse werden im Folgenden erklärt:

Nach dem *Editieren eines Eingabefeldes* mit anschließendem „Herausklicken“ (Fokuswechsel) wird der neue Wert automatisch in den hinterlegten SubTree (siehe Kapitel 4.2) des *Formulars* geschrieben und zum Abspeichern an das Backend geschickt. Dieses Vorgehen ermöglicht eine automatisierte Speicherung der eingegebenen Daten, ohne das eine Nutzeraktion nötig ist.

Öffnen Nutzer:innen eine PIO-XML-Datei, wird nach erfolgreicher Frontend-Inputvalidierung der zweite Prozess im Sequenzdiagramm angestoßen. Der XML-String wird im Backend von einer eigenen Deserialisierungslogik geparst. Daraufhin frägt der UUID-Service die UUIDs aller eingelesenen FHIR-Ressourcen ab und speichert sie zur weiteren Verwendung im Frontend. Über eine Initialisierungslogik holen sich alle *Formulare* ihre benötigten SubTrees selbstständig vom Backend und verwenden dabei die Ressourcen-UUIDs. Diese SubTrees enthalten die geparsten Patientendaten, welche von jedem *Formular* selbstständig in die Eingabefelder übertragen werden. Die Nutzer:innen können nun die Patientendaten einsehen.

4 Entwicklung des PIO-ULB Editors

Bei der Entwicklung des PIO-ULB Editors wurde das Projekt in mehrere Teilprojekte (Repositories) zerlegt, die miteinander interagieren und hauptsächlich in TypeScript geschrieben sind. Dies hilft bei der konzeptionellen Trennung von Komponenten und vermeidet Redundanzen im Code. Zu Beginn werden zwei Repositories (FHIR & Meta) beschrieben, welche Code enthalten, der in mehreren anderen Teilprojekten benötigt und somit ausgelagert wurde. Danach wird auf die beiden Hauptkomponenten (Frontend & Backend) sowie den Validator genauer eingegangen.

4.1 FHIR-Repository

In dieser Komponente werden JSON-Dateien erzeugt, welche strukturelle Informationen des PIO-ULBs, gemäß der FHIR-Spezifikation, in Form von LookUpTables enthalten. Die „ResourceLookUpTable“ enthält alle validen PIO-XML-Pfade mit zugehörigen Datentypen, während die „ResourceLookUpTableSmall“ nur das in Kapitel 3 beschriebene *PIO-Small* darstellt. Kapitel 4.3 geht genauer auf die Verwendung der LookUpTables ein.

Des Weiteren wird eine JSON-Datei generiert, welche alle für den PIO-ULB relevanten Codesysteme enthält. In der PIO-ULB-Spezifikation werden neben Terminologien wie LOINC, ICD und Alpha-ID meist SNOMED CT-Codes ver-

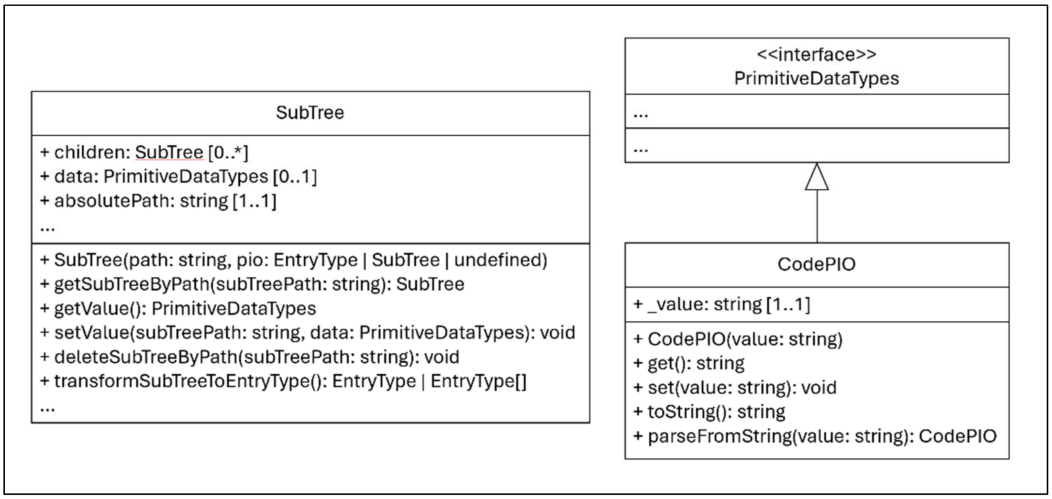

Abbildung 5: UML-Klassendiagramm für SubTrees sowie einen beispielhaften primitiven Datentyp (CodePIO)

wendet, um Interoperabilität zu gewährleisten. Der PIO-ULB Editor unterstützt nur SNOMED CT-Codes in vollem Umfang, alle anderen Codesysteme können zwar eingelesen und dargestellt, aber nicht aktiv über Drop-Down-Menüs ausgewählt werden. Relevante Codesysteme wurden in englischer Sprache vom „International SNOMED CT Browser“ heruntergeladen und mit deutschen Formulierungen angereichert, insofern Mappings auf deutschsprachige Konzepte von der KBV oder der mio42 GmbH in Form von ConceptMaps zur Verfügung gestellt wurden.

4.2 Meta-Repository

Das Meta-Repository hält unter anderem die Klassendefinition eines SubTrees (siehe Abbildung 5).
Ein SubTree ist rekursiv aufgebaut (*property: children*) und stellt einen Teilbereich der XML-Baumstruktur eines PIOs dar. Dabei kann jedes Element einen Wert enthalten (*property: data*). Diese Werte sind analog zu FHIR als „primitive Datentypen“ (z.B. String, Code, Integer, UUID) implementiert. Für jeden dieser Datentypen stellt das Meta-Repository eine eigene Klasse bereit (siehe Abbildung 5). Außerdem wird in jedem SubTree hinterlegt, an welche Stelle im XML-Dokument dessen Daten platziert werden sollen (siehe Abbildung 5, *property: absolutePath*). SubTrees verfügen weiterhin über Methoden zur Datenmanipulation, damit die *Formulare* im Frontend Nutzereingaben im hinterlegten SubTree speichern können.
Weiterhin stellt das Meta-Repository Interfaces bereit, die eine komplette FHIR-Ressource repräsentieren (z.B. repräsentiert IAllergyObject eine Allergieressource). Sie werden für Akkordeon-Menüs und den Redux Store im Frontend sowie das Adressbuch im Backend benötigt.

4.3 Backend-Server

Das Backend ist eine der beiden Hauptkomponenten und kann PIO-ULBs exportieren bzw. importieren. Dies wird durch eine Serialisierungs- bzw. Deserialisierungslogik ermöglicht, welche die interne Datenstruktur (siehe Abbildung 6) in ein PIO-konformes XML-Dokument konvertiert und umgekehrt.
Das gesamte FHIR-Bundle wird ressourcenweise als „EntryType“ bzw. als JSON-Objekt (siehe Abbildung 6) im Backend gehalten, wobei auf Kompatibilität zum verwendeten XML-Parser („fast-xml-parser“) geachtet wurde.
Für alle Operationen stellt das Backend API-Routen (Application Programming Interface) für das Frontend bereit, um z.B. Daten in die Datenbank zu schreiben, bestimmte Operationen auf ein aktuell geöffnetes PIO auszuführen oder Nutzer:innen an- bzw. abzumelden.
Weiterhin ist das Backend möglichst generisch implementiert. Dies bedeutet, dass das Backend lediglich Daten in eine Baumstruktur schreibt, ohne die Struktur zu validieren. Das Frontend muss also den Pfad kennen, unter dem die PIO-Daten laut Spezifikation abgespeichert werden sollen. Um z.B. auf die Postleitzahl der hinterlegten Patientenadresse zuzugreifen, muss folgender absoluter Pfad verwendet werden (vgl. Abbildung 6):

```
24ed8a3b-59f7-47fd-8699-bb908de982c0.KBV_
PR_MIO_ULB_Patient.address[0].postalCode
```

Des Weiteren ist zu erwähnen, dass bereits Implementierungen von FHIR-Ressourcen existieren, die deren Struktur als Objekte abbilden und einen XML-Export ermöglichen (z.B. HAPI FHIR [15]). Die vorgestellte Architektur verzichtet jedoch bewusst auf derartige Implementierungen, um das Backend generisch und somit wartungsärmer bzw. flexibler zu halten. Außerdem lässt sich die spezielle PIO-ULB Struktur nicht vollständig durch derartige allgemeingültige Implementierungen abbilden.
Die in Kapitel 4.1 beschriebenen LookUpTables, stellen dem Backend zwar strukturbezogene Datentypinformationen zur Verfügung, diese werden aber lediglich für die Deserialisierungslogik – also das Importieren – benötigt, um die als String gespeicherten Daten des XML-Dokuments in die richtigen primitiven Datentypen zu parsen. Außerdem kann anhand der LookUpTables identifiziert werden, ob ein XML-Pfad zur PIO-ULB Spezifikation, zum

```
{ "24ed8a3b-59f7-47fd-8699-bb908de982c0": {
      "KBV_PR_MIO_ULB_Patient": {
            "…": { … },
            "address": [ {
                  "type": {
                        "_ _value": CodePIO(„postal") },
                  "use": {
                        "_ _value": CodePIO("home") },
                  "line": [ {
                        "_ _value": StringPIO("Tulpenweg 14, c"),
                        "extension": [ {
                              "_ _url": UriPIO("http://hl7.org/fhir/StructureDefinition/iso21090-ADXP-streetName"),
                              "valueString": {
                                    "_ _value": StringPIO("Tulpenweg") } },
                        { … }, { … } ] } ],
                  "postalCode": {
                        "_ _value": StringPIO("86150") },
                  "city": {
                        "_ _value": StringPIO("Augsburg") },
                  "text": {
                        "_ _value": StringPIO("Tulpenweg 14, c, 86150 Augsburg") }
            } ],
            "…": { … },
} } }  as  EntryType
```

Abbildung 6: Beispielhafte Darstellung der Patientenadresse als „EntryType“ (grün: primitive Daten, blau: Pfad bis zur Postleitzahl)

PIO-Small oder zu keinem der beiden zugeordnet werden kann.

4.4 Frontend-Server

Das Frontend ist die zweite Hauptkomponente, welche die Benutzeroberfläche bereitstellt. Das Layout des PIO-ULB Editors wurde in einem iterativen Prozess mittels User-Tests entwickelt und stetig verbessert, um die Gebrauchstauglichkeit zu optimieren [9]. Da es primär um die Erfassung relevanter Pflegedaten geht, ist das Bereitstellen von verständlichen Eingabefeldern sowie die formularbasierte Verarbeitung der Daten die Kernaufgabe des Frontends. Wo benötigt, verfügen Eingabefelder über eine Frontendvalidierung und überprüfen zum Beispiel das Vorhandensein von Pflichtangaben. Wie die *Formulare* Daten mit dem Backend austauschen und dabei auf SubTrees zurückgreifen, ist in Kapitel 3.2 beschrieben.
Ein weiterer Aspekt während der Entwicklung war die Handhabung von Referenzen. FHIR ermöglicht die Erstellung eines Informationsnetzwerks durch das Referenzieren von Ressourcen mittels UUIDs. So kann es z.B. passieren, dass Nutzer:innen die verantwortliche Person im *Diagnose Formular* auswählen müssen, wobei behandelnde Personen in einem anderen *Formular* erstellt werden. Damit die Nutzer:innen nicht hin und her springen müssen, falls die diagnosestellende Person noch nicht erstellt wurde, ermöglicht das Drop-Down-Menü im *Diagnose Formular* das Hinzufügen einer neuen behandelnden Person an Ort und Stelle. Solche Informationen – wie z.B. die der behandelnden Personen – müssen in beiden *Formularen* synchron gehalten werden, was mit Hilfe des zentralen Redux Stores realisiert wurde.

4.5 Validator

Da der PIO-ULB Editor nicht nur für das Erstellen von PIO-ULBs gedacht ist, sondern diese auch einlesen kann, ist die Validierung einer PIO-XML-Datei ein wichtiger Prozessschritt. Hierfür bietet Firely eine API an [16], welche FHIR-Strukturdefinitionen einbinden und anschließend gesamte FHIR-Bundles offline validieren kann. Nach einem initialen Setup sowie gewissen Anpassungen entsprechend der *PIO-Small*-Spezifikation (Herausfiltern von Fehlermeldungen, die aufgrund der Umfangsreduzierung auftreten), kann der Validator nun vom Frontend über eine eigene API angesprochen und genutzt werden. Der Service ist in *C-Sharp (C#)* geschrieben.

5 Lessons Learned

In diesem Kapitel sollen Vor- und Nachteile der gewählten Softwarearchitektur diskutiert werden. Zuerst wird auf Punkte eingegangen, die rückblickend **Verbesserungspotential** aufweisen und in weiterführenden Implementierungen des PIO-ULB Editors berücksichtigt werden sollten. Der PIO-ULB Editor wurde als Prototyp konzipiert.
Weil das Backend generisch implementiert wurde (siehe Kapitel 4.3), wandert viel strukturbezogene PIO-Logik ins Frontend, obwohl Datenverarbeitung nicht die Kernaufgabe eines Frontends ist. So müssen die einzelnen *Formulare* im Frontend genau wissen, an welcher Stelle im FHIR-Bundle für sie relevante Daten zu finden sind. Im Quellcode des Frontends muss also mit String-basierten XML-Pfaden gearbeitet werden (siehe Kapitel 4.3), bei denen sich schnell schwer identifizierbare Tippfehler einschleichen. In diesem Fehlerfall würde das Backend einfach einen falschen XML-Pfad erzeugen, obwohl dieser laut PIO-ULB-Spezifikation nicht valide ist. Rückblickend würden wir von einem generischen Backend absehen und die FHIR-Logik in einem statischen Backend implementieren, sodass das Frontend einfach Operationen wie „createContactPerson(...)“ aufrufen kann und nicht alle Daten einer Kontaktperson einzeln mit Hilfe der „setValue(...)“ Methode der SubTree Klasse (siehe Abbildung 5) schreiben muss. Ein statisches Backend hätte die Ver-

wendung von bereits existierenden Implementierungen von FHIR-Ressourcen (z.B. HAPI-FHIR [15]) ermöglicht.
Die in Kapitel 3.2 beschriebene SubTree-Kommunikation zwischen Front- und Backend führt in unserem Prototyp zu vielen Konvertierungsschritten. Wird beispielsweise eine Allergie-Ressource als SubTree vom Backend an das Frontend geschickt, muss zuerst die Backend-Datenstruktur (siehe Abbildung 6) in einen SubTree konvertiert werden. Weil Daten, die über HTTP-Requests verschickt werden, als JSON übermittelt werden und somit Datentypen verloren gehen, müssen vor dem Versenden Informationen über die verwendeten primitiven Datentypen an jeden einzelnen SubTree angehängt werden. Das Frontend verwendet diese Informationen, um wieder Objekte des richtigen primitiven Datentyps zu erzeugen. Da Allergien mehrfach vorkommen können und in einem *Ant Design Akkordeon-Menü* gerendert werden, wird der rekursive SubTree im letzten Schritt in ein Interface mit flacher Hierarchie konvertiert (IAllergyObject), welches die Verarbeitung innerhalb des Akkordeon-Menüs vereinfacht und vom Meta-Repository bereitgestellt wird. Rückblickend hätte man genau solche Interfaces – also JSON-Objekte – anstelle von SubTrees zwischen Front- und Backend austauschen können. Dies impliziert jedoch ein statisches Backend.
Wie im Abschnitt oben bereits erwähnt,haben die primitiven Datentypen für Mehraufwand bei der SubTree-Kommunikation gesorgt. In der Ausbaustufe, auf der sich unser PIO-ULB Editor befindet, bringen sie aber wenig Vorteile. Rückblickend hätten alle primitiven Daten (egal ob String, Code oder Integer) als String repräsentiert werden können, da die Daten im XML-Dokument sowieso als String hinterlegt sind. Dadurch wäre auch die aufwendige LookUpTable überflüssig geworden, die der Deserialisierungslogik Datentypinformationen für jeden PIO-XML-Pfad bereitstellt (siehe Kapitel 4.1).
Aus dem generischen Backend ergeben sich auch **Vorteile**. Da das Backend losgelöst von der PIO-Struktur entwickelt wurde, ist es eine sehr flexible und wartungsarme Komponente. Bei strukturellen Änderungen in der PIO-ULB-Spezifikation muss demnach nur die Frontend-Logik überarbeitet werden. Das generische Backend könnte außerdem nach geringfügigen Änderungen auf andere MIO/PIO-Spezifikationen angewandt werden.
Das Aufspalten des XML-Bundles in Teilbäume (SubTrees) ermöglicht es, dass jedes *Formular* nur den für sich relevanten Teil der Daten hält. Da beim Speichern von Daten nur dieser kleine Teil an das Backend geschickt wird, konnte die Netzwerkauslastung auf einem geringen Niveau gehalten werden.
Zu guter Letzt ist zu erwähnen, dass die PIO-ULB-Spezifikation den Umfang von herkömmlichen, papierbasierten Pflegeüberleitungsberichten weit übertrifft. Um den Nutzer durch eine hohe Anzahl an Eingabemöglichkeiten nicht zu überfordern, wurde im Sinne des Pflegefachpersonals das *PIO-Small* entwickelt. Uns ist bewusst, dass die Definition eines bundesweiten Standards eine herausfordernde Aufgabe ist. Es müssen Anforderungen von vielen Parteien berücksichtigt und Kompromisse gefunden werden. Dennoch wäre eine weniger komplexe FHIR-Struktur wünschenswert gewesen.

6 Ausblick

Die Pilotierung des PIO-ULB Editors im pflegerischen Alltag wurde im Sommer 2024 durchgeführt. In diesem Rahmen sind Patientenüberleitungen zwischen kooperierenden Pflegeeinrichtungen und dem Uniklinikum Augsburg durchgeführt worden. Die erstellte XML-Datei versendete das Pflegepersonal mit Hilfe des KIM-Dienstes der TI. Der PIO-ULB Editor dient in dem Prozess als temporäre Brückenlösung, welche langfristig gesehen durch einen automatischen PIO-ULB-Export aus dem Pflegedokumentationssystem ersetzt werden sollte. Diese Exportfunktion wird zukünftig einen durchgängig automatisierten Pflegedatenüberleitungsprozess ermöglichen und manuelle Schritte, wie das Eingeben von Daten in den PIO-ULB Editor, überflüssig machen. Die Ergebnisse dieser Pilotierungsphase werden in einem weiteren wissenschaftlichen Beitrag veröffentlicht.
Mit der Veröffentlichung des Editors hoffen wir, anderen Softwareherstellern eine Beispielimplementierung für die leichtere Entwicklung PIO-basierter Pflegesoftware zu bieten.

Anmerkungen

Danksagung

Wir bedanken uns herzlich bei unseren Kooperationspartnern für die Bereitschaft, ihre fachliche Expertise in die Entwicklung des PIO-ULB Editors einfließen zu lassen. Ohne ihr regelmäßiges Feedback wäre die nutzerzentrierte Konzeption und Entwicklung des PIO-ULB Editors nicht möglich gewesen.
Die Forschungsarbeit ist Teil des Verbundforschungsprojektes CARE REGIO, welches durch das Bayerische Staatsministerium für Gesundheit, Pflege und Prävention gefördert wird.

Beiträge der Autor:innen

MR: Substantielle Überarbeitung des Manuskripts; MR, VW, EM, SB: Schreiben des Manuskripts; MR, VW, LK: Entwicklung und Implementierung der Software; EM: Entwicklung der Benutzeroberfläche; PH: Expertenhilfe bei der Entwicklung der Softwarearchitektur; LD, ST: Pflegewissenschaftliche Unterstützung; AM, CR, AT: Projektleitung.

Interessenkonflikte

Die Autoren erklären, dass sie keine Interessenkonflikte in Zusammenhang mit diesem Artikel haben.

Literatur

1. Statistisches Bundesamt (Destatis). Mehr Pflegebedürftige. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://www.destatis.de/DE/Themen/Querschnitt/Demografischer-Wandel/Hintergruende-Auswirkungen/demografie-pflege.html
2. Ahnert J, Ladwig J, Holderied A, Brüggemann S, Vogel H. Optimierung des Reha-Entlassungsberichts der Deutschen Rentenversicherung - die Sichtweisen der Adressaten bzw. Nutzer [Optimising the rehabilitation discharge report of the German statutory pension insurance: the recipients' and users' perspectives]. Gesundheitswesen. 2014 Jun;76(6):351-8. DOI: 10.1055/s-0033-1348224
3. Fachinger U, Mähs M. Digitalisierung und Pflege. In: Klauber J, Geraedts M, Friedrich J, Wasem J, Hrsg. Krankenhausreport 2019. Berlin: Springer-Verlag; 2019.
4. Mess EV, Balic S, Daufratshofer L, Kleybolte L, Regner M, Seifert N, Tilmes S, Waibel AK, Swoboda W, Teynor A, Mahler A. Review of Care Transition Records and Their Transmission Process in Nursing Facilities and Hospitals in Germany – Results of an Online Questionnaire. Stud Health Technol Inform. 2023 Jun 29;305:240-243. DOI: 10.3233/SHTI230473
5. Bundesministerium für Gesundheit. Digitale-Versorgung-und-Pflege-Modernisierungs-Gesetz. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://www.bundesgesundheitsministerium.de/service/gesetze-und-verordnungen/guv-19-lp/dvpmg
6. Schulte G, Hübner U, Rienhoff O, Quade M, Rottmann T, Fenske M, Egbert N, Kuhlisch R, Sellemann B. Evaluation einer elektronisch unterstützten pflegerischen Überleitung zwischen Krankenhaus und Pflegeheim unter Nutzung einer Test-Telematikinfrastruktur: eine Fallanalyse [Evaluation of electronically supported nursing transfers between hospital and nursing home based on a test health telematics infrastructure: a case analysis]. GMS Med Inform Biom Epidemiol. 2017;13(1):Doc05. DOI: 10.3205/mibe000172
7. GitHub Technische Hochschule Augsburg – Institut für agile Softwareentwicklung. GitHub; 2024 [zitiert am 21.10.2024]. Verfügbar unter: https://github.com/THAias
8. Hochschule für angewandte Wissenschaften Augsburg. Care Regio – PIO-ÜB Editor. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://www.pio-editor.de/
9. Mess EV, Balic S, Kleybolte L, Regner M, Reuter C, Werlitz V, et al. Design und Entwicklung einer Nutzeroberfläche zur Darstellung des PIO-Überleitungsbogens. In: Boll S, Hein A, et al., Hrsg. Zukunft der Pflege. Tagungsband der 6. Clusterkonferenz 2023. Oldenburg: University of Oldenburg Press; 2023. S. 30-34. URN: urn:nbn:de:gbv:715-oops-59467
10. Health Level Seven International (HL7). HL7 FHIR Release 4. 2019 [zitiert am 05.09.2024]. Verfügbar unter: https://hl7.org/fhir/r4/
11. PIO Überleitungsbogen V1.0.0. In: Simplifier.net. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://simplifier.net/ulb
12. Regner M. PIO-ULB Editor Limitationen. GitHub; 2023 [zitiert am 24.09.2024]. Verfügbar unter: https://github.com/THAias/PIO_Editor_Backend/blob/main/assets/PIOEditorLimitationenMitAnhang.pdf
13. Meta Open Source Project. React. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://react.dev/
14. Ant Group; Ant Design Community. Ant Design 5.0. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://ant.design/
15. HAPI FHIR. Dokumentation von HAPI FHIR. 2024 [zitiert am 05.09.2024]. Verfügbar unter: https://hapifhir.io/
16. Firely. Validating data against profiles. 2023 [zitiert am 05.09.2024]. Verfügbar unter: https://docs.fire.ly/projects/Firely-NET-SDK/en/latest/validation/profile-validation.html

Korrespondenzadresse:
Matthias Regner
Technische Hochschule Augsburg, An der Hochschule 1, 86161 Augsburg, Deutschland
matthias.regner@hs-augsburg.de

Bitte zitieren als
Regner M, Werlitz V, Kleybolte L, Mess EV, Balic S, Daufratshofer L, Tilmes S, Mahler A, Heidegger P, Reuter C, Teynor A. Entwicklung eines Editors zur Erstellung und Bearbeitung Pflegerischer Informationsobjekte (PIOs) zur Pflegeüberleitung. GMS Med Inform Biom Epidemiol. 2024;20:Doc12. DOI: 10.3205/mibe000268, URN: urn:nbn:de:0183-mibe0002687

Artikel online frei zugänglich unter
https://doi.org/10.3205/mibe000268

Veröffentlicht: *20.11.2024*