alpha-Offsync: Verteilte Datensynchronisation in Form von IMAP-basiertem Mailtransfer

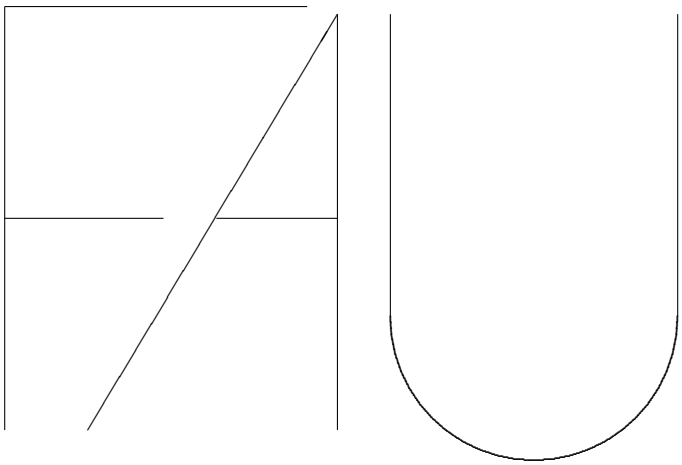
Bachelorarbeit

Andreas Maximilian Wahl

Lehrstuhl für Informatik 6
(Datenmanagement)

Department Informatik
Technische Fakultät

Friedrich Alexander-
Universität
Erlangen-Nürnberg



alpha-Offsync: Verteilte Datensynchronisation in Form von IMAP-basiertem Mailtransfer

Bachelorarbeit im Fach Informatik

vorgelegt von

Andreas Maximilian Wahl

geb. 19.02.1989 in Erlangen

angefertigt am

Department Informatik
Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Univ.-Prof. Dr.-Ing. habil. Richard Lenz
Dipl.-Inf. Christoph P. Neumann

Beginn der Arbeit: 01.06.2011
Abgabe der Arbeit: 31.10.2011

Erklärung zur Selbständigkeit

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Informatik 6 (Datenmanagement), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Bachelorarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 31.10.2011

(Andreas Maximilian Wahl)

Kurzfassung

alpha-Offsync: Verteilte Datensynchronisation in Form von IMAP-basiertem Mailtransfer

Aus der zunehmenden Komplexität interdisziplinärer und institutionsübergreifender Workflows im Gesundheitswesen erwächst die Notwendigkeit einer elektronischen Prozessunterstützung, die die Charakteristika derartiger Prozesse berücksichtigt. Das Forschungsprojekt α-Flow bietet einen Ansatz zur Prozessunterstützung auf Basis von aktiven Dokumenten, der besonders für dynamische Prozesse mit initial unbekannter Anzahl von Beteiligten geeignet ist. Dabei werden die passiven medizinischen Dokumente in einer elektronischen Fallakte gesammelt und mit aktiven Eigenschaften versehen, die eine direkte Interaktion mit der Akte ermöglichen. Zu diesen Eigenschaften gehören unter anderem die automatische Synchronisation mit anderen Akteuren und die Verwaltung des Kreises der Prozessteilnehmer.

Im Rahmen der vorliegenden Arbeit werden Konzepte entwickelt, mit denen neue Informationen unter den Teilnehmern propagiert und die Existenz nebenläufiger globaler Änderungskonflikte der gemeinsamen Informationen auf Basis lokal verfügbarer Informationen erkannt werden können. Darüber hinaus werden Strukturen und Verfahren für den Beitritt neuer Akteure und die konsistente Verwaltung eigenständiger Replikate der Fallakte konzipiert.

Auf Basis des fachlichen Lösungsansatzes werden die erarbeiteten Konzepte in α-Flow integriert. Neben dem Entwurf generischer Softwareschnittstellen wird eine E-Mail-basierte prototypische Implementierung angefertigt.

Abstract

alpha-Offsync: Distributed data synchronization in form of IMAP-based mail transfer

The growing complexity of interdisciplinary and inter-institutional workflows in healthcare sharpens the demand for electronic process support regarding the characteristics of such processes. The research project α-Flow provides an approach to process support based on active documents, which is especially suitable for dynamic processes with an initially unknown number of participants. Thereby passive medical documents are aggregated in an electronic case file provided with active properties, thus allowing a direct interaction with the file itself. These properties include, among others, automatic synchronization with remaining actors and administration of the group of collaborating participants.

In this thesis concepts are developed, which allow the propagation of new information between the participants and enable the detection of distributed global modification conflicts of the shared process status, based on locally available information. Moreover structures and mechanisms are conceived to facilitate joining of new actors and consistent administration of independent copies of the distributed case file.

Based on the conceptual approach to a solution the elaborated concepts are incorporated into α-Flow. In addition to the design of generic software interfaces, an email-based prototypical implementation is developed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ACAP	Application Configuration Access Protocol
ASCII	American Standard Code for Information Interchange
APA	Adornment Prototype Artifact
API	Application Programming Interface
BC API	Bouncy Castle Crypto API
CRA	Collaboration Resource Artifact
CRC-Card	Class-Responsiblity-Collaboration-Card
DSN	Delivery Status Notification
IMAP	Internet Message Access Protocol
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JAXB	Java Architecture for XML Binding
MDA	Mail Delivery Agent
MIME	Multipurpose Internet Mail Extensions
MTA	Mail Transfer Agent
MUA	Mail User Agent
NTP	Network Time Protocol
P-IMAP	Push Extensions to the IMAP Protocol
PGP	Pretty Good Privacy

PSA	Process Structure Artifact
SMAP	Simple Mail Access Protocol
SMTP	Simple Mail Transfer Protocol
UID	Unique Identifier
UML	Unified Modelling Language
UUID	Universally Unique Identifier
XML	Extended Markup Language
XMPP	Extensible Messaging and Presence Protocol

1 Einleitung

Durch stetige Fortschritte in Wissenschaft und Forschung hat sich die Medizin in den vergangenen Jahrzehnten rasant weiterentwickelt. Parallel dazu ist die Komplexität der Patientenbehandlung durch die zunehmende Ausdifferenzierung der einzelnen medizinischen Fachbereiche angestiegen. Heute sind interdisziplinäre und institutionsübergreifende Behandlungsprozesse mit einer unter Umständen hohen Anzahl beteiligter Individuen und Institutionen die Regel.

Der effizienten Koordination der Zusammenarbeit zwischen den Prozessbeteiligten fällt daher eine bedeutende Rolle zu, um Behandlungsepisoden zum Wohle des Patienten erfolgreich durchführen zu können. Besonders wichtig ist in diesem Rahmen die Möglichkeit über elektronische Kommunikationswege die involvierten Parteien schnell und zuverlässig kontaktieren zu können, um diesen die jeweils aktuellen Informationen über den Stand der Behandlung zu übermitteln.

1.1 Motivation und Hintergründe

Eine Erhöhung der Komplexität der medizinischen Behandlungsprozesse führt zugleich auch zu einer Erhöhung der Wahrscheinlichkeit von Fehlern während der Patientenbehandlung. Nach den Ausführungen von [KCD+99] und [CKD+01] starben um die Jahrtausendwende in den USA Jahr für Jahr zwischen 44000 und 98000 Menschen aufgrund vermeidbarer medizinischer Fehler. Viele weitere bleiben aufgrund schwerwiegender Fehlbehandlung für den Rest ihres Lebens körperlich und psychisch beeinträchtigt. Die in amerikanischen Krankenhäusern durch unnötig falsche oder fehlerhafte Behandlungen verursachten Kosten beliefen sich auf 17 bis 29 Milliarden Dollar pro Jahr und verursachten so einen beträchtlichen volkswirtschaftlichen Schaden.

Die Gründe für diese alarmierenden Zahlen sind vielschichtig [KCD+99]. Auf der einen Seite existieren fachliche Fehler, zu denen beispielsweise der Einsatz unzeitgemäßer Behandlungsmethoden, ärztliche Fehler bei der Ausführung von Eingriffen, Fehler bei der Medikamentendosierung oder falsche Diagnosen gehören. Auf der anderen Seite stehen

organisatorische Fehler, zum Beispiel mangelnde Absprachen zwischen den behandelnden Ärzten oder eine fehlerhafte Therapieplanung.

Einer der Faktoren, die die genannten Fehlerursachen begünstigen, ist das Fehlen von Informationen, die zur fundierten Beurteilung des Behandlungsverlaufs benötigt werden. Darüber hinaus ist oftmals nur unzureichendes Wissen über die Bedeutung der vorliegenden medizinischen Informationen vorherrschend.

Gerade diese beiden Einflussfaktoren verdeutlichen das Potential moderner Informationstechnologie im Gesundheitswesen [CKD+01]. So kann beispielsweise durch die Einführung wissensverarbeitender Systeme die Zuverlässigkeit der Patientenbehandlung gesteigert werden. Solche Systeme sind in der Lage, auf Basis patientenbezogener Daten Diagnose- und Therapievorschläge zu unterbreiten oder beim Auftreten kritischer Parameterwerte automatisiert einen Alarm auszulösen. Das Fehlen zeitnah benötigter Informationen kann durch den Einsatz informationsverarbeitender Systeme verhindert werden, indem der Behandlungsprozess mit Hilfe von Software-Unterstützung effizienter koordiniert wird.

Das Forschungsprojekt ProMed hat sich die softwarebasierte Unterstützung von adaptiv-evolutionären Informationssystemen im Gesundheitswesen zum Ziel gesetzt. Zentraler Bestandteil von ProMed ist das Subprojekt α-Flow, das den Informationsaustausch zwischen heterogenen, lose-gekoppelten Informationssystemen im Rahmen der Patientenbehandlung begünstigen soll. Bezogen auf individuelle Institutionen basiert der Austausch von Informationen im Gesundheitswesen vielfach bereits auf internen Standards und Richtlinien zur elektronischen Kommunikation. Der externe Informationstransfer zwischen verschiedenen Institutionen findet jedoch meist noch in Form papierbasierter Schreiben statt.

Um den Charakteristika der zu modellierenden Prozesse, wie beispielsweise dem Fehlen eines zentralen Koordinators oder der initial unbekannten Anzahl von Teilnehmern, Rechnung zu tragen, greift α-Flow das dokumenten-orientierte Paradigma der traditionellen papiergebundenen Kommunikation auf. Die an sich passiven medizinischen Dokumente werden um zusätzliche Funktionalitäten erweitert, die eine direkte Interaktion mit dem Inhalt ermöglichen. Derartige aktive Dokumente enthalten sämtliche prozessbezogenen Informationen und Mechanismen, die zur unmittelbaren Partizipation an einer Behandlungsepisode notwendig sind.

Eine zentrale Funktion aktiver Dokumente im Kontext von α-Flow ist die zuverlässige und schnelle automatisierte Propagation der aktuellsten Behandlungsinformationen auf elektronischem Wege. Im Gegensatz zum zeitintensiven physischen Transport von papier-

basierten Dokumenten stehen neue Informationen unmittelbar nach ihrer Entstehung für alle berechtigten Interessenten zur Verfügung. Die automatische Synchronisation des aktiven Dokuments ermöglicht es somit allen Beteiligten zu jedem Zeitpunkt Zugriff auf die verfügbaren medizinischen Informationen zu geben, um das Risiko einer Fehlbehandlung aufgrund unzureichender Entscheidungsgrundlagen zu minimieren.

1.2 Zielsetzungen der Arbeit

Die vorliegende Arbeit hat es zum Ziel, am Beispiel von α-Flow ein Modell für die zuverlässige Synchronisation von Informationen zwischen Akteuren in dokumenten-orientierten verteilten Workflows zu konzipieren. Dieses Modell soll die Offline-Fähigkeit der einzelnen Prozessteilnehmer realisieren, ohne dabei aufgrund unterbrochener Kommunikation die Konsistenz der verteilten Informationen dauerhaft zu beeinträchtigen.

Komponenten des Fachkonzepts

Für die Erfüllung dieser Aufgabe sind unter Beachtung von Aspekten der Authentisierung und des Datenschutzes spezielle Kommunikationsprotokolle für den strukturierten Austausch von Informationen und den koordinierten Beitritt neuer Prozessteilnehmer zu entwerfen. Um die Konsistenz der gemeinsamen Informationen garantieren zu können, wird ein geeigneter Mechanismus zur Generierung von Zeitstempeln entwickelt. Dieser erlaubt es, die im Gesamtsystem auftretenden Ereignisse zu ordnen sowie unter Erhaltung der Provenienz ausgetauschter Informationen miteinander in Konflikt stehende Aktionen einzelner Akteure zu erkennen. Als Grundlage für die Erkennung der globalen Konflikte werden hierbei ausschließlich die lokal bei einem Akteur vorliegenden Informationen verwendet. Um einen flexiblen Einsatz des entwickelten Gesamtkonzepts zu ermöglichen, wird ein Verfahren ausgearbeitet, das die interne Verwaltung aller physischen Arbeitsplätze der jeweiligen Akteure regelt.

Integration in α-Flow und prototypische Implementierung

Ausgehend von diesem technologieneutralen Fachkonzept wird auf Basis der Java Platform eine generische Softwarearchitektur für die Integration der entwickelten Strukturen und Verfahren in das α-Flow-System entworfen. Diese Architektur unterstützt die Verwendung von Transportprotokollen und Sicherheitskonzepten, die den in der vorliegenden Arbeit identifizierten Grundanforderungen von α-Flow genügen.

Im Rahmen der prototypischen Implementierung werden, gemäß der vorgegebenen Aufgabenstellung dieser Arbeit, das Simple Mail Transfer Protocol (SMTP) und das Internet Message Access Protocol (IMAP) eingesetzt, um einen E-Mail-basierten Informationsaustausch zwischen den Akteuren zu realisieren. Zur Absicherung des Datentransfers und zur Authentisierung der beteiligten Akteure kommt eine Implementierung des OpenPGP-Standards zum Einsatz. Für die Umsetzung des entwickelten Synchronisationskonzepts wird die Anbindung an die Schnittstellen des in α-Flow integrierten Versionsverwaltungssystems realisiert.

2 Methodik

In diesem Kapitel wird die Vorgehensweise erläutert, nach der die in der vorliegenden Arbeit dargelegten Ergebnisse gewonnen werden. Die Beschreibungen der durchgeführten Entwurfsschritte werden den jeweiligen Kapiteln der Ausarbeitung zugeordnet, um dem Leser einen Überblick über den Aufbau dieser Arbeit zu geben.

In Kapitel 3 erfolgt eine Analyse der wissenschaftlichen Grundlagen, die für das zu entwickelnde fachliche Lösungskonzept und die technische Umsetzung relevant sind. Näher eingegangen wird dabei insbesondere auf das Domänenmodell des übergeordneten Forschungsprojekts α-Flow und auf Grundlagen der Kommunikation in verteilten Systemen. Ein weiterer Schwerpunkt wird auf den Umgang mit logischer Zeit in verteilten Systemen gelegt, indem verschiedene Varianten logischer Uhren vorgestellt werden.

An die Betrachtung des wissenschaftlichen Kontextes schließt sich ein Überblick an über die technischen Grundlagen, die zur Realisierung der Lösungskonzepte benötigt werden. Das zugehörige Kapitel 4 geht zu Beginn auf Möglichkeiten zur internen Strukturierung von E-Mail-Nachrichten ein. Anschließend werden die Konzepte der für den Nachrichtentransport vorgesehenen Protokolle Simple Mail Transfer Protocol (SMTP) und Internet Message Access Protocol (IMAP) dargestellt. Da das existierende α-Flow System auf der Java Platform Standard Edition realisiert ist, wird auf für die vorliegende Arbeit wichtige Konzepte der Java Platform eingegangen. Diese beinhalten eine kurze Vorstellung des JavaMail Application Programming Interface und der Grundlagen der Java-basierten Kryptographie.

Ausgehend von der Betrachtung der wissenschaftlichen und technischen Grundlagen wird der fachliche Lösungsansatz entwickelt. Hierzu erfolgt in Kapitel 5 zunächst eine Analyse der im Rahmen der vorliegenden Arbeit zu behandelnden Problemstellungen. Basierend auf einem möglichen Einsatzszenario wird erläutert, welche Teilkomponenten für das zu entwickelnde Synchronisationskonzept benötigt werden. Die Anforderungen an die einzelnen Komponenten, die erfüllt werden müssen, um den Benutzern einen angemessenen Mehrwert liefern zu können, werden schrittweise erarbeitet und zu einem Anforderungsprofil der gesuchten Lösung zusammengefasst.

Kapitel 6 beinhaltet eine Diskussion verwandter Arbeiten mit Bezug zu den vorgesehenen Lösungskonzepten, um die Notwendigkeit eines eigenständigen Lösungsansatzes zu motivieren. Die Ausführungen umfassen Ansätze zur Synchronisation physischer Uhren und weiterführende Konzepte zur Generierung und Verwendung logischer Zeitstempel. Daneben werden Verfahren zur Datensynchronisation in verteilten Datenbanken erläutert, bei denen logische Zeitstempel und dynamische Änderungsrechte zum Einsatz kommen.

Auf Basis des entworfenen Anforderungsprofils und der verwandten Arbeiten wird in Kapitel 7 der fachliche Lösungsansatz erarbeitet und detailliert präsentiert. Es wird auf die zur Synchronisation zwischen den Akteuren benötigten Strukturen und Verfahren eingegangen, mit denen unter Verwendung logischer Zeitstempel eine Ordnung auf den im Gesamtsystem durchgeführten Modifikationen hergestellt werden kann. Darüber hinaus werden die Konzepte zur Verwaltung unterschiedlicher physischer Arbeitsplätze pro Akteur erläutert. Als weitere Komponente des Lösungsansatzes wird ein Kommunikationsprotokoll entwickelt, mit dessen Hilfe neue Akteure dem Kreis der existierenden Prozessteilnehmer dynamisch beitreten können.

Ausgehend von diesen theoretischen Ausführungen wird in Kapitel 8 die Integration der fachlichen Lösungskonzepte in α-Flow anhand einer Architekturübersicht beschrieben. Es werden die Zuständigkeitsbereiche der Teilkomponenten untersucht und die Beziehungen zwischen einzelnen Komponenten aus konzeptioneller Sicht beschrieben.

Um diesen Grobentwurf in Software überführen zu können, werden im Feinentwurf in Kapitel 9 die generischen Schnittstellen und Konzepte des entwickelten Kommunikationssubsystems und der Synchronisationsmechanismen definiert. Diese bilden die Grundlage für die Erstellung von Implementierungen, die spezifische Transportprotokolle zum Nachrichtenaustausch benutzen und auf Umsetzungen der Konzepte zur Verwaltung logischer Zeitstempel und des entwickelten Verfahrens zur Ordnung von Modifikationen zurückgreifen.

Eine prototypische Implementierung dieser Schnittstellen hat Kapitel 10 zum Thema. Hier wird eingegangen auf ausgewählte Aspekte der Umsetzung der fachlichen Lösungskonzepte unter Verwendung von E-Mail-Nachrichten, die mittels SMTP und IMAP zwischen den Akteuren ausgetauscht werden.

Abschließend werden in Kapitel 11 Anregungen für weiterführende, auf den Ergebnissen der vorliegenden Arbeit beruhende Forschungen im Kontext von α-Flow vorgestellt.

3 Wissenschaftliche Grundlagen

Als Einstieg in die Thematik wird mit einem Überblick über den wissenschaftlichen Kontext begonnen, in den die vorliegende Arbeit eingebettet ist. Am Anfang der Ausführungen stehen die konzeptionellen Grundlagen des übergeordneten Forschungsprojekts α-Flow. Darauf folgend werden theoretische Grundlagen zu verteilten Systemen vermittelt. Den Abschluss bildet eine Übersicht grundlegender Ansätze zur verteilten Synchronisation unter Verwendung von logischen Zeitstempeln.

3.1 α-Flow Domänenmodell

Detaillierte Beschreibungen der einzelnen Elemente des Domänenmodells von α-Flow finden sich in den Veröffentlichungen [NL09], [NL10], [NL12] sowie in den vorangegangenen Abschlussarbeiten [Tod10] und [Han10]. Als Referenz für die Struktur des evolutionsfähigen Metadatenmodells von α-Flow sei auf [Sch11] verwiesen. Für ein leichteres Verständnis der Ausführungen in späteren Kapiteln werden die zentralen Aspekte des Domänenmodells zusammengefasst.

3.1.1 α-Episode und α-Doc

Im Fokus der Betrachtung von α-Flow stehen institutionsübergreifende verteilte Workflows zur Behandlung von Patienten im Gesundheitswesen. Ein derartiger Workflow wird im Folgenden als α-Episode bezeichnet und umfasst die Behandlung eines bestimmten Patienten. An einer α-Episode kann eine Vielzahl von Personen bzw. Institutionen beteiligt sein. Alle diese Akteure streben als gemeinsames Ziel den erfolgreichen Abschluss einer Patientenbehandlung an.

Nach den Grundsätzen von α-Flow wird davon ausgegangen, dass sich, gemäß des diagnostisch-therapeutischen Zyklus [BMH97], Entscheidungen in institutionsübergreifenden Prozessen als Anforderungen zusätzlicher Informationen modellieren lassen. Durch solche Informationsanforderungen kann der Prozess von den einzelnen Akteuren in eine individuelle Richtung gesteuert werden. Eine α-Episode endet, wenn das Ziel des Behand-

lungsprozesses erreicht ist. Dies ist genau dann der Fall, wenn es keine Notwendigkeit mehr zum Einfordern neuer Informationen gibt und alle Akteure auf Basis der ihnen vorliegenden Informationen fundierte medizinische Entscheidungen treffen konnten.

Repräsentiert wird jede α-Episode in Form eines α-Doc. Dabei handelt es sich um eine elektronische Fallakte, die alle Dokumente enthält, die für den Behandlungsprozess relevant sind und im Verlauf der Patientenbehandlung entstanden sind. Diese umfassen somit die expliziten Anforderungen neuer Inhalte und die daraufhin von den Akteuren eingeholten Informationen.

3.1.2 α-Cards

Die einzelnen Informationseinheiten innerhalb des α-Doc werden durch die sogenannten α-Cards repräsentiert. Diese Modellierung als eigenständige Prozessartefakte erlaubt ein hohes Maß an Flexibilität und ermöglicht die feingranulare Vergabe von Änderungsrechten und Verantwortungsbereichen unter den beteiligten Akteuren. α-Cards stellen die atomare Einheit der Validierung und Freigabe an andere Akteure dar. Jede α-Card besteht aus zwei unterschiedlichen Teilen: ihrem Deskriptor und dem zugehörigen Payload (Abbildung 3.1).

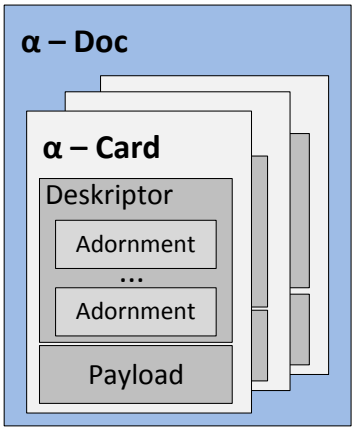


Abbildung 3.1: Aufbau eines α-Doc aus den einzelnen α-Cards

Der Deskriptor enthält prozessrelevante Statusattribute, die sogenannten α-Adornments. Diese können zur Steuerung des Prozessverlaufs eingesetzt werden. Mit ihrer Hilfe kann eine α-Card eindeutig identifiziert und benannt werden. Sie erlauben zudem die Speicherung von identifizierenden Daten des betroffenen Patienten und des für die α-Card verantwortlichen Akteurs. Daneben existieren weitere Adornments, mit denen die semantische Bedeutung einer α-Card sowie ihre Gültigkeit und Sichtbarkeit charakterisiert werden können.

Unter Verwendung des adaptiv-evolutionären Metadatenmodells für α-Adornments wird es den beteiligten Akteuren ermöglicht, dynamisch zur Laufzeit, neue domänenspezifische Adornments zu erstellen und sie α-Cards zuzuweisen. Auf diese Weise können weitere patientenbezogene Informationen, wie zum Beispiel Parameterwerte von Laborergebnissen, persistiert und zur Steuerung des weiteren Prozessverlaufs herangezogen werden.

Im zu einer α-Card gehörenden Payload werden die eigentlichen Nutzinformationen abgespeichert. Welche Art von Informationen in diesem Payload enthalten ist, wird durch den Typ der jeweiligen α-Card festgelegt. Zur Trennung von medizinischen Inhalten und prozessrelevanten Metadaten wird grundlegend zwischen Content-α-Cards und Coordination-α-Cards unterschieden, die eine jeweils eigene Semantik aufweisen.

3.1.2.1 Content-α-Cards

Content-α-Cards stellen die medizinischen Nutzinformationen dar, die im Rahmen der Patientenbehandlung entstehen und als Teildokument in das α-Doc aufgenommen werden. Sie können beispielsweise die Repräsentation eines Befundes, einer therapeutischen Maßnahme oder einer Überweisung sein. Ihr Payload besteht aus binären Dateien in beliebigem Format, die als Ergebnis der durchgeführten medizinischen Handlung im lokalen Informationssystem eines beteiligten Akteurs zu Dokumentationszwecken entstanden sind.

3.1.2.2 Coordination-α-Cards

Die Coordination-α-Cards hingegen dienen zur Verwaltung von prozessrelevanten Metainformationen. Es existieren drei verschiedene Typen von Coordination-α-Cards: Das Process Structure Artifact (PSA), das Collaboration Resource Artifact (CRA) und das Adornment Prototype Artifact (APA).

Aufgabe des PSA ist die Verwaltung aller angelegten α-Cards. Sein Payload besteht zu diesem Zweck aus einer Liste aller bekannten α-Cards und einer Liste, in der alle Abhängigkeiten zwischen einzelnen α-Cards verzeichnet sind.

Mit dem CRA kann der Kreis der am Prozess teilnehmenden Akteure verwaltet werden. Im zugeordneten Payload werden die identifizierenden Daten der einzelnen Akteure gespeichert. Zudem wird hinterlegt, unter welcher elektronischen Adresse die jeweiligen Akteure kontaktiert werden können.

Der Payload des APA enthält alle von den Akteuren angelegten domänenspezifischen α-Adornments und deren Attribute. Zu diesen Eigenschaften gehören unter anderem die gültigen Wertebereiche einzelner Adornments und die zugrundeliegenden Datentypen.

3.1.3 Das α-Doc als aktives Dokument und verteilte Fallakte

Traditionelle papierbasierte Fallakten verhalten sich grundsätzlich passiv und dienen als einfache Sammelbehälter für die im Behandlungsverlauf von den beteiligten Akteuren angelegten Dokumente. Im Gegensatz hierzu bietet das α-Doc in seiner Rolle als aktives Dokument ein breites Spektrum zusätzlicher Funktionalitäten. Aktive Dokumente erlauben dem Benutzer eine direkte Interaktion mit den im Dokument hinterlegten Inhalten und besitzen aktive Eigenschaften. Das α-Doc bietet hierzu benutzerdefinierte Sichten auf die in ihm enthaltenen Informationen und erlaubt eine einfache Bearbeitung der Deskriptoren einzelner Prozessartefakte.

Darüber hinaus repräsentiert ein α-Doc eine physikalisch verteilte, logisch jedoch zentralisierte patientenspezifische Fallakte. Das bedeutet, dass alle existierenden Repliken eines α-Doc, die zur selben α-Episode gehören, automatisch untereinander synchronisiert werden (Abbildung 3.2).

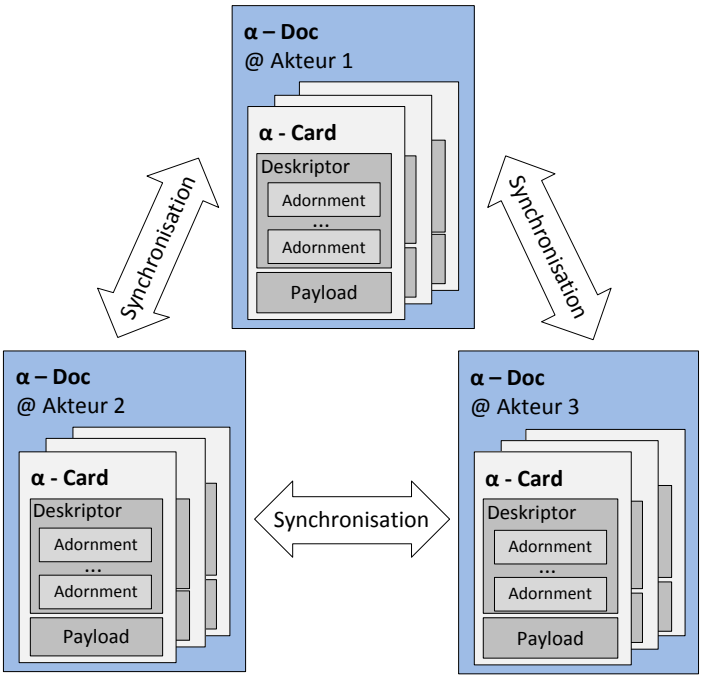


Abbildung 3.2: Synchronisation einer verteilten Fallakte in Form eines α-Doc

Jedem teilnehmenden Akteur muss eine individuelle Kopie des α-Doc übersandt werden. Beim ersten Öffnen seiner Kopie kann der Akteur seine identifizierenden Daten sowie die Adresse eines elektronischen Kommunikationskanals eingeben. Alle Änderungen, die von anderen Akteuren an α-Cards innerhalb des α-Doc durchgeführt werden, können ihm über diesen Kanal mitgeteilt werden. Das α-Doc wird bis zum Ende der α-Episode automatisch zwischen den Akteuren synchronisiert, um allen Prozessbeteiligten die jeweils aktuellsten organisatorischen und medizinischen Informationen zur Verfügung zu stellen.

3.2 Verteilte Systeme

Vernetzte Rechnersysteme haben in den letzten Jahrzehnten und vor allem in jüngster Zeit eine stetige Weiterentwicklung durchlaufen. Durch ihre inhärente Flexibilität eignen sie sich besonders zur Förderung der Interaktion zwischen verschiedenen Teilnehmern in heterogenen Systemen, wie sie etwa für die sinnvolle Nutzung von α-Flow notwendig ist. Dieser Abschnitt behandelt die Grundlagen solcher verteilten Systeme. Neben deren Definition, Charakteristika und möglichen Problemen wird detailliert auf die grundlegenden Konzepte zur geordneten und zuverlässigen Kommunikation zwischen den Netzknoten eingegangen. Als Voraussetzung für das Verständnis der nachfolgenden Abschnitte wird zudem der fundamentale Unterschied zwischen physischer und logischer Zeit erläutert.

3.2.1 Definition und charakteristische Eigenschaften

Andrew S. Tanenbaum definiert den Begriff eines verteilten Systems in [TS03] folgendermaßen:

Definition 1. *Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen.*

Abbildung 3.3 veranschaulicht diese Sicht auf verteilte Systeme: Details einzelner Netzknoten bleiben dem Benutzer verborgen, da er aus seiner Perspektive nur mit dem Gesamtsystem in Interaktion tritt. Ausgehend von dieser allgemeinen Definition lassen sich verschiedene grundlegende Charakteristika verteilter Systeme identifizieren. Basierend auf den Ausführungen von [CDK02] und [Ham05] sind dies folgende Eigenschaften:

- **Heterogenität:** Verteilte Systeme können eine Vielzahl von Netzwerken, Betriebssystemen, Hardwarearchitekturen und Programmiersprachen verbinden.

- **Offenheit:** Verteilte Systeme sind bei Verfügbarkeit geeigneter Schnittstellen und Kommunikationsprotokolle leicht um zusätzliche Netzknoten und Komponenten erweiterbar.
- **Sicherheit:** Mittels kryptographischer Verfahren lässt sich die Kommunikation zwischen voneinander entfernten Netzknoten absichern. Durch geeignete Maßnahmen lassen sich einzelne Knoten oder das Gesamtsystem vor Angriffen schützen.
- **Skalierbarkeit:** Das Hinzufügen neuer Netzknoten und Komponenten verursacht im Verhältnis zu den benötigten Gesamtressourcen des Systems einen konstanten Kostenaufwand.
- **Fehlerverarbeitung:** Ausfälle einzelner Netzknoten oder Komponenten sind kompensierbar, ohne die Funktionalität des Gesamtsystems zu gefährden.
- **Nebenläufigkeit:** Mit steigender Anzahl von Netzknoten und Benutzern steigt das Risiko von Konflikten beim Zugriff auf gemeinsam genutzte Ressourcen. Alle verfügbaren Ressourcen werden so verwaltet, dass sich nicht-deterministische Zugriffskonflikte vermeiden lassen.
- **Transparenz:** Bestimmte Aspekte der Verteilung werden vor Anwendungsprogrammierern und Benutzern verborgen, um die Interaktion mit dem Gesamtsystem zu vereinfachen.

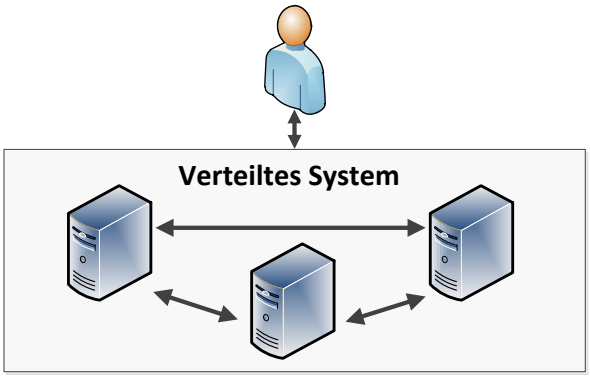


Abbildung 3.3: Benutzersicht auf ein verteiltes System

3.2.2 Overlay-Netzwerke

In verteilten Systemen kommunizieren die verschiedenen Netzknoten über ein darunterliegendes Netzwerk von Datenverbindungen miteinander. Bei diesem Netzwerk muss es sich jedoch nicht zwingend um ein speziell für das verteilte System geschaffenes

physisches Netzwerk handeln. Vielmehr bilden die einzelnen Knoten zumeist ein sogenanntes Overlay-Netzwerk. Dabei handelt es sich um ein logisches Netzwerk, das auf ein darunterliegendes physisches Netzwerk aufsetzt. Für die einzelnen Netzknoten ist nur die Struktur des logischen Netzwerks sichtbar. Durch die Verwendung eines logischen Netzwerks kann die darunterliegende physische Infrastruktur weiterhin genutzt werden. Für die Adressierung der Netzknoten und die Wegewahl für die Kommunikation können jedoch individuelle Protokolle definiert werden, um das Overlay-Netzwerk im Hinblick auf die beabsichtigte Nutzung zu optimieren.

Ein Beispiel für ein derartiges Overlay-Netzwerk ist in Abbildung 3.4 dargestellt. Das zugrundeliegende physische Netzwerk besteht aus fünf Knoten, die in Sterntopologie miteinander verbunden sind. Das Overlay-Netzwerk hingegen besteht aus nur vier Knoten, die in einem Ring miteinander kommunizieren, ohne dem Benutzer Details über die Struktur des darunterliegenden Netzwerks zugänglich zu machen. Daten, die aus Benutzersicht im Overlay-Netzwerk direkt von Client zu Client übertragen werden, müssen den zentralen Netzknoten des physischen Netzwerks passieren und werden erst von dort aus an den zum jeweiligen Client gehörenden physischen Knoten weitergeleitet. Als Beispiel für eine praktische Anwendung von Overlay-Netzwerken sind unter anderem Peer-to-Peer-Protokolle zu nennen.

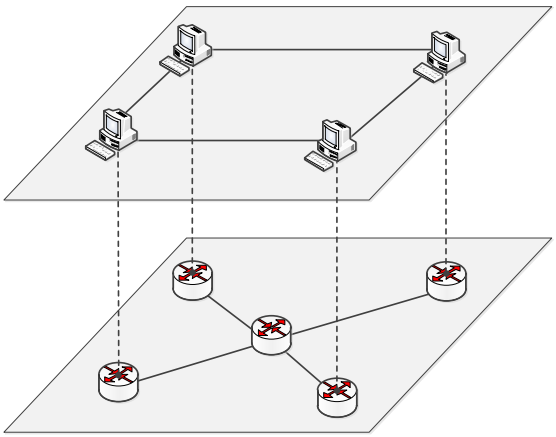


Abbildung 3.4: Overlay-Netzwerk mit Ringtopologie aufsetzend auf einem physischen Netzwerk mit Sterntopologie

3.2.3 Nachrichtenorientierte Kommunikation

Für die Kommunikation in verteilten Systemen lassen sich verschiedene Ansätze für den Datenaustausch identifizieren. Bei der synchronen Kommunikation wartet der aufrufende

Teilnehmer solange, bis die von ihm übermittelten Daten vom Empfänger vollständig verarbeitet worden sind. Bei der asynchronen Kommunikation wird nicht auf die Verarbeitung durch andere Netzknoten gewartet. Falls erforderlich, werden Netzknoten durch einen vom Empfänger initiierten Datenaustausch über die erfolgreiche Verarbeitung der Daten benachrichtigt.

Um einzelne Netzknoten in einem verteilten System nicht länger als nötig zu blockieren, sollten die Netzknoten auf synchronen Datenaustausch untereinander verzichten und asynchron kommunizieren. Asynchrone Kommunikation kann beispielsweise über den Austausch von Nachrichten zwischen Netzknoten realisiert werden. Zu den wichtigsten Aspekten dieser nachrichtenorientierten Kommunikation gehören die Verwaltung von Warteschlangen, die zuverlässige Übermittlung von Nachrichten sowie ereignisbasierte Benachrichtigungen.

3.2.3.1 Warteschlangenmodell für Nachrichten

Bei der nachrichtenorientierten Kommunikation besitzen Netzknoten eine Nachrichtenwarteschlange, deren Verwendung in Abbildung 3.5 illustriert ist.

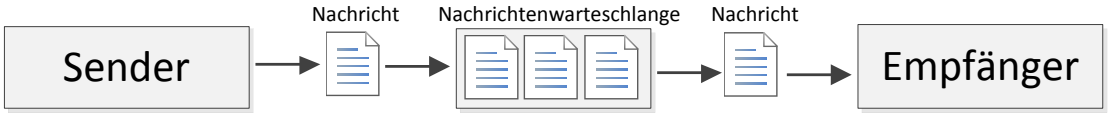


Abbildung 3.5: Funktionsweise einer Nachrichtenwarteschlange

Zu übermittelnde Nachrichten werden vom Absender in die Warteschlange des Empfängers eingefügt. Das hat den Vorteil, dass der Absender nicht wissen muss, ob der jeweilige Empfänger zum Zeitpunkt des Versandes gerade Nachrichten verarbeiten kann oder nicht. Sendende Knoten können also direkt nach dem Versand einer Nachricht weiterarbeiten, ohne explizit auf deren Verarbeitung zu warten. Zudem kann der Empfänger durch Benutzung einer Warteschlange selbst den Zeitpunkt bestimmen, zu dem er empfangene Nachrichten verarbeiten möchte, indem er sie aus seiner Warteschlange entnimmt.

3.2.3.2 Zuverlässige Nachrichtenübermittlung

Um eine Aussage über die Korrektheit des Informationsaustausches zwischen verschiedenen Netzknoten treffen zu können, ist es erforderlich, die Zuverlässigkeit der Nachrichtenübermittlung zu analysieren. Es existieren verschiedene Konzepte, die von Seiten des verwendeten Kommunikationsprotokolls zu implementieren sind, um eine zuverlässige Kommunikation zu ermöglichen.

Versand von Empfangsbestätigungen

Versendet ein Netzknoten eine Nachricht an den vorgesehenen Empfänger, so sollte der Absender darüber informiert werden, ob die Nachricht auch tatsächlich auf der Gegenseite angekommen ist. Hierfür wird für jede ankommende Nachricht vom Empfänger eine Empfangsbestätigung an den Absender der Nachricht gesendet (Abbildung 3.6).

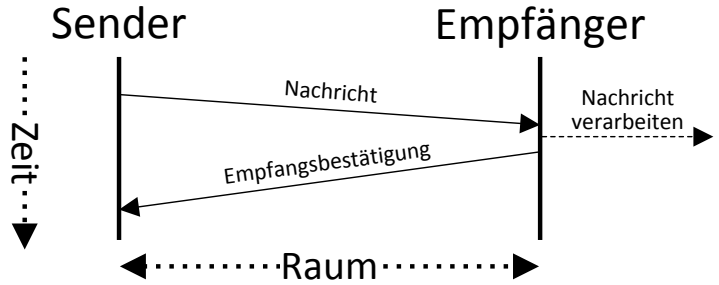


Abbildung 3.6: Versand von Empfangsbestätigungen

Trifft diese Bestätigungsnachricht innerhalb einer festgelegten Zeitspanne beim Sender der ursprünglichen Nachricht ein, so klassifiziert dieser die Nachricht als erfolgreich zugestellt. Verstreicht die zuvor festgelegte Zeitspanne jedoch ohne den Eingang der Empfangsbestätigung, so gilt die Nachricht als fehlerhaft zugestellt. Es obliegt dann dem Absender der ursprünglichen Nachricht, diese gegebenenfalls erneut zu versenden, um eine erfolgreiche Zustellung zu ermöglichen.

Einhaltung der korrekten Nachrichtenreihenfolge

Aufgrund von technischen Verzögerungen auf dem Transportweg vom Sender zum Empfänger ist es möglich, dass Nachrichten nicht in exakt der Reihenfolge empfangen werden, in der sie versandt worden sind (Abbildung 3.7).

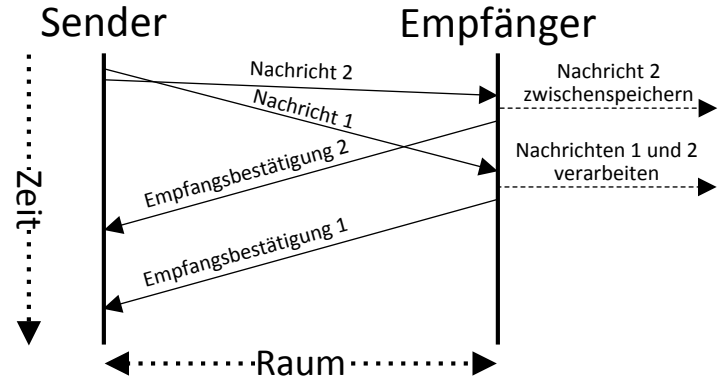


Abbildung 3.7: Empfang von Nachrichten in abweichender Reihenfolge

Zwischen den Inhalten verschiedener einzelner Nachrichten können jedoch semantische Abhängigkeiten bestehen, die es erforderlich machen, dass die Nachrichten in genau der Reihenfolge verarbeitet werden, in der sie versandt worden sind. Es ist daher notwendig, dass der Empfänger erkennen kann, ob alle Vorgänger einer Nachricht bereits empfangen worden sind und die Nachricht problemlos verarbeitet werden kann. Zu diesem Zweck muss jeder Nachricht auf Ebene des Transportprotokolls eine eindeutige Sequenznummer zugeordnet werden. Wird Nachricht 1 vor Nachricht 2 versandt, so muss garantiert sein, dass die Sequenznummer von Nachricht 1 kleiner als die von Nachricht 2 ist.

Wird eine Nachricht empfangen, deren Vorgänger noch nicht vollständig empfangen worden sind, wird die Nachricht solange vom Empfänger zwischengespeichert, bis die fehlenden Nachrichten eingetroffen sind. Sind alle notwendigen Nachrichten lokal verfügbar, so werden diese in der Reihenfolge ihrer Sequenznummer und somit in der Reihenfolge des Versands verarbeitet.

Behandlung von Duplikaten von Nachrichten

Erhält der Absender einer Nachricht keine rechtzeitige Empfangsbestätigung, so kann eines von zwei möglichen Fehlerszenarien eintreten sein. Zum einen ist es möglich, dass die ursprüngliche Nachricht auf dem Weg zum Empfänger verloren gegangen ist und daher keine Empfangsbestätigung verschickt wurde (Abbildung 3.8a). In diesem Fall kann die Nachricht erneut an den Empfänger gesendet werden, ohne weitere Probleme zu erzeugen. Wird dieses Duplikat erfolgreich übertragen, antwortet der Empfänger entsprechend mit einer Empfangsbestätigung und beginnt mit der lokalen Verarbeitung der Nachricht.

Es ist jedoch auch möglich, dass zwar die Nachricht korrekt an den Empfänger ausgeliefert und von diesem lokal verarbeitet worden ist, die Empfangsbestätigung jedoch zu spät beim Absender der ursprünglichen Nachricht ankommt oder auf dem Transportweg verloren geht (Abbildung 3.8b). Wird in einem solchen Fall ein Duplikat der Nachricht an den Empfänger gesendet, muss sichergestellt werden, dass dieser erkennen kann, dass es sich bei einer ankommenden Nachricht um ein Duplikat handelt. Hierzu kann beispielsweise die eindeutige Sequenznummer der Nachrichten verwendet werden. Werden Duplikate nicht korrekt aussortiert, ist es möglich, dass das System aufgrund der daraus resultierenden mehrfachen Verarbeitung von Anfragen in einen fehlerhaften oder inkonsistenten Zustand gelangt.

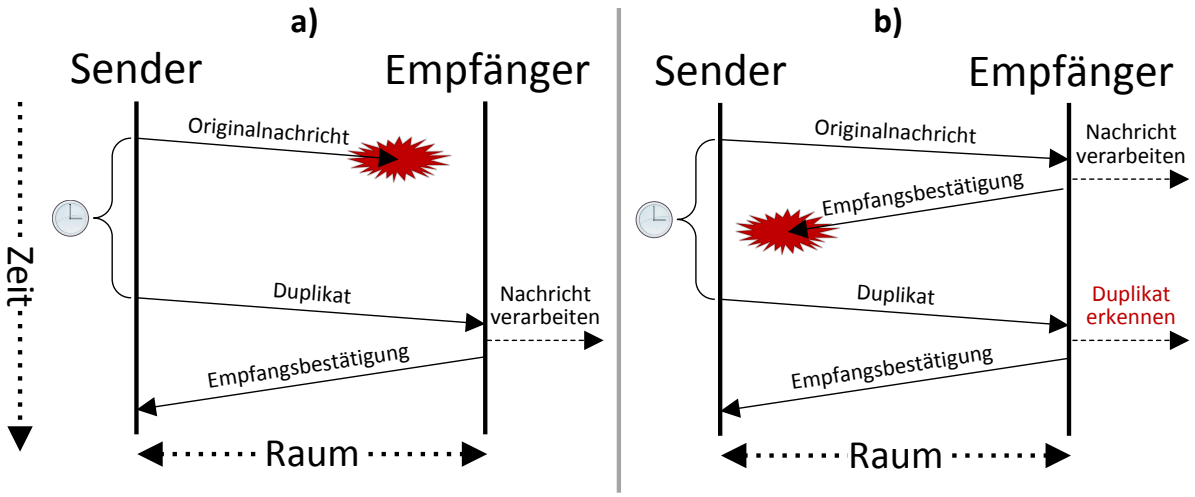


Abbildung 3.8: Mehrfacher Versand einer Nachricht

Persistente Kommunikation

Um bei Verwendung des Warteschlangemodells neben den zuvor genannten Aspekten der zuverlässigen Nachrichtenübermittlung zu garantieren, dass keine Nachrichten, die erfolgreich in die Transportwarteschlange eingefügt wurden, verloren gehen, muss diese Warteschlange persistiert werden. So kann nach Fehlern, die ein Wiederaufsetzen des Empfängerknotens erfordern, der letzte gültige Zustand der Warteschlange wiederhergestellt werden, um alle eingegangenen Nachrichten nach Behebung des Fehlers und dem Wiederaufsetzen verarbeiten zu können. Zudem ist es nicht erforderlich, dass der vorgesehene Empfänger zum Zeitpunkt des Nachrichtenversands mit dem Netzwerk verbunden ist, da alle ankommenden Nachrichten solange zwischengelagert werden, bis er das nächste Mal zum Empfang bereit ist. Persistente Kommunikation sichert dem Sender somit zu, dass jede von ihm an die Warteschlange übermittelte Nachricht verarbeitet wird, trifft aber keine Aussage darüber, zu welchem Zeitpunkt eine Verarbeitung zu erwarten ist.

3.2.3.3 Ereignisbasierte Benachrichtigungen

Netzknoten müssen in zahlreichen Anwendungen über den lokalen Eintritt bestimmter Ereignisse innerhalb des Systems benachrichtigt werden. Dabei gibt es zwei grundlegend unterschiedliche Herangehensweisen an die Verbreitung der Informationen über den Eintritt eines Ereignisses unter den Netzknoten.

Pull-Modell

Die erste Möglichkeit ist die Verwendung des sogenannten Pull-Modells. Dabei fragen alle Netzknoten, die sich für den eventuellen Eintritt von Ereignissen interessieren, in periodischen Abständen explizit bei anderen Netzknoten an, ob Informationen über neue Ereignisse vorliegen. Ist dies der Fall, erhalten die anfragenden Knoten eine Antwort, die sie über das jeweils eingetretene Ereignis benachrichtigt.

Vorteilhaft ist, dass die anfragenden Knoten selbst bestimmen können, wann sie Informationen über neue Ereignisse abrufen möchten. Problematisch ist jedoch die Wahl der Dauer zwischen zwei Anfragen, da nicht immer bekannt ist, zu welchen Zeitpunkten Ereignisse eintreten. Somit kann es sein, dass unnötig viele Anfragen gesendet werden. Es ist aber auch möglich, dass die Periodendauer zu groß ist, um auf schnell aufeinander folgende Ereignisse, die eine zügige Verarbeitung erfordern, angemessen zu reagieren.

Push-Modell

Um das Problem der Wahl einer geeigneten Aktualisierungsfrequenz zu umgehen, kann alternativ das Push-Modell [BMR+96] verwendet werden. Bei diesem Interaktionsmodell registriert sich jeder Knoten, der über den Eintritt von Ereignissen benachrichtigt werden möchte, einmalig bei Knoten, von denen er Benachrichtigungen erhalten möchte. Tritt nun ein Ereignis ein, wird jeder zuvor registrierte Interessent unverzüglich darüber informiert.

Der Vorteil des Push-Modells ist, dass keine unnötigen Nachrichten versandt werden müssen, um festzustellen, ob neue Ereignisse eingetreten sind. Nur im Fall eines tatsächlichen Ereigniseintritts kommt es zum Nachrichtenaustausch. Kritisch zu sehen ist jedoch, dass der Empfänger den Zeitpunkt, an dem er Informationen über Ereignisse erhält, nicht mehr selbst beeinflussen kann und zu jeder Zeit in der Lage sein muss, ankommende Nachrichten in Empfang zu nehmen.

3.2.4 Probleme der Uhrensynchronisation in verteilten Systemen

Typische Computersysteme besitzen eine integrierte Uhr, welche die lokal aktuelle Uhrzeit festlegt. Vielfach handelt es sich dabei um einen Quarz-Kristall, der mittels elektrischer Spannung in Schwingung versetzt wird [TS03]. Dieser Zeitgeber generiert in periodischen Abständen Benachrichtigungen, die dem Computer das Fortschreiten der lokalen Zeit signalisieren.

Bei Softwaresystemen, die nur auf einem einzigen Computer ausgeführt werden, macht es dabei in den meisten Fällen keinen Unterschied, ob dieser lokale Zeitgeber geringfügig von der tatsächlichen physischen Zeit abweicht. Alle auf dem Computer ausgeführten Programme arbeiten mit derselben, potenziell von einer globalen Referenz abweichenden, Uhrzeit.

Anders ist die Situation bei verteilten Systemen. Da hier mehrere Computer in einem gemeinsamen Netz interagieren, gibt es in diesem Netz mehrere lokale Zeitgeber und dementsprechend unter Umständen mehrere voneinander abweichende lokale Uhrzeiten. Darüber hinaus kann auch die Zählgeschwindigkeit verschiedener Uhren unterschiedlich sein, was dazu führt, dass die Uhren an verschiedenen Netzknoten unterschiedlich schnell fortlaufen.

Für zahlreiche Anwendungen und Messungen im Kontext verteilter Systeme ist es jedoch erforderlich, eine gemeinsame globale Zeit für alle Netzknoten festlegen zu können. So kann es beispielsweise notwendig sein, Nachrichten mit einem Zeitstempel zu versehen, da sie nur strikt in ihrer Erstellungsreihenfolge verarbeitet werden dürfen, um die Konsistenz des Gesamtsystems zu erhalten.

Um die unterschiedlichen lokalen Uhren in verteilten Systemen zu synchronisieren, wurden verschiedene Verfahren entwickelt. Die grundlegende Funktionsweise von Verfahren, die eine fortlaufende Annäherung einzelner Zeitgeber anstreben, wird in Kapitel 6 im Rahmen der Betrachtung verwandter Arbeiten erläutert.

Ist jedoch weniger die physische Uhrzeit von Ereignissen für die Korrektheit eines verteilten Systems relevant, sondern die Möglichkeit, Ereignisse innerhalb des Systems in eine Ordnung bringen zu können, so kann auf die Synchronisation physischer Uhren verzichtet werden. Es ist in diesen Fällen ausreichend, wenn sich alle Netzknoten auf eine gemeinsame Zeit einigen. die jedoch nicht mit der realen physischen Zeit in Verbindung stehen muss.

3.3 Logische Zeit

Als Vorbereitung auf den in Kapitel 7 beschriebenen fachlichen Lösungsansatz der vorliegenden Arbeit, werden in diesem Abschnitt grundlegende Konzepte des Umgangs mit logischer Zeit in verteilten Systemen präsentiert. Als Ausgangspunkt dient die Betrachtung der Theorie logischer Zeitmessung. Darauf aufbauend, werden Algorithmen zur Synchronisierung von Ereignissen in verteilten Systemen unter Verwendung logischer Zeit diskutiert, die unterschiedlichen Anforderungen an logische Uhren genügen.

3.3.1 Theoretische Grundlagen

Leslie Lamport präsentierte 1978 in seinem Werk *Time, Clocks, and the Ordering of Events in a Distributed System* [Lam78] erste Konzepte zum Herstellen einer Ordnung auf der Menge der Ereignisse innerhalb eines verteilten Systems mit Hilfe von logischen Zeitstempeln.

3.3.1.1 Die Happened-Before-Relation

Bei seinen Ausführungen geht Lamport von einem verteilten System aus, in dem eine endliche Anzahl von Prozessen untereinander über den Austausch von Nachrichten kommuniziert. Ein einzelner Prozess wird in diesem Modell als eine Folge von Ereignissen betrachtet, d.h. als eine Menge deren Elemente sich in eine totale Ordnung bringen lassen. Tritt also ein Ereignis a vor einem Ereignis b ein, so ist a in der Folge der Ereignisse eines Prozesses vor b einzuordnen. Ereignisse sind dabei neben internen Vorgängen eines Prozesses insbesondere der Versand oder Empfang einer Nachricht innerhalb des verteilten Systems.

Ausgehend von diesen Annahmen wird zur Formalisierung des Begriffs der Abfolge von Ereignissen eine Relation „$\rightarrow$“ (*Happened-Before*) wie folgt definiert:

Definition 2. *Die Relation „$\rightarrow$“ auf einer Menge von Ereignissen ε in einem verteilten System ist der schwächste Zusammenhang zwischen Elementen dieser Menge, der die folgenden Bedingungen erfüllt:*

1. *Wenn a und b Ereignisse innerhalb desselben Prozesses sind und a vor b eintritt, dann gilt $a \rightarrow b$.*
2. *Wenn a das Senden einer Nachricht in einem Prozess ist und b der Empfang dieser Nachricht durch einen anderen Prozess, dann gilt $a \rightarrow b$.*
3. *Wenn $a \rightarrow b$ und $b \rightarrow c$ gelten, dann gilt $a \rightarrow c$.*

Für jedes Ereignis a gilt, unter der Annahme einer kontinuierlich fortlaufenden Zeit, $\neg(a \rightarrow a)$.
Zwei unterschiedliche Ereignisse a und b werden als nebenläufig bezeichnet, wenn $\neg(a \rightarrow b) \wedge \neg(b \rightarrow a)$ gilt.

Beispiel

Abbildung 3.9 veranschaulicht die Bedeutung der obigen Definition. Das Raum-Zeit-Diagramm stellt den Ablauf von Ereignissen in zwei unabhängigen Prozessen dar.

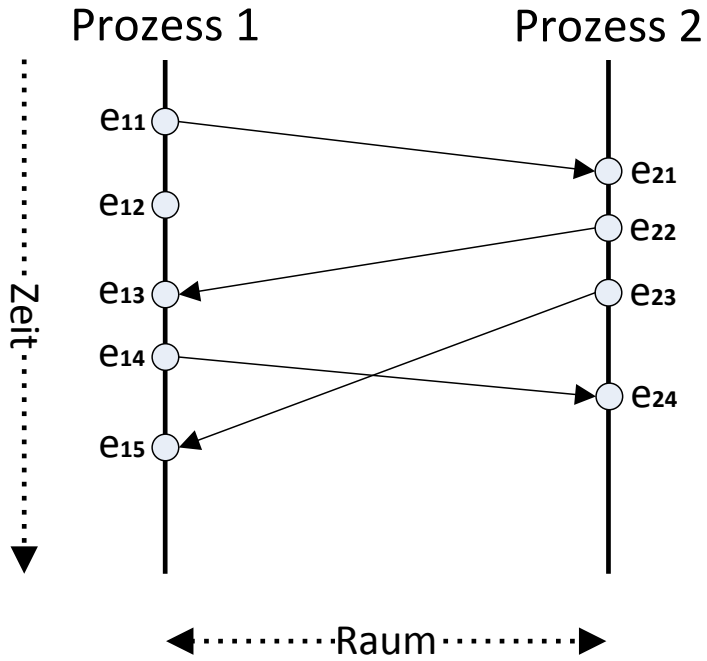


Abbildung 3.9: Einfaches Beispiel zu Lamports Happened-Before-Relation

Die Ereignisse e_{11} und e_{12} finden beide in Prozess 1 statt. Gemäß obiger Definition gilt somit $e_{11} \rightarrow e_{12}$. Mit selbiger Begründung gilt auch $e_{12} \rightarrow e_{14}$. Aus der Transitivitätseigenschaft der Happened-Before-Relation kann gefolgert werden, dass auch $e_{11} \rightarrow e_{14}$ gültig ist. Der Empfang der von Prozess 1 mit e_{11} versandten Nachricht durch Prozess 2 wird durch e_{21} repräsentiert. Diese Beziehung zwischen den beiden Ereignissen resultiert in $e_{11} \rightarrow e_{21}$. Beide Prozesse senden mit e_{14} bzw. e_{23} eine Nachricht an den jeweils anderen. Da weder $e_{14} \rightarrow e_{23}$ noch $e_{23} \rightarrow e_{14}$ gelten, werden die Ereignisse als nebenläufig klassifiziert.

Tatsächlich ist e_{23} jedoch zu einem früheren physischen Zeitpunkt als e_{14} eingetreten. Es ist an diesem Beispiel ersichtlich, dass die Happened-Before-Relation nicht die reale zeitliche Abfolge von Ereignissen erfasst. Vielmehr identifiziert sie die kausalen Beziehungen zwischen verschiedenen Ereignissen. Ereignisse werden also immer genau dann als *nebenläufig* bezeichnet, wenn die Prozesse, in denen sie stattfinden, zum Zeitpunkt des Ereigniseintritts nicht über alle Ereignisse in anderen Prozessen informiert sind.

3.3.1.2 Logische Uhren und Kriterien zur Quantifizierung ihrer Konsistenz

Um die obigen theoretischen Ausführungen praktisch nutzen zu können, ist es erforderlich, das Konzept der logischen Uhren einzuführen. Je nach beabsichtigtem Einsatzzweck, müssen diese unterschiedlichen Konsistenzbedingungen genügen.

Logische Uhren

Unter einer logischen Uhr ist allgemein eine Funktion zu verstehen, die einem Ereignis einen bestimmten Funktionswert zuweist: Jeder Prozess in einem verteilten System muss eine eigene Uhr besitzen, die ausschließlich von Ereignissen beeinflusst werden kann, die im jeweils zugehörigen Prozess auftreten. Man definiert somit für jeden Prozess p_i eine logische Uhr in Form der Funktion C_i, die jedem in p_i eintretenden Ereignis a den Wert $C_i(a)$ zuweist.

Die Gesamtheit aller Uhren in einem verteilten System wird durch die Funktion C beschrieben, die jedem Ereignis b den Wert $C(b)$ zuweist. Dabei gilt $C(b) = C_j(b)$, falls es sich bei b um ein in Prozess p_j eingetretenes Ereignis handelt.

Die Funktion C muss in keiner Verbindung zur tatsächlichen physischen Zeit stehen, sondern erfordert lediglich den Bezug zu den Ereignissen innerhalb des verteilten Systems. Ausgehend von dieser Aussage lassen sich Bedingungen an die logische Uhr formulieren, die Rückschlüsse auf ihre Konsistenz ermöglichen.

Schwache Uhrenbedingung

Eine dieser Bedingungen stellt die sogenannte „schwache Uhrenbedingung“ dar, die vielfach nur als „die Uhrenbedingung“ bezeichnet wird. Sie ist wie folgt definiert:

Definition 3. *Eine logische Uhr C genügt der schwachen Uhrenbedingung, wenn für alle Ereignisse a und b aus der Menge aller Ereignisse ε folgendes erfüllt ist:*
Wenn $a \rightarrow b$ gilt, dann gilt $C(a) < C(b)$.

Die Erfüllung der Uhrenbedingung ist unter Verwendung von „$\rightarrow$“ immer dann gegeben, wenn eine der folgenden Bedingungen zutrifft:

1. Wenn a und b Ereignisse in einem Prozess p_i sind und a vor b eintritt, dann gilt $C_i(a) < C_i(b)$.
2. Wenn a das Senden einer Nachricht in einem Prozess p_i ist und b der Empfang dieser Nachricht durch einen Prozess p_j, dann gilt $C_i(a) < C_j(b)$.

Die Uhrenbedingung ermöglicht es ausschließlich, Aussagen über logische Uhren zu treffen, wenn Informationen über den möglichen kausalen Zusammenhang zwischen den ihnen zugrundeliegenden Ereignissen bekannt sind. Der Vorgang, durch Analyse logischer Zeitpunkte auf die kausalen Zusammenhänge zwischen Ereignissen zu schließen, wird jedoch explizit nicht erfasst.

Umkehrung der Uhrenbedingung

Um aus vorliegenden logischen Zeitpunkten von Ereignissen die realen kausalen Zusammenhänge folgern zu können, ist es notwendig, dass für eine logische Uhr C folgende Bedingung gilt:

Definition 4. *Eine logische Uhr C genügt der Umkehrung der Uhrenbedingung, wenn für alle Ereignisse a und b aus der Menge aller Ereignisse ε folgendes erfüllt ist:*
Wenn $C(a) < C(b)$ gilt, dann gilt $a \rightarrow b$.

3.3.2 Lamport-Uhren

Ein Verfahren, das die Einhaltung der schwachen Uhrenbedingung garantiert, ist die Lamport-Uhr [Lam78]. Dabei handelt es sich um ein einfaches Verfahren, das Ereignissen in verteilten Systemen monoton steigende Elemente aus der Menge der natürlichen Zahlen zuordnet.

Lamport-Uhren lassen sich unter Verwendung einer Menge von Zählern implementieren, wobei jeder Prozess p_i innerhalb des verteilten Systems einen individuellen Zähler C_i besitzt. Das Verfahren besteht aus den folgenden Regeln:

1. Jeweils vor Eintritt eines internen Ereignisses oder dem Senden einer Nachricht in Prozess p_i schreitet die logische Zeit von p_i fort:

   $C_i := C_i + d \ (d \in \mathbb{N})$

2. Jede Nachricht wird vom Sender mit einem Zeitstempel versehen, dessen Wert die für ihn jeweils aktuelle logische Zeit ist.

3. Beim Empfang einer Nachricht, die mit einem Zeitstempel t versehen ist, durch einen Prozess p_i, schreitet dessen logische Zeit fort:

   $C_i := \max(C_i,t) + d \ (d \in \mathbb{N})$

Beispiel

Die Anwendung der Regeln in einem verteilten System wird in Abbildung 3.10 veranschaulicht. Bei Eintritt von e_{11} wird die logische Zeit von Prozess 1 auf den Wert 1 gesetzt. Die an Prozess 2 versandte Nachricht wird mit diesem Zeitstempel versehen. Beim empfangenden Prozess steht der Zähler der logischen Zeit noch auf 0. Bei Eintritt des Empfangsereignisse e_{21} wird er zuerst auf max(0,1) gesetzt und anschließend um 1 inkrementiert. Das Eintreten von e_{22} und e_{23} erhöht den Zähler ebenfalls um jeweils 1. Die an Prozess 1 versandte Nachricht trägt dementsprechend den Zeitstempel 4. Prozess 1 setzt bei e_{12} seinen Zähler zuerst auf max(1,4). Anschließend wird er um 1 inkrementiert und enthält den Wert 5.

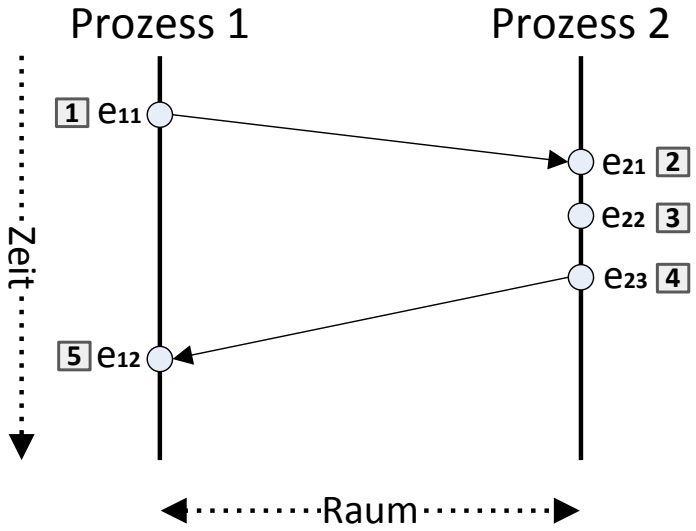


Abbildung 3.10: Beispiel zur Generierung von logischen Zeitstempeln mit Lamport-Uhren

3.3.3 Vektoruhren

Lamport-Uhren genügen zwar der schwachen Uhrenbedingung, nicht jedoch deren Umkehrung. Im Kontext verteilter Systeme ist es für praktische Anwendungen erforderlich, durch Vergleich der logischen Zeitpunkte von Ereignissen eine gesicherte Aussage über deren möglichen kausalen Zusammenhang und somit die Abfolge der betrachteten Ereignisse bestimmen zu können.

Problematisch ist in dieser Hinsicht bei Lamports Verfahren vor allem die Tatsache, dass einzelnen Prozessen der Überblick über die aktuelle logische Zeit der übrigen Prozesse des Gesamtsystems fehlt. Deswegen gibt es keine Möglichkeit Zeitstempel zu vergeben,

die systemweit eindeutig sind. Eine Möglichkeit zur Umgehung dieser Unzulänglichkeiten bieten die 1988 parallel von Schwarz/Mattern [SM94] und Fidge [Fid88] entwickelten Vektoruhren.

Bei diesen besitzt jeder Prozess p_i eines verteilten Systems mit insgesamt n Prozessen eine logische Uhr bestehend aus einem n-elementigen Vektor V von Zählern. Dieser Vektor enthält für jeden Prozess genau einen Eintrag, in dem die logische Zeit des jeweiligen Prozesses gespeichert wird.

Die Regeln des Verfahrens von Lamport werden wie folgt abgeändert, um die Vorteile der geänderten Datenstruktur zu nutzen:

1. Jeweils vor Eintritt eines internen Ereignisses oder dem Senden einer Nachricht in Prozess p_i schreitet die logische Zeit von p_i fort:

   $V_i[i] := V_i[i] + 1$

2. Jede Nachricht wird vom Sender mit einem Zeitstempel versehen, dessen Wert der für ihn jeweils aktuelle Vektor von Zählern ist.

3. Beim Empfang einer Nachricht von Prozess p_j, die mit einem Zeitstempel t versehen ist, durch einen Prozess p_i schreitet dessen logische Zeit fort:

   a) $V_i := \sup(V_i,t)$ wobei $\sup(V_i,t)$ das elementweise Maximum beider Vektoren darstellt, d.h $\sup(V_i,t) = w$ mit $w[i] = \max(V_i[i],t[i])\ \forall i\ (i = 1...n)$

   b) $V_i[i] = V_i[i] + 1$

Vergleicht man nun elementweise zwei nach obigem Vorgehen erzeugte Vektoren v_1 und v_2 von zwei Ereignissen $e_1, e_2 \in \varepsilon$ miteinander, so ergeben sich die folgenden vier Möglichkeiten für die Beziehung der beiden Vektoren:

1. $v_1 = v_2$ genau dann, wenn $v_1[i] = v_2[i]\ \forall i\ (i = 1...n)$
2. $v_1 \leq v_2$ genau dann, wenn $v_1[i] \leq v_2[i]\ \forall i\ (i = 1...n)$
3. $v_1 < v_2$ genau dann, wenn $v_1[i] \leq v_2[i]$ und $\neg(v_1 = v_2)$
4. $v_1 \mid\mid v_2$ genau dann, wenn $\neg(v_1 < v_2)$ und $\neg(v_2 < v_1)$

Dabei charakterisiert $v_1 \mid\mid v_2$ den Fall, dass die beiden verglichenen Vektoruhren nebenläufig sind, also in keinem kausalen Zusammenhang zueinander stehen.

Mit Hilfe von Definition 2 (Happened-Before-Relation) lässt sich nun das Verhältnis zwischen den Vektoruhren auf die ihnen zugeschriebenen Ereignisse übertragen. Demnach können über die Ereignisse e_1 und e_2 nach [SM94] die folgenden Aussagen getroffen werden:

1. $e_1 = e_2$ genau dann, wenn $v_1 = v_2$
2. $e_1 < e_2$ genau dann, wenn $v_1 < v_2$
3. $e_1 \mid\mid e_2$ genau dann, wenn $v_1 \mid\mid v_2$

Schwarz und Mattern zeigen in ihrer Arbeit formal, dass Vektoruhren die umgekehrte Uhrenbedingung erfüllen, da sie die Bestimmung des kausalen Zusammenhangs zwischen Ereignissen durch Betrachtung der zugehörigen logischen Zeitstempel ermöglichen.

Beispiel

Die Anwendung der Regeln zur Erzeugung von Zeitstempeln in Form von Vektoruhren und die Klassifikation der kausalen Beziehung zwischen verschiedenen Ereignissen werden in Abbildung 3.11 illustriert.

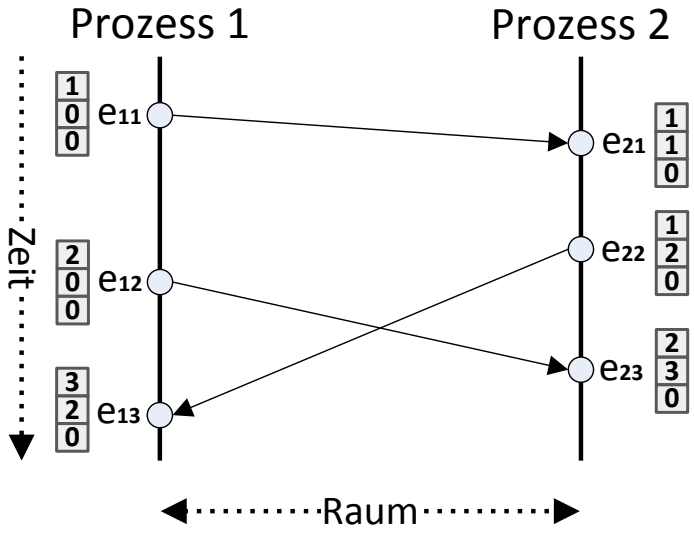


Abbildung 3.11: Beispiel zur Generierung von logischen Zeitstempeln mit Vektoruhren

Bei Eintritt von e_{11} setzt Prozess 1 seinen Eintrag im Vektor auf 1 und versendet den Vektor zusammen mit der eigentlichen Nachricht an Prozess 2. Beim Empfang mit e_{21} setzt dieser seinen lokalen Vektor auf das Maximum des lokalen und des ankommenden Vektors. Anschließend erhöht er seinen eigenen Eintrag um 1. Für e_{11} und e_{21} kann nun durch einen Vergleich der jeweiligen Zeitstempel gefolgert werden, dass $e_{11} < e_{21}$ gilt.

Beide Prozesse senden mit e_{12} und e_{22} eine Nachricht an den jeweils anderen. Ein Vergleich der zugehörigen Zeitstempel ergibt, dass $e_{12} || e_{22}$ gilt. Beide Prozesse bilden das elementweise Maximum des lokalen und des ankommenden Vektors und erhöhen abschließend ihren eigenen Zähler. Für e_{13} und e_{23} gilt danach ebenfalls, dass beide Ereignisse nebenläufig sind.

3.3.4 Versionsvektoren

Ein mögliches Einsatzgebiet für Mechanismen zur Uhrensynchronisation, die auf den Prinzipien der logischen Zeit basieren, ist die Verwaltung der Modifikationen von Dateien, von denen Replikate an mehreren Orten innerhalb eines verteilten Systems gespeichert werden. Hierbei ist es notwendig, parallele Änderungen durch verschiedene Prozesse sicher zu erkennen, um Inkonsistenzen zu vermeiden.

Ein möglicher Ansatz zur Bewältigung dieser Aufgabe wird von Parker et al. in *Detection of Mutual Inconsistency in Distributed Systems* [PJPR+83] in Form der Versionsvektoren präsentiert. Dieses, den Vektoruhren sehr ähnliche, Verfahren dient vor allem in Situationen, in denen eine starke Partitionierung des verwendeten Netzwerks vorliegt, dem Erkennen von konfligierenden Änderungsoperationen auf gemeinsamen Dateien.

Regeln zur Erzeugung und Verwaltung von Versionsvektoren

Der zu einer bei n verschiedenen Teilnehmern gespeicherten Datei f gehörende und bei Netzknoten S_i gespeicherte Versionsvektor v_i ist eine Sequenz aus n Paaren der Form (S,v). Die Komponenten S_j und v_j von $v_i[j]$ bezeichnen den Namen des j-ten Netzknotens und die Anzahl von Aktualisierungen, die von diesem Netzknoten an f durchgeführt wurden. So charakterisiert der Versionsvektor *<(A,2),(B,1),(C,0)>* etwa eine Datei, die von Netzknoten A zweimal, von Netzknoten B einmal und von Netzknoten C bisher noch nicht modifiziert wurde.

Versionsvektoren werden auf analoge Art und Weise generiert wie Vektoruhren. Zu beachten ist lediglich neben den neuen Regeln, die spezifisch bei der Verwaltung von Dateien sind, dass die logische Zeit jeweils nur Ereignissen, die eine Datei modifizieren und bei der Konfliktbehebung fortschreitet, nicht aber bei anderen Ereignissen.

1. Bei jeder Veränderung (Aktualisierung, Umbenennung, Löschung) von f durch einen Netzknoten S_i schreitet dessen logische Zeit fort:

   $v_i[i] := v_i[i] + 1$

2. Bei Empfang eines Versionsvektors t durch S_i schreitet dessen logische Zeit fort, falls die Behebung eines Konfliktes erforderlich ist:

   a) $v_i := \sup(v_f,t)$ wobei $\sup(v_f,t)$ das elementweise Maximum beider Vektoren darstellt, d.h $\sup(v_f,t) = w$ mit $w[h] = \max(v_f[h],t[h])\ \forall h\ (h = 1...n)$

   b) $v_i[i] := v_i[i] + 1$, falls S_i die Konfliktbehebung initiiert hat.

3. Werden Replikate von f bei neu hinzugekommenen Netzknoten erzeugt oder Repliken aus dem System entfernt, so kann die Größe des Versionsvektors dynamisch angepasst werden, indem entsprechende Einträge hinzugefügt oder entfernt werden.

Informationsaustausch zwischen einzelnen Netzknoten

Versionsvektoren eignen sich für die Synchronisation in partitionierten Netzwerken. Unter Partitionierung eines Netzwerks versteht man eine Situation, in der ein Netzwerk in verschiedene logisch getrennte Komponenten, Partitionen genannt, zerteilt ist. Zwischen den einzelnen Komponenten ist kein Informationsaustausch möglich.

Die Ursache für eine solche Teilung kann unter anderem der Ausfall einzelner Netzknoten oder der Datenverbindungen zwischen diesen sein. Die Ausdehnung einzelner Partitionen kann dynamisch variieren, indem sie weiter in Subpartitionen aufgesplittet werden oder mit anderen Partitionen vereinigt werden. Dadurch ist es möglich, dass Änderungsoperationen an alle Netzknoten propagiert werden, ohne dass das Netzwerk zu einem bestimmten Zeitpunkt nur aus einer einzigen Partition bestehen muss.

Alle Netzknoten, die sich in derselben Partition befinden, können sich durch engmaschige Kommunikation über den gültigen Inhalt einer gemeinsam genutzten Datei einigen. Dabei werden geänderte Dateien bei jeder durchgeführten Modifikation mit einem logischen Zeitstempel versehen. Besteht zu einem bestimmten Zeitpunkt eine aktive Verbindung zu einer weiteren Partition des Netzwerkes, so ist es notwendig, durch Vergleich der Zeitstempel aus beiden Partitionen zu bestimmen, ob beide Änderungshistorien miteinander kompatibel sind. Ist dies nicht der Fall, so liegt ein Änderungskonflikt vor, der entsprechend behandelt werden muss.

Erkennung paralleler konfligierender Modifikationen

Versionsvektoren können auf dieselbe Art und Weise miteinander verglichen werden, wie Vektoruhren (vgl. Abschnitt 3.3.3). Da auch sie der umgekehrten Uhrenbedingung genügen, lassen sich mittels Anwendung von Definition 2 die Verhältnisse zwischen Zeitstempeln analog zu den Vektoruhren auf die kausalen Zusammenhänge zwischen Ereignissen übertragen.

Es liegt genau dann und nur dann ein nebenläufiger Änderungskonflikt vor, wenn die Zeitstempel zweier Partitionen beim elementweisen Vergleich als nebenläufig klassifiziert werden. In diesen Fällen muss ein applikationsspezifischer Vorgang zur Auflösung des Konflikts angestoßen werden, damit das verteilte System weiterhin in einem konsistenten Zustand ausgeführt werden kann.

Beispiel

Abschließend wird anhand eines verteilten Systems mit drei Netzknoten in Abbildung 3.12 der Einsatz von Versionsvektoren zur Synchronisation von Änderungen einer bestimmten, bei allen Netzknoten repliziert gespeicherten Datei veranschaulicht. Die Versionsvektoren reflektieren die jeweils durchgeführten Modifikationen der betrachteten Datei.

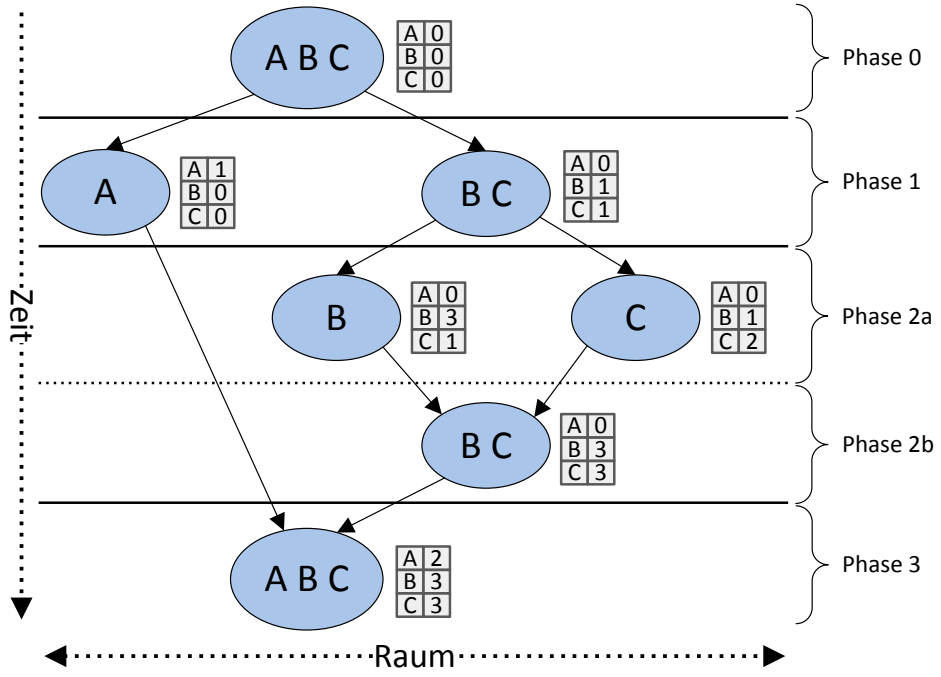


Abbildung 3.12: Partitionsgraph und zugehörige Versionsvektoren

Zu Beginn des Beispielszenarios sind in Phase 0 alle Netzknoten untereinander verbunden. In Phase 1 zerfällt das Netzwerk in zwei Partitionen: In einer ist Netzknoten A enthalten, in der anderen sind es B und C. Unabhängig von den übrigen Netzknoten modifiziert A die Datei ein einziges Mal. In der verbleibenden Partition wird die Datei je einmal von B und C modifiziert.

Diese Partition zerfällt anschließend in Phase 2a erneut und jetzt sind auch B und C voneinander isoliert. Netzknoten B bearbeitet die Datei zweimal Mal und C genau einmal. B und C sind einige Zeit später in Phase 2b wieder miteinander verbunden. Durch elementweisen Vergleich ihrer Versionsvektoren wird festgestellt, dass nebenläufige Änderungen vorliegen, die entsprechend behandelt werden müssen. C stößt daraufhin eine Auflösung des Konfliktes an. Innerhalb der Partition von B und C wird das elementweise Maximum der lokalen Versionsvektoren gebildet. Zusätzlich wird der Eintrag von Netzknoten C um 1 inkrementiert, da C die Konfliktbehandlung initiiert hat.

Analog wird verfahren, wenn in Phase 3 die Verbindung zu A wiederhergestellt ist. Der durch die nebenläufige Modifikation der Datei durch A entstandene Konflikt wird aufgelöst und die lokalen Versionsvektoren werden entsprechend der geschilderten Regeln angepasst.

3.4 Zusammenfassung

Zusammenfassend betrachtet bietet α-Flow einen neuartigen Ansatz zur Koordinierung interdisziplinärer, institutionsübergreifender Prozesse. Die einzelnen Elemente des Domänenmodells ermöglichen eine flexible Modellierung der abzubildenden Prozesse in Form von elektronischen Dokumenten. Durch die aktiven Eigenschaften des α-Doc wird für die beteiligten Individuen und Institutionen ein deutlicher Mehrwert gegenüber passiven papierbasierten Dokumenten und existierenden Softwarelösungen zur Systemintegration geschaffen.

Verteilte Systeme ermöglichen über Overlay-Netzwerke den effizienten Nachrichtenaustausch zwischen entfernten Rechnern. Dabei wird zu jedem Zeitpunkt die Integrität und Sicherheit des Datentransfers garantiert. Durch ihre inhärente Flexibilität und Skalierbarkeit sind verteilte Systeme in der Lage, dynamische Gruppen beteiligter Benutzer zu verwalten.

Mit Hilfe der Konzepte logischer Zeitstempel können Ereignisse in einem verteilten System in eine eindeutige Ordnung gebracht bzw. konfliktbehaftete parallele Änderungen erkannt werden. Die vorgestellten Grundlagen und Verfahren vektorbasierter logischer Zeitstempel fließen in das Synchronisationskonzept für α-Flow ein, das im Rahmen der vorliegenden Arbeit entwickelt wurde.

Die erläuterten Aspekte bilden das Fundament für den Lösungsansatz und die technischen Grundlagen der angefertigten prototypischen Implementierung. Auf dieses Fundament wird in den nachfolgenden Kapiteln Bezug genommen, um die eingesetzten Technologien und den entwickelten fachlichen Lösungsansatz detailliert zu beschreiben.

4 Technische Grundlagen

Dieses Kapitel beschreibt die für die spätere Umsetzung relevanten technischen Grundlagen. Vorangegangene Forschungen im Rahmen von α-Flow haben sich bereits mit verschiedenen Transportprotokollen zum Informationsaustausch beschäftigt [Ram09], [Peh10]. Dort wurden das Extensible Messaging and Presence Protocol (XMPP) [SA11a], [SA11b] und das Peer-to-Peer-Protokoll Juxtapose (JXTA)[1] untersucht.

Eines der Ziele von α-Flow ist jedoch die Kompatibilität zur zukünftigen Infrastruktur der elektronischen Gesundheitskarte (eGK) im deutschen Gesundheitswesen[2]. Die Kommunikation innerhalb des eGK-Ökosystems ähnelt stark dem Austausch verschlüsselter E-Mail-Nachrichten, die bis zum expliziten Abruf durch den Benutzer serverseitig zwischengespeichert werden.

Aus diesem Grund soll in der vorliegenden Arbeit ein E-Mail-basierter Lösungsansatz verfolgt werden, um die spätere Kommunikation bei Verwendung der eGK zu simulieren. Als vorteilhaft erweist sich in diesem Zusammenhang die Tatsache, dass die für den Austausch von E-Mails notwendige Infrastruktur nahezu flächendeckend vorhanden ist und somit die Verfügbarkeit eines E-Mail-Kontos seitens der beteiligten Akteure, im Gegensatz zu einem XMPP-Account, vorausgesetzt werden darf.

Als Ausgangspunkt der Betrachtungen wird die Struktur und Inhaltsrepräsentationen von E-Mails gemäß den Prinzipien des Internet Mail Modells erläutert. Die heutzutage weit verbreiteten Transportprotokolle Simple Mail Transfer Protocol und Internet Message Access Protocol, die Standards für den Versand und Empfang von E-Mails definieren, werden in eigenen Abschnitten vorgestellt.

Für die durchgeführte Implementierung im Rahmen des α-Flow-Projekts wird auf das JavaMail Application Programming Interface zurückgegriffen. Grundlagen dieses Frameworks zur E-mail-basierten Kommunikation werden in einem separaten Abschnitt präsentiert. Abschließend folgt eine Betrachtung der Möglichkeiten zur sicheren Authen-

1 Für weitere Informationen siehe `http://jxta.kenai.com`.
2 Für weiterführende Informationen zur eGK siehe `http://www.bundesgesundheitsministerium.de`.

tisierung von Benutzern und der Einhaltung des Datenschutzes unter Verwendung der Java Platform.

4.1 Struktur von E-Mail-Nachrichten

Bei der Klassifikation von E-Mails muss zwischen verschiedenen Typen von Nachrichten unterschieden werden. Zum einen ist es möglich rein textuelle Nachrichten zu versenden. Daneben existiert jedoch auch die Möglichkeit, Informationen, wie beispielsweise Mediendaten, in binärer Form zu versenden. Neben dem zu übertragenden Nutzinhalt kann eine Nachricht Metainformationen enthalten, die den Inhalt und die Semantik der Nachricht charakterisieren.

4.1.1 Einfache textuelle Nachrichten

Die nachfolgenden Ausführungen beschreiben den Aufbau rein textbasierter E-Mail-Nachrichten in Konformität zu dem auf [Cro82] und [Res01] basierenden Standard des Internet Message Format [Res08]. Auf einzelne technische Details und ihre Umsetzung wird an dieser Stelle nicht eingegangen. Der Leser sei hierzu auf die umfangreiche Dokumentation der zugrundeliegenden Standards verwiesen.

In ihrer ursprünglichen Form sind E-Mails ausschließlich zum Austausch von nach dem American Standard Code for Information Interchange (ASCII) kodierten Texten [Cer69] konzipiert. Der eigentliche Nutzinhalt einer E-Mail, die vom Absender eingegebene Nachricht, wird als endliche Abfolge von Zeilen aufgefasst. Diese Zeilen werden mittels Wagenrücklauf und anschließendem Zeilenvorschub terminiert und unterliegen einer Längenbeschränkung von je maximal 998 Zeichen.

Darüber hinaus enthält jede E-Mail vor Beginn des Nutzinhalts in der sogenannten Header Section Metainformationen über die jeweilige Nachricht, die in einzelnen Zeilen, den Header Fields, hinterlegt sind. Ein einzelnes Header Field setzt sich zusammen aus einer Bezeichnung und einer Beschreibung des Inhalts. Beide Bestandteile werden durch einen Doppelpunkt getrennt. Die im vorherigen Abschnitt beschriebenen Einschränkungen bezüglich Zeichenkodierung und Zeilenlänge gelten in gleicher Weise für Header Fields.

In Konformität zu den genannten Standards muss eine E-Mail mindestens nachfolgende Header Fields enthalten:

- `Date`: Datum der Übergabe an das E-Mail-System durch den Absender
- `From`: Liste von Absender-E-Mail-Adressen

Alle weiteren Metadaten zu Empfänger, Betreff, Schlüsselwörtern des Inhalts, oder benutzerdefinierten Informationen sind rein optional und müssen nicht in jeder E-Mail enthalten sein.

4.1.2 Multipurpose Internet Mail Extensions (MIME)

Um eine flexiblere Nutzung von E-Mails für den Informationsaustausch zu realisieren, wurde das Konzept der Multipurpose Internet Mail Extensions (MIME) entwickelt (spezifiziert in [FB96b], [FB96c], [Moo96], [FKP96] und [FB96a]). Bei MIME handelt es sich um einen Industriestandard, der E-Mail-Nachrichten in beliebiger Kodierung, binäre Dateianhänge, mehrteilige Nachrichten sowie Header Fields in beliebiger Kodierung erlaubt.

Header Fields bei MIME-Nachrichten

Um das gewünschte Maß an Flexibilität in Hinblick auf den Inhalt von E-Mail-Nachrichten zu erreichen, müssen zusätzliche Metainformationen eingebettet werden, die dem Empfänger das Dekodieren und Interpretieren von Nachrichten ermöglichen. Diese Metainformationen werden, konform zum Internet Message Format [Res08], in der Header Section der jeweiligen E-Mail gespeichert. Dabei werden zusätzlich zu den im vorherigen Abschnitt angesprochenen Feldern folgende Header Fields definiert:

- `MIME-Version`: Version des MIME-Protokolls, mit der die Nachricht erstellt wurde.
- `Content-Type`: Typ der zu transportierenden Daten
- `Content-Transfer-Encoding`: Senderseitig verwendete Kodierung der zu transportierenden Daten
- `Content-Disposition`: Beschreibung von Dateiname, Erstellungsdatum und Darstellungsform eines Nachrichtenteils
- `Content-ID`: Eindeutiger Schlüssel zur Identifizierung eines Nachrichtenteils
- `Content-Description`: Textuelle Beschreibung eines Nachrichtenteils

Über die vorgegebenen Content-Typen `text, image, audio, video, application, multipart, message` und ihre jeweiligen Subtypen hinaus können auch benutzerspezifische Typen eingeführt werden.

4.1.2.1 MIME-Multipart-Nachrichten

Eine zentrale Eigenschaft von MIME ist die Unterstützung von Nachrichten, die aus mehreren, insbesondere hierarchisch verschachtelten, Nachrichtenteilen zusammengesetzt werden. Diese sogenannten Multipart-Nachrichten können mehrere verschiedene Typen von Informationen enthalten. Eine einzelne MIME-konforme E-Mail kann beispielsweise einen einleitenden Text, darauf folgend mehrere Bilder und zum Abschluss eine Audiodatei enthalten.

Um einzelne Nachrichtenteile einer Nachricht bei der Syntaxanalyse zuverlässig voneinander unterscheiden zu können, ist das Festlegen des `boundary`-Attributs erforderlich, das die zur eindeutigen Trennung verwendete Zeichenkette enthält.

In der Spezifikation [FB96b] sind die nachfolgenden Typen von Multipart-Nachrichten erfasst, die jeweils unterschiedliche Rückschlüsse auf die Semantik einer Nachricht erlauben:

- `multipart/mixed`: Nachrichtenteile sind unabhängig, sollen aber in festgelegter Reihenfolge gebündelt werden
- `multipart/parallel`: Nachrichtenteile sind unabhängig und sollen ohne festgelegte Reihenfolge gebündelt werden
- `multipart/alternative`: Nachrichtenteile repräsentieren alle die gleiche Information, jedoch in unterschiedlichen Darstellungsformen
- `multipart/digest`: Nachrichtenteile sind eigenständige Nachrichten, die gebündelt versendet werden

Für spezielle Einsatzzwecke existieren weitere Arbeiten, die in MIME-Nachrichten unter anderem die Verwendung elektronischer Signaturen oder den Versand verschlüsselter Nachrichten unterstützen [GMCF95].

4.2 Simple Mail Transfer Protocol (SMTP)

Das Simple Mail Transfer Protocol (SMTP) [Kle08] definiert ein zeilenorientiertes Protokoll zum Versand von E-Mail-Nachrichten in Computernetzen. Unter Verwendung von SMTP lässt sich die Kommunikation zwischen dem E-Mail-Clientprogramm des sendenden Benutzers, dem sogenannten Mail User Agent (MUA), und dem für den Empfang und Versand von Nachrichten verantwortlichen Softwaremodul des Mail Transfer Agent (MTA) auf dem SMTP-Server, wie in Abbildung 4.1 dargestellt, formalisieren.

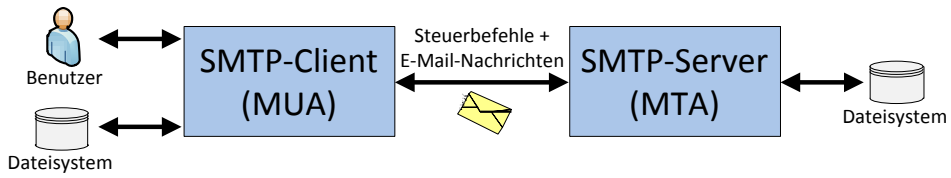


Abbildung 4.1: SMTP Kommunikation zwischen Client und Server

Bei der Übermittlung von Nachrichten kommunizieren Client und Server über eine gemeinsame aktive Datenverbindung. Über diese Verbindung wird von Seiten des Benutzers die zu versendende Nachricht übertragen. Die zu übertragenden E-Mails bestehen aus Inhalten des lokalen Dateisystems des Benutzers und müssen dem in Abschnitt 4.1 beschriebenen Format entsprechen.

Als zusätzliche Informationen muss der Client die Absenderadresse, sowie die Adressinformationen des bzw. der Empfänger an den Server übergeben, um eine korrekte Zustellung der Nachricht zu ermöglichen. Diese Informationen werden, in Anlehnung an die Terminologie herkömmlicher Briefe, als Umschlag der zu versendenden Nachricht bezeichnet.

Nach erfolgreicher Beendigung der Übertragung wird die Nachricht im Dateisystem des SMTP-Servers abgelegt. Von dort aus wird sie an den MTA des nächsten SMTP-Servers auf dem Weg zum vorgesehenen Empfänger der Nachricht übermittelt. Auf diese Weise kann eine Nachricht entlang einer Kette von verschiedenen MTAs sequenziell weitergereicht werden, bevor schließlich der MTA erreicht wird, der mit Hilfe seines lokalen Mail Delivery Agent (MDA) die Nachricht im Postfach des Empfängers ablegen kann (Abbildung 4.2). Von dort aus kann die Nachricht von ihrem Empfänger mit Hilfe seines MUA abgerufen werden.

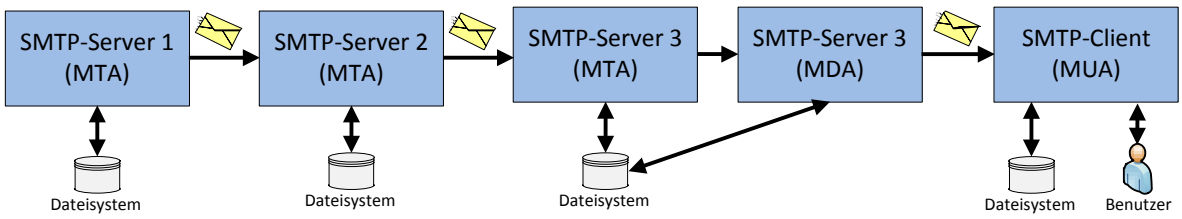


Abbildung 4.2: Weiterreichen einer Nachricht an den Ziel-SMTP-Server

Kritische Eigenschaft: Zuverlässige Nachrichtenübermittlung

SMTP bietet keine zuverlässige Übermittlung von Nachrichten, da außer der Persistierung von Nachrichten keine der in Abschnitt 3.2.3.2 formulierten Anforderungen an zuverlässige Kommunikationsprotokolle erfüllt werden.

Da der SMTP-Client des sendenden Benutzers nicht direkt mit dem SMTP-Server des vorgesehenen Empfängers kommuniziert, erhält er von diesem standardmäßig keine direkte Rückmeldung, ob eine Nachricht beim Empfänger angekommen oder verloren gegangen ist. Um die Zuverlässigkeit des Nachrichtentransfers zu verbessern, kann auf das Prinzip der Delivery Status Notification (DSN) [Moo03] zurückgegriffen werden. Diese Erweiterung von SMTP ermöglicht den Versand von Empfangsbestätigungen unter der Bedingung, dass alle Mail-Server auf dem Weg vom Absender zum Empfänger DSN unterstützen. Problematisch bleibt neben der nicht flächendeckenden Verbreitung von DSN die Tatsache, dass es sich bei DSN-Nachrichten ebenfalls um E-Mails handelt, welche wiederum auf dem Transportweg verloren gehen können.

Ist einer der Mail-Server auf dem Weg zum Empfänger temporär nicht erreichbar, so kann die zu versendende Nachricht nicht korrekt zugestellt werden. Wird ein derartiger Fehler festgestellt, so müssen unter Umständen alle MTAs in umgekehrter Reihenfolge der vorherigen Nachrichtenweitergabe darüber informiert werden, um den ursprünglichen Absender benachrichtigen zu können. Es obliegt jedoch jedem einzelnen Server auf diesem Weg, ob solche Meldungen weitergegeben werden oder nicht, weshalb ein Ausbleiben einer solchen Benachrichtigung keine zuverlässige Aussage über den Versandstatus der E-Mail erlaubt.

Weiterhin ist zu beachten, dass auch Nachrichten, die korrekt vom Server des Empfängers entgegengenommen und bestätigt worden sind, nicht in jedem Fall dem Empfänger zugänglich gemacht werden. So können beispielsweise auf dem Server installierte Spam- bzw. Virenschutzmechanismen für ein nachträgliches Verwerfen der Nachricht verantwortlich sein, ohne dass der Absender darüber informiert wird.

Darüber hinaus bietet SMTP selbst keine Unterstützung für die Verwaltung nachrichtenspezifischer Sequenznummern und Duplikaterkennung. Alle empfangenen Nachrichten werden gemäß ihrer Eingangsreihenfolge auf dem Server verarbeitet.

Insgesamt betrachtet ist das Problem der Zuverlässigkeit bei SMTP fundamental und darf nicht vernachlässigt werden. Unter diesen Voraussetzungen erfolgt die Verwendung von E-Mails im Kontext von α-Flow ausschließlich als vorläufige Alternative zu den zukünftigen zuverlässigen Transportkonzepten der elektronischen Gesundheitskarte.

4.3 Internet Message Access Protocol (IMAP)

Das seit 1986 entwickelte Internet Message Access Protocol (IMAP) ist ein zeilenorientiertes Internetprotokoll zum Zugriff auf E-Mail-Postfächer und zur Manipulation von E-Mails in den jeweiligen Postfächern [Cri03]. Dieser Abschnitt stellt die wichtigsten Aspekte und Konzepte des Protokolls dar. Dazu gehören, neben der serverseitigen Nachrichtenorganisation, der individuelle Abruf einzelner Teile einer Nachricht und die Möglichkeit zur Realisierung ereignisbasierter Echtzeitbenachrichtigungen.

Auf eine Darstellung der Syntax der vom Client an den Server gesendeten Anfragen sowie der zugehörigen Antwortnachrichten wird aus Gründen der Übersichtlichkeit in dieser Arbeit verzichtet. Der Leser sei an dieser Stelle auf die umfangreichen Beschreibung in den Standardisierungsdokumenten [Cri03] und auf die Darstellungen in [MM01] verwiesen.

4.3.1 Organisation von Nachrichten

IMAP bietet vielfältige Möglichkeiten, E-Mails direkt auf dem Server zu organisieren und auch große Mengen von Nachrichten mehreren Benutzern zeitgleich zur Verfügung zu stellen. Abbildung 4.3 gibt einen Überblick über die organisatorische Struktur eines IMAP-Accounts, dessen einzelne Elemente im Folgenden genauer beschrieben werden.

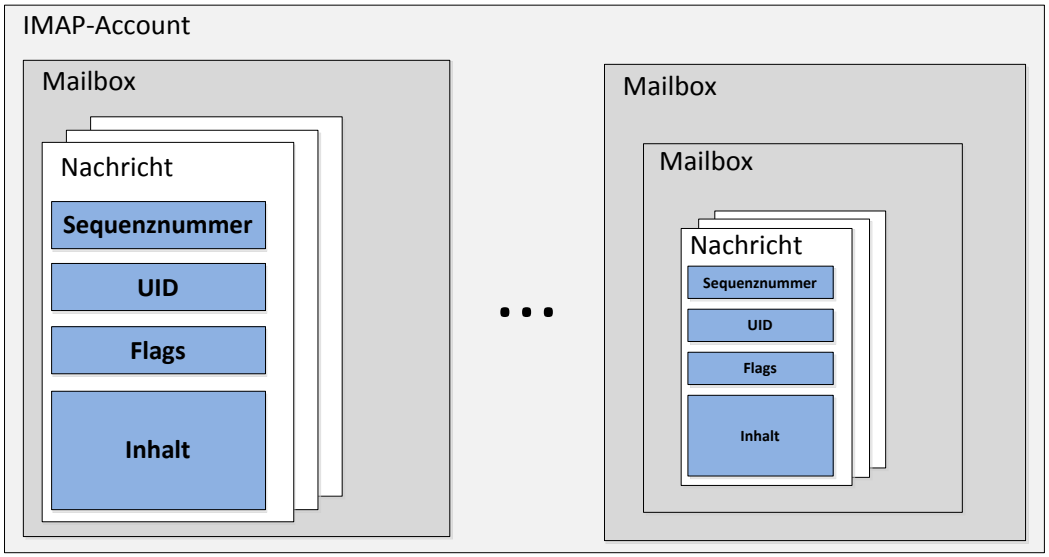


Abbildung 4.3: Organisation von Nachrichten mittels IMAP

Hierarchie von Mailboxen

Jeder IMAP-Account kann mehrere Mailboxen enthalten. Standardmäßig ist eine Mailbox mit der Bezeichnung „INBOX“ zu jeder Zeit auf dem Server vorhanden. Alle eingehenden Nachrichten werden in dieser Mailbox abgelegt.

Mailboxen können neben den eigentlichen Nachrichten auch noch andere Mailboxen enthalten. So entsteht eine hierarchische Struktur, die aus Benutzersicht den Verzeichnissen in einem Dateisystem entspricht und zur sortierten Ablage von Nachrichten benutzt werden kann.

Nachrichtenspezifische Attribute

Für den komfortablen Umgang mit großen Mengen von E-Mails ist es erforderlich, den einzelnen Nachrichten, über den eigentlichen Nutzinhalt hinaus, Attribute zuzuweisen, die ihre Organisation vereinfachen.

Sequenznummer: Eines dieser nachrichtenspezifischen Attribute ist die Sequenznummer. Sie beschreibt die relative Position einer Nachricht in der Mailbox. Eine Sequenznummer ist in keiner Weise persistent. Es gibt demnach keine Garantie dafür, unter derselben Sequenznummer nach einem erneuten Verbindungsaufbau die erwartete Nachricht vorzufinden. Aber auch während einer laufenden Sitzung kann sich die Sequenznummer einer Nachricht ändern. Das kann beispielsweise beobachtet werden, wenn neue Nachrichten in der Mailbox eintreffen oder ein zeitgleich eingeloggter Benutzer die Inhalte der Mailbox manipuliert.

Unique Identifier (UID): Um Nachrichten zu jedem Zeitpunkt eindeutig identifizieren zu können, existiert der Unique Identifier. Dabei handelt es sich um einen 32-bit Identifikator, der jeweils nur genau einer Nachricht in der Mailbox zugewiesen wird. Die Werte dieser Identifikatoren sind monoton steigend, aber implementierungsabhängig nicht zwingend kontinuierlich. Unique Identifiers werden serverseitig persistiert und stehen auch nach einem erneuten Verbindungsaufbau wieder zur Verfügung.

Da es durch serverseitige Datenverluste trotz allem dazu kommen kann, dass die bis zu einem bestimmten Zeitpunkt vergebenen UIDs ungültig werden, gibt es den Mechanismus der sogenannten UIDValidity. Bei dieser 32-bit Zahl handelt es sich um ein Attribut der Mailbox, welche es erlaubt zu überprüfen, ob die UIDs seit dem letzten Zugriff invalidiert wurden. Ist die vom Client empfangene UIDValidity größer als der ihm von

der vorhergehenden Sitzung bekannte Wert, so kann der Client davon ausgehen, dass allen Nachrichten neue UIDs zugeteilt worden sind.

Flags: Zur weiterführenden Kennzeichnung von Nachrichten existieren die IMAP Flags. Bei Flags handelt es sich um mit einer Nachricht assoziierte Token in Form einer Zeichenkette. Sie erlauben es, Nachrichten mit einer Art Etikett zu versehen, um dadurch Status bzw. Inhalt detailliert zu charakterisieren. Verschiedene Flags sind in [Cri03] definiert und auf allen standardkonformen Serverimplementierungen verfügbar. Alle standardisierten Flags sind persistent, können also auch über das Ende einer Sitzung hinaus gespeichert werden und dienen somit zur permanenten Klassifizierung von Nachrichten. Viele Serverimplementierungen erlauben dem Benutzer außerdem das Anlegen eigener nicht-standardisierter Flags, die in den meisten Fällen auch dauerhaft serverseitig persistiert werden können.

Nutzinhalt und weitere nachrichtenspezifische Attribute: Neben dem eigentlichen vom Absender verschickten Nutzinhalt werden zusätzlich das Eingangsdatum der Nachricht auf dem Server und die Nachrichtengröße auf dem Server abgelegt. Zudem werden komprimierte Darstellungen des Headers und der enthaltenen MIME-Informationen parallel zu den eigentlichen Nachrichten gespeichert.

4.3.2 Transfer einzelner Nachrichtenteile

Im Gegensatz zu anderen Protokollen für den Abruf von E-Mail-Nachrichten, wie beispielsweise dem Post Office Protocol [MR96], müssen bei IMAP Nachrichten nicht vollständig vom Server auf den Client heruntergeladen werden, um ihre Inhalte anzuzeigen. IMAP erlaubt es, die Header einer Nachricht getrennt vom eigentlichen Nutzinhalt abzurufen. Darüber hinaus ist es bei MIME-Nachrichten möglich, jeden MIME-Part einzeln vom Server anzufragen und zum Client zu transferieren.

Auf diese Weise kann das benötigte Übertragungsvolumen reduziert werden, da nur diejenigen Teile einer Nachricht heruntergeladen werden, die für den Benutzer tatsächlich relevant sind. So werden beispielsweise große Dateianhänge nur dann übertragen, wenn diese Übertragung explizit angefordert worden ist. Die restliche Nachricht kann jedoch auch schon zuvor gelesen und bearbeitet werden.

4.3.3 Ereignisbasierte Echtzeitbenachrichtigungen mit IMAP IDLE

Um jederzeit vom Server über den Eingang neuer Nachrichten und alle weiteren Änderungen innerhalb einer Mailbox informiert zu werden, gibt es für den Client die Möglichkeit, das IMAP IDLE Kommando zu verwenden. Dieser Befehl ist eine in [Lei97] definierte Erweiterung von IMAP, die den Empfang von ereignisbasierten Echtzeitbenachrichtigungen ermöglicht. Grundlage dafür bildet das in Abschnitt 3.2.3.3 beschriebene Push-Modell, bei dem der Server alle interessierten Clients über den Eintritt bestimmter Ereignisse informiert. Damit wird verhindert, dass die Clients in kurzen Intervallen beim Server nachfragen müssen, ob neue Ereignisse vorliegen.

Der Ablauf der Kommunikation zwischen Server und Client bei Verwendung von IMAP IDLE ist in Abbildung 4.4 zu sehen. Durch Senden des Befehls `IDLE` an den Server

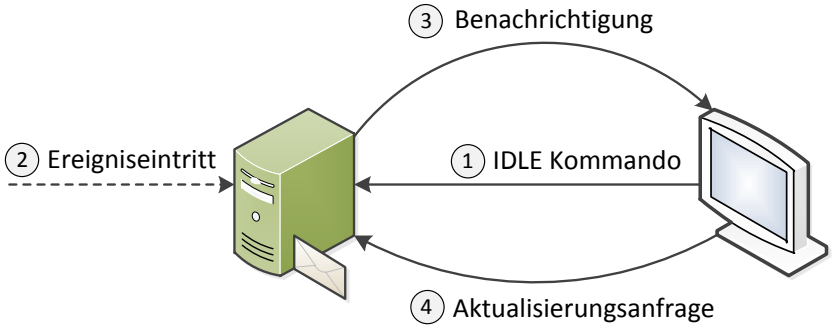


Abbildung 4.4: Ereignisbasierte Echtzeitbenachrichtigungen mit IMAP IDLE

signalisiert der Client, dass er über das Eintreten von Ereignissen durch den Server in Echtzeit informiert werden will. Tritt nun ein neues Ereignis ein, z.B. der Empfang einer neuen Nachricht, so wird der Client vom Server darüber informiert.

Die Antwort des Servers enthält ausschließlich die Benachrichtigung über ein neues Ereignis, nicht jedoch die von diesem Ereignis modifizierten Informationen. In einem letzten Schritt fragt deshalb der Client beim Server nach eben diesen geänderten Inhalten, um seinen lokalen Status zu aktualisieren.

Das IMAP IDLE Kommando wird in α-Flow dazu benutzt, die einzelnen Prozessteilnehmer über die Modifikation von Prozessartefakten und weitere Änderungen in Echtzeit zu informieren. Weitere Ausführungen dazu finden sich in Abschnitt 10.1.5.2.

4.4 JavaMail Application Programming Interface

Beim JavaMail Application Programming Interface (JavaMail API) handelt es sich um eine Programmierschnittstelle für den Abruf und Versand von E-Mails unter Verwendung der Java Platform. Die von Sun/Oracle entwickelte, ehemals proprietäre Referenzimplementierung der JavaMail API ist seit März 2009 unter verschiedenen freien Lizenzen als Open-Source-Software verfügbar.

4.4.1 Architekturübersicht

Um ein großes Maß an Flexibilität zu gewährleisten, folgt die JavaMail API dem Entwurfsmuster „Abstrakte Fabrik“ [GHJV94]. Die angebotenen abstrakten Klassen beschreiben auf hohem Abstraktionsniveau die einzelnen Bestandteile eines E-Mail-Systems. Auf diese Weise wird es ermöglicht, E-Mail-Funktionalität zu einer Applikation hinzuzufügen, ohne detaillierte Kenntnisse über konkrete Implementierungen der abstrakten Superklassen zu besitzen.

Neben diesem Modell werden zusätzlich Implementierungen der in Abschnitt 4.1 vorgestellten Standards des Internet Message Format und der MIME Messages bereitgestellt, um die Benutzung der JavaMail API für die E-Mail-Kommunikation über das Internet zu erleichtern. Die resultierende mehrschichtige Softwarearchitektur bei Verwendung der JavaMail API gemäß [Sun06] wird in Abbildung 4.5 illustriert.

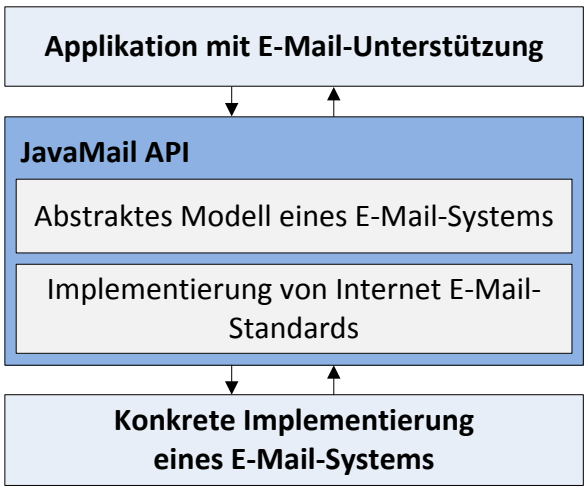


Abbildung 4.5: Schichtenarchitektur bei Verwendung des JavaMail API

Konkrete Implementierungen bestimmter E-Mail-Systeme werden mit Hilfe sogenannter Service-Provider eingebunden. Jeder Service-Provider muss alle abstrakten Klassen der

JavaMail API implementieren, die zur korrekten Benutzung der gewünschten E-Mail-Protokolle erforderlich sind.

Die aufrufende Applikation kann festlegen, welches Protokoll für die Kommunikation genutzt werden soll. Die JavaMail API erzeugt daraufhin eine Instanz eines passenden Service-Providers. Im weiteren Programmablauf fungiert die JavaMail API als Middleware zwischen Applikation und Service-Provider, indem zu versendende und ankommende Nachrichten weitergereicht werden.

Die Referenzimplementierung der JavaMail API liefert standardmäßig Service-Provider für SMTP, IMAP und das Post Office Protocol mit. Weitere Protokolle können durch von Drittanbietern bereitgestellte Service-Provider verwendet werden.

4.4.2 Zentrale Komponenten

Das abstrakte Modell eines E-Mail-Systems wird von der JavaMail API durch ein Zusammenspiel verschiedener Komponenten charakterisiert. Zu den zentralen Komponenten gehören die folgenden abstrakten Klassen und Schnittstellen:

- `Session`: Die gesamte Kommunikation mit dem Netzwerk läuft im Rahmen einer Sitzung ab. Eine `Session` definiert und verwaltet alle Attribute der Kommunikation zwischen der Applikation und dem Netzwerk.
- `Message`: Nachrichten stellen die Einheiten des Informationsaustausches dar. Die Klasse `Message` definiert grundlegende Attribute von Nachrichten, die für eine erfolgreiche Zustellung zwingend benötigt werden.
- `Folder`: Nachrichten werden serverseitig in Mailboxen aufbewahrt. Diese Mailboxen können als äquivalent zu den Verzeichnissen eines Dateisystems betrachtet werden. Repräsentiert werden sie durch die Klasse `Folder` und können neben Nachrichten auch weitere Unterordner vom Typ `Folder` enthalten.
- `Store`: Für den geordneten Zugriff auf einzelne serverseitig gespeicherte Ordner ist es erforderlich, eine Übersicht der verfügbaren Ordner in Form eines `Store` zu verwalten. Diese Klasse definiert zudem Schnittstellen für den Zugriff auf einzelne Ordnerinhalte und deren Transfer vom Server zum lokalen Client.
- `Transport`: Die konkrete Datenübertragung an das Netzwerk wird unter Verwendung von `Transport` durchgeführt. Dabei werden Wegewahl und technische Aspekte der Kommunikation mit dem Netzwerk vor dem Aufrufer verborgen.

Das grundlegende Vorgehen bei der Verarbeitung von Nachrichten lässt sich durch Kooperationen zwischen den genannten Komponenten beschreiben. Abbildung 4.6 visualisiert diese Zusammenarbeit.

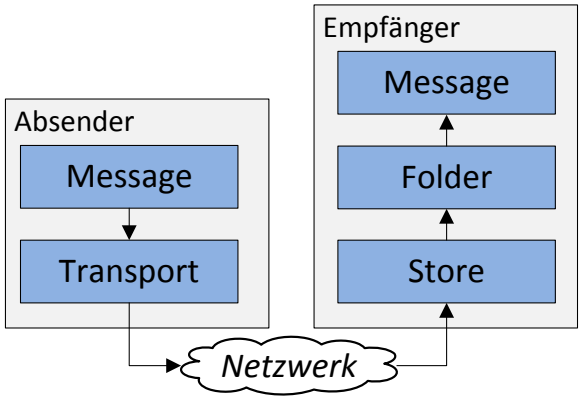


Abbildung 4.6: Zusammenspiel der zentralen Komponenten der JavaMail API

Auf der Seite des Absenders wird eine Nachricht vom Typ `Message` erzeugt, mit den zu übertragenden Nutzinhalten gefüllt und mit den benötigten Metainformationen versehen. Anschließend wird die Nachricht an die `Transport`-Komponenten übergeben, um sie über das Netzwerk an den Empfänger zu übertragen.

Dieser verbindet sich mit dem `Store`, der zur gewünschten E-Mail-Adresse gehört. Die versandte Nachricht wird in einem `Folder` abgelegt und kann von dort aus abgerufen werden. Nach dem erfolgreichen Abruf steht dem Empfänger die ursprüngliche Nachricht zur Verfügung, die er nun applikationsspezifisch weiterverarbeiten kann.

4.4.3 Behandlung von Ereignissen

Für eine effiziente Nutzung von E-Mail-Kommunikation in eigenen Applikationen ist es notwendig, automatisch über den Eintritt bestimmter Ereignisse informiert zu werden. Zu diesen Ereignissen zählen beispielsweise das Eintreffen neuer Nachrichten, das Löschen vorhandener Nachrichten oder die Restrukturierung der Verzeichnishierarchie auf dem Server.

Für diesen Zweck kann sich eine Applikation vorab bei den Komponenten der JavaMail API registrieren. An alle registrierten Interessenten wird gemäß dem Entwurfsmuster „Beobachter“ [GHJV94] bei Ereigniseintritt eine Benachrichtigung versandt. Die Komponenten `Store`, `Folder` und `Transport` dienen als Quellen für derartige Benachrichtigungen, da sie den serverseitigen Status des E-Mail-Systems repräsentieren und mit dem Netzwerk kommunizieren. Neben verschiedenen vorgegebenen Ereignistypen können

noch zusätzlich weitere definiert werden, um den speziellen Eigenschaften bestimmter E-Mail-Systeme Rechnung zu tragen.

4.5 Kryptographie-Infrastruktur der Java Platform

Beim Austausch sensibler Informationen zwischen verschiedenen Computersystemen kommt der Sicherheit des Datentransfers eine wesentliche Bedeutung zu. Zwei Aspekte, mit denen sich der Begriff Sicherheit in diesem Kontext charakterisieren lässt, sind Authentifizierung und Geheimhaltung. Unter Authentifizierung versteht man die Garantie, dass alle miteinander kommunizierenden Parteien auch wirklich diejenigen Personen oder Organisationen sind, die sie vorgeben zu sein. Geheimhaltung hingegen beschreibt den Schutz des Inhalts der ausgetauschten Informationen vor dem unberechtigten Zugriff Dritter.

Diese beiden Sicherheitsaspekte lassen sich durch den Einsatz von Verschlüsselung und der Verwendung von elektronischen Signaturen sicherstellen. Für eine allgemeine Einführung in diese beiden Wissenschaftsfelder sei an dieser Stelle auf die einschlägige Literatur verwiesen. Kompakte Erklärungen zu den Grundlagen finden sich beispielsweise bei Stephan Spitz in [SPS11].

Der folgende Abschnitt behandelt die von der Java Platform angebotene Infrastruktur, die zur Sicherstellung der beschriebenen Konzepte bei der Implementierung von α-Flow verwendet wird. Ausgehend von den Grundlagen der Java Cryptography Architecture folgt ein Überblick über die Funktionen der Java-basierten Bibliothek BouncyCastle Crypto API.

4.5.1 Java Cryptography Architecture (JCA) und Java Cryptography Extension (JCE)

Die zentralen Sicherheitskonzepte in Java sind in der Java Cryptography Architecture (JCA) gebündelt. Die Architektur der JCA gründet sich auf die Entwicklungsprinzipien von Implementierungsunabhängigkeit, Implementierungsinteroperabilität sowie der Möglichkeit zur Einbindung benutzerdefinierter Algorithmen [Ora11b]. Dies vereinfacht die Integration von Kryptographie in bestehende Softwaresysteme. Konkret handelt es sich bei der JCA um eine Sammlung verschiedener sogenannter Engine-Klassen, die in Form von abstrakter Modellierung, d.h. ohne konkrete Implementierung, bestimmte sicherheitsrelevante Dienste beschreiben. Zu diesen Diensten gehören unter anderem die

Generierung von Schlüsseln und Zertifikaten, die Erzeugung von Zufallszahlen, sowie das Erstellen elektronischer Signaturen.

Sämtliche Schnittstellen zu Diensten, deren Aufgaben die Verschlüsselung und Entschlüsselung von Daten, Schlüsselverwaltung oder Kommunikations-Authentifizierung sind, befinden sich in der Java Cryptography Extension (JCE), die ebenfalls Teil der JCA ist. Diese Trennung hat weniger technische Gründe, sondern ist politischen Interventionen geschuldet. Aufgrund von Exportbeschränkungen der Vereinigten Staaten von Amerika für kryptographische Verfahren mussten einige Funktionalitäten in der Vergangenheit aus den regulären Versionen der Java Platform ausgelagert werden, um die Verbreitung von Java außerhalb der USA zu legalisieren. Als Ergebnis von Veränderungen der rechtlichen Rahmenbedingungen, darf die JCE inzwischen wieder direkt als Teil der Java Platform ausgeliefert werden, die historisch gewachsene Trennung zwischen JCA und JCE bleibt jedoch weiterhin erhalten.

Um die in der JCA definierten Dienste verwenden zu können, müssen die jeweiligen Implementierungen ergänzt werden. Dafür sieht die JCA das Konzept von sogenannten Providern vor. Bei einem Provider handelt es sich um eine Sammlung von Klassen, die die benötigten abstrakten Modelle der Engine-Klassen mit konkreter Funktionalität füllen. Zum einen können vorgegebene Provider-Implementierungen eingesetzt werden, es besteht aber auch die Möglichkeit benutzerdefinierte Provider, die bestimmte Richtlinien befolgen [Ora11a], einzusetzen. Abbildung 4.7 visualisiert die Benutzung der JCA durch Anwendungsprogramme.

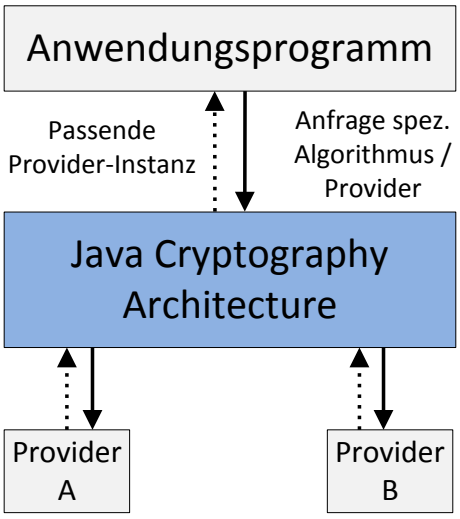


Abbildung 4.7: Providerkonzept der JCA

Diese interagieren initial ausschließlich mit dem Provider Framework der JCA. Dabei können Anfragen an die einzelnen Engine-Klassen formuliert werden, in denen die Anwendung spezifiziert, welchen Algorithmus sie zur Benutzung eines kryptographischen Dienstes benutzen möchte. Wird keine Bezeichnung für einen konkreten Provider von der Anwendung mitgeliefert, so wird aus der Menge der bekannten Provider der erste ausgewählt, der eine Implementierung des gewünschten Algorithmus anbietet. Die JCA erzeugt daraufhin eine Provider-Instanz, welche exakt die gewünschte Implementierung beinhaltet, und übergibt diese dem Anwendungsprogramm zur weiteren Benutzung.

4.5.2 Bouncy Castle Crypto API

Bei der Bouncy Castle Crypto API (BC API) handelt es sich um eine von der Entwicklergemeinschaft „Legion of the Bouncy Castle“[1] vertriebene freie Sammlung von Programmierschnittstellen zur erleichterten Benutzung kryptographischer Verfahren in Java-basierter Software. Der Funktionsumfang umfasst Implementierungen zahlreicher kryptographischer Algorithmen und bietet Unterstützung für ein breites Spektrum standardisierter Verschlüsselungsverfahren.

Darüber hinaus wird ein Provider für die JCA zur Verfügung gestellt, der zahlreiche Engine-Klassen implementiert. Dabei enthält er insbesondere kryptographische Verfahren, die mit weitaus längeren Schlüsseln arbeiten als die mit der Java Platform standardmäßig ausgelieferten Provider. Denn aufgrund der Tatsache, dass der Hauptentwicklungsstandort der „Legion of the Bouncy Castle“ Australien ist, existieren gemäß der dortigen Rechtslage keinerlei Beschränkungen in Hinblick auf die Stärke der verwendeten Algorithmen.

Die BC API kommt bei der Implementierung der Synchronisationskomponente von α-Flow zum Einsatz, da es derzeit die einzige frei verfügbare aktiv entwickelte Java-basierte Implementierung des in [CDF+07] definierten Industriestandards OpenPGP. Dieser Standard bildet die Grundlage zur kryptographischen Sicherung der zwischen den Teilnehmern einer α-Episode übertragenen Nachrichten.

4.6 Zusammenfassung

Mit Hilfe der Multipurpose Internet Mail Extensions (MIME) lassen sich beliebige Nutzinhalte versenden und E-Mail-Nachrichten in strukturierter Form gliedern. Dies

1 Für weitere Informationen siehe `http://www.bouncycastle.org`.

ermöglicht dem Empfänger eine unkomplizierte Verarbeitung binärer Inhalte und erlaubt einen Zugriff auf einzelne Informationsaspekte einer Nachricht.

Trotz der dargestellten Probleme beim zuverlässigen Nachrichtentransport bilden die beiden Protokolle SMTP und IMAP zusammen ein ausgereiftes E-Mail-System. SMTP realisiert auf unkomplizierte Weise den Versand von Nachrichten, IMAP bietet komplexe Möglichkeiten zum Nachrichtenempfang. Besonders bei der Verwaltung von E-Mails mit großen Dateianhängen und bei der Verwaltung umfangreicher Postfächer zeigt IMAP seine Stärken und ermöglicht einen bandbreitenschonenden Abruf der auf dem Server hinterlegten E-Mails. Darüber hinaus begünstigen die Konzepte ereignisbasierter Echtzeitbenachrichtigungen die Verwendung von E-Mails in mobilen Einsatzszenarien.

Die JavaMail API und die Java Cryptography Architecture ermöglichen durch ihre flexible Systemarchitektur die einfache Integration von E-Mail-Kommunikation und fortschrittlichen Mechanismen zu deren Absicherung in Java-Applikationen. Durch das eingesetzte Providerkonzept ist es zudem leicht möglich, die Frameworks an individuelle Anforderungen anzupassen.

Die präsentierten Aspekte der Kommunikation mittels E-Mail-Nachrichten umfassen diejenigen Technologien, auf deren Grundlage die prototypische Implementierung und ihre Integration in das α-Flow-System durchgeführt werden. Durch die vorgestellten Konzepte kann α-Flow die Kommunikation zwischen den beteiligten Akteuren realisieren.

5 Problemanalyse

Aufbauend auf den bisher präsentierten wissenschaftlichen und technischen Grundlagen werden die im Rahmen der Arbeit zu bearbeitenden Problemstellungen vorgestellt und auf die Anforderungen für ihre Umsetzung hin analysiert.

In Abschnitt 5.1 wird, ausgehend von einem medizinischen Beispielszenario, der Mehrwert der institutionsübergreifenden Synchronisation von Fallakten im Gesundheitswesen in Abgrenzung zur klassischen papierbasierten Kommunikation vermittelt. Im Anschluss daran werden in Abschnitt 5.2 mögliche Problemfelder identifiziert, die bei der automatischen Synchronisation von Fallakten auftreten und die Konsistenz der Kommunikation beeinträchtigen können.

Für eine zuverlässige Synchronisation der verfügbaren Informationen zwischen den Akteuren müssen von Seiten des verwendeten Kommunikationskanals verschiedene grundlegende Anforderungen erfüllt werden, um das benötigte Maß an Zuverlässigkeit der Kommunikation garantieren zu können. Diese Anforderungen werden in Abschnitt 5.3 erläutert.

Abschnitt 5.4 befasst sich schließlich detailliert mit den einzelnen Bestandteilen, die für ein zuverlässiges Synchronisationskonzept erforderlich sind, um den Anforderungen der Prozessbeteiligten zu genügen.

5.1 Institutsübergreifende Synchronisation von Fallakten im Gesundheitswesen

Im Gesundheitswesen lassen sich zahlreiche Beispiele für dynamische institutionsübergreifende Workflows identifizieren, bei denen Informationen über den Fortschritt des Prozesses zwischen den beteiligten Akteuren ausgetauscht werden. Einer dieser Workflows ist die Klassifikationsepisode bei Verdacht auf eine Brustkrebserkrankung [NL10]. Anhand der für diesen Behandlungsprozess notwendigen Abläufe sollen typische Charakteristika derartiger Prozesse aufgezeigt werden.

Hierbei wird zunächst die klassische papierbasierte Arbeitsweise der beteiligten Akteure betrachtet, bei der die im Behandlungsverlauf entstandenen Dokumente untereinander auf nicht-elektronischem Wege ausgetauscht werden, bis entsprechend dem therapeutisch-diagnostischen Zyklus genügend Informationen für eine fundierte Behandlungsentscheidung zur Verfügung stehen. Als dokumenten-orientierter Workflow lässt sich die Brustkrebsklassifkation somit als Folge von Dokumenten modellieren. Die Ergebnisdokumente einzelner Behandlungsschritte und ihre jeweiligen Urheber sind in Abbildung 5.1 visualisiert.

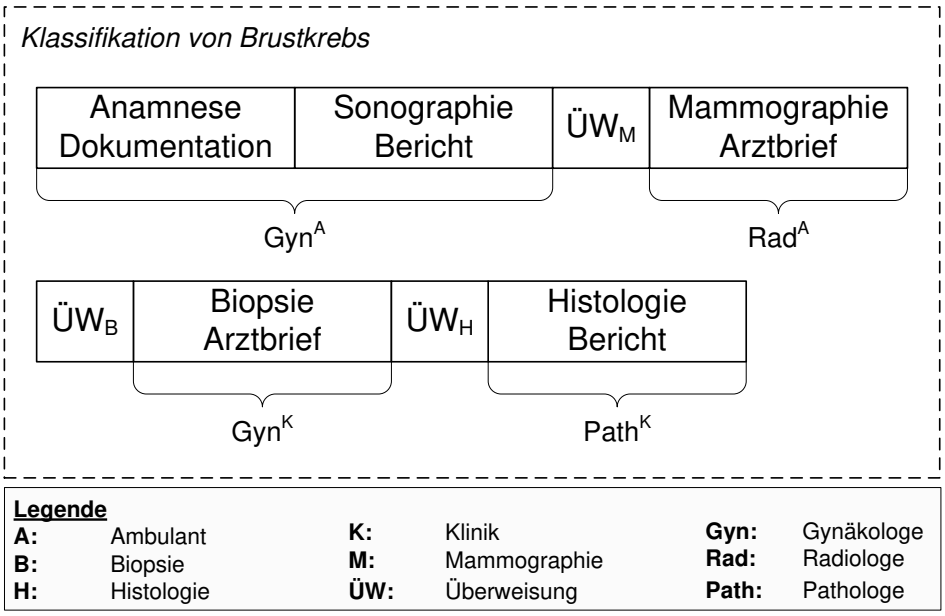


Abbildung 5.1: Klassifikationsepisode bei Verdacht auf Brustkrebs

Am Anfang der beispielhaften Behandlungsepisode steht die Anamnese des ambulanten Gynäkologen Gyn^A. Die betroffene Patientin sucht ihn auf, da sie vor wenigen Tagen beim Abtasten ihrer Brust einen Knoten erfühlt hat. Aufgrund ihrer Schilderungen führt Gyn^A nach der Dokumentation des Anamneseverlaufs eine Sonographie durch. Die Ergebnisse dieser Untersuchung hält er in einem Zwischenbericht fest.

Die Sonographie hat den im Raum stehenden Verdacht auf eine bösartige Neubildung erhärtet. Deswegen sind weitere diagnostische Maßnahmen erforderlich. Gyn^A ordnet daher die Durchführung einer Mammographie an. Dazu ist es notwendig, die Patientin zu einem Radiologen zu überweisen, da Gyn^A nicht über die benötigten medizinischen Geräte verfügt.

Gyn^A erstellt einen entsprechenden Überweisungsschein $\ddot{U}W_M$ mit weiteren Informationen für den Radiologen. Die Patientin bekommt diesen Überweisungsschein ausgehändigt und kann nun einen Radiologen zur weiteren Diagnostik aufsuchen.

Der Radiologe Rad^A erhält von der Patientin den Überweisungsschein und führt die angeforderte Mammographie durch. Nach dieser Untersuchung erstellt Rad^A einen Arztbrief, der die Ergebnisse der Untersuchung und eine Einschätzung der Gesundheitssituation der Patientin enthält. Eine Kopie dieses Arztbriefs wird anschließend in Papierform an den überweisenden Arzt Gyn^A geschickt.

Das Ergebnis der Mammographie legt im vorliegenden Fall nahe, zur weiteren Abklärung des Krankheitsbildes eine Biopsie des betroffenen Gewebes durchzuführen. Nach Erhalt des Arztbriefes von Rad^A veranlasst Gyn^A daher die Durchführung dieser Untersuchung. Zu diesem Zweck wird die Patientin mit Hilfe eines Überweisungsscheins $\ddot{U}W_B$ an den klinischen Gynäkologen Gyn^K überwiesen, der die Biopsie durchführen soll.

Nach Entnahme der Gewebeproben Gyn^K müssen diese weiterführend untersucht werden. Dazu werden sie zusammen mit $\ddot{U}W_H$, der Anforderung der vorgesehenen Untersuchungen, an den klinischen Pathologen $Path^K$ weitergegeben.

Nach Abschluss der Laboruntersuchungen erstellt $Path^K$ seinen Bericht über die histologischen Befunde. Dieser Bericht wird im Anschluss daran an Gyn^K übermittelt. Gyn^K verfasst einen zusammenfassenden Arztbrief, um die Ergebnisse der Biopsie an Gyn^A zurückzuleiten.

Nach Erhalt einer papierbasierten Kopie dieses Arztbriefs liegen Gyn^A alle Informationen vor, die er für eine fundierte Einschätzung des Gesundheitszustandes seiner Patientin benötigt. Er kann nun seine Entscheidung über den weiteren Verlauf der Behandlung treffen.

Problemfelder bei der klassischen papierbasierten Kommunikation

Das beschriebene Vorgehen bei der Klassifikation von Brustkrebs zeigt auf, dass die Leistungsfähigkeit medizinischer Workflows bei der klassischen papierbasierten Kommunikation unnötig eingeschränkt wird.

Zum einen verläuft der Informationsaustausch aufgrund der Passivität papierbasierter Dokumente rein manuell, d.h. der Versand von medizinischen Ergebnissen muss in jedem Fall vom Urheber der Informationen explizit angestoßen werden. Zu diesem Zweck müssen Kopien der Ergebnisdokumente in papiergebundener Form an die vorgesehenen Empfänger übermittelt werden, beispielsweise auf dem Postweg oder durch die Patientin selbst.

Problematisch ist zudem die Tatsache, dass neu gewonnene Informationen aufgrund des ansonsten erhöhten Verwaltungsaufwands zumeist nur direkt an den unmittelbar anfordernden Akteur weitergegeben werden und nicht an alle am Behandlungsprozess beteiligten Akteure. Im dargestellten Szenario wird beispielsweise der Bericht von $Path^K$ nur an den anfordernden Gyn^K versandt. Gyn^A, der für die initiale Anforderung verantwortlich ist, wird erst in Form des Biopsieberichts von Gyn^K über die Untersuchungsergebnisse unterrichtet. Somit stehen nicht allen Prozessbeteiligten zu jedem Zeitpunkt die jeweils aktuellsten Informationen zur Verfügung.

Hinzu kommt, dass aufgrund der langen Transportdauer von Papierdokumenten, der Zeitpunkt der medizinischen Entscheidungsfindung hinausgezögert wird. Besonders bei Workflows mit einer großen Anzahl beteiligter Akteure beeinträchtigt dies die Dynamik des Behandlungsprozesses.

Der aus den geschilderten Problemen resultierende unzureichende Informationsfluss und das Fehlen bzw. verspätete Eintreffen wichtiger Informationen behindert die institutionsübergreifende Zusammenarbeit zwischen den Akteuren. Dies wird von den partizipierenden Medizinern als wesentliche Problemstellung bei der Kommunikation und Koordinierung mit anderen Beteiligten empfunden [LBM+05].

Brustkrebsklassifikation unter Verwendung einer elektronischen Fallakte

Zur Vermeidung der beschriebenen Probleme können elektronische Fallakten eingesetzt werden. In einer solchen Fallakte in Form eines aktiven Dokuments werden alle medizinischen Ergebnisse gebündelt.

Bei der Brustkrebsklassifikation erstellt Gyn^A dafür zu Beginn des Behandlungsprozesses mit Hilfe einer speziellen Applikation auf seiner Workstation die elektronische Fallakte. Dieser fügt er neben den beiden Berichten zu Anamnese und Sonographie auch eine elektronische Überweisung zur Mammographie als Dokument hinzu. Zusätzlich hinterlegt Gyn^A die Adressinformationen eines Kommunikationskanals, auf dem er über Aktualisierungen der Behandlungsepisode seiner Patientin informiert werden kann. Diese Adressinformationen kann man sich beispielsweise als Adresse eines E-Mail-Kontos vorstellen.

Anschließend wird das aktive Dokument zum Transport auf einen geeigneten Datenträger, zum Beispiel die elektronische Gesundheitskarte, kopiert, damit die Patientin es einem Radiologen ihrer Wahl übergeben kann. Durch das Öffnen des aktiven Dokuments und die Eingabe seiner Adressinformationen kann dieser Radiologe Rad^A dem Kreis der

Prozessteilnehmer der vorliegenden Behandlungsepisode beitreten. Unter Verwendung der hinterlegten Adresse wird Gyn^A der Beitritt von Rad^A automatisch mitgeteilt.

Der Radiologe fügt anschließend den Arztbrief, der die Ergebnisse der Mammographie dokumentiert, dem aktiven Dokument hinzu. Ein im Hintergrund arbeitender Mechanismus sorgt dafür, dass Gyn^A über seine angegebene Adresse automatisch über das Vorliegen des Berichtes informiert wird, wenn er seine Kopie des aktiven Dokuments das nächste Mal öffnet.

Die erforderliche Überweisung zur Biopsie wird von Gyn^A nach Kenntnisnahme des Mammographieberichts als weiteres Dokument dem aktiven Dokument hinzugefügt. Die daraufhin neu erzeugte Kopie des aktiven Dokuments wird von der Patientin an Gyn^K weitergegeben.

Auch Gyn^K tritt mit Eingabe seiner Adressinformationen erfolgreich dem Behandlungsprozess bei. Anstatt eines Überweisungsscheins werden die entnommenen Gewebeproben zusammen mit einer aktualisierten Kopie des aktiven Dokuments an den klinischen Pathologen $Path^K$ weitergegeben.

Nachdem auch dieser dem Kreis der behandelnden Akteure beigetreten ist, fügt er seinen Bericht über die histologischen Befunde dem aktiven Dokument hinzu. Diese Informationen werden daraufhin automatisch an alle beteiligten Mediziner weitergeleitet, um sie über den Status der Diagnosefindung zu informieren.

Beim nächsten Öffnen seiner Kopie des aktiven Dokuments kann Gyn^K auf Basis der histologischen Ergebnisse seinen Arztbrief zum Ergebnis der Biopsie dem aktiven Dokument hinzufügen, um ihn wiederum an alle Akteure zu verteilen.

Vorteile der automatischen Synchronisation verfügbarer Informationen

Die Vorteile, die sich durch den Einsatz eines aktiven Dokuments mit automatischer Synchronisation gegenüber einer papierbasierten Fallakte ergeben, sind weitreichend. Zum einen wird die gesamte technische Koordination des Prozessverlaufs vor den Akteuren verborgen. Nach Angabe ihrer Adressinformationen werden sie automatisch über Aktualisierungen von Prozessartefakten des aktiven Dokuments benachrichtigt. Die Verteilung aktualisierter Informationen erfordert keine zusätzliche manuelle Interaktion des jeweiligen Urhebers, sondern wird selbsttätig vom aktiven Dokument auf Basis der hinterlegten Adressen der weiterbehandelnden und mitbehandelnden Akteure übernommen.

Zum anderen wird nicht nur der unmittelbar anfordernde Akteur über neue Ergebnisse benachrichtigt, sondern alle an der Behandlungsepisode beteiligten Akteure. Damit

wird sichergestellt, dass alle Akteure zu jedem Zeitpunkt vollständigen Zugang zu den gemeinsamen Informationen bezüglich der Patientenbehandlung haben.

Zu berücksichtigen ist auch die Geschwindigkeit der Weitergabe von neuen medizinischen Erkenntnissen. Der Zeitaufwand für den manuellen Transport von Ergebnisberichten entfällt vollständig, da alle Informationen sofort nach ihrer Modifikation an alle Akteure propagiert werden. Die schnelle automatische Weitergabe von Informationen macht es zudem praktikabel, vorläufige Zwischenergebnisse den übrigen Akteure als unverbindliche Vorabindikatoren des Gesundheitszustands des Patienten zugänglich zu machen. Auf diese Weise können vorliegende Befunde bereits weitergegeben werden, noch bevor der zugehörige ausführliche Arztbrief erstellt worden ist. Die Akteure können diese Ergebnisse somit bereits als erste Informationsquelle nutzen. Bei der Brustkrebsklassifikation haben so alle Beteiligten bereits Zugriff auf die Ergebnisse der von $Path^K$ durchgeführten Untersuchung der Gewebeproben, bevor der Arztbrief von Gyn^K fertiggestellt ist.

Die genannten Aspekte erlauben eine höhere Dynamik des Behandlungsprozesses. Informationen werden auf schnellstem Wege verbreitet und stehen daher jeweils rechtzeitig zum Zeitpunkt der medizinischen Entscheidungsfindung zur Verfügung.

5.2 Mögliche Probleme bei der Nutzung verteilt verwalteter Fallakten

Neben den im vorangegangenen Abschnitt dargestellten Vorteilen entstehen bei der Nutzung verteilt verwalteter Fallakten, deren Kopien automatisch untereinander synchronisiert werden, neue Problemfelder, die die Konsistenz der verteilten Informationen beeinträchtigen können.

Die Ursache für diese Probleme liegt darin, dass jeder der beteiligten Akteure schreibenden Zugriff auf gemeinsam genutzte Ressourcen zur Prozesskoordination besitzt. Zu diesen Ressourcen gehören beispielsweise die Liste der beteiligten Akteure oder der gemeinsam verwaltete Therapieplan.

Überschneiden sich mehrere, von verschiedenen Akteuren durchgeführte Änderungen einer solchen Ressource zeitlich derart, dass ein Akteur vor Durchführung seiner eigenen Änderung nicht über sämtliche Änderungen anderer Akteure informiert ist, so liegt ein nebenläufiger Änderungskonflikt vor. Tritt ein derartiger Fall ein, dann ist ab diesem Zeitpunkt kein konsistentes Fortfahren im Behandlungsprozess mehr möglich, da nicht alle Akteure über die gleichen Informationen zum Status der Behandlung verfügen.

Am Beispiel des gemeinsamen Therapieplans lässt sich zeigen, dass nebenläufige Änderungen die fachliche Konsistenz des Behandlungsprozesses beeinträchtigen und zu einem unerwünschten Prozessverlauf führen können. Sind beispielsweise weitere diagnostische Informationen notwendig, um eine medizinische Entscheidung über die weitere Behandlung des Patienten zu treffen, kann es passieren, dass zwei Akteure unabhängig voneinander die Durchführung ähnlicher Untersuchungen anordnen. Da beide Ärzte ihre Anordnung unter Umständen nicht gegeben hätten, wenn sie von der Entscheidung des anderen Akteurs gewusst hätten, befindet sich der Therapieplan in einem fehlerbehafteten Zustand, da unbeabsichtigt eine überflüssige Untersuchung eingeplant ist.

Spezifische Probleme im Kontext von α-Flow

In α-Flow werden gemeinsam genutzte Informationen mit Hilfe der Coordination-Cards verwaltet. Zu den Coordination-α-Cards gehören, wie in Abschnitt 3.1 beschrieben, das Collaboration Resource Artifact (CRA), das Process Structure Artifact (PSA) und das Adornment Prototype Artifact (APA).

Jeder Akteur einer α-Episode besitzt Schreibrechte auf diesen Coordination-Cards. Aktualisierungen werden ausgelöst durch den Eintritt von Ereignissen, die das α-Flow-System aus organisatorischer Sicht betreffen und sich auf ein oder mehrere α-Cards auswirken.

Der Payload des CRA ändert sich immer dann, wenn neue Teilnehmer dem Prozess beitreten oder vorhandene Teilnehmer ihre Adressinformationen aktualisieren. Der Payload des PSA wird immer dann modifiziert, wenn von einem Akteur eine neue α-Card angelegt wird. Änderungen am Payload des APA treten bei Modifikation benutzerdefinierter adaptiver Adornments auf. Da alle diese Änderungen aber an die übrigen Akteure zur lokalen Verarbeitung propagiert werden, kann bei gleichzeitigem Zugriff durch mehrere Akteure ein nebenläufiger Änderungskonflikt entstehen. Änderungen an Coordination-Cards müssen somit unter allen Akteuren synchronisiert werden.

Wird es zugelassen, dass Content-Cards, die beispielsweise Überweisungen, Arztbriefe oder Befunde repräsentieren, ebenfalls von mehr als jeweils genau einem Akteur modifiziert werden können, bergen auch diese Schreibzugriffe Konfliktpotential. Bei ungünstiger Überschneidung von Änderungen werden unter Umständen medizinische Entscheidungen aufgrund inkonsistenter Informationen getroffen.

5.3 Eigenschaften des verwendeten Kommunikationskanals

Um die Bestandteile des benötigten Synchronisationskonzepts identifizieren zu können, muss bekannt sein, welche Garantien der verwendete Kommunikationskanal im Hinblick auf die Zuverlässigkeit der Nachrichtenübermittlung bietet (vgl. hierzu auch die Ausführungen in Abschnitt 3.2.3.2). Für α-Flow wird vorausgesetzt, dass der für die Kommunikation genutzte Kanal mindestens über die nachfolgend dargestellten Eigenschaften verfügt:

1. Der Kommunikationskanal muss in der Lage sein, Nachrichten aufzubewahren, falls der jeweilige Empfänger temporär nicht erreichbar ist. Baut der Empfänger erneut eine Verbindung zum Netzwerk auf, müssen ihm die zwischengelagerten Nachrichten übergeben werden.
2. Eine Nachricht, die von einem Akteur im Prozessverlauf abgeschickt wird, muss ihre jeweiligen Empfänger erreichen, um allen Akteuren die notwendigen Informationen zur lokalen Erkennung globaler Konflikte zur Verfügung zu stellen.

Es ist nicht erforderlich, dass die Reihenfolge der Nachrichten auf dem Transportweg erhalten bleibt. Die Reihenfolge, in der Nachrichten von einem Akteur empfangen werden, muss daher nicht notwendigerweise derselben Reihenfolge entsprechen, in der sie vom Sender verschickt wurden.

5.4 Bestandteile eines zuverlässigen Synchronisationskonzepts

Um die Anforderungen der Akteure an den Informationsaustausch in interdisziplinären und institutionsübergreifenden Workflows erfüllen zu können, muss das Konzept zur zuverlässigen Synchronisation der verteilten Fallakten verschiedene Funktionalitäten anbieten. Basierend auf den geschilderten Vor- und Nachteilen automatischer Synchronisation und den Charakteristika der zu modellierenden Prozesse lassen sich die in den nachfolgenden Abschnitten detailliert dargestellten Bestandteile des Synchronisationskonzepts identifizieren.

5.4.1 Herstellen einer Ordnung auf den Versionen von Prozessartefakten

Die zeitliche Abfolge des Nachrichtenempfangs gibt nicht zwangsläufig zuverlässig Auskunft über die zugrundeliegende Reihenfolge der durchgeführten Änderungen. Anomalien, wie das in Abbildung 5.2a dargestellte gegenseitige Überholen von Nachrichten auf dem Transportweg oder das in Abbildung 5.2b illustrierte nebenläufige Verändern eines Prozessartefakts, müssen durch zusätzliche Mechanismen auf der Empfängerseite identifiziert werden.

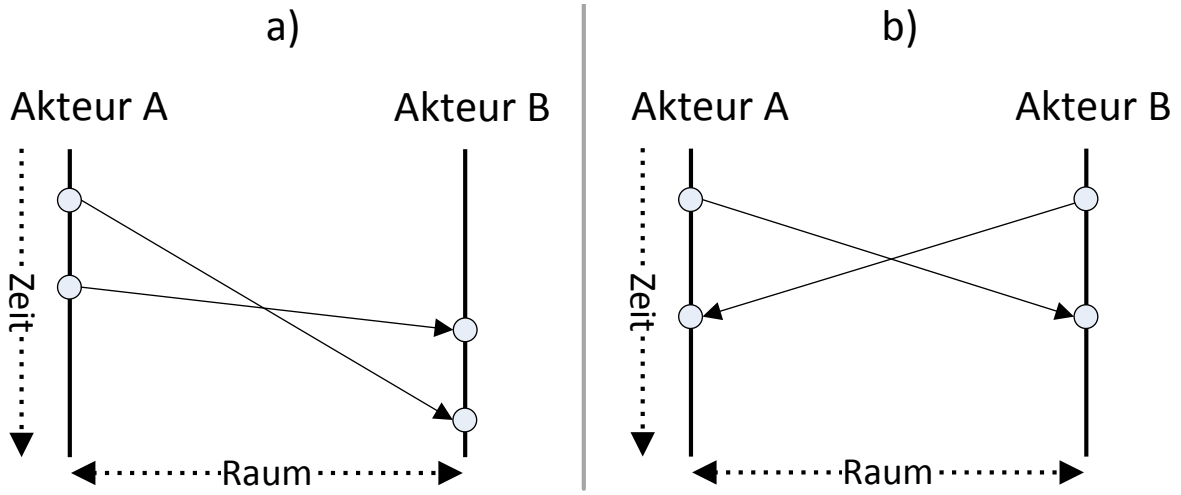


Abbildung 5.2: Mögliche Anomalien bei Aktualisierung eines Prozessartefakts

Um zu verhindern, dass Prozessartefakte dauerhaft in einem inkonsistenten Zustand verbleiben, muss das Synchronisationskonzept in der Lage sein, durch geeignete Strukturen und Mechanismen die ankommenden Nachrichten in eine Ordnung bringen zu können. Dadurch wird es ermöglicht, die Entstehung und Modifikation von Prozessartefakten im globalen Kontext zu dokumentieren. Dabei wird angenommen, dass dem lokalen System technische Mittel zur Verfügung stehen, mit deren Hilfe einzelne Versionen von Prozessartefakten individuell und dauerhaft archiviert werden können. In α-Flow kommt hierzu das von Scott A. Hady entwickelte Versionsverwaltungssystem Hydra [Had11] zum Einsatz, das über gesonderte Schnittstellen in das Gesamtsystem integriert wird.

Zur Verwaltung einer lückenlosen Versionshistorie von Prozessartefakten muss es dem Synchronisationskonzept möglich sein, Versionen, die nicht in ihrer Sendereihenfolge eintreffen, zu Dokumentationszwecken unter Verwendung des eingesetzten Versionsverwaltungssystems nachträglich in die Historie einzufügen. Zu diesem Zweck muss relativ

zu den übrigen Versionen diejenige Position innerhalb der Versionshistorie bestimmt werden können, an der eine ankommende Version kausal einzuordnen ist.

Gemäß Abschnitt 5.2 wird darüber hinaus gefordert, dass das Synchronisationskonzept in der Lage ist, mit nebenläufigen Änderungskonflikten eines Prozessartefakts umzugehen. Derartige Konflikte müssen zuverlässig erkannt werden können, um die Konsistenz aller Replikate des Prozessartefakts aufrechtzuerhalten.

Wird ein Konflikt erkannt, so ist der Benutzer unverzüglich darüber zu informieren, da mögliche fachliche Inkonsistenzen aus Gründen der Transparenz nicht vom System verborgen werden dürfen. Zudem muss das System in der Lage sein, Möglichkeiten zur Behebung des Änderungskonflikts bereitzustellen. Im Rahmen der vorliegenden Arbeit soll das betroffene Prozessartefakt auf die letzte nicht-konfliktbehaftete Version zurückgesetzt werden, es existieren aber verschiedene weitere Strategien, auf die in Abschnitt 11.2 näher eingegangen wird.

Als besondere Herausforderung gilt es zu beachten, dass aufgrund der lose-gekoppelten Struktur dokumenten-orientierter verteilter Workflows einzelne Akteure längere Zeit von den übrigen Prozessbeteiligten isoliert agieren können. Daher muss die Konfliktbehandlung vollständig auf Basis der lokal verfügbaren Informationen durchzuführen sein.

5.4.2 Prozessbeitritt und -austritt von Akteuren

Eine charakteristische Eigenschaft interdisziplinärer institutionsübergreifender Prozesse ist, dass die Anzahl beteiligter Akteure initial nicht bekannt ist [NL09]. Deswegen benötigen die bereits an einem Behandlungsprozess teilnehmenden Akteure in Form eines Beitrittsprotokolls die Möglichkeit, unkompliziert und dynamisch neue Akteure zur Teilnahme einzuladen und ihnen die aktuellsten Informationen über den Prozessverlauf zukommen zu lassen. Neuen Akteuren soll zum Beitritt lediglich eine Kopie des aktiven Dokuments übergeben werden müssen, bei deren erstem Öffnen sie ihre Adressinformationen für den Austausch von Aktualisierungen hinterlegen können.

Dabei muss insbesondere sichergestellt werden, dass der parallele Beitritt mehrerer Teilnehmer nicht zu Inkonsistenzen führt. Jeder Akteur muss zuverlässig über die Identität aller übrigen am Prozessverlauf beteiligten Individuen und Institutionen informiert werden.

Um unnötigen Informationsaustausch zu vermeiden, sollen nicht mehr Nachrichten als minimal erforderlich verschickt werden, um allen Akteuren den aktuellen Prozessstatus zugänglich zu machen. Durch wenige, aber dafür grobgranulare Nachrichten, die alle zu

einem bestimmten Zeitpunkt während des Beitritts eines neuen Akteurs erforderlichen Informationen bündeln, kann erreicht werden, dass ein erfolgreicher Beitritt auch dann möglich ist, wenn einige Akteure zu bestimmten Zeitpunkten offline sind.

Möchte ein Akteur seine Teilnahme an einer Behandlungsepisode zu einem bestimmten Zeitpunkt beenden, so muss er dafür sorgen, dass allen zukünftig beitretenden Akteuren alle Informationen über die von ihm verantworteten Prozessartefakte zugänglich gemacht werden können. Hierfür ist es erforderlich, dass er den übrigen Akteuren seinen Austritt aus dem Kreis der Prozessteilnehmer entsprechend anzeigt und die von ihm verantworteten Informationen abschließend propagiert. Der Austritt von Akteuren ist jedoch explizit nicht Gegenstand der vorliegenden Arbeit und bleibt offen für weitere Forschungen im Rahmen von α-Flow.

5.4.3 Verwaltung der Divergenz zwischen Akteur, Adressinformation und physischem Arbeitsplatz

Nachdem ein Akteur dem Prozess beigetreten ist, wird er über alle Aktualisierungen von Prozessartefakten informiert. Der neu beigetretene Akteur kann nun von ihm selbst an einer Kopie des aktiven Dokuments durchgeführte Aktualisierungen an die übrigen Akteure propagieren.

Um die Möglichkeiten einer elektronischen Fallakte flexibel nutzen zu können, muss es darüber hinaus jedem Akteur ermöglicht werden, Replikate der ihm zugeordneten Kopie des aktiven Dokuments anzufertigen. Damit ist eine Nutzung des aktiven Dokuments von mehreren physischen Arbeitsplätzen aus möglich. Das aktive Dokument kann dann beispielsweise von einem Akteur sowohl im Sprechzimmer als auch in gesonderten Untersuchungsräumen oder an Befundungsarbeitsplätzen auf verschiedenen Computern verwendet werden.

Um dieses Benutzungsverhalten zu unterstützen, müssen Mechanismen für die Synchronisation zwischen den Kopien des aktiven Dokuments eines Akteurs bereitgestellt werden. Dazu muss es möglich sein, Akteure und ihre physischen Arbeitsplätze eindeutig identifizieren zu können, um trotz der nicht Arbeitsplatz-spezifischen Adressinformationen der Akteure eine Weiterleitung von Aktualisierungen an alle existierenden Replikate der elektronischen Fallakte in Form des aktiven Dokuments zu erlauben.

5.5 Zusammenfassung

Im Rahmen der vorliegenden Arbeit soll die zuverlässige Synchronisation der verfügbaren Informationen zwischen den einzelnen Akteuren unter Verwendung von elektronischen Fallakten realisiert werden. Zu diesem Zweck müssen konzeptionelle Lösungen für die verschiedenen Aspekte der zu bearbeitenden Problemstellung bereitgestellt werden.

Als Bestandteil des fachlichen Lösungsansatzes wird ein Konzept zur Herstellung einer Ordnung auf den verschiedenen Versionen eines Prozessartefakts benötigt. Dabei müssen globale nebenläufige Änderungskonflikte auf Basis der lokal verfügbaren Informationen zuverlässig erkannt und behoben werden können. Daneben muss es möglich sein, in Kooperation mit einem Versionsverwaltungssystem die Versionshistorie eines Prozessartefakts nachträglich zu vervollständigen, um verzögert ankommende Versionen für Dokumentationszwecke zu archivieren.

Weiterer Bestandteil des fachlichen Lösungsansatzes ist ein Protokoll zum Beitritt neuer Akteure, das den parallelen Beitritt einer beliebigen Anzahl von neuen Akteuren unterstützt. Dabei ist sicherzustellen, dass alle Akteure untereinander bekannt gemacht und die aktuellsten Informationen über den Prozessverlauf an alle neu beigetretenen Akteure propagiert werden.

Schließlich muss der fachliche Lösungsansatz in der Lage sein, die Divergenz zwischen Akteur, Adressinformation und physischem Arbeitsplatz zu verwalten. Die hierfür zu entwickelnden Mechanismen sind dafür verantwortlich, den Akteuren eine unbeschränkte Replikation des aktiven Dokuments und gleichzeitiger Einhaltung der Konsistenz der verteilt gespeicherten Replikate zu ermöglichen.

6 Verwandte Arbeiten

Verschiedene andere Arbeiten und Technologien sind mit den in der vorliegenden Arbeit verwendeten Konzepten verwandt:

Neben der im fachlichen Lösungsansatz vorgesehenen, auf logischer Zeit basierenden Synchronisation existieren Mechanismen zur Synchronisation von physischen Uhren. Daneben sind weiterführende Konzepte zur Verwaltung logischer Zeitstempel verfügbar, die auf den in Abschnitt 3.3 vorgestellten Grundlagen basieren.

Auch im Kontext verteilter Datenbanken spielt die Synchronisation von Modifikationen auf repliziert gespeicherten Daten eine bedeutende Rolle. Hierfür sind vielfältige Konzepte verfügbar, die es erlauben, einen verteilten Datenbestand bei allen Netzknoten konsistent zu halten bzw. nebenläufige Änderungskonflikte zu vermeiden.

6.1 Synchronisation physischer Uhren

Wie bereits in Kapitel 3.2.4 erwähnt, können physische Uhren durch den paarweisen Austausch von Zeitstempeln untereinander synchronisiert werden. Dabei kann unterschieden werden zwischen Verfahren zur externen Synchronisationen, die auf eine externe Zeitquelle zurückgreifen, und Verfahren zur internen Synchronisation, bei denen die gültige Uhrzeit ausschließlich durch Informationsaustausch innerhalb einer definierten Gruppe von Netzknoten bestimmt wird.

6.1.1 Algorithmus von Cristian

Eine probalitistische Methode zur externen Uhrensynchronisation beschreibt Cristian in [Cri89]. Der von ihm entwickelte Algorithmus stützt sich auf die Annahme, dass ein zuverlässiger, vertrauenswürdiger Zeitserver verfügbar ist. Dieser muss zu jedem Zeitpunkt von den Netzknoten kontaktiert werden können, um eine dauerhafte Synchronisation zu ermöglichen. Desweiteren muss gefordert werden, dass die Round-Trip-Zeiten zwischen Client und Zeitserver kleiner sind als die gewünschte Genauigkeit der globalen Uhrzeit.

Die Synchronisation einer lokalen Uhr mit der zentralen Serverzeit läuft nach folgendem Schema ab:

1. Der Client C fordert mit einer Nachricht zum Zeitpunkt t_0 die aktuelle Zeit vom Server S an.
2. S verarbeitet diese Anfrage innerhalb einer Zeitspanne I.
3. S sendet eine Antwortnachricht an C, in der die aktuelle Serverzeit t_1 enthalten ist.
4. C setzt seine lokale Uhr auf $t_1 + \frac{t_{RoundTrip}}{2}$, wobei die Round-Trip-Zeit $t_{RoundTrip} = t_1 - t_0$.
   Ist C darüber hinaus I bekannt, so kann durch $t_{RoundTrip} = t_1 - t_0 - I$ eine verbesserte Annäherung zwischen den lokalen Uhrzeiten erreicht werden.

Für kleine Round-Trip-Zeiten, wie sie beispielsweise in lokalen Netzwerken auftreten, kann auf diese Weise eine sinnvolle Uhrensynchronisation umgesetzt werden.

6.1.2 Berkeley-Algorithmus

Bleibt der Zeitserver bei Cristian rein passiv, so übernimmt er beim Berkeley-Algorithmus [GZ89], einem Verfahren zur internen Uhrensynchronisation, eine aktive Rolle. In periodischen Zeitabständen sendet derjenige Client, der von den übrigen als Zeitserver ausgewählt worden ist, Anfragen an alle Netzknoten, um deren aktuelle lokale Zeit zu erfragen.

Mit einem Verfahren ähnlich dem von Cristian berechnet der Server die einzelnen Round-Trip-Zeiten. Mit diesen Informationen können die lokalen Uhrzeiten aller Clients geschätzt werden. Der Server bildet aus den lokalen Uhrzeiten einen fehlertoleranten Mittelwert, der die neue globale Uhrzeit darstellt. Abschließend werden jedem Client die Informationen übersandt, wie er seine lokale Uhr anpassen muss, um konform zur Systemzeit zu agieren.

Auch bei diesem Verfahren ist es für eine aussagekräftige Annäherung an die reale Uhrzeit notwendig, dass die Round-Trip-Zeiten zwischen Client und Server nicht zu groß werden.

6.1.3 Network Time Protocol (NTP)

Im Gegensatz zu den beiden vorherigen Ansätzen erlaubt das Network Time Protocol (NTP) [MMBK10] eine externe Uhrensynchronisation in Netzwerken mit großen und

variablen Nachrichtenverzögerungen sowie zeitweise unerreichbaren Netzknoten. NTP stützt sich auf das User Datagram Protocol [Pos80] und greift auf eine Hierarchie von weltweit verteilt platzierten Zeitservern zurück. Auf diese Weise können Ausfälle einzelner Server kompensiert werden, ohne die Uhrensynchronisation in nennenswertem Maße zu behindern.

Für eine verbesserte Genauigkeit der Zeitschätzung tauschen die unterschiedlichen NTP-Server untereinander nach festgelegten Protokollen lokale Uhrzeiten aus. Clients können jederzeit die serverseitig nach einem Mittelwert-Algorithmus gebildete gültige Zeit von einem Server ihrer Wahl abfragen.

Trotz der möglichen Störungen der Verbindung und der Datenübertragung ist es möglich, unter Verwendung von NTP, Uhren über das Internet mit einer Abweichung zwischen 1 und 50 Millisekunden zu synchronisieren. [TS03].

6.1.4 Eignung physischer Zeitstempel im Kontext von α-Flow

Als Grundlage der Synchronisation von Modifikationen an Prozessartefakten sind physische Zeitstempel im Kontext von α-Flow aus mehreren Gründen ungeeignet.
Verfahren zur internen Uhrensynchronisation widersprechen der Idee, dass neben Kopien des α-Doc so wenig weitere Infrastruktur wie möglich erforderlich ist, um den Prozessverlauf zu koordinieren. Um zu garantieren, dass jederzeit ein zentraler Zeitserver verfügbar ist, muss dieser außerhalb des α-Doc dediziert betrieben werden. Ein solcher Server, der individuell für die Akteure einer α-Episode betrieben wird, erfordert jedoch zusätzlichen Administrationsaufwand, welcher die Komplexität der Koordination des Prozessverlaufs erhöht.

Im Gegensatz dazu ist bei der externen Uhrensynchronisation nicht zwingend ein individueller Zeitserver pro α-Episode erforderlich, da auf öffentlich verfügbare Zeitserver zurückgegriffen werden kann. Die ununterbrochene Verfügbarkeit dieser Server kann jedoch nicht garantiert werden. Sind die Zeitserver aufgrund von Verbindungsproblemen nicht von allen Akteuren erreichbar, so können durch die daraufhin divergierenden lokalen Uhrzeiten Inkonsistenzen bei der Synchronisation auftreten. Daher wird im Rahmen dieser Arbeit auch von der Verwendung externer Uhrensynchronisation abgesehen.

6.2 Alternative Verfahren zur Verwaltung logischer Zeitstempel

Neben den in Abschnitt 3.3 vorgestellten Zeitstempelkonzepten gibt es zahlreiche weitere vektorbasierte Ansätze zur Modellierung logischer Zeitpunkte.

So stellen die sogenannten Matrixuhren eine Generalisierung des Konzepts der Vektoruhren dar [DB03]. Dabei besteht ein Zeitstempel aus einem Vektor von Vektoruhren. Allen Netzknoten ist somit zu jedem Zeitpunkt bekannt, auf welchem Kenntnisstand sich andere Knoten befinden. Diese Informationen können unter anderem für die Herstellung eines gemeinsamen globalen Status oder die Überwachung von Berechnungen auf gemeinsamen Ressourcen verwendet werden.

Ein Nachteil von vektorbasierten Zeitstempeln ist ihre eingeschränkte Skalierbarkeit. Für jeden Netzknoten muss ein individueller Vektoreintrag verwaltet werden. Zudem kann es bei einer hohen Modifikationsfrequenz zu einem Überlaufen der Wertebereiche von logischen Zählern kommen, wodurch betroffene Zeitstempel ungültig werden. Unter der Annahme, dass alle zwischen den Netzknoten ausgetauschten Nachrichten in der Sendereihenfolge beim Empfänger ankommen und keine Nachrichten verloren gehen, kann durch Optimierungen der Zeitstempelstruktur und der Regeln zu deren Verwaltung eine verbesserte Skalierbarkeit erreicht werden. So bieten beispielsweise die Plausible Clocks [TRA99] oder die effizienten Vektoruhren von Signhal/Kshemkalyani [SK92] Ansätze zur Realisierung von logischen Uhren konstanter Größe. Auch unter Verwendung der Bounded Version Vectors [AAB04] kann eine Platzersparnis erreicht werden.

Zu beachten ist beim Einsatz komprimierter Zeitstempel jedoch die Tatsache, dass in speziellen Fällen Abstriche bei der Konsistenz bzw. der Korrektheit der Ordnung von Ereignissen gemacht werden müssen. So ist es beispielsweise möglich, dass Plausible Clocks unter bestimmten Bedingungen zwei Ereignisse als nebenläufig klassifizieren, obwohl diese tatsächlich in eine gültige Ordnung zu bringen sind.

Aufgrund dieser Problemstellungen und in Hinblick auf die Charakteristika der erwarteten Kommunikationskanäle (vgl. Abschnitt 5.3) wird von einer Verwendung derartig optimierter Zeitstempelkonzepte im Rahmen dieser Arbeit abgesehen. Zudem sind aufgrund der Struktur der von α-Flow zu modellierenden Prozesse weder Überläufe der logischen Zähler noch Skalierungsprobleme aufgrund der Anzahl von Vektoreinträgen zu erwarten.

6.3 Synchronisation in verteilten Datenbanken

Auch die Synchronisation von repliziert an verschiedenen Netzwerkorten gespeicherten Datenbanken bedarf spezieller Mechanismen, um die gegenseitige Konsistenz der Repliken sicherstellen zu können. In der vorliegenden Arbeit werden ausgewählte Verfahren vorgestellt und auf ihre Anwendbarkeit im Rahmen von α-Flow untersucht. Eine weiterführende Zusammenstellung von Synchronisationsverfahren im Kontext adaptiver Datenreplikation in verteilten Systemen ist in [Len97] zu finden.

6.3.1 Synchronisation durch Verwendung von Zeitstempeln

Zur Synchronisation bei verteilten Datenbanken können unter anderem die Konzepte objektspezifischer Zeitstempel angewandt werden. So benutzt beispielsweise das Datenbanksystem „Project Voldemort“[1] den klassischen Ansatz der Versionsvektoren um einzelne Netzknoten zu synchronisieren. Es existieren jedoch verschiedene weitere Zeitstempelbasierte Konzepte, von denen im Folgenden zwei näher erläutert werden.

6.3.1.1 Independent Updates

Das Konzept der Independent Updates [CHKS92], [CHKS95] verwendet Zeitstempel, die den Vektoruhren ähneln. Für jedes repliziert gespeicherte Datenobjekt wird eine zusätzliche Datenstruktur geführt.

Dieser sogenannte Reception Vector enthält einen Eintrag pro Netzknoten. Ein solcher Eintrag besteht aus dem Zeitstempel der letzten von dem zugehörigen Netzknoten durchgeführten und in die Datenbank eingebrachten Modifikation des Datenobjekts.

Der zur Synchronisation eingesetzte Algorithmus garantiert, dass bei Empfang eines beliebigen Reception Vector alle Modifikationen, deren Zeitstempel kleiner als der ankommende sind, bereits erfolgreich in den lokalen Datenbestand des Empfängers eingebracht worden sind.

Die Propagation von Änderungen läuft in zwei Schritten ab. Zunächst versucht der Urheber der Änderung alle übrigen Netzknoten synchron über seine Modifikation zu benachrichtigen. Schlägt dies bei einigen Knoten fehl, so wird in einer separaten Datenstruktur protokolliert, welche Änderungen an welchem Datenobjekt noch an weitere Knoten zur propagieren sind. Sobald diese Netzknoten wieder für einen Datenaustausch

1 Für weitere Informationen siehe `http://www.project-voldemort.com/`.

zur Verfügung stehen, wird eine Zusammenführung angestoßen, in deren Verlauf die noch fehlenden Aktualisierungsinformationen ausgetauscht werden.

Für die Korrektheit dieses Verfahrens müssen alle Operationen auf den Datenobjekten unär und kommutativ sein. Darüber hinaus muss für jede Operation eine entsprechende Kompensationsaktion verfügbar sein, mit der die Auswirkungen der ursprünglichen Operation wieder rückgängig gemacht werden können. In Hinblick auf den Kommunikationskanal wird zudem, anders als bei den für α-Flow getroffenen Annahmen (vgl. 5.3), gefordert, dass alle Nachrichten in der Sendereihenfolge beim Empfänger ankommen und es dem Absender durch einen Timeout-Mechanismus ermöglicht wird festzustellen, ob seine Nachrichten angekommen oder auf dem Transportweg verloren gegangen sind.

6.3.1.2 Timestamped Anti-Entropy

Ein weiteres auf Zeitstempeln basierendes Synchronisationsverfahren stellt das Protokoll der Timestamped Anti-Entropy [Gol92] dar. Das grundsätzliche Vorgehen weicht hierbei von dem der Independent Updates ab.

Die von einem bestimmten Netzknoten an Datenobjekten vorgenommenen Modifikationen werden nicht unmittelbar nach ihrer Durchführung an die übrigen Knoten propagiert. In paarweisen Synchronisationssitzungen werden die Aktualisierungen zwischen den Netzknoten ausgetauscht.

Das Protokoll beschreibt hierfür ein Verfahren zur garantierten Propagation aller Aktualisierungen an alle Netzknoten und Einbringung in korrekter Reihenfolge. Empfängt ein Knoten eine Nachricht, so wird diese vor dem eigentlichen Einbringen in den lokalen Datenbestand in einem Puffer zwischengelagert. Dort verbleibt sie solange, bis die betreffenden Netzknoten alle Aktualisierungen mit einem kleineren Zeitstempel empfangen und eingebracht haben.

6.3.1.3 Anwendbarkeit der Konzepte im Kontext von α-Flow

Für eine angemessene Aktualität der Informationen über den Status des Prozessverlaufs bei allen beteiligten Akteuren wären bei Verwendung der vorgestellten Ansätze im Kontext von α-Flow engmaschige paarweise Synchronisationssitzungen erforderlich. Aufgrund der zulässigen Offline-Zeiten einzelner Akteure sind die Konzepte direkter, paarweiser Synchronisation für die vorliegende Arbeit nicht praktikabel, da nicht garantiert ist, dass die entsprechenden Akteure zu einem bestimmten Zeitpunkt gleichzeitig online sind. Es ist darüber hinaus im Kontext von α-Flow nicht erforderlich, Annahmen über die

Semantik der propagierten Änderungen zu treffen, da immer nur vollständige, in sich konsistente Versionen einer α-Card Gegenstand der Änderungspropagation sind.

6.3.2 Dynamische Verwaltung von Änderungsrechten

Um ein eventuelles Zurücksetzen des Datenbestandes oder aufwendige Zusammenführungen von Datenobjekten zu vermeiden, kann vor dem Durchführen einer Modifikation die Erlaubnis der übrigen Netzknoten eingeholt werden. Damit kann sichergestellt werden, dass nicht zwei Knoten nebenläufig ein Datenobjekt aktualisieren.

Soll eine Änderung durchgeführt werden, so sendet der Urheber eine Anfrage an alle übrigen Knoten. Erlauben diese einstimmig die Modifikation des betreffenden Datenobjekts durch den Anfragenden, so gehen die Änderungsrechte des Datenobjekts auf diesen über. Nun können die gewünschten Aktualisierungen durchgeführt werden, ohne nebenläufige Konflikte mit anderen Netzknoten zu erzeugen.

Nach Beendigung der Änderungsoperationen kann der Urheber freiwillig die Änderungsrechte auf einen anderen Knoten seiner Wahl übertragen. Er kann sie jedoch auch so lange behalten, bis einer Bitte um Änderungsrechte eines anderen Knoten von den übrigen Beteiligten zugestimmt wird.

Dieses sogenannte Token-Konzept kann auch im Kontext von α-Flow Vorteile bieten. Durch Integration eines derartigen Synchronisationsansatzes zusätzlich zum vorhandenen Zeitstempelkonzept könnten bestimmte Fehlersituationen von vornherein vermieden werden. Bei entsprechender, vom verwendeten Kommunikationsprotokoll garantierter, unmittelbarer Erreichbarkeit der beteiligten Akteure kann dies zu einer verbesserten Benutzererfahrung beitragen, da beispielsweise nebenläufige Änderungen von α-Cards ausgeschlossen werden können. Allerdings müssen Verfahren zum Einsatz kommen, die es einzelnen Akteuren weiterhin erlauben, offline zu gehen.

6.4 Zusammenfassung

Die vorgestellten Synchronisationsverfahren zeigen, dass der zuverlässigen Synchronisation und dem schnellen Informationsaustausch in verteilten Systemen im Allgemeinen eine große Bedeutung zukommt. Für ihre korrekte Anwendbarkeit stellen die Verfahren jedoch spezifische Anforderungen an die Beschaffenheit des verwendeten Kommunikationskanals, die vorhandene technische Infrastruktur oder die Erreichbarkeit beteiligter Akteure, die im Kontext von α-Flow nicht erfüllt werden können oder den grundlegenden Konzepten

von α-Flow widersprechen. Dies macht es notwendig, einen eigenständigen Lösungsansatz zur verteilten Datensynchronisation für α-Flow zu konzipieren.

7 Fachlicher Lösungsansatz

Aufbauend auf den bisher präsentierten wissenschaftlichen und technischen Grundlagen, der in Kapitel 5 durchgeführten Analyse der zu bearbeitenden Problemstellungen sowie der Betrachtung verwandter Arbeiten in Kapitel 6 wird der konkrete fachliche Lösungsansatz entwickelt und detailliert vorgestellt.

Abschnitt 7.1 illustriert den internen semantischen Aufbau der zur Kommunikation ausgetauschten Nachrichten. In den Abschnitten 7.2 bis 7.4 werden die Teillösungen für die einzelnen Komponenten und Aspekte des eigentlichen Fachkonzepts erarbeitet.

Dazu werden zunächst die zur zuverlässigen Synchronisation notwendigen Strukturen und Verfahren ausführlich dargestellt. Darüber hinaus wird eine Möglichkeit zur Handhabung der Divergenz zwischen Akteur, Adressinformation und physischem Arbeitsplatz erarbeitet. Abschließend wird das Protokoll zum Beitritt neuer Teilnehmer und der Verbreitung geänderter Adressinformationen stufenweise entwickelt und erläutert.

7.1 Struktur der ausgetauschten Nachrichten

Lokale Änderungen eines Akteurs werden durch den Versand von Nachrichten asynchron an die übrigen Akteure propagiert (vgl. Abschnitt 3.2.3). Diese lokalen Änderungen modifizieren einzelne Attribute und Inhalte des betreffenden Prozessartefakts.

Es erscheint daher naheliegend, anderen Akteuren mitzuteilen, welche Änderungen sie auf ihren lokalen Versionen des Prozessartefakts durchzuführen haben. Aufgrund der in Abschnitt 5.3 beschriebenen erwarteten Eigenschaften des verwendeten Kommunikationskanals ist jedoch damit zu rechnen, dass der erfolgreiche Empfang einer Nachricht nicht automatisch garantiert, dass alle vorhergehenden Nachrichten erfolgreich empfangen worden sind. Ohne Kenntnisse der fachlichen Inhalte kann auf der konzeptionellen Ebene nicht entschieden werden, ob für alle Modifikationen beliebige Kommutativität angenommen werden darf.

Um beim Empfänger aufgrund einer falschen Durchführungsreihenfolge von Aktualisierungen entstandene Inkonsistenzen ausschließen zu können, muss eine geeignete Nachrichtenstruktur gewählt werden (Abbildung 7.1).

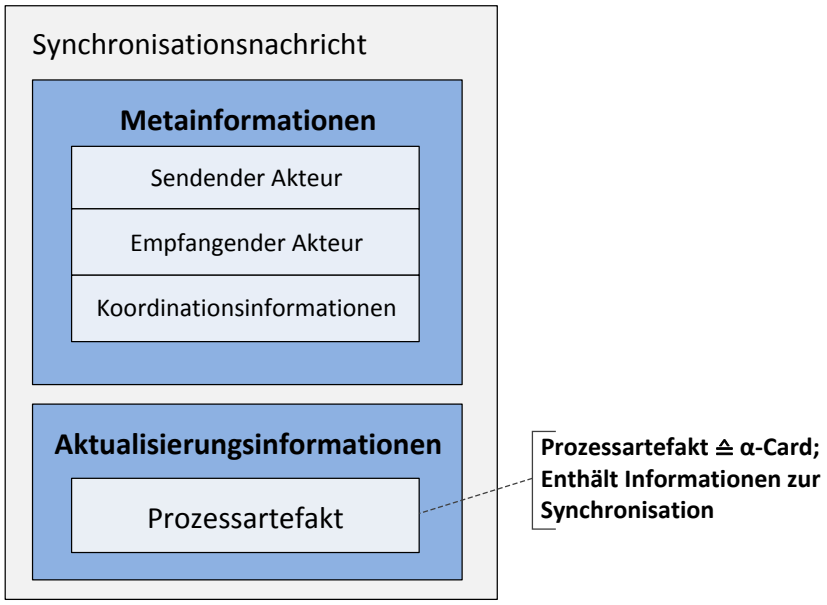


Abbildung 7.1: Struktur der ausgetauschten Synchronisationsnachrichten

Jede propagierte Synchronisationsnachricht ist aus zwei Komponenten aufgebaut: den Metainformationen und den Aktualisierungsinformationen. Erstere enthalten die Informationen, die für die erfolgreiche Zustellung der Nachricht benötigt werden. Dies sind neben den Adressinformationen und identifizierenden Merkmale des Absenders die Adressinformationen des Empfängers . Zusätzlich sind diejenigen Koordinationsinformationen enthalten, die es dem Empfänger erlauben zu entscheiden, ob eine empfangene Nachricht lokal verarbeitet werden muss oder für den modellierten Prozess nicht relevant ist.

Die Aktualisierungsinformationen enthalten eine vollständige Kopie des betreffenden Prozessartefakts. Diese entspricht der Version auf Seiten des Senders nach Durchführung der zu propagierenden lokalen Änderungen. So kann sichergestellt werden, dass sich jedes Prozessartefakt auch bei von der ursprünglichen Sendereihenfolge abweichender Empfangsreihenfolge zu jedem Zeitpunkt beim Empfänger in einem konsistenten Zustand befindet. Informationen, die zur Verwaltung der Synchronisation verteilter Änderungen benötigt werden, sind explizit Teil des Prozessartefakts und werden ebenfalls in jeder Synchronisationsnachricht übermittelt.

Bei α-Flow entspricht ein Prozessartefakt der Gesamtheit von Deskriptor und Nutzinhalt einer α-Card. Die Synchronisationsverwaltungsinformation wird in Form eines logischen Zeitstempels als artefaktspezifisches Attribut im Deskriptor der α-Card gespeichert.

7.2 Synchronisation verteilter Änderungen von Prozessartefakten

Für eine zuverlässige Synchronisation verteilter Änderungen von Prozessartefakten ist es erforderlich, die einzelnen Versionen eines Prozessartefakts in eine Ordnung bringen zu können. Ausgehend von dieser Ordnung kann jedes Prozessartefakt in strukturierter Form versioniert gespeichert werden.

7.2.1 Ordnung der Versionen eines Prozessartefakts

Das im Rahmen dieser Arbeit entwickelte Zeitstempelkonzept ermöglicht es, gemäß den Ausführungen in Abschnitt 3.3.1.2, durch Analyse der artefaktspezifischen logischen Zeitstempel Rückschlüsse auf die kausale Beziehung zwischen Versionen eines Prozessartefakts zu ziehen. Hierfür ist es erforderlich, die Zeitstempel nach einem zuvor definierten Verfahren zu generieren und miteinander zu vergleichen. Das zum Einsatz kommende Verfahren basiert auf den Grundlagen der in Abschnitt 3.3.4 beschriebenen Versionsvektoren.

7.2.1.1 Struktur der Zeitstempel

Ein Zeitstempel, der eine bestimmte Version einer α-Card charakterisiert, bezeichnet im Folgenden eine Datenstruktur V mit n Einträgen, wobei n die Anzahl der partizipierenden Akteure darstellt.
Jeder Eintrag dieser Versionsmap V ordnet hierbei wie bei einem assoziativen Array einem Schlüssel ID_{a_i} einen Wert c_i zu.
ID_{a_i} ist dabei ein Bezeichner, mit dem der i-te Akteur a_i aus der Menge der Akteure A eindeutig zu identifizieren ist.
c_i stellt den diskreten Zähler dar, der die Anzahl der von a_i durchgeführten Modifikationen widerspiegelt. Bei Zugriff auf $V[ID_{a_i}]$ erhält man als Ergebnis c_i, $\forall i$ $(i = 1...n)$.

Zur Illustration dieser formalen Beschreibung ist in Abbildung 7.2 eine nach obigem Schema aufgebaute Versionsmap dargestellt. Es ist zu erkennen, dass drei Akteure an der Modifikation des zugehörigen Prozessartefakts beteiligt sind. Akteur A hat drei Änderungen durchgeführt und Akteur B eine. C hat das Artefakt zweimal modifiziert.

Da die Anzahl der teilnehmenden Akteure des zu modellierenden Prozesses initial nicht zwingend bekannt sein muss, unterstützt das Zeitstempelkonzept zu jedem Zeitpunkt das dynamische Hinzufügen weiterer Einträge zur Versionsmap eines Prozessartefakts. Hierbei

werden jedoch nur diejenigen Akteure durch einen eigenen Eintrag repräsentiert, die das zugehörige Prozessartefakt mindestens einmal bearbeitet haben, da nur schreibende Zugriffe auf Prozessartefakte synchronisiert werden müssen.

Formal ausgedrückt bedeutet dies, dass es nicht erforderlich ist, ein Tupel (ID_{a_i}, c_i) in den Zeitstempel aufzunehmen, falls für a_i gilt $c_i = 0$. Existiert der Eintrag $V[ID_{a_i}]$ nicht, darf bei der weiteren Verarbeitung des Zeitstempels somit umgekehrt $c_i = 0$ implizit angenommen werden.

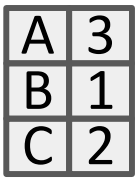


Abbildung 7.2: Beispiel eines Zeitstempels

7.2.1.2 Bestimmung der kausalen Beziehung zwischen Zeitstempeln

Zum Herstellen einer Ordnung auf einer Menge verschiedener Zeitstempel ist es notwendig, die Beziehung zwischen einzelnen Zeitstempeln charakterisieren zu können. Prinzipiell sind dabei die folgenden Möglichkeiten für die Beziehung zweier n-elementiger Zeitstempel V_1 und V_2 von Interesse:

1. $V_1 \equiv V_2$: beide Zeitstempel sind identisch
2. $V_1 > V_2$: V_1 besitzt eine kausale Abhängigkeit zu V_2
3. $V_1 < V_2$: V_2 besitzt eine kausale Abhängigkeit zu V_1
4. $V_1 || V_2$: V_1 und V_2 stehen in keinem kausalen Zusammenhang, sind also das Ergebnis nebenläufiger Modifikationen

Um eindeutig klassifizieren zu können, welcher der obigen vier Fälle für zwei konkrete Zeitstempel zutreffend ist, müssen diese elementweise miteinander verglichen werden. Bei diesem Vergleich kann auf Basis der folgenden Regeln eine Entscheidung getroffen werden:

1. $V_1 \equiv V_2$ genau dann, wenn $\forall i$ (i=1...n): $V_1[ID_{a_i}] = V_2[ID_{a_i}]$
2. $V_1 > V_2$ genau dann, wenn $\forall i$ (i=1...n): $V_1[ID_{a_i}] \geq V_2[ID_{a_i}]$ und $\neg(V_1 \equiv V_2)$
3. $V_1 < V_2$ genau dann, wenn $\forall i$ (i=1...n): $V_1[ID_{a_i}] \leq V_2[ID_{a_i}]$ und $\neg(V_1 \equiv V_2)$

4. $V_1 || V_2$ genau dann, wenn $\neg(V_1 < V_2)$ und $\neg(V_2 < V_1)$

Veranschaulicht werden diese Berechnungen durch die in Abbildung 7.3 dargestellten Paare von Zeitstempeln inklusive ihrem kausalen Zusammenhang.

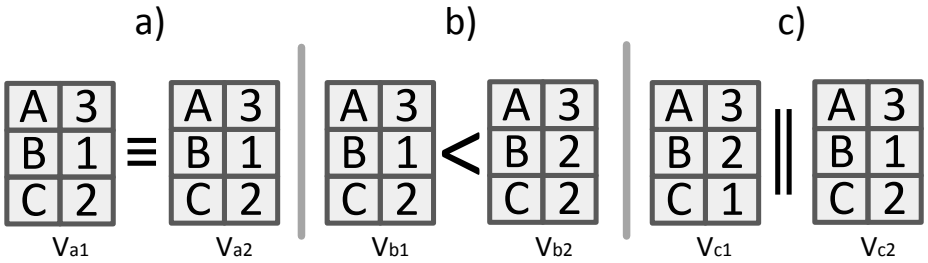


Abbildung 7.3: Beispiele für die kausalen Beziehungen zwischen Zeitstempeln

Die beiden Zeitstempel V_{a1} und V_{a2} aus *(a)* sind identisch, da sie in allen Einträgen übereinstimmen. Der rechte Zeitstempel V_{b2} aus *(b)* dominiert den linken Zeitstempel V_{b1} aufgrund der Tatsache, dass beide identische Einträge für die Akteure A und C enthalten, der rechte Zeitstempel jedoch den größeren Zählerwert für Akteur B aufweist. Die Zeitstempel aus *(c)* zeugen von einem nebenläufigen Änderungskonflikt, da keiner von beiden den anderen dominieren kann: Es gilt $V_{c1}[B] > V_{c2}[B]$, aber gleichzeitig auch $V_{c1}[C] < V_{c2}[C]$.

7.2.1.3 Generierung der Zeitstempel

Gültige Versionsmaps können unter Beachtung der folgenden Regeln erzeugt werden. Die Ausführungen beziehen sich auf den, zu einem Prozessartefakt p gehörenden, Zeitstempel $V_{p_{a_i}}$. Dieser ist lokal bei einem Akteur a_i aus A gespeichert.

1. Bei jeder Veränderung von p durch a_i erhöht dieser den ihm zugeordneten Zähler:

   $V_{p_{a_i}}[ID_{a_i}] := 1$, falls $V_{p_{a_i}}[ID_{a_i}] = 0$ (Erzeugung des Artefakts)
   $V_{p_{a_i}}[ID_{a_i}] := V_{pi}[ID_{a_i}] + 1$, sonst

2. Bei Empfang einer von Teilnehmer a_j versandten Version von p, die den Zeitstempel $V_{p_{a_j}}$ trägt, aktualisiert a_i seinen lokalen Zeitstempel, falls ein nebenläufiger Änderungskonflikt entdeckt wurde:

   $V_{p_{a_i}} := \sup(V_{p_{a_i}}, V_{p_{a_j}})$, wobei $\sup(V_{p_{a_i}}, V_{p_{a_j}})$ das elementweise Maximum beider Versionsmaps darstellt, d.h $\sup(V_{p_{a_i}}, V_{p_{a_j}}) = w$ mit $w[a_h] = \max(V_{p_{a_i}}[h], V_{p_{a_j}}[h])$ $\forall h\ (h = 1...n)$

Regel 2 ist erforderlich, um nach Behebung eines Änderungskonflikts zu vermerken, wie viele Modifikationen insgesamt bereits auf dem Artefakt durchgeführt wurden. Dadurch können auch zukünftig auftretende Konflikte erkannt werden.

Folgendes Beispiel (Abbildung 7.4) eines Kommunikationsszenarios zwischen zwei Teilnehmern, bei dem Aktualisierungen zu einem einzigen Prozessartefakt ausgetauscht werden, illustriert die formalen Regeln zur Generierung der Zeitstempel.

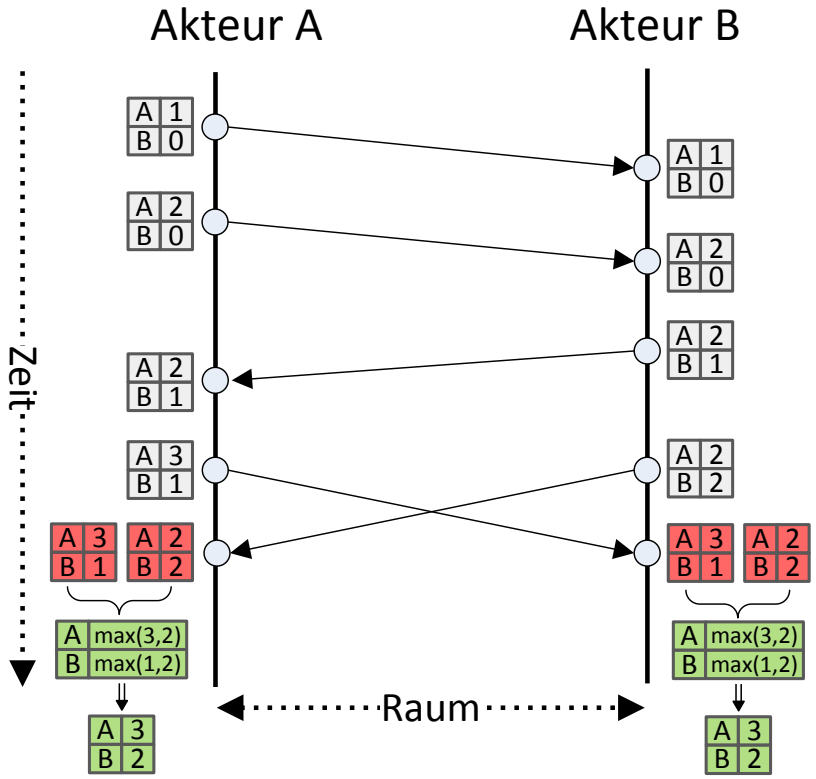


Abbildung 7.4: Beispiel für die Generierung von Zeitstempeln

Bei den ersten drei propagierten Aktualisierungen inkrementiert der jeweilige Urheber, beginnend bei null, seinen Eintrag um eins. Danach modifizieren A und B nebenläufig das Artefakt, ohne über die Änderung des Gegenübers informiert zu sein.

Der Vergleich beim Empfang der entsprechenden Nachrichten resultiert auf beiden Seiten in der Erkennung eines Änderungskonflikts (rote Markierung). Deswegen wird zur Sicherung der Konsistenz bei zukünftigen Änderungen von beiden Akteuren das elementweise Maximum der miteinander in Konflikt stehenden Zeitstempel gebildet (grüne Markierung) und als neuer Zeitstempel gespeichert. Neuer gültiger Zählerwert von Akteur A ist $max(3,2) = 3$, der neue Wert von Akteur B beträgt $max(1,2) = 2$. Auf Basis der entstandenen Versionsmap können nun wieder weitere gültige Versionen erzeugt werden.

7.2.2 Versionierung und Sicherung der Provenienz von Prozessartefakten

Ausgehend von den bisher erarbeiteten Grundlagen folgt die Darstellung des entwickelten Verfahrens zur Synchronisation verteilter Änderungen unter Verwendung des vorgestellten Konzepts logischer Zeitstempel. Dabei ist es den einzelnen Akteuren möglich, die Existenz globaler Änderungskonflikte ausschließlich auf Grundlage der ihnen lokal verfügbaren Informationen zu erkennen und diese Konflikte aufzulösen. Um zu einem späteren Zeitpunkt den Entstehungsprozess von Prozessartefakten nachvollziehen zu können, ist es erforderlich, ihre Provenienz zu dokumentieren. Das bedeutet, dass die vollständige Änderungshistorie aller Prozessartefakte für weitere Auswertungen persistent gesichert wird.

Voraussetzung für die Anwendbarkeit des Verfahrens ist die Einhaltung der Anforderungen an den Kommunikationskanal (vgl. Abschnitt 5.3). Zudem werden geeignete applikationsspezifische Datenstrukturen für die Verwaltung der Persistenz aller Versionen eines Prozessartefakts benötigt. Diese Datenstrukturen müssen die Möglichkeit bieten, die kausalen Beziehungen zwischen einzelnen Versionen analysieren zu können. Daneben muss das Navigieren entlang der Änderungshistorie zwischen den Vorgängern und Nachfolgern einzelner Versionen eines Prozessartefakts realisiert werden können.

7.2.2.1 Vervollständigung der Änderungshistorie

Mithilfe logischer Zeitstempel kann geprüft werden, auf welche Weise Versionen eines Prozessartefaktes kausal voneinander abhängen. Betrachtet werden in diesem Abschnitt die beiden Fälle, die auftreten wenn Sende- und Empfangsreihenfolge von Aktualisierungsnachrichten divergieren:

1. Eine Nachricht mit Versionsmap V_2 kommt beim Empfänger an, bevor eine Nachricht mit Zeitstempel V_1 eintrifft, die zu einem früheren Zeitpunkt versandt wurde.
2. Eine Nachricht mit Versionsmap V_1 kommt beim Empfänger an, nachdem eine Nachricht mit Zeitstempel V_2 eintrifft, die zu einem späteren Zeitpunkt versandt wurde.

In beiden Fällen darf die beim Empfänger vorhandene Version nicht ohne weiteres durch das ankommende Prozessartefakt ersetzt werden. Vielmehr ist es gewollt, dass die ankommende Version an der richtigen Position in die Historie der unterschiedlichen Versionen einsortiert wird. Dadurch ist sichergestellt, dass die Entstehungsgeschichte

eines Prozessartefaktes dokumentiert werden kann und stets die tatsächlich aktuelle Version an oberster Stelle der Änderungshistorie steht.

Um diese Position korrekt bestimmen zu können, kann der Abstand zwischen einzelnen Versionen berechnet werden. Der Abstand definiert hierbei die Anzahl von Versionen, die sich in der Änderungshistorie zwischen den betrachteten Versionen befinden.

Folgende Formel erlaubt die Bestimmung des Abstandes $dist(V_1, V_2)$ zwischen zwei Versionen mit den n-elementigen Zeitstempeln V_1 und V_2:

$$dist(V_1, V_2) = \left(\sum_{i=1}^{n} V_2[ID_{a_i}]\right) - \left(\sum_{i=1}^{n} V_1[ID_{a_i}]\right) - 1$$

Zu beachten ist, dass der Abstand zwischen V_1 und V_2 nur berechnet werden kann, falls $\neg(V_1||V_2)$ gilt, da der Abstand zwischen miteinander in Konflikt stehenden Zeitstempeln undefiniert ist. Der Fall $V_1||V_2$ wird gesondert in Abschnitt 7.2.2.2 diskutiert.

Ist der Abstand zwischen zwei Zeitstempeln V_1 und V_2 gleich null, kann die ankommende Version direkt als neues aktuelles Element in die lokale Änderungshistorie eingefügt werden. Ist der Abstand jedoch größer null, so bedeutet dies, dass alle Versionen zwischen der lokal aktuellen und der ankommenden lokal noch nicht verfügbar sind, die Änderungshistorie also unvollständig ist.

Aufgrund der Eigenschaften des verwendeten Kommunikationskanals kann jedoch sicher mit einem Eintreffen der Nachrichten mit den fehlenden Versionen des Prozessartefakts zu einem späteren Zeitpunkt gerechnet werden. Um diese Versionen nachträglich an die richtige Position innerhalb der Historie einfügen zu können, ist es erforderlich, im Voraus entsprechende Lücken in der Historie freizulassen. Zu diesem Zweck werden in der Datenstruktur zur Verwaltung der Versionen insgesamt $dist(V_1, V_2)$ Versionen ohne Nutzinhalt angelegt. Im Anschluss an diese Folge von Platzhalterversionen kann die ankommende Version schließlich eingefügt werden.

Ist der Abstand negativ bzw. gilt $V_1 > V_2$, so wird in der Datenstruktur $dist(V_1, V_2) + 1$ Schritte in Richtung der Vorgänger des Vergleichspartners navigiert. Mit der ankommenden Version wird anschließend eine an dieser Position vorhandene Lücke in der Historie geschlossen.

Wie die korrekte Einordnung ankommender Versionen gemäß obiger Regeln im Detail funktioniert, wird anhand eines einfachen Kommunikationsszenarios in Abbildung 7.5a aufgezeigt. Die ersten beiden von A propagierten Versionen V_1 und V_2 des zugehörigen Prozessartefakts werden in korrekter Reihenfolge an B ausgeliefert. Die darauf folgenden drei Aktualisierungen V_3 bis V_5 von A erreichen B jedoch aufgrund von Verzögerungen auf dem Transportweg in umgekehrter Reihenfolge. B muss deswegen beim Empfang die

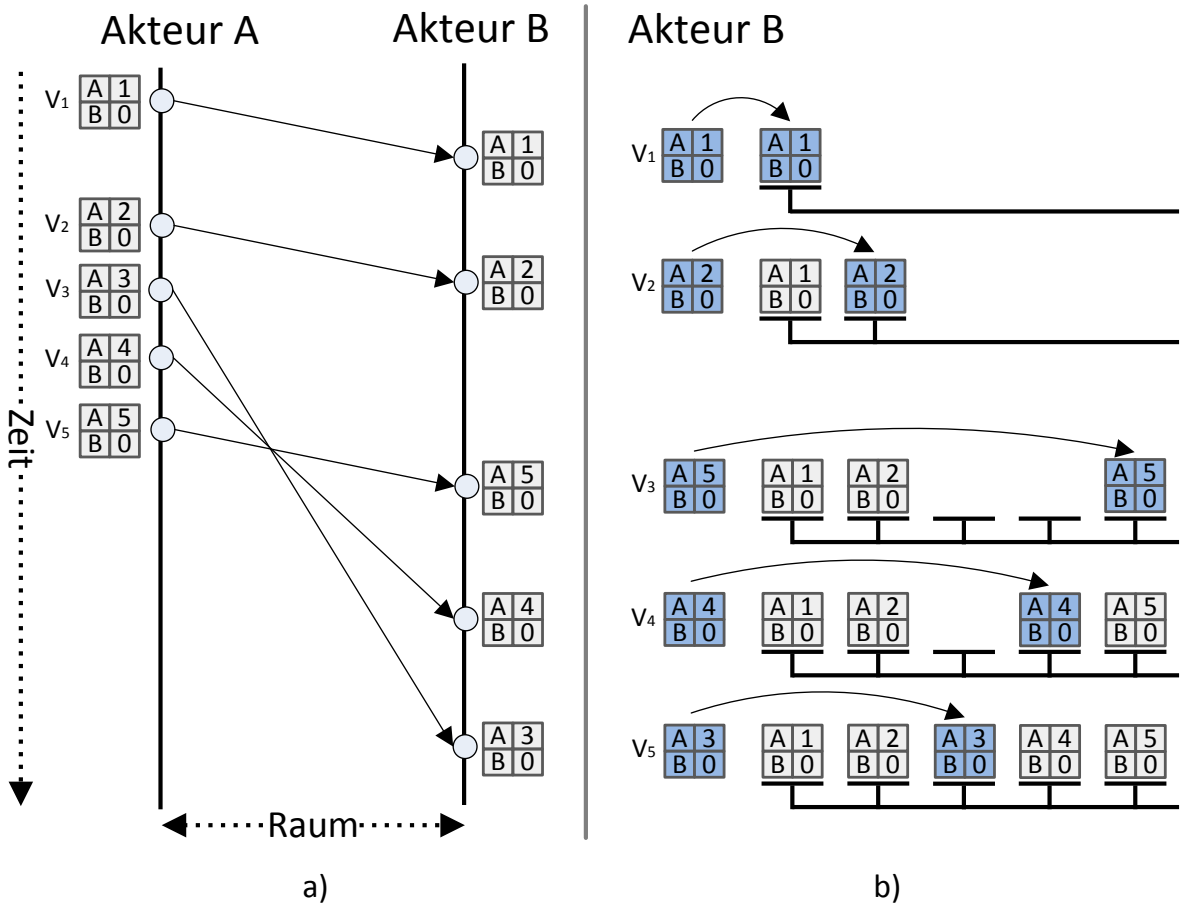


Abbildung 7.5: Kommunikationsszenario zur Vervollständigung der Änderungshistorie

ankommenden Versionen an der korrekten Position in seiner lokalen Historie ablegen. Dies resultiert nach jeder empfangenen Nachricht in einer veränderten Struktur der Historie, deren schrittweise Entwicklung in Abbildung 7.5b dargestellt ist (blaue Markierungen kennzeichnen die jeweils ankommende Version).

Zu Beginn sind noch keine Versionen des Prozessartefakts in der lokalen Historie von B enthalten. Die erste ankommende Version V_1 kann an erster Stelle eingefügt werden, da sie die initiale Erstellung des Prozessartefakts anzeigt.

Der Abstand der zweiten ankommenden Version V_2 zur lokal gespeicherten beträgt gemäß obiger Formel null. Deshalb wird die Nachricht unmittelbar an der nächsten freien Position eingefügt.

Die nächste empfangene Version V_3 weist einen Abstand von zwei zur lokal aktuellen Version auf. Es werden zwei Platzhalter erzeugt und die Version an fünfter Stelle der Historie eingefügt.

Mit dem Empfang der beiden verbleibenden Nachrichten mit den Versionen V_4 und V_5 können die vorhandenen Lücken geschlossen werden. Die Änderungshistorie von B ist nun vollständig und weist keine leeren Platzhalter mehr auf.

7.2.2.2 Behandlung von nebenläufigen Änderungskonflikten

Neben Nachrichten, die in geänderter Reihenfolge ankommen, sind auch parallele Änderungskonflikte zu behandeln. Den Akteuren muss es ermöglicht werden, im Nachhinein nachvollziehen zu können, welche Entscheidungen im Prozessverlauf aufgrund inkonsistenter Informationen getroffen wurden. Dafür ist es erforderlich, alle konfliktbehafteten, aufgrund nebenläufiger Modifikationen entstandenen Versionen zu Dokumentationszwecken aufzubewahren.

Erzeugung von Konfliktzweigen und Zurücksetzen auf letzte gültige Version

Die bisher beschriebene Struktur von Versionshistorien sieht einen streng linearen Aufbau vor. Zur Behandlung nebenläufiger Änderungskonflikte muss dieses lineare Schema der Historie um ein zusätzliches Konzept erweitert werden.

Zur Wahrung der Provenienz eines Prozessartefakts ist es bei der Erkennung von Änderungskonflikten notwendig, Abzweigungen in der Historie zu erstellen. Diese Abzweigungen erlauben es, konfliktbehaftete Versionen beim Navigieren durch die Historie zu umgehen, ohne sie vollständig aus dieser entfernen zu müssen.

Dafür werden alle Versionen, die durch die Entdeckung paralleler Änderungen anderer Teilnehmer ungültig geworden sind, in einen sogenannten Konfliktzweig verschoben. Dies sind alle Versionen, die zwischen der ersten konfliktbehafteten und der ersten konfliktfreien Version in der Historie liegen.

Es wird daraufhin als lokal aktuelle Version eine neue Version erzeugt, die als Wurzel des Konfliktzweiges fungiert. Der Zeitstempel dieser Version muss der Regel zur Generierung von Zeitstempeln bei nebenläufigen Änderungskonflikten entsprechen.

Daneben muss jedoch auch ein gültiger, nicht konfliktbehafteter Nutzinhalt für die neue Version gewählt werden. Unter der Annahme, dass nebenläufige Änderungskonflikte nur sehr selten auftreten und jede lokale Versionshistorie die letzte global gültige Version enthält, ist es naheliegend, das Prozessartefakt auf diese Version zurückzusetzen. Bei dieser Strategie wird der Nutzinhalt der letzten gültigen Version an die Position der neu erzeugten Version in der Historie kopiert, um den dort gespeicherten Inhalt zu ersetzen.

Zum Auffinden des letzten global gültigen Vorgängers einer Version $V_{konflikt}$ wird der Operator $sub(V_{konflikt})$ definiert. Dieser identifiziert, beispielsweise mit Hilfe einer

navigierenden Suche entlang der Versionshistorie, diejenige Version $V_{konfliktfrei}$, die den folgenden Bedingungen genügt:

1. $V_{konfliktfrei} < V_{konflikt}$
2. $\nexists V_{konfliktfrei'} < V_{konflikt} : V_{konfliktfrei'} > V_{konfliktfrei}$

Das Anlegen eines solchen Konfliktzweiges und die Anwendung des *sub*-Operators werden im Folgenden detailliert beschrieben. Der Darstellung liegt das Kommunikationsszenario aus Abbildung 7.6a zugrunde.

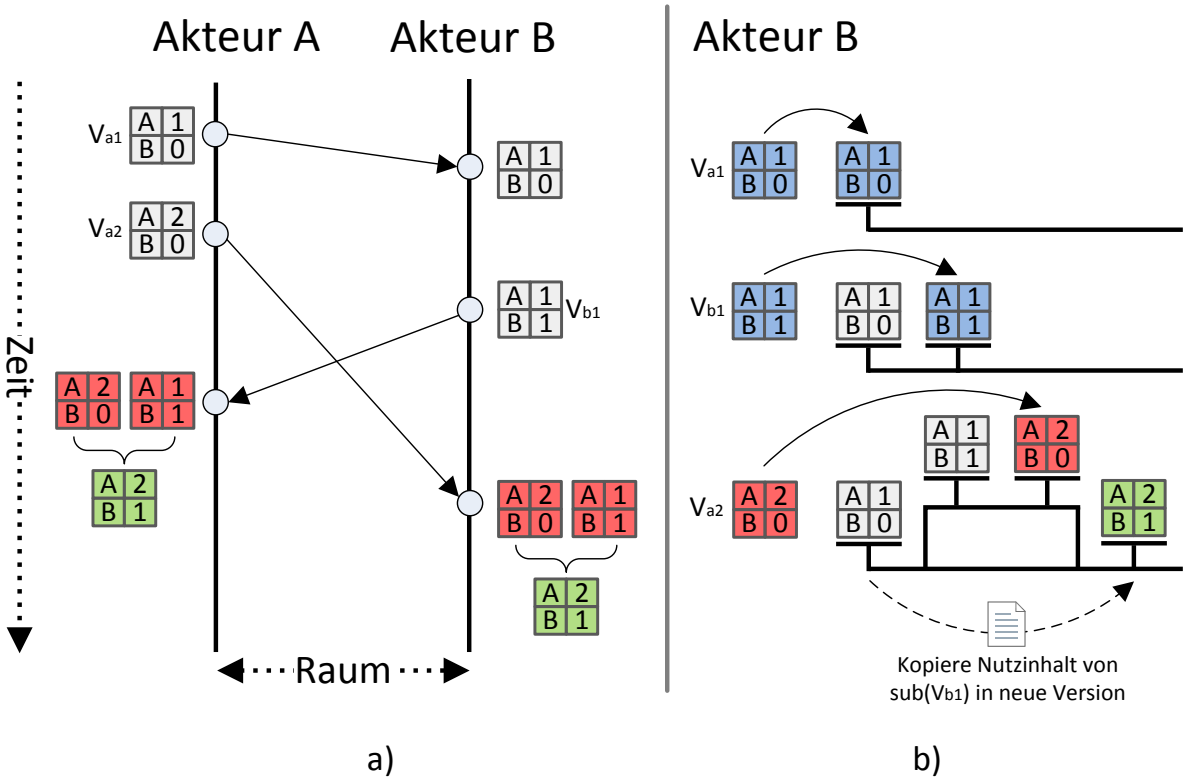


Abbildung 7.6: Behandlung nebenläufiger Änderungskonflikte: Erzeugung eines Konfliktzweiges

Nach der erfolgreichen Propagation der ersten Nachricht von A, die Version V_{a1} enthält, führen beide Akteure nebenläufig je eine Modifikation des Prozessartefakts durch, ohne die Änderung des jeweiligen Gegenübers zu kennen. Dies führt auf beiden Seiten unter Zuhilfenahme der Auswertungsvorschrift aus Abschnitt 7.2.1 zur Erkennung eines parallelen Änderungskonflikts und zu entsprechenden Anpassungen der Historie. Am Beispiel von Akteur B (Akteur A verfährt analog) zeigt Abbildung 7.6b die Entwicklung der Versionshistorie unter Verwendung eines Konfliktzweiges.

Bevor die konfliktbehaftete Version V_{a2} (rote Markierung) bei B eintrifft, wird die Historie mit den Versionen V_{a1} und V_{b1} gefüllt. Nach Erkennen des Änderungskonflikts wird mit Hilfe des Operators $sub(V_{b1})$ nach der letzten gültigen Version in der Historie gesucht, zwischen der und V_{b1} kein Konflikt vorliegt. Im Beispiel ist das die erste von A propagierte Version V_{a1}, da $V_{a1}[A] < V_{b1}[A]$ und $V_{a1}[B] < V_{b1}[B]$ gelten.

Nun wird eine neue Version erzeugt (grüne Markierung), deren Zeitstempel das elementweise Maximum aus V_{a2} und der bis dahin aktuellsten lokalen Version V_{b1} ist. Alle vorhandenen Versionen, die zwischen der ersten konfliktfreien und der neu erzeugten liegen, werden in einen Konfliktzweig ausgelagert.

Auch die ankommende Version wird diesem Zweig hinzugefügt. So sind die ungültig gewordenen Versionen V_{b1} und V_{a2} weiterhin auffindbar, ohne das Navigieren entlang des Pfades der gültigen Versionen zu beeinflussen. Danach wird der Nutzinhalt von $sub(V_{b1}) = V_{a1}$ an die Position der neu erzeugten Version kopiert, um die Konfliktlösung abzuschließen.

Erweiterung von Konfliktzweigen

Nach der einmaligen Erzeugung eines Konfliktzweiges können auch alle später eintreffenden, konfliktbehafteten Versionen entsprechend behandelt werden. Da der Konfliktzweig hier bereits existiert, muss die ankommende Version diesem lediglich hinzugefügt und der Zeitstempel der lokal aktuellen Version angepasst werden. Abbildung 7.7 illustriert diesen Vorgang anhand einer zeitlichen Fortsetzung des zuvor dargestellten Kommunikationsszenarios.

Es wird in Abbildung 7.7a eine Situation betrachtet, in der Akteur A nicht nur eine Änderung durchführt, die in Konflikt mit der von Akteur B erzeugten Version V_{b1} steht, sondern zwei weitere konfliktbehaftete Versionen V_{a3} und V_{a4} propagiert. Beim Empfang von V_{a3} entdeckt B einen nebenläufigen Änderungskonflikt. Die Auswertung von $sub(V_{a3})$ ergibt, dass die Version V_{a1} der letzte gültige Vorgänger der ankommenden Version ist.

Da die lokal aktuelle Version bereits Wurzel eines Konfliktzweiges ist, der bei V_{a1} beginnt, kann auf das Kopieren des Nutzinhalts der gültigen Version verzichtet werden (Abbildung 7.7b). Es wird daher ausschließlich der Zeitstempel der lokal aktuellen Version aktualisiert (grüne Markierung) und V_{a3} am Ende des Konfliktzweiges in die Historie eingefügt (rote Markierung). Selbiges Vorgehen ist auch beim Empfang der ebenfalls konfliktbehafteten Version V_{a4} erforderlich, um die Versionshistorie in einem konsistenten Zustand zu hinterlassen.

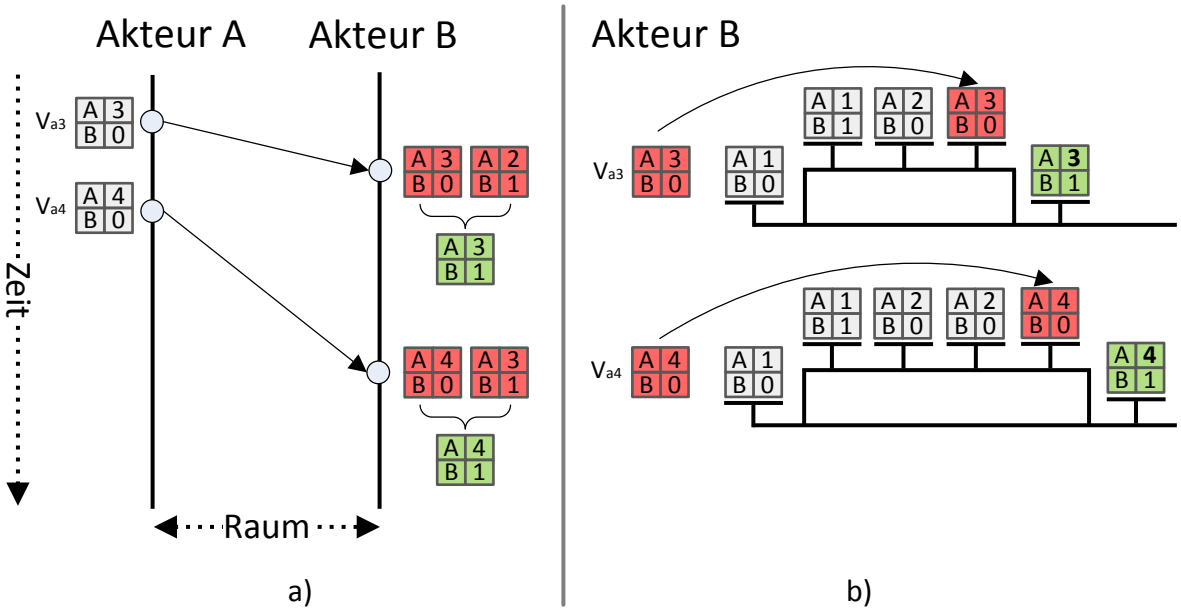


Abbildung 7.7: Kommunikationsszenario zur Behandlung nebenläufiger Änderungskonflikte: Erweiterung eines Konfliktzweiges

Konfliktlösungsstrategien für den Nutzinhalt

Eine Möglichkeit zur Auflösung von nebenläufigen Änderungskonflikten stellt die erläuterte intuitive Strategie des Zurücksetzens auf die letzte global gültige Version dar. Daneben existieren verschiedene alternative Strategien, um den Nutzinhalt eines Prozessartefakts nach dem Erkennen eines Konflikts wieder in einen konsistenten Zustand zu bringen.

So ist es möglich, für bestimmte Typen von Prozessartefakten eine automatische Zusammenführung mehrerer Versionen in Betracht zu ziehen. Bei der Zusammenführung von mehr als zwei Versionen muss jedoch die Kommutativität der auf den einzelnen Versionen durchgeführten Änderungen beachtet werden, um eine semantisch sinnvolle Version zu erzeugen. Denkbar ist auch die Priorisierung bestimmter Akteure derart, dass ihre Aktualisierungen diejenigen von anderen Teilnehmern überschreiben. Zudem kann in jedem Fall der Benutzer aufgefordert werden, durch manuellen Vergleich der in Konflikt stehenden Versionen, eine neue gültige Version zu erzeugen.

Kommt eine nicht-deterministische Strategie zum Einsatz, so wird es erforderlich sein, das Ergebnis der lokalen Konfliktlösung als eigenständige Version des Prozessartefakts an die übrigen Akteure zu propagieren. Durch Priorisierung einer der individuellen Versionen nach bestimmten Kriterien kann sichergestellt werden, dass global bei allen Akteuren dieselbe gültige Version verfügbar ist.

In Abschnitt 11.2 wird näher auf zukünftige Möglichkeiten einer automatischen Zusammenführung von Prozessartefakten im Kontext von α-Flow eingegangen. Weitere Überlegungen betreffend die Lösung von Konflikten für den Nutzinhalt von Prozessartefakten sind nicht Gegenstand dieser Arbeit.

7.2.2.3 Gesamtüberblick über das vorgestellte Verfahren

Abschließend wird das in den vorangegangenen Abschnitten vorgestellte Verfahren zur Verwaltung der Versionshistorie eines Prozessartefakts in formalisierter Form als Pseudocode dargestellt. Dies ermöglicht eine höhere Übersichtlichkeit sowie das leichte Nachvollziehen aller durchgeführten Schritte.

Im Vergleich zu den bisher dargestellten Situationen sind in der nachfolgenden Beschreibung Verhaltensweisen bei zwei zusätzlichen Sonderfällen hinzugekommen. Diese umfassen das Verhalten beim Empfang bereits lokal verfügbarer Versionen und beim Einfügen verspätet empfangener Versionen in einen Konfliktzweig.
$V_{aktuell}$ steht für den Zeitstempel der lokal aktuellsten Version, bei $V_{ankommend}$ handelt es sich um den Zeitstempel der einzufügenden Version.

Wenn $V_{aktuell} \equiv V_{ankommend}$ **Dann**

- Beende den Einfügevorgang.

Wenn $V_{aktuell} < V_{ankommend}$ **Dann**

- Erzeuge ausgehend von $V_{aktuell}$ genau $dist(V_{aktuell}, V_{ankommend})$ viele Platzhalter.
- Füge $V_{ankommend}$ dahinter in die Historie ein.

Wenn $V_{aktuell} > V_{ankommend}$ **Dann**

- Suche in der Historie die Version $V_{vorgaenger} := sub(V_{ankommend})$.
- **Wenn** Beim ersten Nachfolger von $V_{vorgaenger}$ der kein Platzhalter ist, handelt es sich um die Wurzel eines Konfliktzweiges. **Dann**
  - Füge $V_{ankommend}$ dem Konfliktzweig hinzu.

  Sonst
  - Speichere $V_{ankommend}$ an der Platzhalterposition $dist(V_{vorgaenger}, V_{ankommend}) + 1$ Schritte von $V_{vorgaenger}$ ab.

Wenn $V_{aktuell} || V_{ankommend}$ **Dann**

- Erzeuge als neue lokal aktuelle Version $V_{konfliktloesung}$ mit Zeitstempel sup ($V_{ankommend}$,$V_{aktuell}$) sowie Nutzinhalt gemäß gewählter Konfliktlösungsstrategie[1].
- Erzeuge einen Konfliktzweig, der alle Versionen zwischen $V_{konfliktfrei}$ und $V_{konfliktloesung}$ enthält.
- Füge $V_{ankommend}$ dem Konfliktzweig hinzu.

7.3 Divergenz von Akteur, Adressinformation und physischem Arbeitsplatz

Aus Sicht eines beteiligten Akteurs findet der Austausch von Informationen über den Prozessverlauf direkt durch elektronische Kommunikation mit den übrigen Akteuren statt. Die aus den Überlegungen in Abschnitt 5.4.3 resultierende Divergenz von Akteur, seinen Adressinformationen und seinen physischen Arbeitsplätzen muss konzeptionell erfasst und aufgelöst werden, um eine zuverlässige Prozesskoordinierung garantieren zu können. Im Folgenden wird eine Möglichkeit vorgestellt, um unter Wahrung der Konsistenz der Synchronisation die Verwaltung mehrerer Replikate des aktiven Dokuments pro Akteur zu ermöglichen.

7.3.1 Mögliche Probleme bei Replikation des aktiven Dokuments

In Konformität zu den Ausführungen in Abschnitt 7.2.1.1 wird für jeden Akteur ein eigener Eintrag im logischen Zeitstempel angelegt, der durch einen eindeutigen Bezeichner für diesen Akteur identifiziert wird. Für die Bildung dieses eindeutigen Bezeichners werden zunächst die folgenden Informationen herangezogen:

1. Die Bezeichnung des Akteurs

1 Für die Verwendung der beschriebenen Strategie des Zurücksetzens auf die letzte gültige Version sind folgende Schritte durchzuführen:

- Suche in der Historie $V_{konfliktfrei} := sub(V_{ankommend})$.
- Wähle den Nutzinhalt von $V_{konfliktfrei}$ als neuen Nutzinhalt von $V_{konfliktloesung}$.

2. Die Bezeichnung der Institution, für die der Akteur am Prozessgeschehen teilnimmt
3. Die Bezeichnung der Rolle, die der Akteur im Prozessverlauf einnimmt

Situationen, in denen ein einziger Akteur über mehr als genau ein Replikat des aktiven Dokuments verfügt, bergen unter Verwendung von aus obigen Informationen aufgebauten Bezeichnern die Gefahr von Problemen bei der Synchronisierung mit anderen Teilnehmern. Eine mögliche Problematik dieser Art ist in Abbildung 7.8 dargestellt.

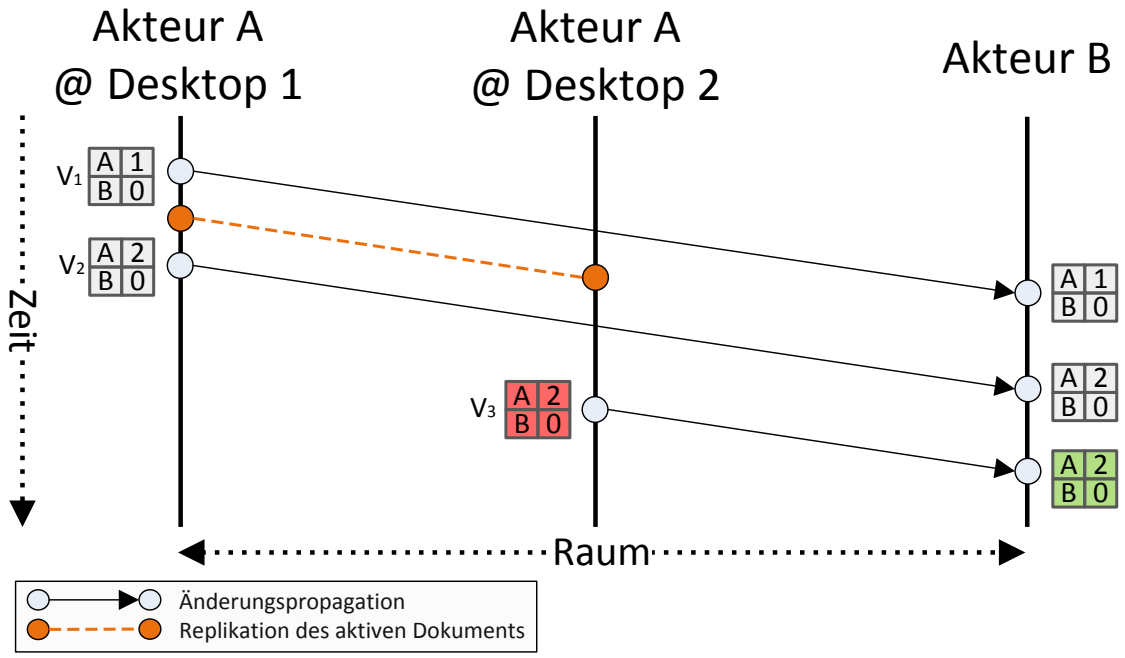


Abbildung 7.8: Mögliche Probleme bei Modifikationen durch einen Akteur von verschiedenen Arbeitsplätzen aus

Das Beispiel erfasst den zeitlichen Verlauf der Propagation von Versionen eines einzelnen Prozessartefaktes. Akteur A bearbeitet dieses Artefakt und somit auch das zugehörige aktive Dokuments sowohl von Desktop 1 aus als auch von Desktop 2. Akteur B agiert von einem einzigen physischen Arbeitsplatz aus. Dabei wird im logischen Zeitstempel des Prozessartefakts jeweils ein Eintrag pro beteiligtem Akteur angelegt.

Das Erstellen des Artefakts (Zeitstempel V_1) an Desktop 1 wird korrekt an B propagiert. Daraufhin erstellt A ein Replikat des aktiven Dokuments zur späteren Verwendung an Desktop 2.

Die nachfolgende Änderungsoperation (Zeitstempel V_2) wird korrekt von Akteur B empfangen. Die Kopie des aktiven Dokuments an Desktop 2 erhält jedoch die Nachricht nicht zugestellt, da das aktive Dokument an Desktop 1 keinen Überblick über existierende

Replikate seiner selbst hat und Nachrichten nicht zusätzlich an die Adresse des Absenders geschickt werden.

Dies führt zu der Situation, dass A beim Durchführen einer weiteren Änderung, diesmal von Desktop 2, eine Nachricht mit einem bereits zuvor aufgetretenen logischen Zeitstempel V_3 an B versendet. Somit ist es für Akteur A nicht möglich zu erkennen, dass ein paralleler Änderungskonflikt zwischen den Replikaten an Desktop 1 und Desktop 2 existiert. Durch diesen unentdeckten Konflikt ist ab diesem Zeitpunkt kein konsistentes Weiterarbeiten mehr möglich, da A und B nicht mehr über synchronisierte Informationen verfügen.

7.3.2 Lösungsansatz zur Vermeidung von Inkonsistenzen

Um die Konsistenz der Synchronisation zu wahren, muss das mehrfache Auftreten einer identischen Versionsmap zu verschiedenen Zeitpunkten vermieden werden. Hierzu wird zur Identifizierung von Einträgen des Zeitstempels zusätzlich ein Identifikator des aktuellen Rechners einbezogen, um zwischen verschiedenen physischen Arbeitsplätzen eines Akteurs unterscheiden zu können. Als möglicher Identifikator für einen Rechner bietet sich beispielsweise die eindeutige Hardware-Adresse der verbauten Netzwerkhardware an. Jeder Zeitstempel enthält somit für jede tatsächlich auftretende Kombination aus Akteur und physischem Arbeitsplatz einen Eintrag.

Darüber hinaus müssen alle Nachrichten zusätzlich an die Adresse des Absenders übermittelt werden, damit weitere Replikate des aktiven Dokuments, die über die selben Adressinformationen kontaktiert werden können, an anderen physischen Arbeitsplätzen über sämtliche durchgeführten Aktualisierungen informiert werden.

Abbildung 7.9 illustriert, wie unter Beachtung dieser Änderungen des Synchronisationsverfahrens der im obigen Szenario von Teilnehmer A verursachte parallele Änderungskonflikt zwischen Desktop 1 und Desktop 2 zuverlässig vermieden werden kann. Die Beobachtungen lassen sich durch Anwendung der in Kapitel 7.2 beschriebenen Grundlagen analog auf ähnlich gelagerte Problemfälle übertragen.

Akteur A propagiert zu Beginn die Erstellung des Prozessartefakts (Zeitstempel V_1) erfolgreich an Akteur B. Anschließend repliziert A das aktive Dokument und transferiert es zu Desktop 2.

Die durchgeführte Änderung mit der Versionsmap V_2 wird nun nicht nur an die Adresse von Akteur B übermittelt, sondern auch an die Adresse des Absenders A. Das erlaubt es der Kopie des aktiven Dokuments an Desktop 2, die Nachricht in Empfang zu nehmen und das Prozessartefakt lokal zu aktualisieren. Die abschließend von Desktop 2 propagierte

Änderung mit Zeitstempel V_3 wird erfolgreich an Desktop 1 und Akteur B gesendet, ohne einen Konflikt auszulösen. Somit sind alle im Umlauf befindlichen Replikate des aktiven Dokuments vollständig synchronisiert, was es den Akteuren erlaubt, ihre Zusammenarbeit auf Basis konsistenter Informationen fortzusetzen.

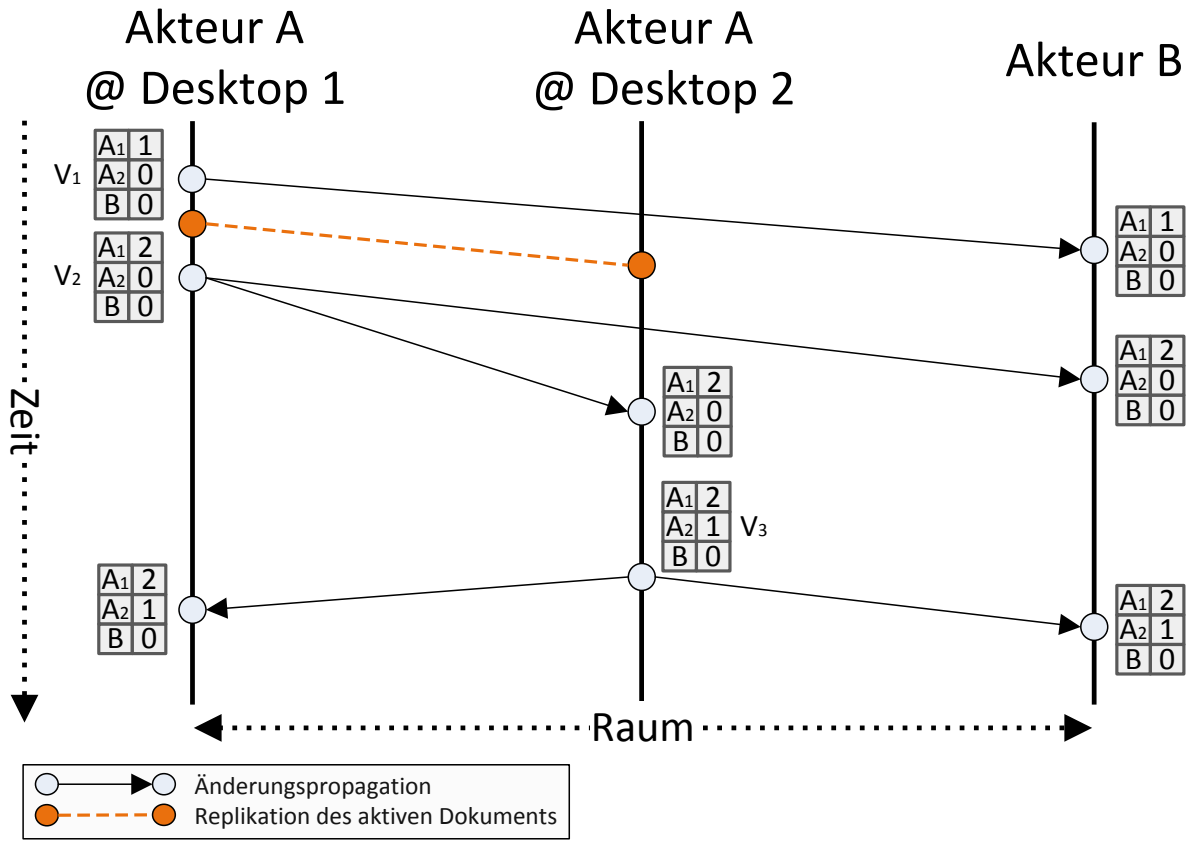


Abbildung 7.9: Verwaltung von mehreren Arbeitsplätzen pro Akteur auf Versionierungsebene

7.4 Beitritt neuer Akteure und Aktualisierung von Adressinformationen

Gemäß den Ausführungen in Abschnitt 5.4.2 soll es externen Interessenten ermöglicht werden, dem Kreis der bereits partizipierenden Akteure unkompliziert beitreten zu können, um dem dynamischen Charakter der zu modellierenden Prozesse Rechnung zu tragen. In den folgenden Abschnitten wird stufenweise ein Kommunikationsprotokoll entwickelt, das den Beitritt bisher unbekannter Akteure zu einem gemeinsamen Prozess ermöglicht.

7.4.1 Einladung potenzieller Akteure

Der Beitritt eines neuen Akteurs setzt voraus, dass diesem zuvor eine Einladung zur Prozessteilnahme durch einen anderen Beteiligten zugegangen ist, die Informationen über den bisherigen Prozessverlauf enthält. Prinzipiell existieren zwei Möglichkeiten, Drittpersonen zur Partizipation aufzufordern:

1. Direkte Adressierung eines spezifischen potenziellen Akteurs, dessen Identität und Kontaktinformationen mindestens einem der Prozessbeteiligten bekannt sind
2. Indirekte Adressierung einer Gruppe von potenziellen Akteuren, deren Identitäten und Kontaktinformationen den Prozessbeteiligten nicht bekannt sein müssen

In beiden Fällen besteht die eigentliche Einladung eines neuen Akteurs darin, dass diesem eine vollständige Kopie des aktiven Dokuments über einen geeigneten Kommunikationskanal bzw. auf einem geeigneten Datenträger übermittelt wird. Da das aktive Dokument alle Informationen und Voraussetzungen, die zu seiner Benutzung notwendig sind, mit sich bringt, kann der neue Akteur mit dem ersten Öffnen der erhaltenen Kopie unmittelbar dem zugehörigen Prozess beitreten.

7.4.2 Entwicklung des Beitrittsprotokolls

In diesem Abschnitt wird das Kommunikationsprotokoll zum Beitritt neuer Akteure erarbeitet und detailliert dargestellt. Die nachfolgenden Kommunikationsbeispiele beziehen sich auf den Fall der indirekten Adressierung einer Gruppe von potenziellen Akteuren, die Ausführungen behalten jedoch auch bei direkter Adressierung ihre Gültigkeit.

7.4.2.1 Initiale Rückmeldung neuer Akteure bei sequentiellem Beitritt

Zum Zeitpunkt der Erstellung der Einladung sind die Identitäten potenzieller Akteure nicht zwingend bereits allen Prozessbeteiligten bekannt. Deswegen ist es erforderlich, dass sich Akteure, die dem Prozess beitreten, den übrigen Beteiligten vorstellen. Vereinfachend wird zunächst angenommen, dass Beitritte immer strikt sequentiell ablaufen, also zu jedem Zeitpunkt maximal eine offene Einladung existiert.

Zur Übermittlung der benötigten Informationen wird ein neuer Nachrichtentyp eingeführt, der im Folgenden *sequentielle Rückmeldung* genannt wird. Nachrichten dieses Typs besitzen den nachfolgenden Inhalt:

1. Informationen, die zur Identifikation des beigetretenen Akteurs erforderlich sind (vgl. Abschnitt 7.3)
2. Adresse, unter der der beigetretene Akteur von den übrigen auf elektronischem Wege erreicht werden kann

Zur Durchführung des Prozessbeitritts verhalten sich alle Akteure wie im Zustandsdiagramm in Abbildung 7.10 visualisiert: Direkt nach dem ersten Öffnen des aktiven Dokuments und der Eingabe von Identifikations - und Adressinformationen sendet der neu beigetretene Akteur eine initiale Rückmeldung an alle in der Kopie des aktiven Dokuments enthaltenen Adressen.

Die bereits dem Prozess beigetretenen Empfänger dieser Nachricht übernehmen die Informationen über den neuen Akteur anschließend in ihr lokales aktives Dokument, um diesen über zukünftige Aktualisierungen benachrichtigen zu können. Somit sind alle zum Zeitpunkt der Erstellung der Einladung am Prozess beteiligten Akteure über die Existenz des neu beigetreten Akteurs unterrichtet.

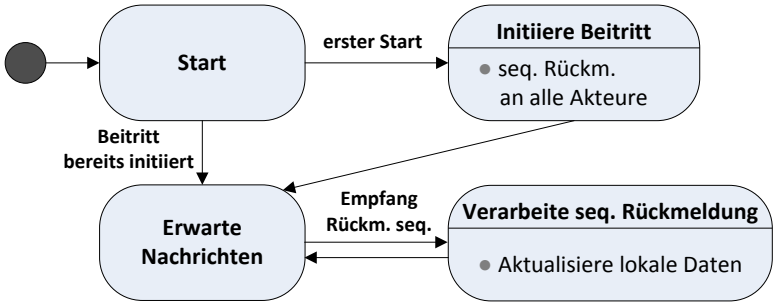


Abbildung 7.10: Verhalten eins Akteurs bei initaler Rückmeldung nach sequentiellem Beitritt

Beispiel

Das Kommunikationsszenario in Abbildung 7.11 zeigt die Verwendung des neu eingeführten Nachrichtentyps gemäß obigem Zustandsdiagramm zur Vorstellung beigetretener Akteure. Dargestellt wird neben den ausgetauschten Nachrichten auch die Liste von Akteuren, die den einzelnen Beteiligten bekannt sind. Die Abläufe im vorgestellten Beispiel übertragen sich analog auf Situationen mit mehr als drei teilnehmenden Akteuren.

Zu Beginn ist lediglich Akteur A am Prozess beteiligt. Von ihm ergeht per indirekter Adressierung eine Einladung an einen weiteren Akteur. Die von A erstellte Kopie des aktiven Dokuments wird zu diesem Zweck an Akteur B übermittelt.

B hat nach dem Öffnen des aktiven Dokuments Kenntnis über die Identität von A und dessen Adressinformationen. Um nun A über den eigenen Prozessbeitritt zu informieren, wird auf der Seite von B eine Nachricht vom Typ *sequentielle Rückmeldung* generiert und an A ausgeliefert. Nach Empfang der Nachricht hat A Kenntnis von B genommen und aktualisiert sein aktives Dokument entsprechend.

Nach diesem erfolgreichen Beitritt soll von A noch ein weiterer Akteur hinzugefügt werden. Hierzu wird erneut eine Kopie des aktiven Dokuments erstellt. Diese wird von Akteur C nach dem Transport in Empfang genommen.

Analog zum vorherigen Fall wird den C bekannten Akteuren A und B eine Nachricht vom Typ *sequentielle Rückmeldung* übersandt. Beide nehmen nach deren Empfang C als neuen Akteur in die Verwaltungsstrukturen ihres lokalen aktiven Dokuments auf. Sämtliche Akteure verfügen nun über die relevanten Informationen zur Identität aller übrigen Prozessbeteiligten.

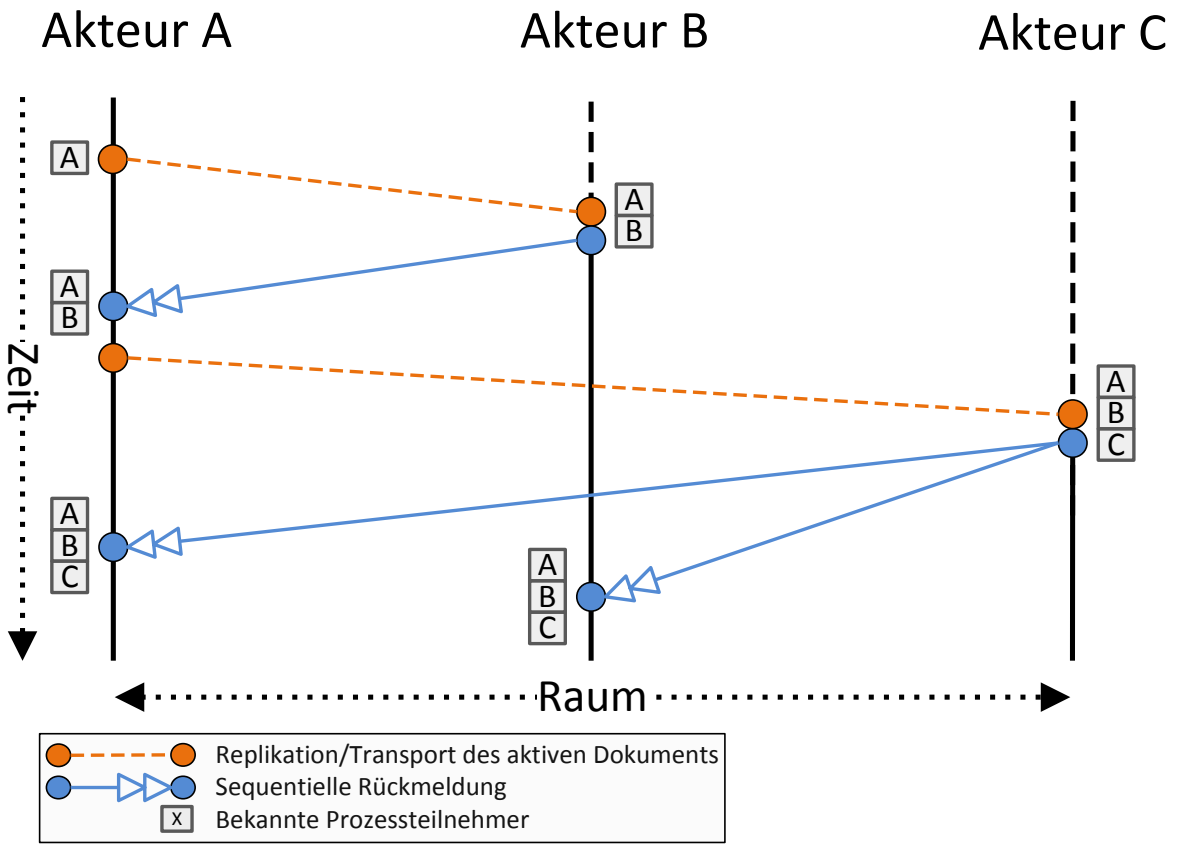


Abbildung 7.11: Beispielszenario zur initialen Rückmeldung neuer Akteure

7.4.2.2 Initiale Synchronisation des aktiven Dokuments neu beigetretener Akteure bei sequenziellem Beitritt

Die Nachrichten vom Typ *sequentielle Rückmeldung* erlauben den Austausch von Identifikations - und Adressinformationen unter den Teilnehmern, nicht aber die nachträgliche Synchronisation von Prozessartefakten. Diese ist jedoch erforderlich, um zu verhindern, dass neu beigetretene Akteure ihre Entscheidungen aufgrund veralteter Informationen treffen.

Die Kopie des aktiven Dokuments, die zu den neuen Akteuren transportiert wird, enthält nur diejenigen Informationen, die zum Zeitpunkt der Replikation des aktiven Dokuments beim Urheber der Einladung verfügbar waren. Aktualisierungen, die zwischen der Replikation und dem tatsächlichen Prozessbeitritt eines Akteurs stattfinden, werden diesem nicht mitgeteilt.

Um diese Problemstellung zu lösen, muss der Kommunikationsablauf modifiziert werden. Dazu wird der Nachrichtentyp *sequentielle Rückmeldung* erweitert und beinhaltet nun folgende Informationen:

1. Informationen, die zur Identifikation des beigetretenen Akteurs erforderlich sind (vgl. Kapitel 7.3)
2. Adresse, unter der der beigetretene Akteur von den übrigen auf elektronischem Wege erreicht werden kann
3. Liste von Tupeln der Form (Identifikator, logischer Zeitstempel) mit jeweils einem Eintrag pro in der lokalen Kopie des aktiven Dokuments enthaltenem Prozessartefakt

Beim Austausch derartiger Nachrichten verfahren die Akteure wie im entsprechend erweiterten Zustandsdiagramm in Abbildung 7.12 (Ergänzungen farblich hervorgehoben). Nach dem Empfang einer Nachricht vom Typ *sequentielle Rückmeldung* können die Akteure durch Vergleich mit den ihnen lokal bekannten Informationen über Prozessartefakte bestimmen, ob seit der Replikation des aktiven Dokuments weitere Aktualisierungen durchgeführt worden sind, die dem neu beigetretenen Akteur unbekannt sind. Dabei berücksichtigt jeder Akteur ausschließlich diejenigen Prozessartefakte, für die er Änderungsrechte besitzt.

Daraufhin senden die Empfänger eine Antwortnachricht an den neu beigetretenen Akteur. Für diese Antwortnachricht wird ein zusätzlicher Nachrichtentyp, *sequentielle Synchronisation*, definiert.

Nachrichten dieses Typs enthalten eine Liste aktueller, vollständiger Kopien von Prozessartefakten. Es sind genau diejenigen Prozessartefakte in der Liste enthalten, die in der vom beigetretenen Akteur versandten Liste fehlen oder deren lokale Version aktueller ist als die aus der per vorheriger Nachricht empfangenen Liste[1]. Nachdem der neue Akteur die Antwortnachrichten aller übrigen ihm bekannten Beteiligten empfangen und sein lokales aktives Dokument entsprechend aktualisiert hat, verfügt er über alle von den ihm bekannten Akteuren bislang durchgeführten Aktualisierungen.

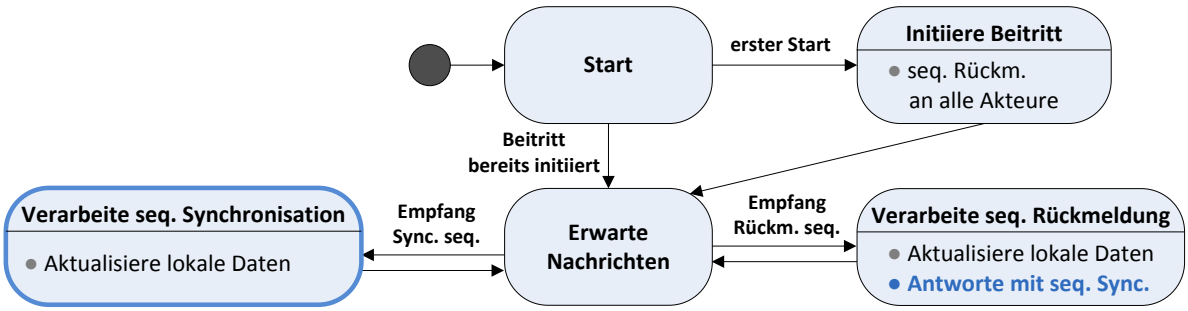


Abbildung 7.12: Verhalten eines Akteurs bei initialer Synchronisation nach sequentiellem Beitritt

Beispiel

Eine Anwendung der beschriebenen Erweiterung des Beitrittsprotokolls wird in Abbildung 7.13 gezeigt. Das dargestellte Kommunikationsszenario entspricht weitgehend demjenigen aus Abbildung 7.11.

Hinzugekommen ist eine Aktualisierung eines Prozessartefakts durch Akteur A zwischen der Replikation des aktiven Dokuments für Akteur B und dessen tatsächlichem Beitritt. Dementsprechend ist B zum Zeitpunkt des Beitritts nicht über diese Änderung informiert.

B schickt nach dem Erhalt und ersten Öffnen des aktiven Dokuments seine *sequentielle Rückmeldung* an A, um sich diesem vorzustellen. Aufgrund der in dieser Nachricht enthaltenen Liste erkennt A, dass B noch nicht über die von ihm durchgeführte Änderung informiert ist. Er antwortet mit einer Nachricht vom Typ *sequentielle Synchronisation*, in der er die aktuellste Version des modifizierten Prozessartefakts einfügt. B empfängt diese und verfügt damit über alle zu diesem Zeitpunkt verfügbaren Informationen über den Prozessverlauf.

1 Anmerkung: Bei Verwendung einer Strategie zur Auflösung nebenläufiger Änderungskonflikte, die Zugriff auf die gesamte Versionshistorie eines Prozessartefakts benötigt, kann es erforderlich sein, alle lokal verfügbaren Versionen des Prozessartefakts zu versenden. Auf diese Weise können eventuelle zukünftige Änderungskonflikte auch durch den neu beigetretenen Akteur aufgelöst werden.

Analog läuft der Beitritt von Akteur C ab. Auch in diesem Fall werden nach der Replikation des aktiven Dokuments Aktualisierungen durchgeführt, bevor C tatsächlich dem Prozess beitritt. C stellt sich ebenfalls mittels *sequentieller Rückmeldung* bei den ihm bekannten Akteuren A und B vor. Diese können daraus ermitteln, dass C ihre letzten Änderungen noch nicht erhalten hat und reagieren mit Nachrichten vom Typ *sequentielle Synchronisation*, die die jeweils aktuellste Version der betreffenden Prozessartefakte enthalten. Auch C ist nach dem Empfang dieser Nachrichten auf dem aktuellen Stand des Prozessgeschehens und somit erfolgreich dem Kreis der partizipierenden Akteure beigetreten.

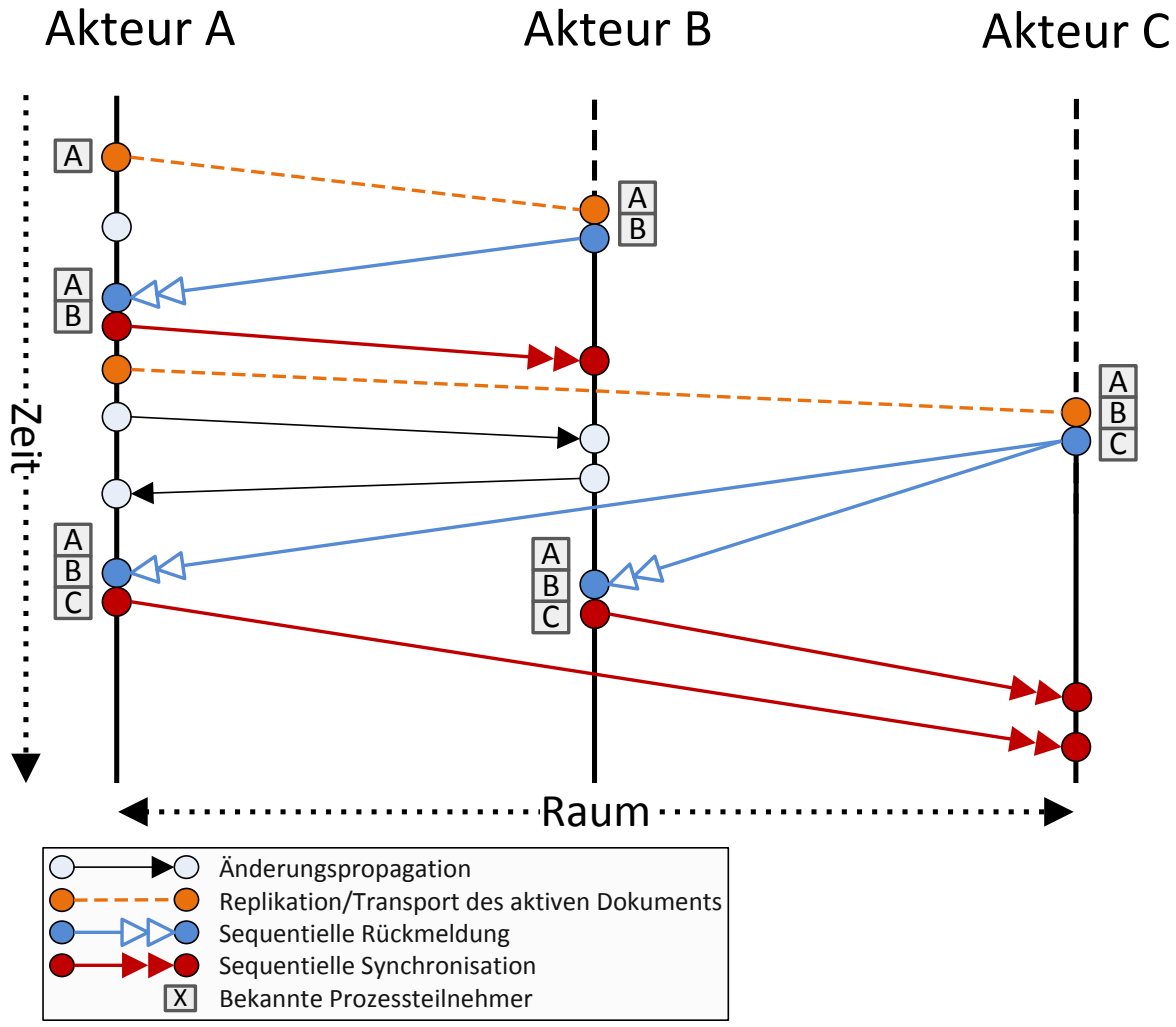


Abbildung 7.13: Beispielszenario zur initialen Synchronisation

7.4.2.3 Synchronisation zwischen parallel beigetretenen Akteuren

Um das Beitrittsprotokoll zu vervollständigen wird in einem letzten Schritt auf die in den Abschnitten 7.4.2.1 und 7.4.2.2 getroffene vereinfachende Annahme eines strikt sequentiellen Beitritts neuer Akteure verzichtet. Durch eine Erweiterung des Beitrittsprotokolls wird es ermöglicht, dass parallel beliebig viele Replikate des aktiven Dokuments an potenzielle Akteure ausgeliefert werden können, ohne Inkonsistenzen hervorzurufen. Das bedeutet, dass Akteure bereits dem Prozess beitreten können, während noch weitere offene Einladungen zur Partizipation vorhanden sind.

Hierfür sind, wie in Abbildung 7.14 veranschaulicht (Erweiterungen farblich hervorgehoben), zusätzliche Kommunikationsschritte zwischen den parallel beigetretenen Akteuren erforderlich. Dementsprechend wird der Nachrichtentyp *sequentielle Synchronisation* erneut um weitere Informationen ergänzt. Er umfasst damit folgende Inhalte:

1. Liste mit aktuellen, vollständigen Kopien derjenigen Prozessartefakte, die in der vorausgegangenen Nachricht vom Typ *sequentielle Rückmeldung* fehlen oder deren lokale Version aktueller ist als die aus der empfangenen Nachricht
2. Liste von Identifikations- und Adressinformationen aller lokal bekannten Akteure

Zusätzlich werden die beiden neuen Nachrichtentypen *parallele Rückmeldung* und *parallele Synchronisation* eingeführt. Nachrichten vom Typ *parallele Rückmeldung* entsprechen dabei syntaktisch denen vom Typ *sequentielle Rückmeldung*, unterscheiden sich aber von deren Semantik. Zur Kennzeichnung eines parallelen Beitritts dienen sie dazu, den Empfänger zu veranlassen, mit einer *parallelen Synchronisation* zu antworten. Sie werden an alle Akteure verschickt, deren Informationen in einer Nachricht zur *sequentiellen Rückmeldung* enthalten sind und die lokal bisher nicht bekannt waren. Dies sind genau diejenigen Akteure, die parallel zum anfragenden Akteur beigetreten sind.

Nachrichten vom Typ *parallele Synchronisation* bestehen aus folgenden Informationen:

1. Liste mit aktuellen, vollständigen Kopien derjenigen Prozessartefakte, die in der vorausgegangenen Nachricht vom Typ *parallele Rückmeldung* fehlen oder deren lokale Version aktueller ist als die aus der empfangenen Nachricht[1]
2. Liste von Tupeln der Form (Identifikator, Version) mit jeweils einem Eintrag pro Prozessartefakt, das im Vergleich zur vorausgegangenen Nachricht vom Typ *parallele Rückmeldung* lokal fehlt oder in einer älteren Version vorliegt

1 Vgl. hierzu auch die zusätzliche Anmerkung zum Nachrichtentyp *sequentielle Rückmeldung* auf Seite 91.

Beantwortet wird diese Synchronisationsnachricht abschließend durch den Versand einer Nachricht vom Typ *sequentielle Synchronisation*, da nun die betreffenden Akteure untereinander bekannt gemacht sind, also eine Situation äquivalent zum sequentiellen Beitritt vorliegt.

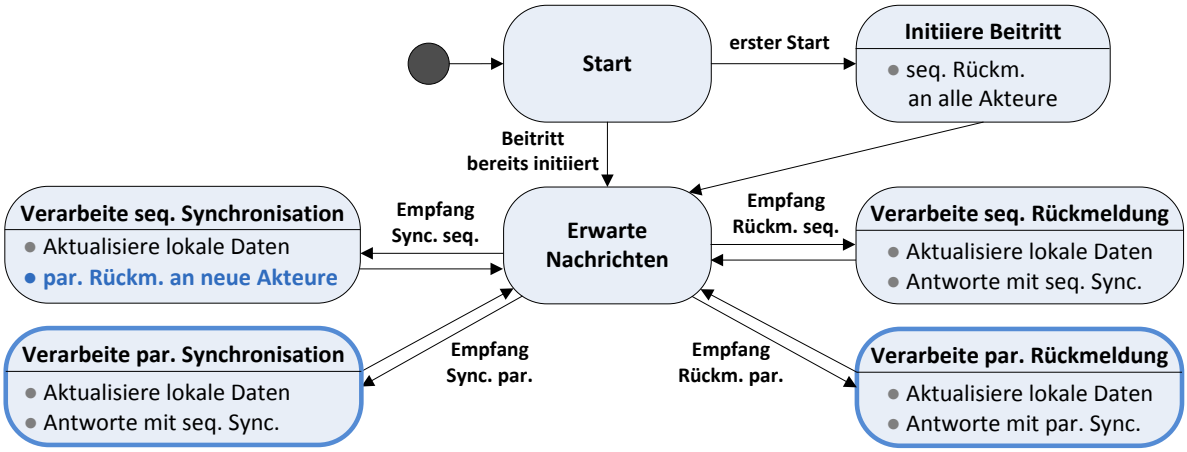


Abbildung 7.14: Vollständiges Zustandsdiagramm für das Beitrittsprotokoll

Beispiel

Der zu obigem komplexen Zustandsdiagramm konforme Informationsaustausch wird von Abbildung 7.15 im Detail dargestellt. Im veranschaulichten Kommunikationsszenario wird das aktive Dokument von Akteur A für C repliziert, bevor Akteur B, für den zuvor ein eigenes Replikat erstellt worden ist, tatsächlich dem Prozess beigetreten ist. Die *sequentielle Rückmeldung* von B und die anschließende *sequentielle Synchronisation* mit A laufen wie in den vorherigen Abschnitten beschriebenen Szenarien ab.

Die vorgenommenen Erweiterungen des Protokolls zeigen jedoch beim Beitritt von C ihre Auswirkungen. In der auf seine *sequentielle Rückmeldung* folgenden Nachricht vom Typ *sequentielle Synchronisation* erfährt C, dass auch Akteur B am Prozess teilnimmt und parallel beigetreten ist. Um sich diesem vorzustellen, verschickt C eine Nachricht des Typs *parallele Rückmeldung*. Nach dem Empfang dieser Nachricht kann B feststellen, dass ihm die zuvor von C vorgenommene Aktualisierung in seinem lokalen aktiven Dokument fehlt.

Mit einer Nachricht vom Typ *parallele Synchronisation* an C fordert er diese Informationen nun explizit an. Zudem übermittelt er in dieser Nachricht die Informationen über die von ihm lokal vorgenommene Aktualisierung, die wiederum C bisher nicht bekannt sind. C empfängt diese Nachricht und lässt B mittels *sequentieller Synchronisation* die angefragten Informationen zukommen.

Alle Akteure sind zum Abschluss des Beispiels miteinander bekannt gemacht und verfügen zudem über die aktuellsten Versionen sämtlicher Prozessartefakte. Die beim parallelen Beitritt von B und C entstandenen temporären Inkonsistenzen sind erfolgreich aufgelöst, das nun vollständige Beitrittsprotokoll ermöglicht also den fehlerfreien sequentiellen und parallelen Beitritt neuer Akteure.

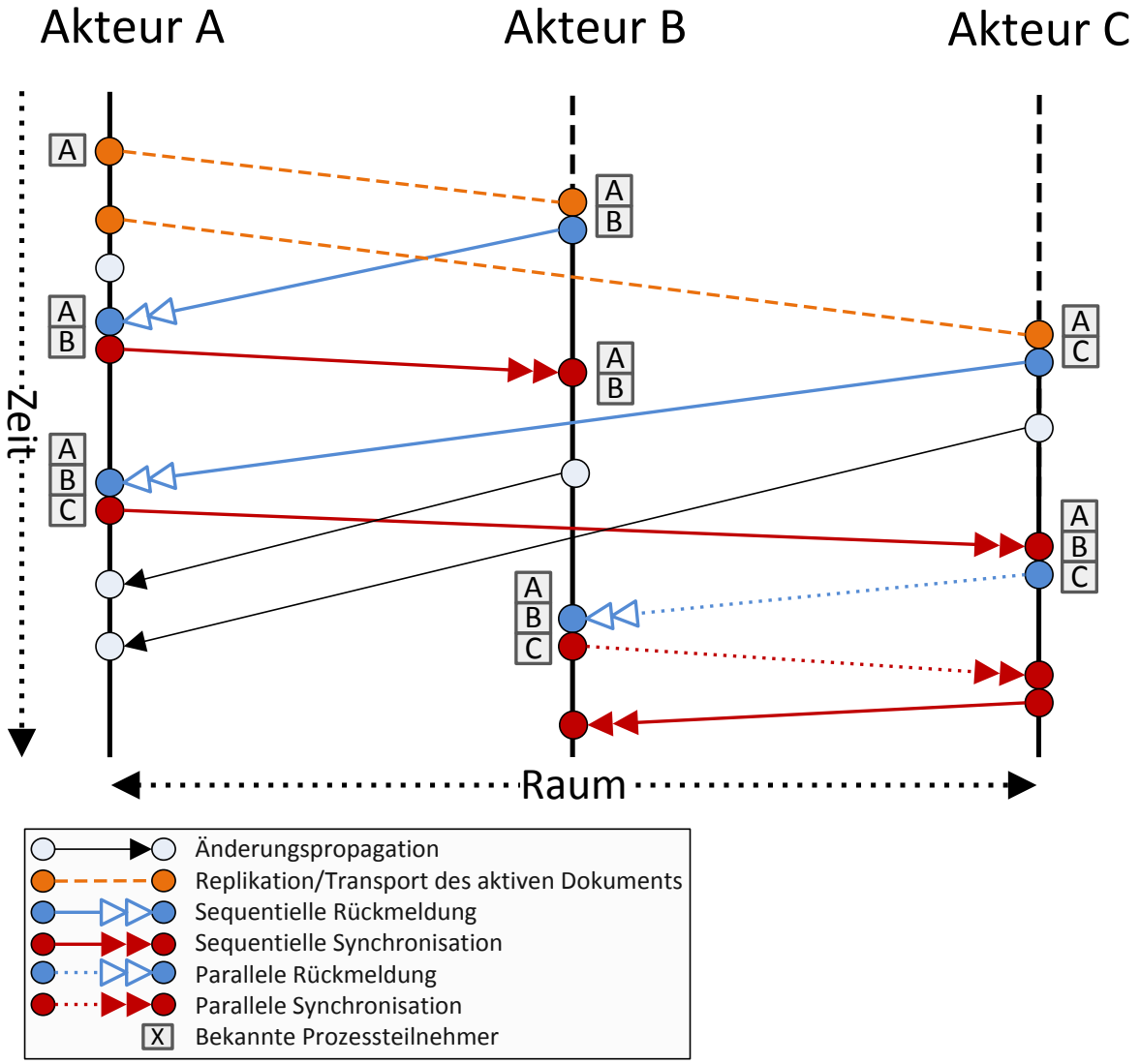


Abbildung 7.15: Beispielszenario zur Synchronisation zwischen parallel beigetretenen Akteuren

7.4.3 Änderung von Adressinformationen

Akteure haben zur Laufzeit die Möglichkeit, die Adresse, unter der sie Nachrichten empfangen, zu ändern. Dabei ist sicherzustellen, dass keine Aktualisierungsnachrichten aufgrund der geänderten Adressinformationen verloren gehen.

Nach der lokalen Änderung der Adressinformationen müssen diese an die übrigen Akteure propagiert werden. Der dazu notwendige Kommunikationsablauf entspricht exakt dem in den vorherigen Abschnitten entwickelten Beitrittsprotokoll. Beim Empfang der Nachricht *sequentielle Rückmeldung* eines bereits lokal bekannten Akteurs modifizieren die übrigen Akteure dessen gespeicherte Adressinformationen und antworten mit den entsprechenden Nachrichten gemäß Abbildung 7.14.

7.5 Zusammenfassung

Der vorgestellte fachliche Lösungsansatz ermöglicht unter Verwendung logischer Zeitstempel die Synchronisation verteilter Änderungen von Prozessartefakten in interdisziplinären, institutionsübergreifenden Workflows, in denen Informationen auf Basis von aktiven Dokumenten ausgetauscht werden. Durch die erläuterten Konzepte zur Verwaltung physischer Arbeitsplätze und der Organisation des Beitritts neuer Akteure wird die Koordinierung dynamisch kooperierender Teilnehmergruppen erleichtert.

Dabei wird die interne Verwaltung der Versionen von Prozessartefakten und der direkte Datenaustausch mit anderen Akteuren vor den Prozessbeteiligten verborgen. Auf diese Weise wird eine intuitive Benutzbarkeit für Implementierungen erreicht, die das Lösungskonzept mit Hilfe konkreter Technologien umsetzen.

8 Architekturübersicht

Die Konzepte des im vorherigen Kapitel präsentierten fachlichen Lösungsansatzes sollen in das Gesamtsystem von α-Flow integriert werden. In diesem Kapitel wird dazu die Architektur der zu entwickelnden Softwarekomponenten beschrieben.

Als Ausgangspunkt dieser Ausführungen wird die existierende Systemarchitektur von α-Flow dargestellt. Darauf aufbauend wird erarbeitet, an welchen Stellen die zu entwickelnden Komponenten in das Gesamtsystem integriert werden und an welchen bereits vorhandenen Teilen von α-Flow weitere Änderungen vorzunehmen sind. Abschließend wird detailliert die Architektur der generischen Kommunikationsschnittstelle erläutert, mit deren Hilfe die Synchronisation zwischen den Akteuren koordiniert wird.

8.1 Einführung in die bestehende Architektur von α-Flow

Die vorhandenen Funktionalitäten von α-Flow sind in mehreren Subsystemen gekapselt. Jedes Subsystem übernimmt dabei einen thematisch abgegrenzten Teil der Gesamtfunktionalität.

Das in Abschnitt 3.1 vorgestellte Domänenmodell von α-Flow ist im Software-Modul α-Model realisiert. Alle anderen Komponenten bauen auf diesem Modul auf und verwenden die Konzepte des Domänenmodells. Diese Komponenten lassen sich, wie in Abbildung 8.1 illustriert, grob in drei verschiedene Schichten strukturieren:

1. Die Schicht der Benutzerinteraktion, die diejenigen Komponenten umfasst, deren Aufgabe die Kommunikation mit dem Benutzer ist
2. Den Systemkern, der für die Verwaltung der Bestandteile des Domänenmodells zuständig ist und die gesamte Geschäftslogik von α-Flow enthält
3. Die Infrastruktur-Schicht, in der die Funktionalitäten zur Kommunikation mit anderen Akteuren und zur Sicherung der lokalen Persistenz von Prozessartefakten gekapselt sind

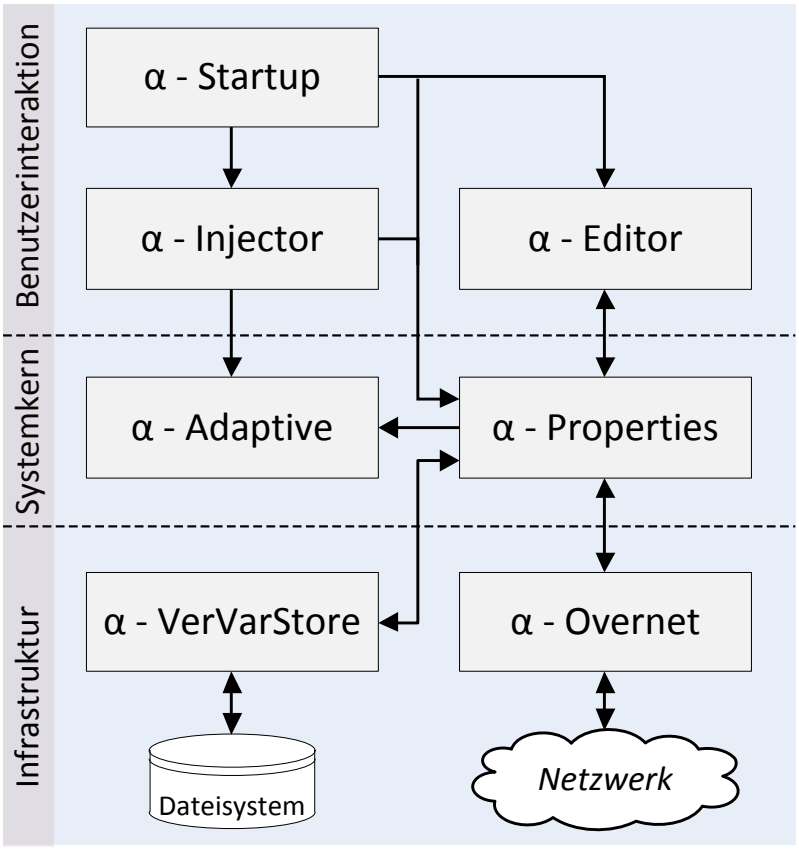


Abbildung 8.1: Ursprüngliche Architektur von α-Flow

Im Folgenden werden die einzelnen Module, aus denen die verschiedenen Schichten aufgebaut sind, in ihrer Funktion näher beschrieben. Zudem werden auf Subsystemebene die Kooperationen zwischen den einzelnen Komponenten erläutert.

Die Schicht der Benutzerinteraktion besteht aus den Modulen α-Startup, α-Injector und α-Editor. α-Startup ist für die Koordination des Startvorgangs verantwortlich. Nach dem Öffnen des α-Doc werden von hier aus alle weiteren benötigten Komponenten initialisiert.

Zum Erstellen eines neuen α-Doc greift α-Startup auf die Funktionalität des α-Injectors zurück. Dieser fügt dem aktiven Dokument die erste α-Card hinzu. Zudem erlaubt er das Hinzufügen von α-Cards in ein bestehendes α-Doc direkt aus dem Dateisystem des verwendeten Betriebssystems heraus.

Nach der erfolgreichen Initialisierung aller Komponenten aus darunterliegenden Schichten wird der α-Editor gestartet. Dieser stellt die graphische Benutzeroberfläche des α-Doc dar. Mit ihm können Benutzer den momentanen Status aller enthaltenen α-Cards

überblicken und manipulieren. Es können neue α-Cards erzeugt oder die Adornments existierender α-Cards aktualisiert werden. Auch das Hinzufügen von Payloadinformationen in Form von lokal vorliegenden Dateien ist durch einfaches Verschieben des Dateisymbols auf die graphische Repräsentation der gewünschten α-Card durchführbar.

Der Systemkern setzt sich zusammen aus den Modulen α-Properties und α-Adaptive. Letzteres umfasst das evolutionäre adaptive Attributmodell zur Verwaltung von Adornments. Dieses erlaubt es den Prozessteilnehmern, zur Laufzeit neue domänenspezifische Adornments zu erstellen, um diese α-Cards zuzuordnen [NSWL11]. Das Modul α-Properties nimmt in der Architektur des Gesamtsystems eine zentrale Rolle ein, da es große Teile der Geschäftslogik von α-Flow in Form einer regelbasierten Bibliothek enthält [TN11]. Im laufenden Betrieb übernimmt dieses Subsystem die Koordinierung der Zusammenarbeit mit den übrigen Komponenten. Benutzereingaben aus dem α-Editor werden von α-Properties entgegengenommen und zur weiteren Verarbeitung an die zuständigen Systemkomponenten oder die regelbasierte Bibliothek delegiert.

Für diese Delegation greift der Systemkern auf die Infrastruktur-Schicht zurück. In dieser sind die Module α-VerVarStore und α-Overnet enthalten. Ersteres umfasst die Schnittstellen eines Versionsverwaltungssystems, mit dessen Hilfe die unterschiedlichen Versionen von α-Cards in strukturierter Form persistiert und verwaltet werden können. Die notwendigen Zugriffe auf das lokale Dateisystem werden über die Schnittstellen des Moduls durchgeführt.

Das Subsystem α-Overnet beinhaltet die Strukturen und Schnittstellen zur Kommunikation mit anderen Akteuren. Es realisiert zudem in seiner ursprünglichen Fassung einen socket-basierten Austausch von Aktualisierungsnachrichten. Alle Akteure müssen aufgrund dieser einfach gehaltenen Verteilungsmethode dauerhaft online sein, um über Aktualisierungen benachrichtigt werden zu können.

8.2 Integration des fachlichen Lösungskonzepts in α-Flow

Zur Umsetzung der im Rahmen dieser Arbeit erarbeiteten Lösungskonzepte ist es erforderlich, die im vorherigen Abschnitt vorgestellte Architektur von α-Flow zu modifizieren und um neue Funktionalitäten zu erweitern. Abbildung 8.2 veranschaulicht das überarbeitete Architekturkonzept. Farblich hervorgehobene Module ohne Schraffur werden komplett neu konzipiert, der Funktionsumfang der schraffiert dargestellten Module wird um neue Aspekte erweitert.

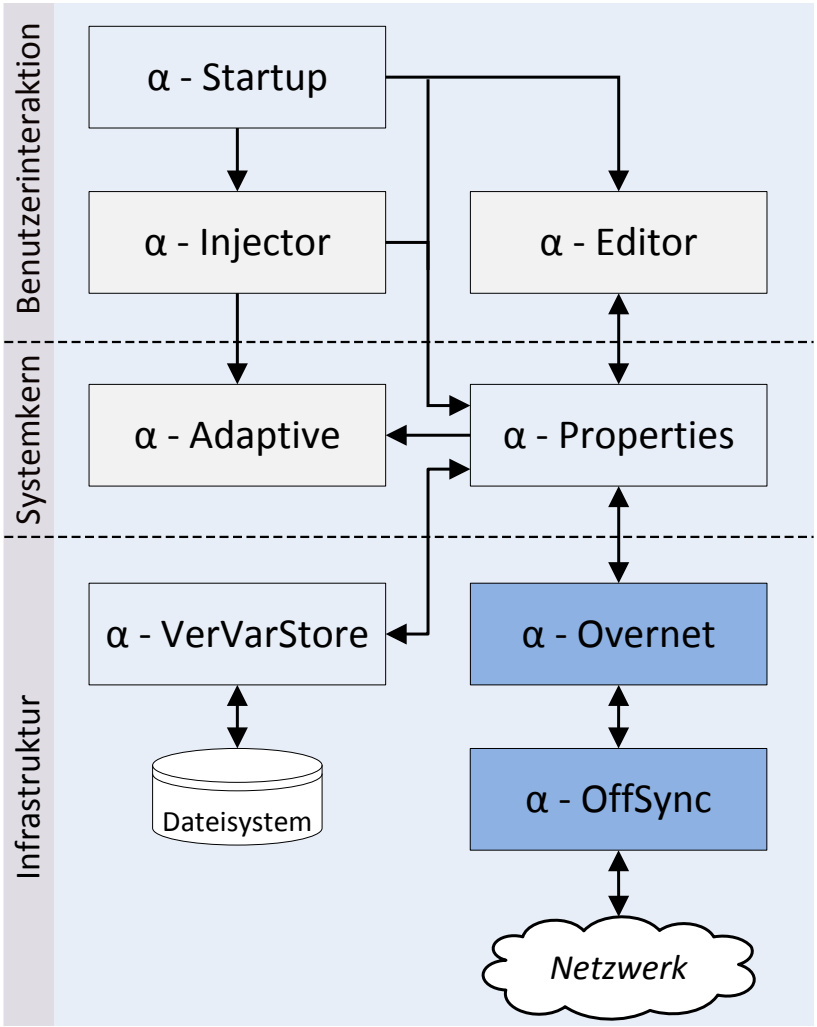


Abbildung 8.2: Gesamtarchitektur von α-Flow nach Erweiterung

Das existierende Subsystem α-Overnet wird einem Generalisierungsprozess unterzogen. Anstatt eine konkrete Implementierung für den nicht offlinefähigen Nachrichtenaustausch anzubieten, stellt es in der modifizierten Fassung eine generische Kommunikationsschnittstelle dar. Diese unterstützt Transportprotokolle und -verfahren, die den in Abschnitt 5.3 formulierten Anforderungen genügen. Alle Implementierungen, die auf konkreten Transportprotokollen basieren, müssen konform zu dieser generischen Schnittstelle sein.

Die in der vorliegenden Arbeit angefertigte prototypische Implementierung unter Verwendung von SMTP und IMAP erfüllt ebenfalls diese Anforderung. Zur Wahrung der Modularität und erleichterten Austauschbarkeit ist sie im neu hinzugefügten Modul α-OffSync gekapselt.

Durch diese Kapselung und die Konformität zur Schnittstelle von α-Overnet ist sichergestellt, dass zu einem späteren Zeitpunkt alternative Implementierungen angeboten werden können, ohne grundlegende Änderungen im eigentlichen Systemkern nach sich zu ziehen. Der Kern erwartet ausschließlich Kompatibilität zur generischen Kommunikationsschnittstelle und trifft ansonsten keine Annahmen über die konkrete Implementierung.

Zur Realisierung des vorgestellten Beitrittsprotokolls sind Anpassungen des Moduls α-Properties notwendig. Die dort gekapselte Regelbibliothek wird um Regeln erweitert, mit denen die Behandlung eintreffender Anfragen gemäß Abschnitt 7.4 koordiniert werden kann.

Das Verfahren zum Herstellen einer Ordnung auf den Versionen einer α-Card und zur Erkennung nebenläufiger Änderungskonflikte wird innerhalb des Moduls α-VerVarStore platziert, da es eng mit dem angebundenen Versionsverwaltungssystem interagiert. Darüber hinaus wird der Zugriff auf das Dateisystem auf eine generische Schnittstelle umgestellt, die unabhängig von konkreten Implementierungen der lokalen Dateiorganisation ist.

Im Modul α-Startup werden verschiedene Anpassungen vorgenommen, um die automatische Konfiguration des lokalen α-Doc an die modifizierte Architektur anzupassen. Diese betreffen das einmalige Einlesen der identifizierenden Informationen des jeweiligen Akteurs und die Initialisierungsreihenfolge der einzelnen Subsysteme.

Das allen Bestandteilen von α-Flow zugrundeliegende Domänenmodell α-Model wird um das dargestellte Konzept logischer Zeitstempel erweitert. Anpassungen des Domänenmodells sind zudem für eine, den Kommunikationscharakteristika angepasste, Umgestaltung der Repräsentation der identifizierenden Attribute einzelner Akteure erforderlich.

8.3 Architektur der generischen Kommunikationsschnittstelle

In diesem Abschnitt wird die neu konzipierte Architektur der Kommunikationsschnittstelle α-Overnet vorgestellt. Abbildung 8.3 visualisiert den internen Aufbau des Subsystems und dessen Anbindung an die übrigen Komponenten von α-Flow.

α-Overnet ist in Sendekomponente, Empfangskomponente, Sicherheitskomponente und Kommunikationsfassade gegliedert. Durch diese Aufteilung wird eine strikte Trennung von Zuständigkeiten erreicht. Bezüglich gemeinsamer Funktionalitäten gibt es zwischen den Komponenten keine Überschneidungen, da die gemeinsam benötigte Funktionali-

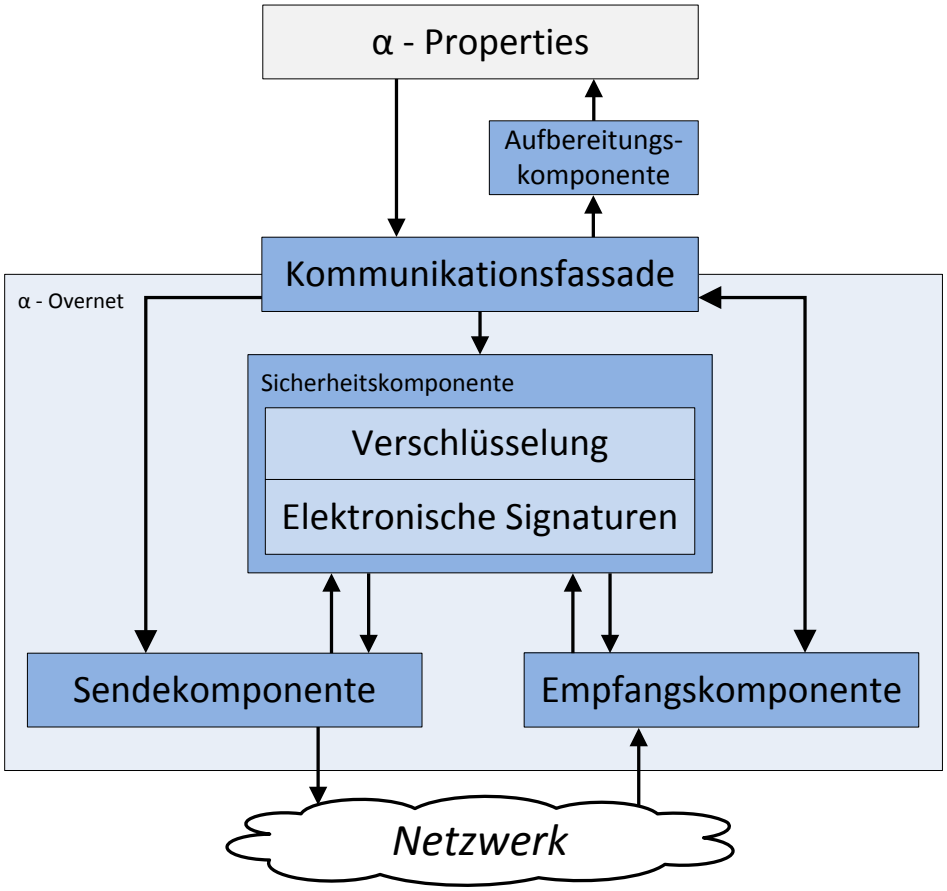


Abbildung 8.3: Architektur der generischen Kommunikationsschnittstelle

tät zur Behandlung verschlüsselter und signierter Nachrichten in der eigenständigen Sicherheitskomponente gekapselt ist.

Durch die interne Trennung der zu realisierenden Funktionen in die oben beschriebenen Komponenten ist ein hohes Maß an Modularität gewährleistet. Alle Komponenten können einzeln ausgetauscht bzw. erweitert werden, ohne Änderungen an den übrigen Teilmodulen zu erzwingen.

Die im obigen Architekturdiagramm enthaltenen Komponenten werden in den nachfolgenden Unterabschnitten bezüglich ihrer vorgesehenen Zuständigkeiten näher charakterisiert. Dabei wird insbesondere auf die in Abbildung 8.3 visualisierten Beziehungen zwischen den Komponenten eingegangen.

Zur Veranschaulichung der erarbeiteten Zuständigkeiten und Beziehungen kommen kompakte Übersichten gemäß der Methode der Class-Responsibility-Collaboration-Cards (CRC-Cards) [BC89] zum Einsatz. Bei diesem Vorgehen werden jedem Teil eines zu entwickelnden Softwaresystems bestimmte grobgranulare Aufgabenbereiche zugeordnet.

Die identifizierten Aufgaben werden zusammen mit einer Liste von Komponenten, die zum Durchführen der Aufgaben verwendet werden, in einer zweispaltigen Darstellung notiert. In der linken Spalte einer CRC-Card werden die von der entsprechenden Komponente zu erledigenden Aufgaben gelistet, wohingegen auf der rechten Seite die kooperierenden Komponenten vermerkt sind.

8.3.1 Zuständigkeitsbereich der Sendekomponente

Die Sendekomponente kapselt das gesamte Verhalten, welches zum eigentlichen Versand von Aktualisierungsinformationen erforderlich ist. Abbildung 8.4 fasst ihren Zuständigkeitsbereich und die Komponenten, mit denen sie in Interaktion tritt, zusammen.

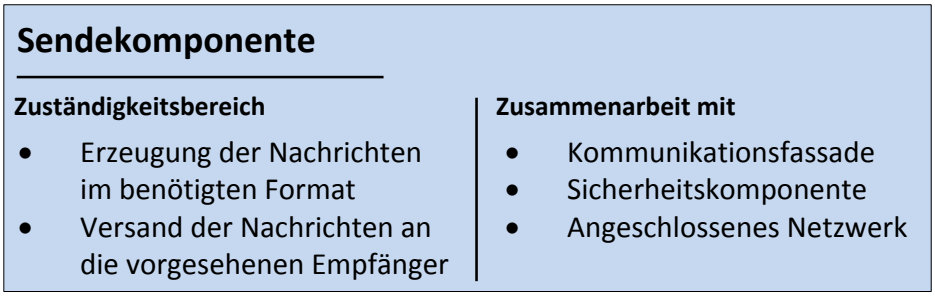


Abbildung 8.4: CRC-Card der Sendekomponente

Zu ihren Aufgaben gehört das Erzeugen von zu versendenden Nachrichten in einem Format, das für den gewählten Kommunikationskanal geeignet ist. Gemäß des beabsichtigten Vorgehens zur Nachrichtenübermittlung (Abbildung 8.5) kümmert sie sich im Anschluss an die Nachrichtenerzeugung und -verarbeitung um den Versand an die vorgesehenen Empfänger. Die Zuständigkeit der Sendekomponente endet, nachdem die Nachricht erfolgreich an das Netzwerk übertragen wurde.

Zur Wahrnehmung dieser Aufgaben muss die Sendekomponente mit anderen Komponenten von α-Overnet in Interaktion treten. Den Nutzinhalt der zu versendenden Nachrichten erhält sie in Form von zu propagierenden Informationen von der Kommunikationsfassade des Subsystems. Zur Sicherung der Authentizität einer neu erzeugten Nachricht und des Datenschutzes auf dem Transportweg greift sie auf die öffentlich zugängliche Funktionalität der Sicherheitskomponente zurück. Zu versendende Nachrichten werden zuerst mit der elektronischen Signatur des Absenders versehen und anschließend verschlüsselt.

Zu beachten ist, dass die genaue Struktur der Nachrichten erst auf der Ebene einer konkreten Implementierung festgelegt wird. Ebenso werden aus konzeptioneller Sicht

keine Annahmen über die konkrete Kommunikation mit dem Netzwerk getroffen, da diese maßgeblich von den verwendeten Transportprotokollen beeinflusst wird.

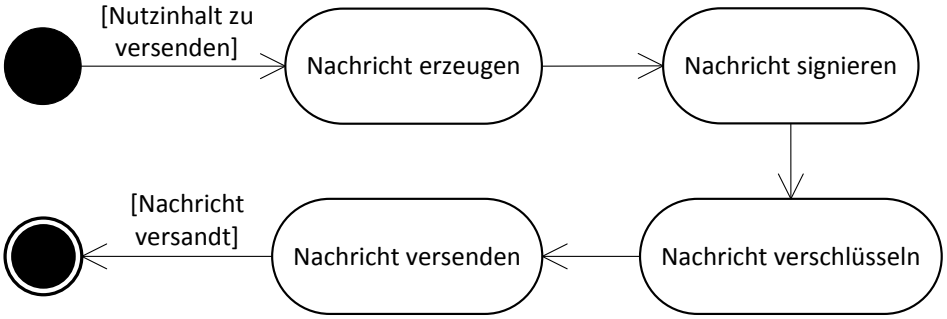


Abbildung 8.5: Aktivitätsdiagramm zum Versand von Nachrichten

8.3.2 Zuständigkeitsbereich der Empfangskomponente

Die Empfangskomponente fungiert in α-Overnet als Gegenstück zur im vorherigen Abschnitt vorgestellten Sendekomponente. Ihre Aufgabe ist es, auf das Netzwerk zu lauschen, um neu ankommende Nachrichten unverzüglich entgegenzunehmen.

Die Zuständigkeiten der Empfangskomponente und ihre Kooperationen mit anderen Teilen des Systems sind in Abbildung 8.6 aufgeführt.

Empfangskomponente	
Zuständigkeitsbereich	**Zusammenarbeit mit**
• Empfang von Nachrichten • Extraktion des übertragenen Nutzinhalts	• Kommunikationsfassade • Sicherheitskomponente • Angeschlossenes Netzwerk

Abbildung 8.6: CRC-Card der Empfangkomponente

Unter Verwendung von Schnittstellen der Sicherheitskomponente wird eine ankommende Nachricht in ihre ursprüngliche Form überführt (Abbildung 8.7). Hierfür wird zuerst versucht, die Nachrichten zu entschlüsseln, um im Anschluss durch Überprüfung der elektronischen Signatur ihre Authentizität zu überprüfen. Schlägt einer dieser beiden Schritte fehl, so wird die Nachricht verworfen und nicht weiter berücksichtigt.

Im Erfolgsfall wird der übertragene Nutzinhalt extrahiert, um die propagierten Informationen weiter verarbeiten zu können. Diese Verarbeitung findet jedoch aus Gründen einer optimierten Trennung von Verantwortlichkeiten nicht in der Empfangskomponente

statt. Die extrahierten Nutzinhalte werden deshalb an die Kommunikationsfassade von α-Overnet weitergegeben, die das weitere Vorgehen koordiniert.

Die generische Empfangskomponente trifft keine Annahmen über Inhalt und Struktur der empfangenen Nachrichten sowie über Charakteristika der Kommunikation mit dem Netzwerk. Konkrete Implementierungen sind dafür verantwortlich, die benötigte Logik zum Auswerten spezifischer Nachrichtenformate und zur Netzwerkkommunikation bereitzustellen.

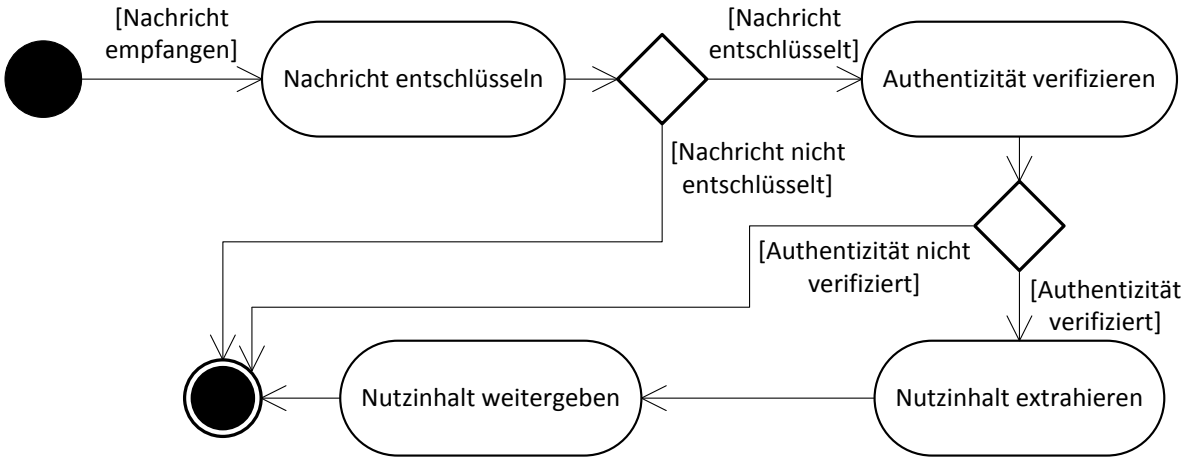


Abbildung 8.7: Aktivitätsdiagramm zur Verarbeitung einer empfangenen Nachricht

8.3.3 Zuständigkeitsbereich der Sicherheitskomponente

Die Sicherheitskomponente verantwortet die Aufrechterhaltung der Authentizität der ausgetauschten Nachrichten sowie die Wahrung des Datenschutzes auf dem Weg vom Absender zum Empfänger. Wie in Abbildung 8.8 ersichtlich, umfasst der Zuständigkeitsbereich dieser Komponente zwei grundlegende Aspekte.

Sicherheitskomponente

Zuständigkeitsbereich	Zusammenarbeit mit
• Ver- und Entschlüsselung von Nachrichten • Signierung von Nachrichten und Verifikation von elektronischen Signaturen	• Sendekomponente • Empfangskomponente • Kommunikationsfassade

Abbildung 8.8: CRC-Card der Sicherheitskomponente

Zum einen regelt sie die Verschlüsselung von Nachrichten beim Absender inklusive korrekter Entschlüsselung beim Empfänger. Zum anderen ist sie verantwortlich für den Einsatz von elektronischen Signaturen. Ihre Schnittstellen erlauben es, Nachrichten vor dem Versand zu signieren und die Signaturen empfangener Nachrichten zu verifizieren.

Diese angebotenen Funktionalitäten werden sowohl von der Empfangskomponente als auch von der Sendekomponente in Anspruch genommen. Da die Anforderungen dieser beiden Komponenten sich sehr ähnlich sind, werden die benötigten Funktionen zwecks verbesserter Kapselung und vereinfachter Wartbarkeit in die Sicherheitskomponente ausgelagert. Alle notwendigen Verwaltungsinformationen zum Erfüllen der angesprochenen Aufgaben werden der Sicherheitskomponente von der Kommunikationsfassade zur Verfügung gestellt.

Auf konzeptioneller Ebene werden keine Annahmen über mögliche Algorithmen zur Implementierung der geforderten Funktionalität getroffen, da diese stark vom verwendeten Kommunikationskanal abhängen. Zur Maximierung der erreichten Sicherheit wird jedoch ausdrücklich der Einsatz eines asymmetrischen Kryptosystems empfohlen.

8.3.4 Zuständigkeitsbereich der Kommunikationsfassade

Ein wichtiger Knotenpunkt der Architektur von α-Overnet ist die Kommunikationsfassade. Der Kommunikationsfassade fällt die Aufgabe zu, das Zusammenspiel der übrigen Komponenten des Subsystems zu koordinieren und deren Initialisierung anzustoßen. Ihre Zuständigkeiten und die erforderlichen Kooperationen mit anderen Komponenten werden in Abbildung 8.9 aufgezeigt.

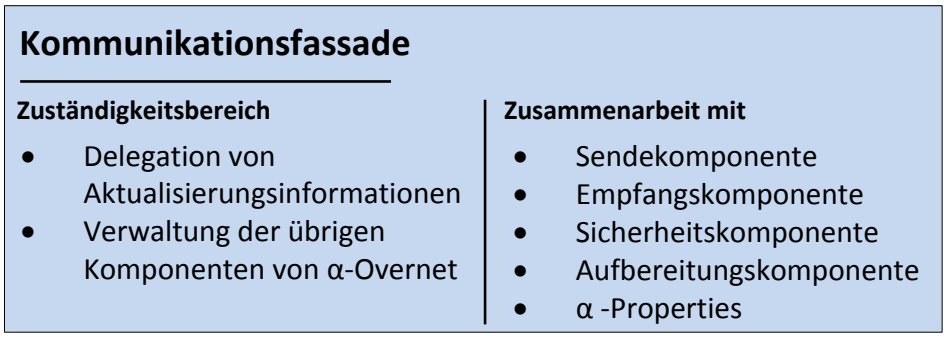


Abbildung 8.9: CRC-Card der Kommunikationsfassade

Die Kommunikationsfassade nimmt Aktualisierungsinformationen von α-Properties entgegen und delegiert diese an die Sendekomponente. Empfangene Nachrichten werden von der Empfangskomponente an die Fassade weitergereicht, von wo aus sie an eine

Aufbereitungskomponente delegiert werden. Diese modifiziert die ihr übergebenen Informationen derart, dass sie anschließend direkt in die Wissensdatenbank der in α-Properties gekapselten regelbasierten Bibliothek übernommen werden können.

Die Kommunikationsfassade fasst aus der Sicht von α-Properties die Funktionalitäten der einzelnen Komponenten unter einer einzigen übersichtlichen Schnittstelle zusammen. Auf diese Weise muss das aufrufende Subsystem keine Kenntnis über die interne Struktur von α-Overnet und die spezifischen Schnittstellen der Teilkomponenten besitzen.

Dies trägt zur Reduzierung der Kopplung zwischen α-Overnet und dem Gesamtsystem von α-Flow bei. Zudem wird die Modularität und Wartbarkeit erhöht, da Änderungen an der internen Struktur von α-Overnet keine Änderungen in α-Properties erfordern, solange die Konformität zur öffentlichen Schnittstelle der Kommunikationsfassade gewahrt bleibt.

8.4 Zusammenfassung

Der vorgestellte Architekturentwurf ermöglicht eine einfache Integration der generischen Kommunikationsschnittstelle in die existierende Subsystemstruktur von α-Flow. Durch Kapselung der für den Informationsaustausch mit anderen Akteuren benötigten Funktionalität wird sichergestellt, dass die übrigen Module bei Verwendung der Kommunikationsfassade unabhängig von der internen Struktur von α-Overnet agieren können.

Auf diese Weise können die für den Versand und Empfang von Nachrichten verwendeten Kommunikationsprotokolle ausgetauscht werden, ohne die Gültigkeit der öffentlichen Schnittstellen von α-Overnet zu tangieren. Auch die Sicherheitskomponente kann flexibel modifiziert werden, um den Einsatz benutzerdefinierter Verfahren zur Verschlüsselung und Verwaltung von elektronischen Signaturen zu ermöglichen.

Die in den Abschnitten 7.2, 7.3 und 7.4 beschriebenen Aspekte und Verfahren des fachlichen Lösungsansatzes werden in thematisch verwandte Subsysteme von α-Flow integriert, um auch für diese Module ein hohes Maß an Kapselung zu erreichen. Für die Kommunikation mit anderen Akteuren wird von diesen Subsystemen wiederum auf die generische Kommunikationsschnittstelle zugegriffen.

9 Feinentwurf der benötigten generischen Softwarekomponenten

Auf Grundlage der dargestellten allgemeinen Architektur wird der technische Feinentwurf der zu entwickelnden Java-Softwarekomponenten erarbeitet. Dazu wird zunächst auf die Struktur der generischen Kommunikationsschnittstelle α-Overnet und die Beziehungen zwischen den internen Teilmodulen eingegangen.

Darauf folgend werden die Schnittstellen zur Umsetzung der im fachlichen Lösungsansatz beschriebenen Mechanismen und Verfahren zur Herstellung einer Ordnung zwischen verschiedenen Versionen einer α-Card und der Erkennung nebenläufiger Änderungskonflikte präsentiert. Ein separater Abschnitt befasst sich mit dem grundlegenden Aufbau der Nachrichtentypen und Schnittstellen, die zur Realisierung des entwickelten Beitrittsprotokolls benötigt werden.

9.1 Schnittstellen von α-Overnet

Die einzelnen Komponenten des architektonischen Entwurfs für die generische Kommunikationsschnittstelle α-Overnet werden im Folgenden in Schnittstellen und Klassen der Programmiersprache Java überführt. Durch Kenntnis dieser Schnittstellen ist es den übrigen Komponenten von α-Flow möglich, mit anderen beteiligten Akteuren einer α-Episode über das angeschlossene Netzwerk zu kommunizieren.

9.1.1 Schnittstelle der Sendekomponente

Die öffentliche Schnittstelle der Sendekomponente `OvernetSender` ist, wie aus Abbildung 9.1 hervorgeht, sehr kompakt. Sie besteht aus einer einzigen Methode `sendUpdate(Object, Set<Participant>)`.

Beim Aufrufparameter vom Typ `Object` handelt es sich um die zu propagierenden Nutzinhalte. Diese sind das Ergebnis einer Benutzerinteraktion mit dem α-Doc über den

α-Editor oder eines internen technischen Vorgangs und werden vom Systemkern an die Kommunikationsschnittstelle übergeben.

Der zweite Parameter vom Typ `Set<Participant>` stellt die Datenstruktur für die Repräsentation aller Empfänger der zu sendenden Nachricht dar. Jeder Eintrag enthält die Identifikations- und Adressinformationen eines beteiligten Akteurs. Mit Hilfe dieser Informationen können von der Sendekomponente Nachrichten generiert und für den Transport vorbereitet werden.

Durch den Rückgabewert der Methode lässt sich eine Aussage über den Status der Nachrichtenübertragung treffen. Schlägt diese fehl, so wird `false` zurückgegeben. Im Erfolgsfall wird dem Aufrufer `true` zurückgemeldet.

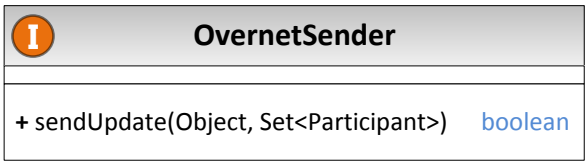


Abbildung 9.1: Schnittstelle der Sendekomponente

9.1.2 Schnittstelle der Empfangskomponente

Von konkreten Umsetzungen der Empfangskomponente wird verlangt, dass sie die Schnittstelle `OvernetReceiver` (Abbildung 9.2) implementieren. Diese besteht aus den beiden Methoden `receiveUpdates()` und `shutdown()`.

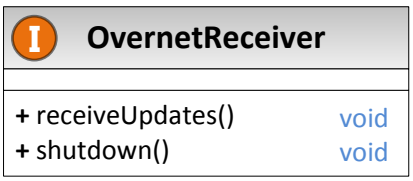


Abbildung 9.2: Schnittstelle der Empfangskomponente

Mit Hilfe von `receiveUpdates()` wird das Empfangen von Nachrichten angestoßen. Nachdem der Aufruf zurückgekehrt ist, kann sich der Aufrufer darauf verlassen, nebenläufig über neue Nachrichten informiert zu werden, sobald diese verfügbar sind. Da es im Kontext von α-Flow wichtig ist, ankommende Aktualisierungen so schnell wie möglich in das lokale α-Doc zu übernehmen, muss die Empfangskomponente nach erfolgreichem Start permanent auf das Netzwerk lauschen und ankommende Nachrichten unverzüglich entgegennehmen.

Ein Fehlschlagen der Methode ist ein schwerwiegender Systemfehler, der auf der Ebene der Empfangskomponente nicht behandelt werden kann, sondern an darüberliegende Module gemeldet werden muss. Damit wird sichergestellt, dass der Benutzer unmittelbar informiert wird, sobald er vom Kreis der übrigen Akteure abgetrennt agiert.

Die Methode `shutdown()` ermöglicht es, den nebenläufigen Empfang ankommender Nachrichten in einem konsistenten Zustand zu beenden. Konkrete Implementierungen müssen durch geeignete Maßnahmen sicherstellen, dass alle bereits empfangenen Nachrichten vor dem Beenden der Empfangskomponente an die übergeordneten Module zur Weiterverarbeitung übergeben werden.

9.1.3 Schnittstellen der Sicherheitskomponente

Um ein hohes Maß an Modularität zu gewährleisten, enthält die Sicherheitskomponente von α-Overnet zwei separate Schnittstellen. Diese definieren jeweils generische Methoden für einen der beiden Sicherheitsaspekte Verschlüsslung bzw. elektronische Signaturen.

Die Schnittstelle `CryptographyUtility` stellt die beiden Methoden `encrypt(Object)` und `decrypt(Object)` zur Verfügung (Abbildung 9.3).

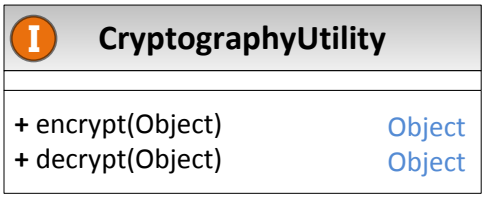


Abbildung 9.3: Schnittstelle der Verschlüsselungskomponente

Mit deren Hilfe wird die Verschlüsselung auf Seiten des Absenders einer Nachricht und die Entschlüsselung beim Empfänger realisiert. Beide erhalten als Eingabeparameter ein Java `Object`, das den Inhalt der zu verarbeitenden Nachricht repräsentiert. Nach der Verschlüsselung mit `encrypt(Object)` wird die verschlüsselte Nachricht zurückgegeben. Nach der entsprechenden Entschlüsselung durch `decrypt(Object)` erhält der Aufrufer die Klartextrepräsentation der Nachricht. Konkrete Implementierungen der Schnittstelle arbeiten mit spezialisierten Nachrichtentypen, welche zum Zeitpunkt der Schnittstellenentwicklung nicht antizipiert werden sollen. Die Umwandlung des übergebenen Parameters in einen geeigneten Typ bleibt der jeweiligen Implementierung überlassen.

Die Verarbeitung elektronischer Signaturen erfolgt über die gesonderte Schnittstelle `ElectronicSignatureUtility`. Wie in Abbildung 9.4 dargestellt, beinhaltet diese die Methoden `sign(Object)` und `verifySignature(Object)`.

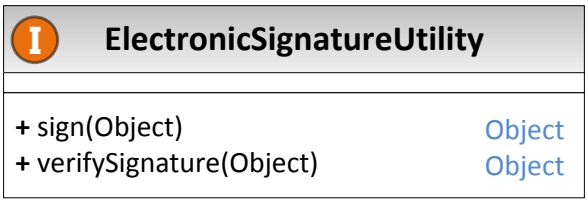


Abbildung 9.4: Schnittstelle der Komponente zur Verarbeitung elektronischer Signaturen

In beiden Fällen wird die zu verarbeitende Nachricht als Aufrufparameter übergeben. Nach der Signierung mittels `sign(Object)` wird die signierte Nachricht zurückgegeben. Wird die Authentizität einer ankommenden Nachricht bestätigt und ihre Signatur als gültig akzeptiert, so gibt `verifySignature(Object)` die unsignierte Repräsentation des Eingabeobjekts zurück, ansonsten `null`.

Zur Reduktion der Komplexität der beiden Schnittstellen nach außen hin, werden diese in einer gemeinsamen abstrakten Klasse gebündelt (Abbildung 9.5). Die Sende- und Empfangskomponenten müssen somit nur Kenntnis über die kombinierte Schnittstelle von `SecurityUtility` besitzen.

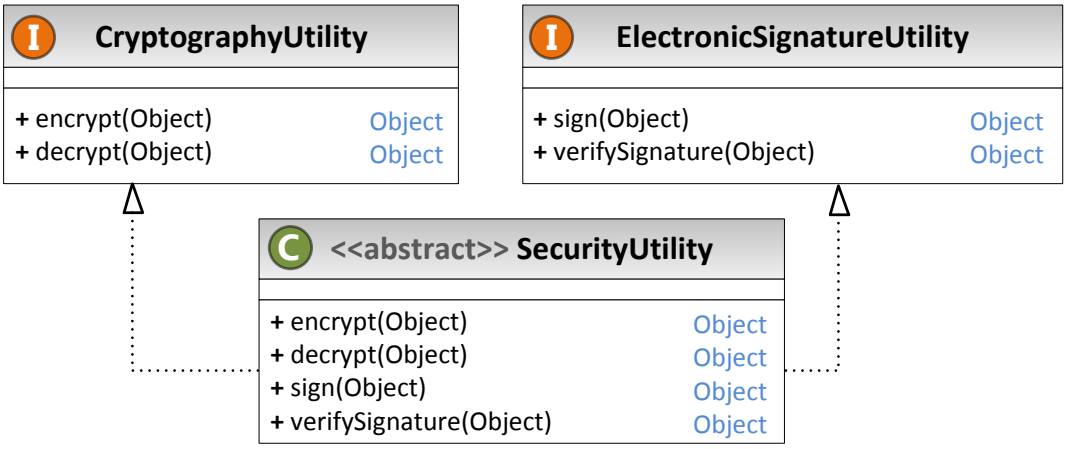


Abbildung 9.5: Gesamtstruktur der Sicherheitskomponente

9.1.4 Schnittstelle der Aufbereitungskomponente

Die Kommunikation zwischen α-Overnet und der regelbasierten Bibliothek von α-Properties wird über die Verwendung einer Drools[1] Pipeline [Tod10] abgewickelt. Diese Pipeline wird zur Laufzeit von einer Klasse verwaltet, die die Schnittstelle `PipelineManager` implementiert. Abbildung 9.6 zeigt die von dieser Schnittstelle definierten Methoden.

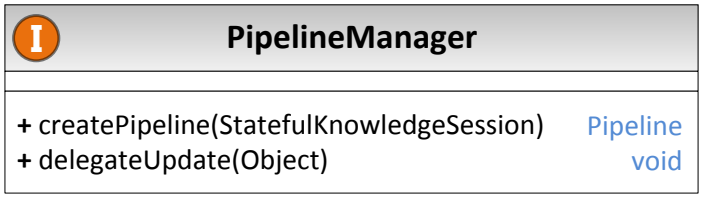


Abbildung 9.6: Schnittstelle der Aufbereitungskomponente

Durch einen Aufruf von `createPipeline(StatefulKnowledgeSession)` kann eine neue Pipeline erzeugt werden, die Datentransfers in eine laufende Instanz von Drools injiziert. Als Aufrufparameter vom Typ `StatefulKnowledgeSession` wird eine Referenz auf die gewünschte Instanz der Wissensdatenbank übergeben.

Mit `delegateUpdate(Object)` können Java-Objekte am Anfang der erzeugten Pipeline eingefügt werden. Am Ende dieser Pipeline werden sie nach verschiedenen Schritten der stufenweisen Aufbereitung in die Wissensdatenbank der regelbasierten Bibliothek von α-Flow übernommen.

9.1.5 Schnittstelle der Kommunikationsfassade

Wie auf der Ebene des Architekturentwurfs, so spielt die Kommunikationsfassade auch auf der Ebene der Software-Schnittstellen von α-Overnet eine zentrale Rolle. Die von `AlphaOvernetFacade` definierten Methoden (Abbildung 9.7) bieten diejenigen Funktionalitäten nach außen an, die von den übrigen Subsystemen von α-Flow zur Kommunikation mit anderen Akteuren benötigt werden.

Die Kommunikationsfassade ist nach dem Entwurfsmuster „Fassade“ [GHJV94] konzipiert. Dieses wird verwendet, um mehrere Schnittstellen eines Subsystems zu einer einzigen zusammenzufassen, um den Aufrufern den Umgang mit dem betreffenden Subsystem zu erleichtern. Dadurch wird die Kopplung zwischen Aufrufer und aufgerufenem

1 Für weitere Informationen siehe `www.jboss.org/drools`.

Subsystem reduziert. Durch Verbergen der einzelnen Schnittstellen wird zudem aus Sicht des Aufrufers die Komplexität des darunterliegenden Subsystems reduziert.

Die Methode `sendUpdate(Object, Set<Participant>)` kann verwendet werden, um die Nachrichten an andere Akteure zu versenden. Es ist vorgesehen, dass dieser Aufruf von der Kommunikationsfassade an die Methode `sendUpdate(Object, Set<Participant>)` der Schnittstelle `OvernetSender` delegiert wird. Auf dieselbe Art und Weise wird der Aufruf von `receiveUpdates()` an die Schnittstelle `OvernetReceiver` weitergeleitet.

Durch diese zusätzliche Indirektionsstufe muss die interne Struktur von α-Overnet nicht nach außen hin bekannt gemacht werden. Es ist zudem nicht notwendig, die Komponenten des Subsystems einzeln durch einen externen Aufrufer initialisieren zu lassen. Dies kann zentral von einer konkreten Implementierung von `AlphaOvernetFacade` übernommen werden.

Die Methode `delegateUpdate(Object)` verbindet α-Overnet mit einer Aufbereitungskomponente, die konform zur Schnittstelle `PipelineManager` implementiert ist. Die von der Empfangskomponente übergebenen Nachrichten werden durch einen Aufruf der entsprechenden Methode der Aufbereitungskomponente zur weiteren Verarbeitung durchgereicht.

Mit Hilfe der Methode `shutdown()` kann die Kommunikationsschnittstelle heruntergefahren werden. Implementierungen müssen aus Gründen der Konsistenz garantieren, dass alle Teilkomponenten von α-Overnet zuverlässig und sicher beendet werden. Dies bedeutet, dass alle ausstehenden Nachrichten noch versandt und alle bereits empfangenen Nachrichten dem verarbeitenden Subsystem α-Properties zugeführt werden.

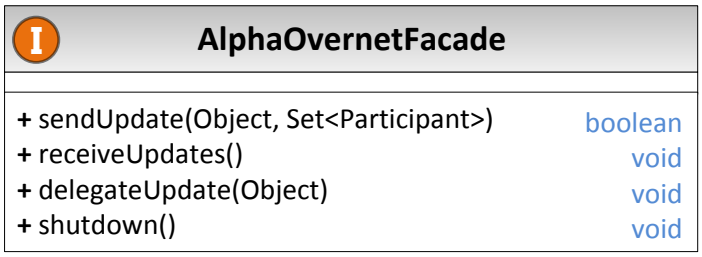


Abbildung 9.7: Schnittstelle der Kommunikationsfassade

9.1.6 Kommunikation zwischen den generischen Schnittstellen

Im Anschluss an die Betrachtung der statischen Struktur der einzelnen Komponenten von α-Overnet folgt eine Übersicht über die Kommunikationsabläufe zwischen den einzelnen Schnittstellen. Die vorgestellten Abläufe zeigen das auf konzeptioneller Ebene

intendierte Vorgehen. Die sichtbaren Schnittstellen der einzelnen Komponenten sind auf die illustrierte Weise zu verwenden, um ein konsistentes Systemverhalten zu gewährleisten.

9.1.6.1 Versand von Nachrichten

Am Versand einer Nachricht sind das Modul α-Properties sowie die Schnittstellen `AlphaOvernetFacade`, `OvernetSender` und die abstrakte Klasse `SecurityUtility` beteiligt (Abbildung 9.8).

Durch einen Aufruf der `sendUpdate`-Methode von `AlphaOvernetFacade` werden die zu propagierenden Nutzinhalte und die vorgesehenen Empfänger an α-Overnet übermittelt. Der Aufruf soll dabei nicht auf den Abschluss des Nachrichtenversands warten müssen, sondern sofort weiterarbeiten können. Dies ist von konkreten Implementierungen durch geeignete Datenstrukturen und Verfahren sicherzustellen.

Die erhaltenen Informationen werden an `OvernetSender` zur weiteren Verarbeitung delegiert. Dieser erzeugt eine implementierungsspezifische Nachricht und übergibt diese anschließend mit `sign(Object)` dem `SecurityUtility` zur Signierung. Die signierte Nachricht wird in einem zweiten Schritt erneut an das `SecurityUtility` übermittelt, um verschlüsselt zu werden.

Die auf diese Weise gesicherte Nachricht wird abschließend von `OvernetSender` an das Netzwerk versandt. Die Ausgestaltung dieses Übertragungsvorgangs liegt im Verantwortlichkeitsbereich konkreter Implementierungen.

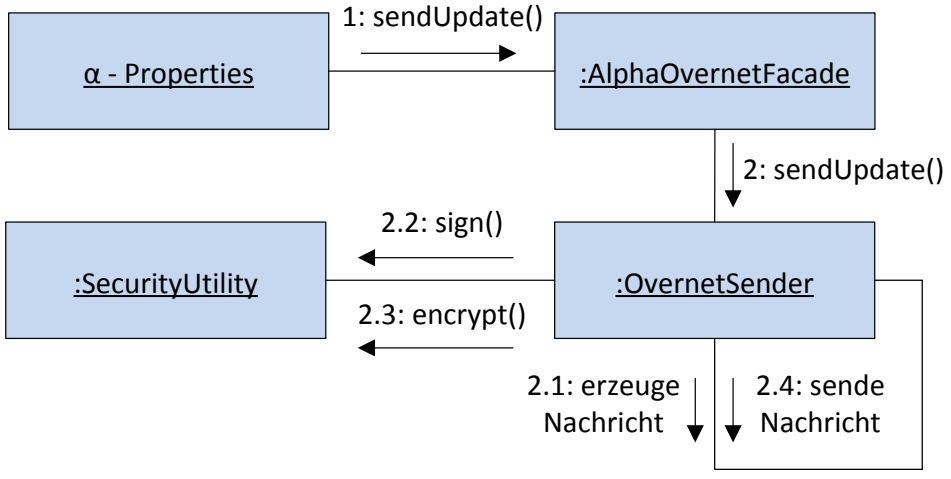


Abbildung 9.8: Kommunikation zwischen den generischen Schnittstellen beim Versand von Nachrichten

9.1.6.2 Empfang von Nachrichten

Auch der Empfang von Nachrichten erfordert die Kooperation mehrerer Komponenten. Beteiligt sind neben dem `OvernetReceiver` das `SecurityUtility`, die `AlphaOvernetFacade` und das Modul α-Properties (Abbildung 9.9).

Ankommende Nachrichten werden vom `OvernetReceiver` in Empfang genommen. Mit Hilfe des `SecurityUtility` wird die Nachricht zuerst mit `decrypt(Object)` entschlüsselt, um anschließend ihre elektronische Signatur mit `verifySignature(Object)` zu verifizieren. Ist dies erfolgreich, so wird aus der unsignierten Repräsentation der Nachricht der eigentliche Nutzinhalt extrahiert.

Mittels der `delegateUpdate`-Methode wird dieser an die `AlphaOvernetFacade` weitergeleitet. Von dort aus wird der Nutzinhalt dem `PipelineManager` übergeben, um schließlich in die Wissensdatenbank der regelbasierten Bibliothek von α-Properties eingefügt zu werden.

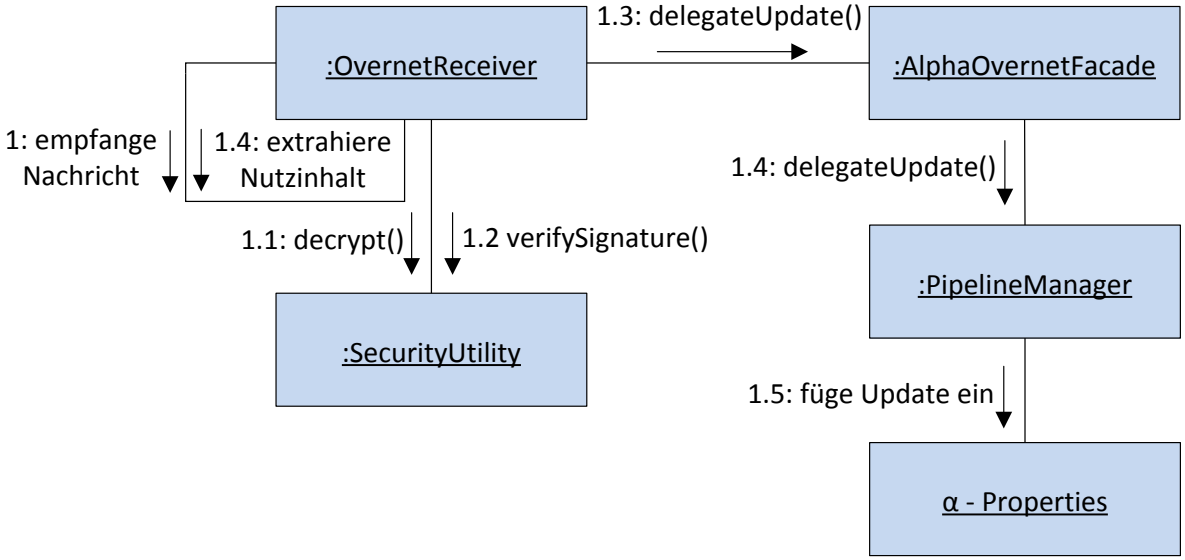


Abbildung 9.9: Kommunikation zwischen den generischen Schnittstellen beim Empfang von Nachrichten

9.2 Schnittstellen für die Verwaltung der Versionshistorie und die Konflikterkennung

Auch für die Umsetzung der in Kapitel 7.2 vorgestellten Konzepte zur Herstellung einer Ordnung auf den verschiedenen Versionen einer α-Card und der Erkennung nebenläufiger

Änderungskonflikte werden generische Schnittstellen definiert. Auf diese Weise wird sichergestellt, dass zukünftige interne Änderungen und Optimierungen sich nicht auf die aufrufenden Subsysteme von α-Flow auswirken.

9.2.1 Generische Schnittstelle für logische Zeitstempel

Die nachfolgend beschriebene Schnittstelle für logische Zeitstempel macht keine Annahmen über die zur internen Realisierung verwendeten Datenstrukturen. Auf diese Weise ist es möglich, auf Änderungen der Kommunikationscharakteristika des Nachrichtenaustausches zwischen Akteuren zu reagieren. Durch erweiterte Transportgarantien seitens des eingesetzten Kommunikationskanals, die über die in Abschnitt 5.3 geforderten hinausgehen, sind unter Umständen spezialisierte Implementierungen möglich (siehe dazu auch die Diskussion verwandter Arbeiten in Kapitel 6).

Die von jedem Zeitstempelkonzept zwingend zu implementierende Schnittstelle `LogicalTimestamp` ist in Abbildung 9.10 dargestellt.

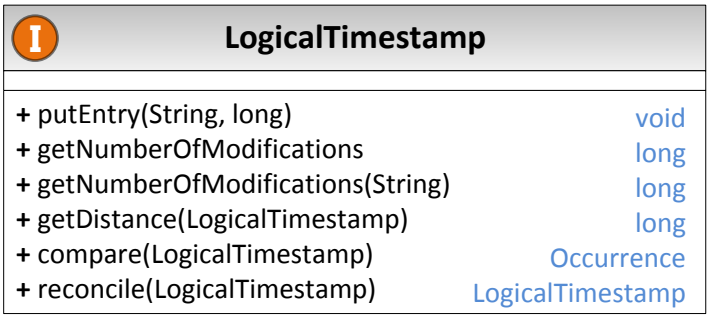


Abbildung 9.10: Schnittstelle der logischen Zeitstempel

Durch Aufruf der Methode `putEntry(String, long)` wird derjenige Eintrag des Zeitstempels, der durch den ersten Parameter charakterisiert wird, auf den durch den zweiten Parameter angegebenen Zählerwert gesetzt. Existiert der entsprechende Eintrag noch nicht, so wird er erzeugt.

Mit Hilfe von `getNumberOfModifications()` kann berechnet werden, wie viele Modifikationen durch den betrachteten Zeitstempel abgebildet werden. Für eine verbesserte Auswertbarkeit muss zudem die Funktion `getNumberOfModifications(String)` angeboten werden, die die Anzahl von Modifikationen durch einen spezifischen Akteur zurückgibt.

Die Aufgabe von `getDistance(LogicalTimestamp)` ist es, den Abstand zweier Zeitstempel auf der Versionshistorie des zugehörigen Prozessartefakts auszugeben. Diese

Berechnung kann, abhängig von der konkreten Implementierung, beispielsweise unter Verwendung von `getNumberOfModifications()` durchgeführt werden.

Aufrufe von `compare(LogicalTimestamp)` liefern die kausale Beziehung zwischen zwei logischen Zeitstempeln zurück. Für dieses Ergebnis stehen die in der Enumeration `Occurrence` (Abbildung 9.11) definierten Möglichkeiten zur Verfügung.

Zwei Zeitstempel können zum einen identisch sein (`Occurrence.IDENTICAL`) oder aufeinander folgen (`Occurrence.PRECEDING`, bzw. `Occurrence.FOLLOWING`). Daneben ist es jedoch auch möglich, dass die verglichenen Zeitstempel in keinem kausalen Verhältnis zueinander stehen (`Occurrence.CONCURRENT`). Dies ist genau dann der Fall, wenn ein nebenläufiger Änderungskonflikt auf dem zugehörigen Prozessartefakt vorliegt. Der Wert `Occurrence.UNDEFINED` repräsentiert ein Fehlschlagen des Vergleichs. Dies kann unter anderem durch unzulässige nebenläufige Speicherzugriffe zur Laufzeit oder durch den Vergleich von Instanzen untereinander inkompatibler Implementierungen der Schnittstelle `LogicalTimestamp` ausgelöst werden.

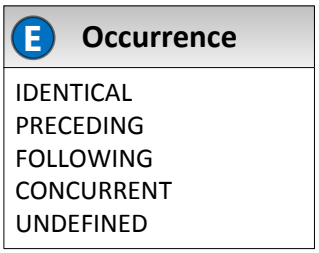


Abbildung 9.11: Möglichkeiten für Beziehungen zwischen Zeitstempeln

Schließlich wird für die Konsolidierung der logischen Zeitstempel nach der erfolgreichen Auflösung eines parallelen Änderungskonflikts eine Funktion zur Erzeugung eines korrekten Zeitstempels, der die Konfliktlösung widerspiegelt, benötigt. Dafür wird `reconcile(LogicalTimestamp)` angeboten, das auf Basis zweier Zeitstempel einen neuen, gültigen erstellt.

9.2.2 Schnittstelle des Verfahrens zur Ordnung von Versionen von α-Cards

Die Einordnung einer ankommenden α-Card in die Versionshistorie wird in Kooperation mit dem angebundenen Versionsverwaltungssystem realisiert. Die in Abschnitt 7.2.2.3 dargestellten Abläufe werden von einer dedizierten Softwarekomponente übernommen, die kompatibel zur generischen Schnittstelle `LogicalTimestampHistoryUtility` sein

muss (Abbildung 9.12). Die Modellierung als eigenständige Schnittstelle gewährleistet eine verbesserte Wartbarkeit und sichert die zukünftige Erweiterbarkeit.

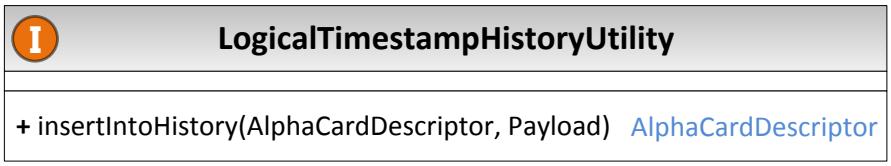


Abbildung 9.12: Schnittstelle des Verfahrens zur Ordnung von Versionen

Durch Aufruf der Methode `insertIntoHistory(AlphaCardDescriptor, Payload)` wird aus der regelbasierten Bibliothek heraus der Einfügevorgang angestoßen (Abbildung 9.13). Eine konkrete Implementierung ist dafür zuständig, unter Verwendung der Schnittstellen des Versionsverwaltungssystems, durch Analyse des logischen Zeitstempels der α-Card den Deskriptor und einen eventuell vorhandenen Payload an der korrekten Position in der Versionshistorie zu platzieren.

Als Rückgabewert liefert die Methode den `AlphaCardDescriptor`, der nach dem Einfügen der ankommenden Version an erster Stelle der Versionshistorie steht. Mit diesem kann der aktuell im Arbeitsspeicher vorgehaltene Deskriptor, der direkt zur graphischen Darstellung der zugehörigen α-Card im α-Editor verwendet wird, ersetzt werden. Auf diese Weise erhält der Benutzer eine direkte graphische Rückmeldung über die durchgeführte Aktualisierung.

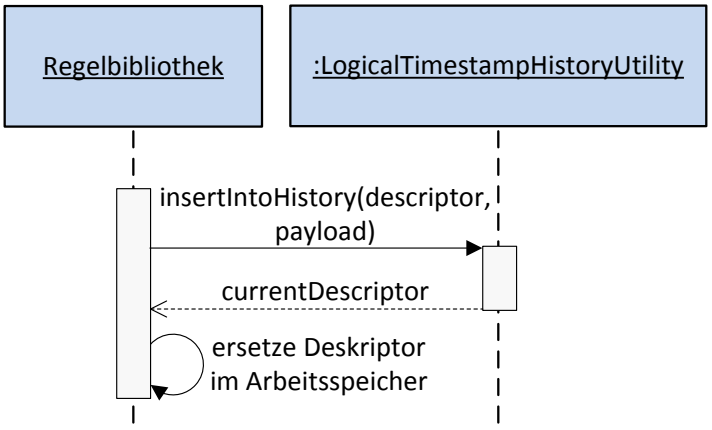


Abbildung 9.13: Verwendung der Schnittstelle `LogicalTimestampHistoryUtility`

9.3 Schnittstellen für den Beitritt neuer Akteure

Um das vorgestellte Konzept zum zuverlässigen Beitritt neuer Akteure angemessen umzusetzen, werden eigene Schnittstellen geschaffen, mit denen sich die benötigten Strukturen und Verfahren in das Gesamtsystem integrieren lassen.

9.3.1 Umsetzung der benötigten Nachrichtentypen

Wie in Abschnitt 7.4 zusammengestellt, sind zur Umsetzung des entwickelten Beitrittsprotokolls vier verschiedene neue Typen von Nachrichten erforderlich. Deren Inhalte werden im Rahmen des Feinentwurfs in je einer eigenen Klasse modelliert.

Für den sequentiellen Beitritt von neuen Akteuren werden Repräsentationen der Nachrichtentypen *sequentielle Rückmeldung* und *sequentielle Synchronisation* benötigt. Zu diesem Zweck werden die in Abbildung 9.14 gezeigten Klassen `SequentialJoinCallback` und `SequentialJoinSynchronisation` eingeführt.

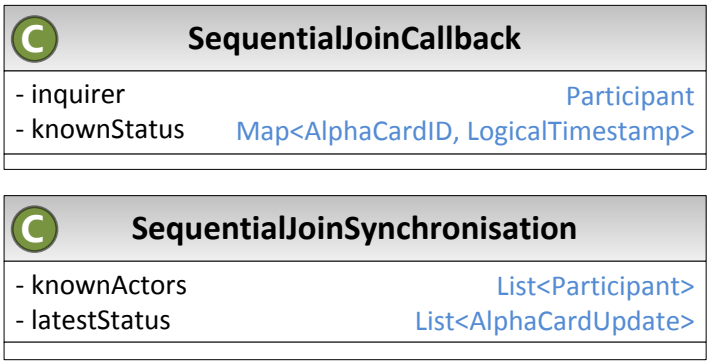


Abbildung 9.14: Nachrichtentypen für den sequentiellen Beitritt

Instanzen beider Klassen funktionieren als passive Behälter für die zu übertragenden Informationen. `SequentialJoinCallback` beinhaltet die Informationen über den beigetretenen Akteur in Form eines Objektes vom Typ `Participant` und den lokal bekannten Prozesstatus, dargestellt durch eine `Map<AlphaCardID,LogicalTimestamp>` mit einem Element pro bekannter α-Card und zugehöriger Version.

Objekte der Klasse `SequentialJoinSynchronisation` enthalten eine Liste der lokal bekannten Akteure. Darüber hinaus beinhalten sie auch die neuesten lokal bekannten Versionen der von ihnen modifizierbaren α-Cards, um den anfragenden Akteur auf den neuesten Stand des Prozessverlaufs zu bringen.

Der parallele Beitritt kann durch die Auswertung von Instanzen der Klassen `ParallelJoinCallback` und `ParallelJoinSynchronisation` koordiniert werden. Ab-

bildung 9.15 zeigt, dass auch diese beiden Klassen die gemäß Beitrittsprotokoll benötigten Informationen enthalten. `ParallelJoinCallback` ist identisch aufgebaut wie die korrespondierende Klasse zur sequentiellen Rückmeldung, besitzt jedoch eine entsprechend abweichende Semantik.

`ParallelJoinSynchronisation` enthält genau wie der Nacchrichtentyp zur *sequentiellen Synchronisation* eine Liste mit den neuesten lokal bekannten Versionen der α-Cards. Zum Anstoßen einer weiteren Antwort des Empfängers wird darüber hinaus der lokale Kenntnisstand über den Prozess in Form einer `Map<AlphaCardID,LogicalTimestamp>` eingebunden.

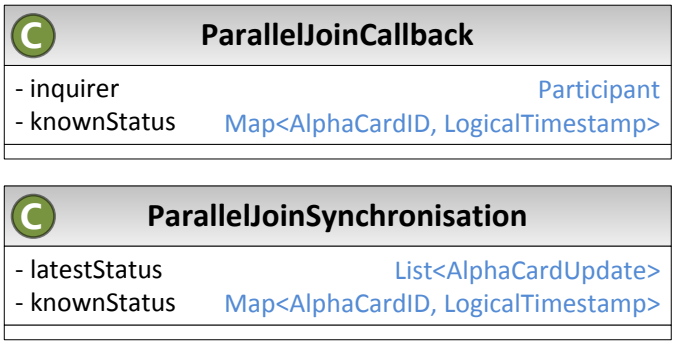


Abbildung 9.15: Nachrichtentypen für den parallelen Beitritt

9.3.2 Schnittstelle zur Auswertung und Generierung von Beitrittsnachrichten

Gemäß Abschnitt 7.4 läuft der Beitritt neuer Akteure nach einem festen Kommunikationsprotokoll ab. Die Generierung der benötigten Beitrittsnachrichten und die Auswertung der Beitrittsanfragen anderer Akteure wird zur besseren Wartbarkeit in einer eigenen Schnittstelle `JoinUtility` gekapselt (Abbildung 9.16). Diese kommuniziert mit der regelbasierten Bibliothek des Moduls α-Properties.

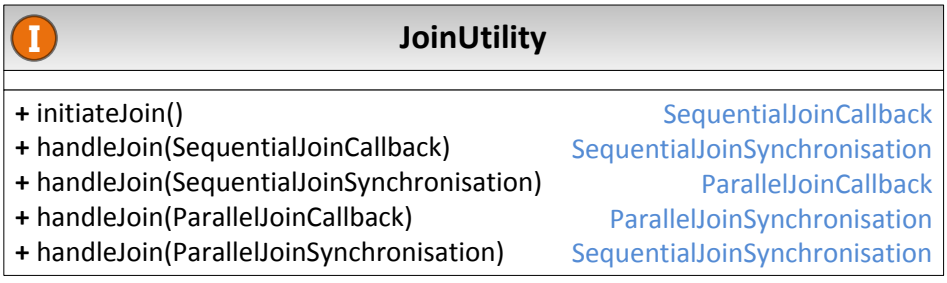


Abbildung 9.16: Schnittstelle zur Auswertung und Generierung von Beitrittsnachrichten

Durch einen Aufruf von `initiateJoin()` stößt ein neuer Akteur seinen Beitritt zur α-Episode an. Den verschiedenen `handleJoin`-Methoden werden ankommende Beitrittsanfragen als Eingabeparameter übergeben (Abbildung 9.17). Deren Inhalt ist von einer konkreten Implementierung auszuwerten. Die dazu benötigten Informationen erhält `JoinUtility` auf Anfrage aus dem Speicher der regelbasierten Bibliothek. Mit diesen Informationen wird eine Antwortnachricht erzeugt, die nach ihrer Rückgabe an die Regelbibliothek an die jeweiligen Empfänger versandt wird.

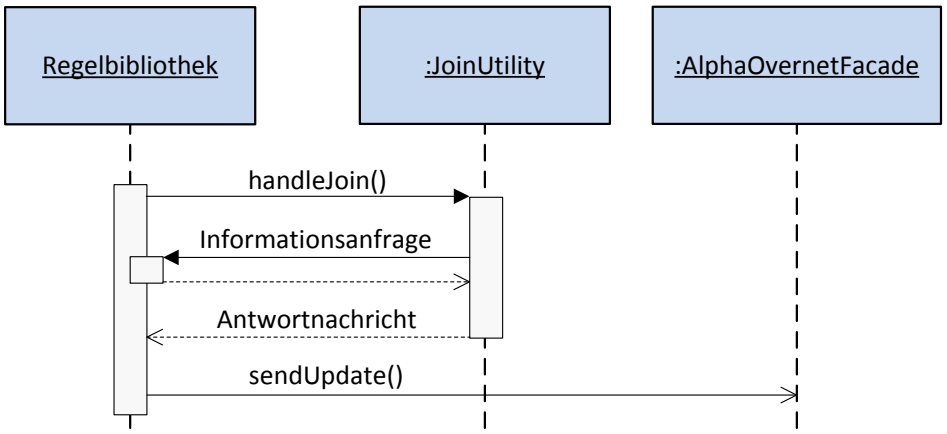


Abbildung 9.17: Verwendung der Schnittstelle `JoinUtility`

9.4 Zusammenfassung

Die entworfenen generischen Schnittstellen legen das Fundament für individuelle Implementierungen der Konzepte des fachlichen Lösungsansatzes. Die vorgestellten allgemeinen Kommunikationsabläufe für die Benutzung der Schnittstellen ermöglichen die zielgerichtete Kooperation der einzelnen Komponenten von α-Flow.

Implementierungen spezifischer Kommunikationsprotokolle müssen die Schnittstellen des Moduls α-Overnet realisieren und diese um eigene Konzepte zur Generierung und Auswertung von Nachrichten ergänzen. Für eine Umsetzung der erarbeiteten Konzepte zur Synchronisation und Erkennung nebenläufiger Änderungskonflikte sowie zum Beitritt neuer Akteure stehen eigene generische Schnittstellen zur Verfügung. Angepasst an das eingesetzte Zeitstempelkonzept und das verwendete Versionsverwaltungssystem können mit Hilfe dieser Schnittstellen sämtliche Aspekte des fachlichen Lösungsansatzes in α-Flow integriert werden.

10 Ausgewählte Aspekte der technischen Umsetzung

Aufbauend auf dem entwickelten Architekturkonzept sowie den beschriebenen generischen Schnittstellen und Klassen wird im Rahmen der vorliegenden Arbeit eine prototypische Implementierung zur verteilten Datensynchronisation in α-Flow angefertigt. Ausgewählte Aspekte dieser technischen Umsetzung werden im Folgenden näher erläutert. Die Ausführungen beschränken sich dabei auf die wichtigsten Kernpunkte, um dem Leser einen fundierten Überblick über die Konzepte und Funktionalitäten der resultierenden Softwaremodule zu vermitteln.

Die vorgestellten Aspekte umfassen zum einen die E-Mail-basierte Implementierung der Schnittstellen von α-Overnet im Modul α-OffSync und die zugehörigen technischen Lösungskonzepte. Zum anderen wird die Implementierung des Synchronisationskonzepts auf Basis des erarbeiteten Zeitstempelkonzepts und des angebundenen Versionsverwaltungssystems thematisiert. Darüber hinaus wird auf relevante Umgestaltungsarbeiten an anderen Modulen von α-Flow eingegangen.

10.1 Struktur von α-OffSync

Entsprechend den Entwürfen aus Abschnitt 8.3 ist der Systembaustein α-OffSync, welcher die E-Mail-basierte Implementierung der generischen Kommunikationsschnittstelle α-Overnet beinhaltet, modular aufgebaut (Abbildung 10.1). Für den Versand der Nachrichten an andere Akteure greift die Sendekomponente auf das Simple Mail Transfer Protocol (siehe Abschnitt 4.2) zurück. Die Empfangskomponente setzt das in Abschnitt 4.3 vorgestellte Internet Message Protocol ein, um ankommende Nachrichten entgegenzunehmen.

Für die Integration beider E-Mail-Protokolle kommt die JavaMail API (vgl. Abschnitt 4.4) zum Einsatz. Auch für die Erzeugung und Verwaltung von E-Mail-Nachrichten werden Schnittstellen und Klassen der JavaMail API genutzt.

Innerhalb der Sicherheitskomponente wird das asymmetrische Kryptosystem OpenPGP [CDF+07] verwendet. Auf diese Weise wird sichergestellt, dass sowohl Prinzipien des Datenschutzes als auch der Authentisierung beim Austausch von Informationen über das angeschlossene Netzwerk gewährleistet bleiben. Als Implementierung der dazu benötigten Algorithmen und Datenstrukturen greift α-OffSync auf die Bouncy Castle Crypto API (vgl. Abschnitt 4.5.2) zurück.

Um die aus dem Internet bzw. Intranet von anderen Akteuren erhaltenen Prozessinformationen lokal im Modul α-Properties zu verarbeiten, werden diese von der Kommunikationsfassade in eine Pipeline der regelbasierten Bibliothek Drools eingefügt.

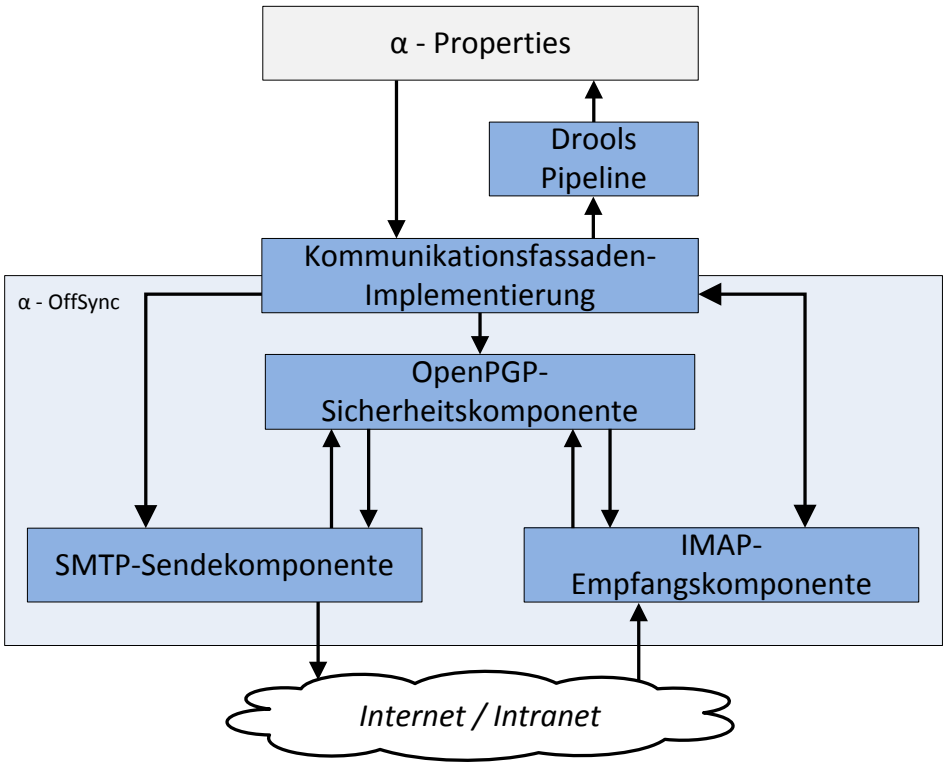


Abbildung 10.1: Architektur von α-OffSync

10.1.1 Benutzerauthentifizierung

Für den Versand und Empfang von E-Mails muss sich der Benutzer zunächst am SMTP- bzw. IMAP-Server anmelden. Um diese Anmeldeprozedur zu vereinfachen, wird das Anmelden an den Server von α-Flow nach einmaliger Eingabe der Zugangsdaten automatisch vorgenommen.

Die benötigten Informationen werden beim ersten Start des α-Doc über einen graphischen Konfigurationsdialog vom Benutzer abgefragt (Abbildung 10.2). Erhoben werden neben der zu verwendenden E-Mail-Adresse des Akteurs der entsprechende Benutzername und das zugehörige Passwort. Für die Kontaktierung des gewünschten IMAP- bzw. SMTP-Servers wird zudem die jeweilige Serveradresse und der zugehörige Port eingelesen.

Abbildung 10.2: Konfigurationsdialog zur Übernahme der Zugangsinformationen

Diese Informationen werden in der lokalen Konfigurationsdatei des α-Doc persistent abgelegt, um beim nächsten Systemstart ohne erneute Benutzerinteraktion zur Verfügung zu stehen. Beim Initialisierungsvorgang des Systems werden die gespeicherten Informationen aus der Konfigurationsdatei ausgelesen und an die Kommunikationsfassade weitergereicht. Zum Start der einzelnen Komponenten von α-OffSync wird von der Kommunikationsfassade auf diese Informationen zurückgegriffen.

Um den Umgang mit den Zugangsdaten für die beteiligten Module zu vereinfachen, werden diese Informationen zur Laufzeit in einer Instanz der Hilfsklasse `MailAuthenticator` in kompakter Form gebündelt (Abbildung 10.3). Über öffentliche Zugriffsmethoden können die benötigten Felder gesetzt und entsprechend wieder ausgelesen werden.

Darüber hinaus wird die von der abstrakten Klasse `Authenticator` aus dem Paket `javax.Mail` geerbte Methode `getPasswordAuthentication()` angeboten. Eine derartige Methode mit entsprechender Signatur wird von den Modulen der JavaMail API vorausgesetzt, um beim Verbindungsaufbau die Zugangsdaten für einen E-Mail-Server zu erhalten. Instanzen von `MailAuthenticator` können somit direkt der JavaMail API übergeben werden, um auf einfache Art und Weise eine Serveranmeldung zu realisieren.

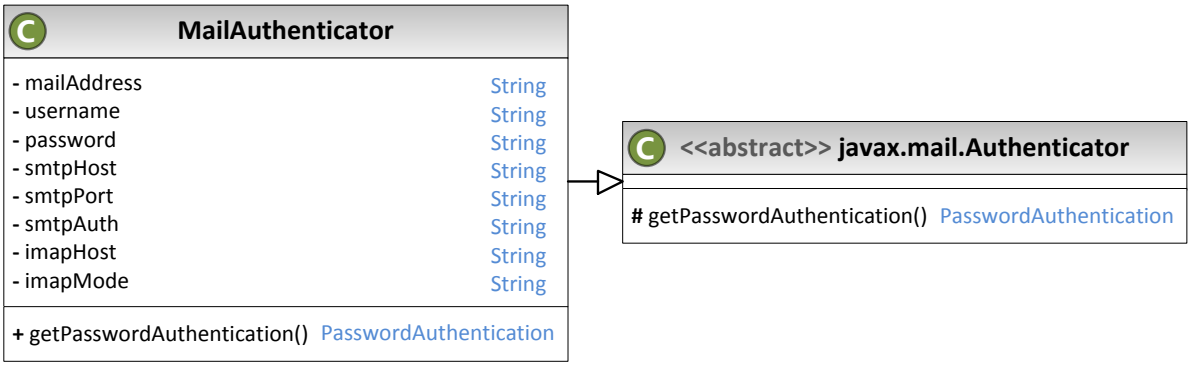


Abbildung 10.3: Struktur der Klasse `MailAuthenticator`

10.1.2 Struktur der ausgetauschten MIME-Nachrichten

Die von α-Flow generierten E-Mail-Nachrichten sind nach den Prinzipien der Multipurpose Internet Mail Extensions (MIME) aufgebaut. Die Grundlagen der Syntax und Semantik dieses standardisierten Nachrichtenformats sind in Kapitel 4.1.2 beschrieben.

Wie in Abschnitt 7.1 dargestellt enthält eine Nachricht im Kontext von α-Flow zwei verschiedene Arten von Informationen. Zum einen sind das die Metainformationen zum Nachrichtenversand und zur Koordinierung der Verarbeitung einer Nachricht. Zum anderen enthält eine Nachricht die eigentlichen Nutzinhalte in Form von Aktualisierungsinformationen.

Die für die Zustellung einer Nachricht benötigten Informationen sind Bestandteil der Header Section gemäß den Ausführungen in Abschnitt 4.2. Die Metainformationen für die α-Flow-spezifische Verarbeitung werden in der Betreffzeile der jeweiligen Nachricht abgelegt. Beim Abruf der E-Mail-Nachrichten vom Server mittels IMAP wird zunächst nur der Inhalt der Betreffzeile zum Client transportiert (vgl. Abschnitt 4.3.2). Durch Unterbringung der α-Flow-spezifischen Koordinationsinformationen im Betreff ist sichergestellt, dass nicht relevante Nachrichten von weiteren Bearbeitungsschritten ausgenommen werden können, bevor sie komplett auf den lokalen Rechner des Benutzers transferiert werden.

Syntaktisch sind die Betreffzeilen der generierten Nachrichten nach folgendem Schema aufgebaut:

`alphaFlow-Message <Identifikator der α-Episode> <Identifikator des sendenden Akteurs und dessen physischem Arbeitsplatz>`

Mit Hilfe des einleitenden Tokens `alphaFlow-Message` wird dem Empfänger angezeigt, dass die zugehörige Nachricht Teil eines verteilten Prozesses im Rahmen von α-Flow

ist. Alle Nachrichten, deren Betreff dieses Token nicht enthält, können bei der weiteren Bearbeitung übergangen werden, weil sie keinen Bezug zu α-Flow aufweisen.

Durch Analyse des zweiten Tokens `<Identifikator der α-Episode>` kann ermittelt werden, ob sich eine α-Flow-Nachricht tatsächlich auf die α-Episode des lokalen α-Doc bezieht. Der Identifikator wird hierbei aus der lokal persistierten Konfigurationsdatei des α-Doc entnommen. Damit ist die Verwendung einer E-Mail-Adresse für die parallele Teilnahme an mehreren α-Episoden möglich, da stets gewährleistet ist, dass die ankommenden Nachrichten der korrekten α-Episode zugeordnet werden können.

Das letzte Token `<Identifikator des sendenden Akteurs und dessen physischem Arbeitsplatz>` wird für die Umsetzung der in Kapitel 7.3 vorgestellten Lösungskonzepte zur Verwaltung der Divergenz zwischen Akteur, E-Mail-Konto und physischem Arbeitsplatz benötigt. Da alle E-Mails zur Unterstützung verschiedener Rechner pro Akteur auch immer an die E-Mail-Adresse des sendenden Akteurs verschickt werden müssen, wird durch das Token aus Gründen der Systemperformanz verhindert, dass die Nachrichten auf dem Rechner, von dem sie versendet wurden, ein zweites Mal verarbeitet werden. Das Token setzt sich zusammen aus dem im lokalen Collaboration Resource Artifact (CRA) hinterlegten Identifikator des sendenden Akteurs und der Hardware-Adresse des sendenden Rechners. Der Empfänger überprüft durch Vergleich mit seinem lokalen CRA und seiner lokalen Hardware-Adresse, ob eine ankommende E-Mail von seinem lokalen Rechner verschickt wurde. Ist dies der Fall, so wird die Nachricht von der weiteren Verarbeitung ausgenommen.

Die eigentlichen Nutzinhalte der E-Mails, die zu propagierenden Informationen, werden in serialisierter Form als mit OpenPGP signierte und verschlüsselte XML-Dateien[1] im Anhang der E-Mail transportiert. Nur wenn die Analyse der Betreffzeile einer E-Mail ergeben hat, dass sie zur lokalen Verarbeitung geeignet ist, wird ihr Anhang auf den lokalen Rechner heruntergeladen.

10.1.3 Implementierung der Kommunikationsfassade

Alle Komponenten von α-OffSync werden von der in Abbildung 10.4 dargestellten Kommunikationsfassade `AlphaOffSyncFacade` initialisiert. Die dazu benötigten Informationen erhält die Kommunikationsfassade vom Subsystem α-Properties, bei dessen Initialisierung die Kommunikation mit dem Netzwerk angestoßen wird.

1 Extensible Markup Language: Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten. Für weitere Informationen siehe `http://www.w3.org/TR/xml/`.

Für einen erfolgreichen Verbindungsaufbau zum jeweiligen SMTP- bzw. IMAP-Server werden die Zugangsinformationen des lokalen Akteurs benötigt, welche in der Konfigurationsdatei des α-Doc hinterlegt sind (vgl. Abschnitt 10.1.1). Diese Informationen werden aus der Konfigurationsdatei extrahiert und zur weiteren Verwendung in einer Instanz der Klasse `MailAuthenticator` hinterlegt. Nach der Initialisierung der Sicherheitskomponente zur Verschlüsselung des Nachrichtentransfers und der Verwaltung von elektronischen Signaturen werden die Komponenten zum Versand und Empfang von E-Mail-Nachrichten sowie die Komponente zur Pipeline-Verwaltung erzeugt.

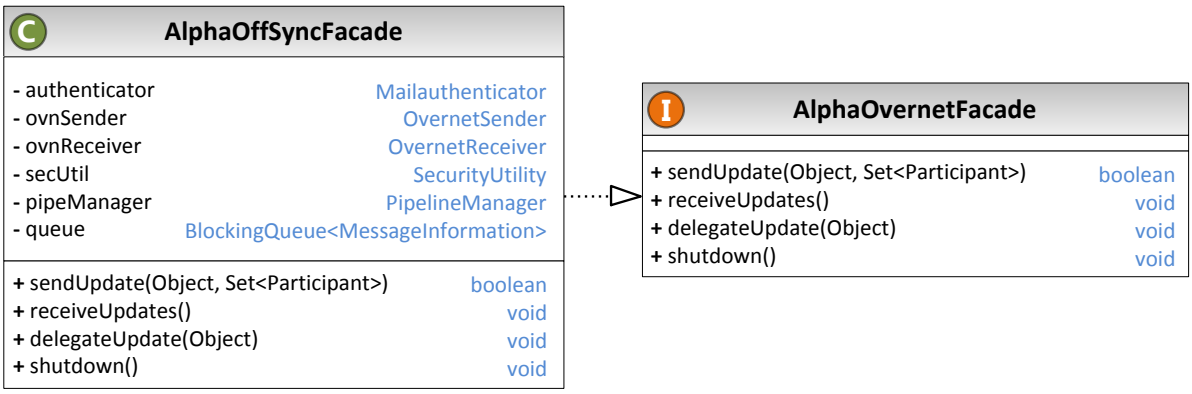


Abbildung 10.4: Struktur der Klasse `AlphaOffSyncFacade`

10.1.3.1 Nebenläufige Kommunikation mit dem Netzwerk

Sowohl die Sendekomponente als auch die Empfangskomponente laufen getrennt von der Kommunikationsfassade jeweils in einem eigenen Thread. Auf diese Weise ist sichergestellt, dass zu jeder Zeit Nachrichten entgegengenommen werden können und die Verwendung des α-Doc nicht durch zeitaufwendige Datenübertragungen an das Netzwerk blockiert wird.

Die Ausführung des Empfangs-Threads wird durch einen Aufruf der Methode `receiveUpdates()` der Kommunikationsfassade angestoßen. Bis zum Aufruf der Methode `shutdown()` werden nun vom Thread der Sendekomponente alle empfangenen Nachrichten asynchron an die Kommunikationsfassade zur weiteren Verarbeitung übermittelt.

Komplexer gestaltet sich der nebenläufige Versand von E-Mail-Nachrichten. Um zu vermeiden, dass die Kommunikationsfassade auf die Rückkehr von Aufrufen der Methode `sendUpdate(Object, Set<Participant>)` warten muss, werden alle zu versendenden Nachrichten in einer Warteschlange abgelegt. Diese ist als thread-sichere `BlockingQueue<MessageInformation>` implementiert (Abbildung 10.5).

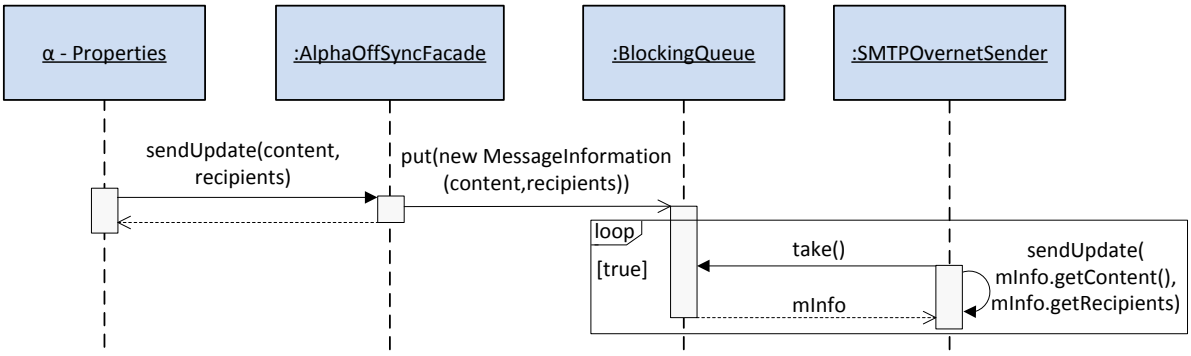


Abbildung 10.5: Verwendung einer `BlockingQueue` zum nebenläufigen Nachrichtenversand

Der Hauptthread von α-Flow, in dem die Kommunikationsfassade läuft, legt bei jedem Aufruf der Sendemethode ein Objekt vom Typ `MessageInformation` in der Warteschlange ab. Jeder dieser Einträge enthält neben dem zu versendenden Nutzinhalt auch eine Liste der vorgesehenen Empfänger (Abbildung 10.6).

Abbildung 10.6: Struktur der Hilfsklasse `MessageInformation`

Die Sendekomponente entnimmt in ihrem Thread solange Einträge aus der Warteschlange und verarbeitet diese, bis keine zu versendenden Nachrichten mehr vorgefunden werden. Dann wartet der Sender passiv, bis die Kommunikationsfassade neue Einträge in die Warteschlange gelegt hat, und fährt dann mit der Verarbeitung fort.

10.1.3.2 Kommunikation mit α-Properties

Die Kommunikationsfassade interagiert bidirektional mit dem Modul α-Properties. Zum einen werden die öffentlichen Methoden der Kommunikationsfassade aus der regelbasierten Bibliothek heraus aufgerufen. Zum anderen übermittelt die Kommunikationsfassade die Nutzinhalte der ankommenden Nachrichten an die regelbasierte Bibliothek zur weiteren Verarbeitung.

Für die Kommunikation von α-Properties zur Kommunikationsfassade registriert sich die Kommunikationsfassade als globale Variable in der regelbasierten Bibliothek. Dadurch kann aus allen Regeln der Bibliothek auf die Netzwerkkomponenten zugegriffen werden.

Für die umgekehrte Richtung wird über eine bei der Initialisierung der Kommunikationsfassade erstellte Instanz der Klasse `PipelineManagerImpl` eine Verbindung zur regelbasierten Bibliothek in Form einer Pipeline aufgebaut (Abbildung 10.7). Die interne Struktur der verwendeten Pipeline entspricht dabei den in [Tod10] näher beschriebenen Konzepten.

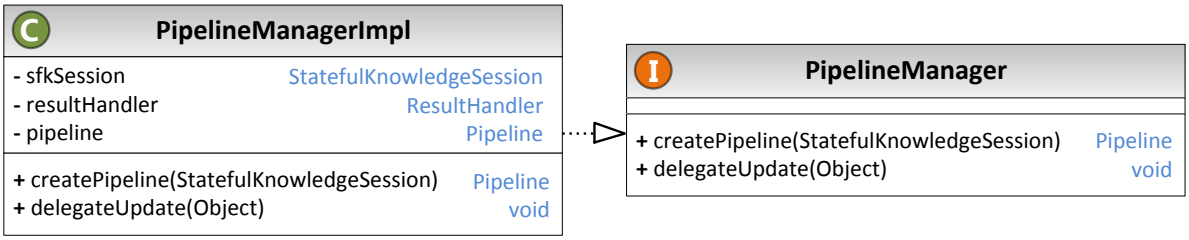


Abbildung 10.7: Struktur der Klasse `PipelineManagerImpl`

10.1.4 Versand von Nachrichten mittels SMTP

Die Klasse `SMTPOvernetSender` (Abbildung 10.8) ist für den Versand von Nachrichten an das Netzwerk verantwortlich.

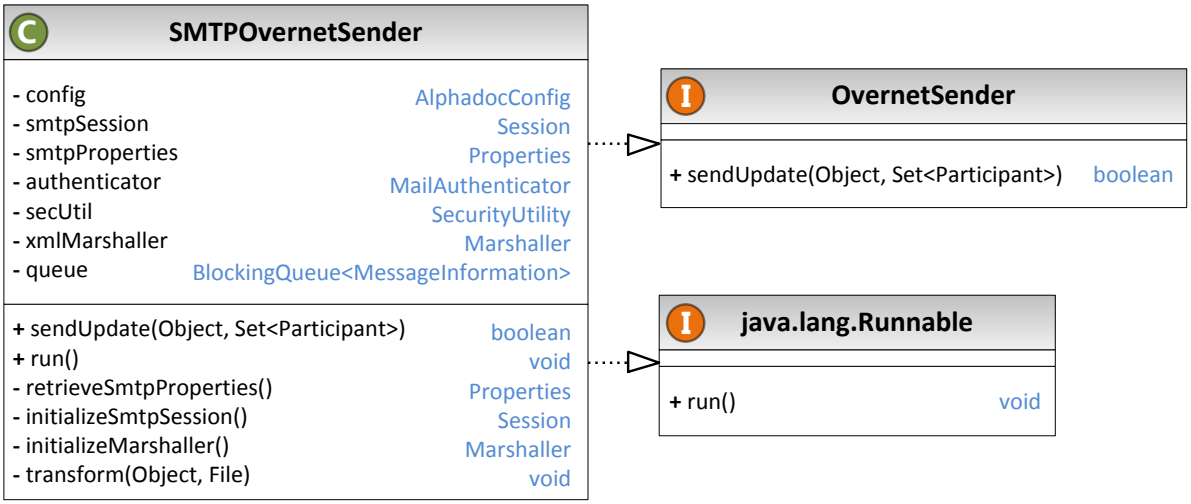


Abbildung 10.8: Struktur der Klasse `SMTPOvernetSender`

Die Informationen über die zu versendenden Nachrichten entnimmt sie der in Abschnitt 10.1.3.1 beschriebenen Warteschlange, in welche die Kommunikationsfassade alle Nachrichten einfügt. Nach dem Start durch den Aufruf der Methode `run()` bearbeitet die Sendekomponente in einer Endlosschleife in einem eigenen Thread die ihr übergebenen Nachrichten.

Für den erfolgreichen Versand der Nachrichten muss eine Verbindung zum SMTP-Server des aktiven Akteurs aufgebaut werden. Die dazu notwendigen Initialisierungen werden von den Hilfsmethoden `retrieveSmtpProperties()` und `initializeSmtpSession()` vorgenommen, die auf die von der Kommunikationsfassade übergebenen Zugangsdaten zurückgreifen.

Bei Aufruf der Methode `sendUpdate(Object, Set<Participant>)` wird eine neue MIME-Nachricht vom Typ `javax.Mail.MimeMessage` erstellt, die nach dem in Abschnitt 10.1.2 vorgestellten Schema mit Inhalt gefüllt wird. Dazu werden die zu versendenden Nutzinhalte mit Hilfe der Java Architecture for XML Binding (JAXB)[1] unter Verwendung der Methode `transform(Object, File)` in XML-Dateien serialisiert und als Anhang der MIME-Nachricht hinzugefügt. Nach dem Hinzufügen der elektronischen Signatur des Absenders und erfolgreicher Verschlüsselung durch eine Implementierung der Schnittstelle `SecurityUtility` wird die Nachricht gemäß dem Vorgehen aus Abschnitt 4.4.2 der Transportkomponente der JavaMail API zum Versand übergeben.

10.1.5 Empfang von Nachrichten mittels IMAP

Der IMAP-basierte Empfang ankommender Nachrichten wird von der gesonderten Klasse `IMAPOvernetReceiver` (Abbildung 10.9) realisiert.

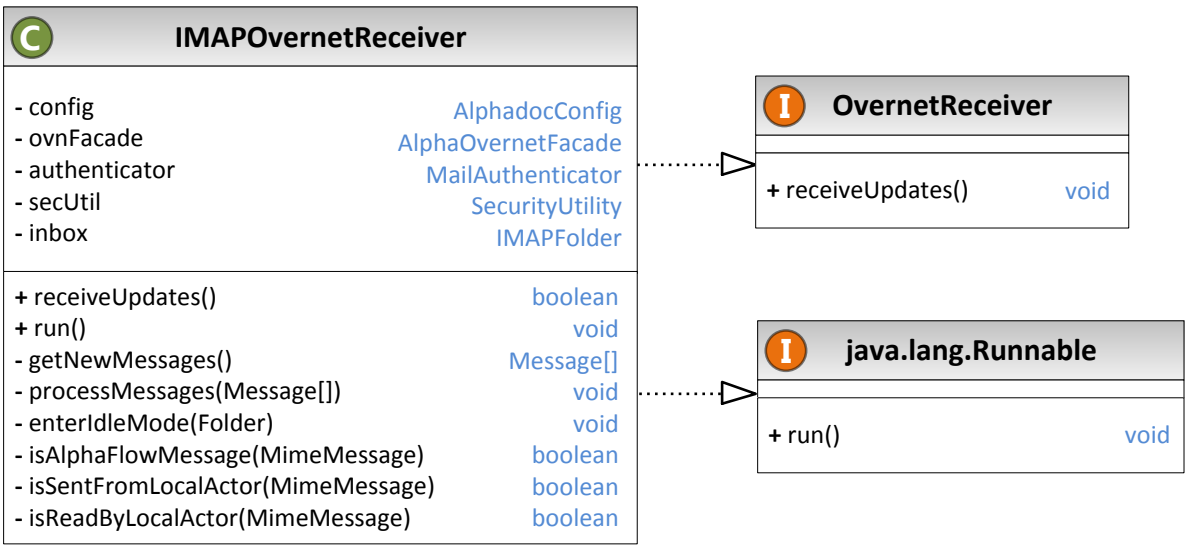


Abbildung 10.9: Struktur der Klasse `IMAPOvernetReceiver`

1 Für weitere Informationen siehe `http://jaxb.java.net/`.

Die Nutzinhalte aus den Dateianhängen aller empfangenen Nachrichten werden an die Kommunikationsfassade weitergereicht, um sie von dort aus an α-Properties zur lokalen Verarbeitung zu übermitteln.

10.1.5.1 Verwendung von IMAP UIDValidity

Nach jedem Start der Empfangskomponente durch die Methode `run()` findet mit `getNewMessages()` ein initialer Abruf von Nachrichten aus dem Postfach mit der Bezeichnung „INBOX“ statt. Da die für α-Flow relevanten Nachrichten zur Umsetzung der Lösungskonzepte aus Abschnitt 7.3 nach ihrer Verarbeitung nicht vom IMAP-Server gelöscht werden, ist es wünschenswert, nicht bei jedem Start alle im Postfach vorhandenen Nachrichten erneut auf ihre Relevanz für die aktuelle α-Episode zu prüfen.

Zu diesem Zweck wird auf das in Kapitel 4.3.1 erläuterte Konzept der UIDValidity zurückgegriffen. Die jeweils letzte bekannte UID einer Nachricht wird zusammen mit der UIDValidity des Postfachs in der lokalen Konfigurationsdatei des α-Doc persistiert.

Ist der aktuelle Wert der UIDValidity größer als der aus der Konfigurationsdatei, so sind die bekannten UIDs der Nachrichten im Postfach ungültig geworden. Dies erfordert eine vollständige Prüfung aller Nachrichten im Postfach, um festzustellen, welche Nachrichten noch nicht verarbeitet worden sind. Andernfalls sind die UIDs der Nachrichten immer noch gültig. Dann genügt es, alle Nachrichten zu analysieren, deren UID größer ist als die in der Konfigurationsdatei persistierte UID der zuletzt verarbeiteten Nachricht.

10.1.5.2 Ereignisbasierte Benachrichtigung über eingetroffene Nachrichten

Um unverzüglich ohne engmaschiges Nachfragen beim Server über das Eintreffen neuer Nachrichten informiert zu werden, verwendet die Klasse `IMAPOvernetReceiver` das in Abschnitt 4.3.3 vorgestellte IDLE Kommando. Hierzu wird gemäß den Konzepten der JavaMail API zur Ereignisverarbeitung (vgl. Abschnitt 4.4.3) mit der Funktion `enterIdleMode(Folder)` ein Beobachter eingerichtet, der mit Hilfe des IDLE Kommandos auf Veränderungen der Anzahl von E-Mails im angegebenen Postfach wartet.

Sobald neue Nachrichten verfügbar sind, wird der Beobachter benachrichtigt und die Methode `processMessages(Message[])` ausgeführt, mit deren Hilfe die hinzugekommenen Nachrichten analysiert und verarbeitet werden. Anschließend wartet der Beobachter wieder passiv auf neue Aktualisierungen.

10.1.5.3 Filterung ankommender Nachrichten

Zur Reduzierung des Datenübertragungsvolumens werden nur diejenigen Nachrichten vollständig vom Server heruntergeladen, die tatsächlich für die aktuelle α-Episode relevante Nutzinhalte transportieren.

Hierfür wird zuerst mit den beiden Methoden `isAlphaFlowMessage(MimeMessage)` und `isSentFromLocalActor(MimeMessage)` überprüft, ob eine ankommende Nachricht gemäß der in Abschnitt 10.1.2 dargestellten Struktur der Betreffzeile zu berücksichtigen ist oder nicht.

Bei den zu diesem Zeitpunkt noch nicht aussortierten Nachrichten wird darüber hinaus unter Verwendung der Methode `isReadByLocalActor(MimeMessage)` sichergestellt, dass keine Nachricht auf dem selben Rechner vom selben Akteur mehr als einmal heruntergeladen und lokal verarbeitet wird. Jede verarbeitete Nachricht wird von der Sendekomponente mit einem IMAP Flag (vgl. Abschnitt 4.3.1) versehen, das den Identifikator des lokalen Akteurs und die Hardware-Adresse des lokalen Rechners enthält. Wird ein derartiges Flag bei einer Nachricht vorgefunden und stimmt dessen Inhalt mit den Informationen über den lokalen Akteur und dessen Rechner überein, so wird die Nachricht von der Verarbeitung ausgenommen.

10.1.6 Verschlüsselung und elektronische Signaturen

Die im Rahmen von α-OffSync eingesetzte Implementierung `OpenPGPSecurityUtility` der abstrakten Klasse `SecurityUtility` stützt sich auf den OpenPGP-Standard zur Verschlüsselung und Verwaltung elektronischer Signaturen.

Da es sich bei OpenPGP um ein asymmetrisches Kryptosystem handelt, müssen private und öffentliche Schlüssel verwaltet werden. Wie in Abbildung 10.10 zu sehen ist, werden die Schlüssel von den Objekten `secretKeyRing` und `publicKeyRing` repräsentiert. Über diese Objekte vom Typ `File` wird auf die lokal persistierten Schlüsselbunde zugegriffen, um die benötigten Informationen zu entnehmen. Als Sicherung der geheimen Schlüssel des lokalen Akteurs ist ein Passwort erforderlich, welches in Form der Variablen `secretKeyRingPassword` vorgehalten wird.

Neben den geerbten Methoden der generischen Schnittstelle besitzt die vorliegende Implementierung zusätzlich die Methode `addPublicKeys(File)`. Mit ihr können neue öffentliche Schlüssel zum lokal persistierten Schlüsselbund hinzugefügt werden. Dies ist beispielsweise dann erforderlich, wenn neue Akteure der α-Episode beigetreten sind und auch ihnen verschlüsselte Nachrichten zugestellt werden sollen.

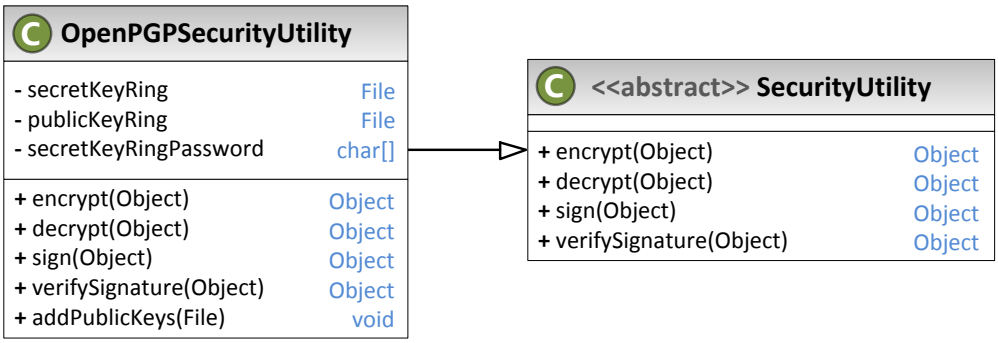


Abbildung 10.10: Struktur der Sicherheitskomponente `OpenPGPSecurityUtility`

Die verwendete Bouncy Castle Crypto API (BC API) greift zur Umsetzung der Konzepte von OpenPGP auf Komponenten der Java Cryptography Architecture (vgl. Abschnitt 4.5.1) zurück, als Provider im Kontext der JCA kommt jedoch weiterhin die Referenzimplementierung von Oracle zum Einsatz. Da die aktuelle Version der BC API aufgrund interner Umstrukturierungen nicht mehr unmittelbar als Provider eingebunden werden muss, sind keine Modifikation an der lokalen virtuellen Maschine von Java oder eine Signierung der auszuführenden Programmdateien notwendig, um starke Verschlüsselungsalgorithmen zu verwenden. Dies vereinfacht die Integration der kryptographischen Verfahren in α-Flow.

10.2 Verwaltung der Versionshistorie und Erkennung nebenläufiger Änderungskonflikte

Das fachliche Lösungskonzept zur Verwaltung der Versionshistorie und zur Erkennung nebenläufiger Änderungskonflikte ist zentraler Bestandteil der vorliegenden Arbeit. Für seine Umsetzung kommt eine Implementierung des vorgestellten Zeitstempelkonzepts zum Einsatz. Die Analyse und Modifikation der Versionshistorie erfolgt über eine generische Schnittstelle, die vom verwendeten Versionsverwaltungssystem entkoppelt ist.

10.2.1 Implementierung des Konzepts logischer Zeitstempel

Zur Umsetzung des in Kapitel 7.2 definierten Zeitstempelkonzepts existiert eine konkrete Implementierung der Schnittstelle `LogicalTimestamp`. Die Klasse `VersionMap` bietet alle Methoden an, die für eine korrekte Behandlung der Zeitstempel erforderlich sind (Abbildung 10.11).

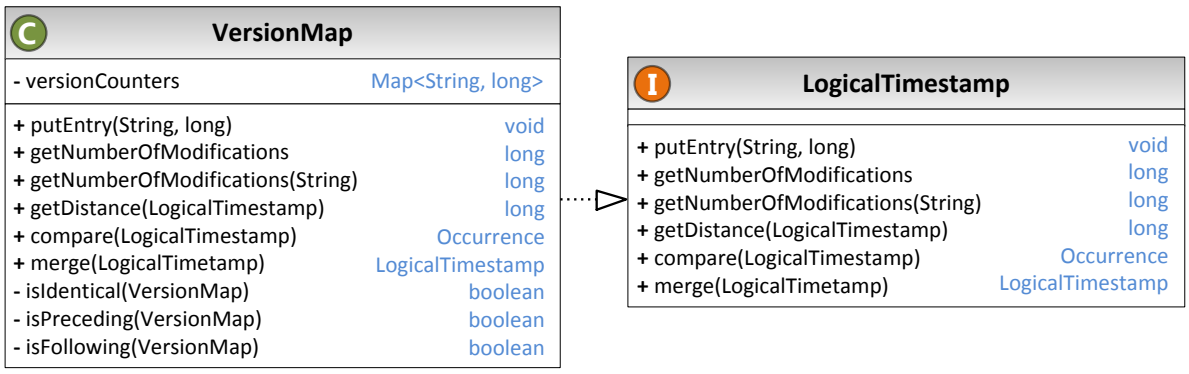


Abbildung 10.11: Struktur der Implementierung des entworfenen Zeitstempelkonzepts

Zur Speicherung der einzelnen Werte der logischen Zähler kommt eine parametrisierte `Map<String,long>` zum Einsatz. Das Argument vom Typ `String` bezeichnet hierbei den Identifikator des jeweiligen Akteurs, wohingegen das zweite Argument vom Typ `long` den zugehörigen aktuellen Zählerwert darstellt.

Auf die Verwendung von einfachen Vektoren bzw. Arrays wurde bei der Implementierung verzichtet, da aufgrund unterschiedlicher Empfangsreihenfolgen von Beitrittsnachrichten nicht gewährleistet ist, dass bei allen Akteuren die Reihenfolge der Dateneinträge identisch ist. Um fehlerhafte Vergleiche zwischen Zeitstempeln zu vermeiden, ohne die Zugriffsgeschwindigkeit zu beinträchtigen, wird eine Datenstruktur verwendet, die Zugriffe auf einzelne Einträge über Schlüsselwerte erlaubt.

Um einen Vergleich zweier Instanzen von `VersionMap` über einen Aufruf von `compare(LogicalTimestamp)` zu realisieren, werden sequentiell die Berechnungsvorschriften aus Abschnitt 7.2.1 angewendet. Die verschiedenen Möglichkeiten einer kausalen Beziehung zwischen den zu vergleichenden Objekten werden hierbei durch die Hilfsmethoden `isIdentical(VersionMap)`, `isPreceding(VersionMap)` und `isFollowing(VersionMap)` abgeprüft.

10.2.2 Umsetzung des Versionierungsverfahrens

Zur Umsetzung des entwickelten Verfahrens zur Ordnung von verschiedenen Versionen einer α-Card kommt die Klasse `VersionMapHistoryUtility` (Abbildung 10.12) zum Einsatz, die die Schnittstelle `LogicalTimestampHistoryUtility` implementiert. Für den Zugriff auf die persistierten Versionen der α-Cards wird die Schnittstelle `Workspace` verwendet, die in Abschnitt 10.3.1 näher vorgestellt wird. Für die Navigation entlang der Versionshistorie wird auf eine Implementierung der abstrakten Klasse `Historian`

zurückgegriffen, die mit dem integrierten Versionsverwaltungssystem von α-Flow kooperiert und häufig benötigte Funktionalitäten kapselt. Dadurch ist es möglich, die Versionshistorie einer α-Card zu durchsuchen und einzelne Einträge zu modifizieren. Dies von Scott A. Hady entworfene komplexe Schnittstelle von `Historian` und die zugehörige Implementierung werden in der Diplomarbeit [Had11] zum Versionsverwaltungssystem Hydra detailliert erläutert, weswegen an dieser Stelle nicht weiter auf sie eingegangen wird.

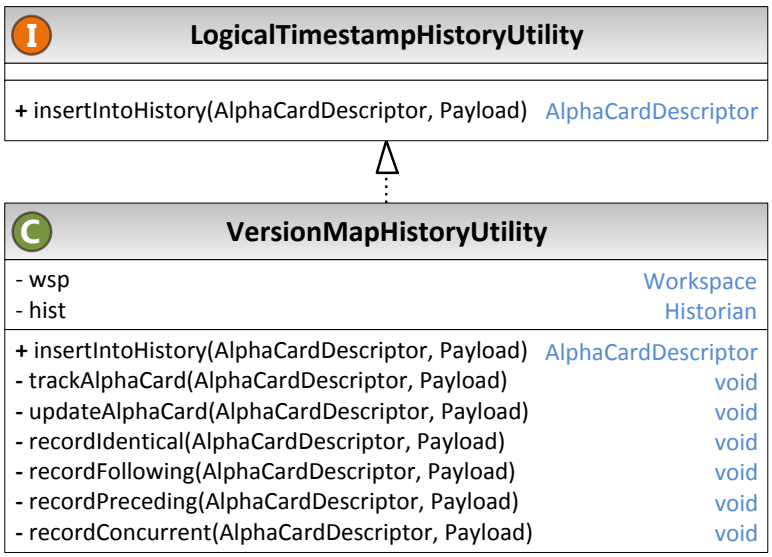


Abbildung 10.12: Struktur der Implementierung zur Verwaltung der Versionshistorie

Der Aufruf der Methode `insertIntoHistory(AlphaCardDescriptor, Payload)` initiiert die Verarbeitung des logischen Zeitstempels des übergebenen Deskriptors gemäß dem Verfahren aus Abschnitt 7.2.2.3. Die einzelnen zu behandelnden Fälle der kausalen Beziehung zwischen der aktuellen Version und der einzufügenden werden mittels individueller Hilfsfunktionen bearbeitet.

Für die Suche nach der korrekten Einfügeposition in der Versionshistorie werden die lokal bekannten Versionen der betreffenden α-Card schrittweise in umgekehrt chronologischer Reihenfolge vom Versionsverwaltungssystem in den Speicher geladen. Nach dem Ladevorgang werden die jeweiligen Versionsmaps analysiert, um zu entscheiden, ob die korrekte Einfügeposition bereits gefunden wurde.

Sobald dies der Fall ist, werden der übergebene Deskriptor und der eventuell vorhandene Payload der empfangenen Version der α-Card lokal persistiert. Über öffentlich zugängliche Funktionen des Versionsverwaltungssystems wird in der Versionshistorie an

der Einfügeposition eine neue gültige Version erzeugt, die auf die persistent abgelegten Informationen verweist.

Im Fall nebenläufiger Änderungskonflikte wird, wie im fachlichen Lösungsansatz in Abschnitt 7.2.2.3 beschrieben, ein Konfliktzweig erzeugt. Die vom Konflikt betroffenen Versionen werden in diesen Zweig verschoben. Anschließend wird eine neue aktuell gültige Version an der ersten Stelle der Versionshistorie erzeugt. Deren Versionsmap entspricht dem elementweisen Maximum der Versionsmaps der bisher aktuellen und der ankommenden Version. Als Nutzinhalt der neuen Version wird, gemäß der allgemein anwendbaren Rücksetzstrategie, der Nutzinhalt der letzten nicht konfliktbehafteten Version verwendet. Um diese Version zu bestimmen, wird mit Hilfe des Versionsverwaltungssystems solange entlang der Historie der α-Card in die Vergangenheit navigiert, bis eine Version gefunden wird, deren Versionsmap nicht mit der der ankommenden Version in Konflikt steht.

Über den Rückgabewert der Methode `insertIntoHistory(AlphaCardDescriptor, Payload)` wird der aufrufenden Regelbibliothek im Subsystem α-Properties der lokal aktuelle Deskriptor übergeben. Die Regelbibliothek verfährt anschließend wie in Abschnitt 9.2.2 illustriert, um den im Arbeitsspeicher vorgehaltenen Deskriptor zu ersetzen.

10.3 Refaktorisierung und Umgestaltung der vorhandenen Systembausteine

Zur Umsetzung der vorgestellten Lösungskonzepte waren an den vorhandenen Systembausteinen von α-Flow umfangreiche Anpassungen vorzunehmen. Für eine verbesserte Lesbarkeit des Quellcodes und eine einfachere Wartbarkeit wurden im Zuge dieser Anpassungen Refaktorisierungen verschiedener existierender Klassen durchgeführt. Ausgewählte Punkte dieser Umgestaltung sollen im Folgenden kurz vorgestellt werden.

10.3.1 Integration einer generischen Schnittstelle zur Dateiverwaltung

Zum erleichterten Umgang mit den lokal persistierten α-Cards wurde von Scott A. Hady eine generische Schnittstelle `Workspace` zur Dateiverwaltung konzipiert (Abbildung 10.13), die im Verlauf der vorliegenden Arbeit in α-Flow integriert wurde.

Unter Verwendung einer `Workspace`-Implementierung kann das α-Flow-System unabhängig von konkreten Operationen des lokalen Dateisystems arbeiten. Anstatt aus-

schließlich direkt mit Dateien des Dateisystems zu arbeiten, kann auf Wunsch auf die Repräsentationen von persistierten Prozessartefakten zugegriffen werden.

Zur Integration der Schnittstelle `Workspace` und einer mit dem Versionsverwaltungssystem Hydra kompatiblen Implementierung waren Modifikationen des Moduls zur Koordination des Systemstarts erforderlich. Darüber hinaus wurde das Modul α-Properties, über das Zugriffe auf persistierte Prozessartefakte erfolgen, auf die Verwendung der von `Workspace` definierten Schnittstelle umgestellt.

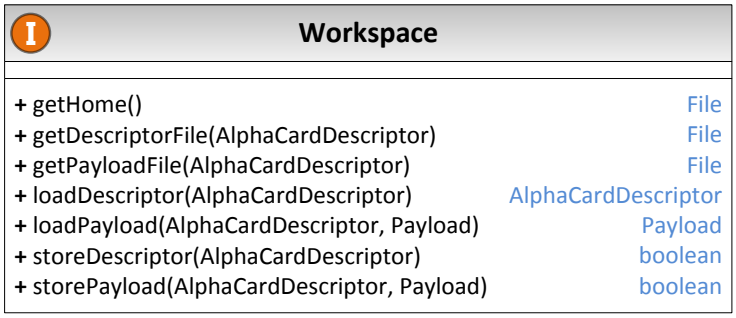


Abbildung 10.13: Struktur der Schnittstelle `Workspace`

10.3.2 Überarbeitung der integrierten Regelbibliothek

Aufgrund der Integration des entwickelten Synchronisationskonzepts und Refaktorisierungen an verschiedenen Modulen von α-Flow musste die regelbasierte Bibliothek angepasst werden. Hierbei wurde vor allem Wert auf eine Vereinfachung der vorhandenen Regeln und eine Reduktion der Gesamtanzahl von Regeln gelegt, um die Wartbarkeit des integrierten Regelpakets zu verbessern.

Zur Vermeidung von Timing-Problemen aufgrund der nebenläufigen Ausführung einzelner Systembausteine wurden die Regeln zur Bestimmung der Änderungsrechte einzelner Adornments in die Fassade des Moduls α-Properties verlagert. Sie liegen nun in Form von gewöhnlichem Java-Code vor und können leicht an geänderte Anforderungen angepasst werden.

Die zahlreichen Hilfsfunktionen zur Weitergabe von Prozessinformationen an α-Overnet wurden zu einer einzigen generischen Funktion zusammengefasst. Auf diese Weise konnte auch die Schnittstelle der Sendekomponente übersichtlicher gestaltet werden, da nicht für jeden Typ von zu versendenden Ereignissen individuelle Sendemethoden bereitgestellt werden müssen.

Es war zudem auch möglich, Regeln zur Modifikation bestimmter, bei jeder α-Card vorhandener, Adornments aus der Regelbasis zu entfernen. Diese werden vollständig von den generischen Regeln des adaptiven Metadatenmodells α-Adaptive subsumiert.

10.3.3 Refaktorisierung der existierenden Ereignistypen

Um unter Berücksichtigung der in Kapitel 5.3 beschriebenen Charakteristika möglicher Kommunikationsprotokolle sicherzustellen, dass alle versandten Nachrichten vom Empfänger korrekt verarbeitet werden können, war es erforderlich, die existierenden Ereignistypen umzugestalten.

Bisher enthielten die Ereignisnachrichten ausschließlich die geänderten Adornments bzw. die geänderten Payloadinformationen einer α-Card. Unter der Annahme, dass die Empfangsreihenfolge der Nachrichten nicht mit ihrer Sendereihenfolge übereinstimmt, kann es somit aufgrund fehlender Kommutativität von Änderungsoperationen zu Inkonsistenzen kommen.

Um zu garantieren, dass beim Empfänger nach Übernahme der empfangenen Informationen immer exakt der selbe Status eines Prozessartefakts wie beim Urheber der entsprechenden Änderungen vorliegt, müssen alle status-bezogenen Informationen vollständig in den propagierten Nachrichten enthalten sein.

Die vorhandenen Nachrichtentypen wurden daher wie folgt modifiziert: Ereignisse, die den Deskriptor einer α-Card modifizieren, enthalten nun anstatt der Informationen über geänderte Adornments den kompletten Deskriptor. Falls die α-Card beim Empfänger noch nicht existiert, wird diese angelegt, ansonsten wird der lokale Deskriptor als Ganzes mit dem ankommenden überschrieben.

Ereignisse, die den Payload einer α-Card modifizieren, enthalten nun zusätzlich den Deskriptor der betreffenden Card. Auf diese Weise ist sichergestellt, dass Payload-Aktualisierungen auch dann vom Empfänger durchgeführt werden können, wenn alle Nachrichten, die den Deskriptor der zugehörigen α-Card betreffen, verzögert übertragen werden.

Die aufgrund dieser Änderungen auftretende Erhöhung der Nachrichtengröße ist zu vernachlässigen, da die XML-basierten Deskriptoren gegenüber zu erwartenden großen binären Payloads, wie z.B. Bilddaten, kaum ins Gewicht fallen. Die Geschwindigkeit des Nachrichtentransports wird somit kaum tangiert.

10.3.4 Integration von Universally Unique Identifiers (UUIDs)

Für interne Administrationszwecke und die Korrektheit der Verarbeitung von Prozessartefakten ist es erforderlich, dass jede α-Episode, der zugehörige Patient und jede α-Card eindeutig identifizierbar sind. Vor Beginn der vorliegenden Arbeit war es dem Benutzer überlassen, einen eindeutigen Identifikator für die aktuelle Behandlungsepisode und den Patienten zu wählen. Die Identifikatoren der α-Cards wurden durch einen einfachen, auf dem Paket `java.util.Random` aufbauenden Zufallszahlengenerator generiert.

Um sicherzustellen, dass die Identifikatoren der genannten Teilinformationen garantiert eindeutig sind, wird anstatt des obigen Vorgehens auf das Konzept der Universally Unique Identifier (UUID) [LMS05] zurückgegriffen. Diese ermöglichen das Erzeugen eines weltweit eindeutigen Bezeichners in Form einer hexadezimal notierten 16-Byte-Zahl.

Die benötigen UUIDs werden unter Verwendung des in der Java Platform enthaltenen Pakets `java.util.UUID` generiert. Diese Implementierung greift auf einen kryptographisch starken Zufallszahlengenerator zurück, um die Wahrscheinlichkeit des Auftretens von Duplikaten unter den erzeugten UUIDs zu minimieren und daraus resultierende Konflikte zu vermeiden. Die erzeugten UUIDs werden in Form von Zeichenketten an α-Flow zur weiteren Benutzung übergeben.

10.4 Zusammenfassung

Die angefertigte Implementierung α-OffSync der generischen Schnittstellen von α-Overnet realisiert den SMTP-/IMAP-basierten Austausch von MIME-Nachrichten zur Propagation von Prozessinformationen einer α-Episode. Durch Verwendung des OpenPGP-Standards in der Sicherheitskomponente werden hohe Anforderungen an Datenschutz und Integrität beim Informationsaustausch erfüllt.

Mit der Umsetzung des entwickelten Konzepts zur Synchronisation verteilter Änderungen und der Erkennung nebenläufiger Änderungskonflikte ist sichergestellt, dass die Akteure jeweils über gültige Informationen über den Prozessverlauf verfügen. Die vorgenommenen Anpassungen verschiedener Module von α-Flow sorgen für eine erhöhte Wartbarkeit des Gesamtsystems und tragen wesentlich zur Integration der von α-Overnet und α-OffSync bei.

11 Ausblick

Während der Anfertigung der vorliegenden Arbeit haben sich verschiedene zusätzliche Aspekte und Problemfelder ergeben, deren konzeptionelle Ausgestaltung und technische Umsetzung nicht Gegenstand dieser Arbeit, für α-Flow jedoch in hohem Maße wünschenswert sind. Weitere Forschungen am α-Flow-Projekt sollten sich mit diesen offenen Punkten in detaillierter Form auseinandersetzen.

11.1 Atomare Nachrichten

Das erarbeitete Synchronisationsprotokoll ist in der Lage, Nachrichten die nicht gemäß ihrer Erstellungsreihenfolge beim Empfänger eintreffen, nachträglich zu ordnen. Auf diese Weise ist nach dem Empfang aller versandten Nachrichten wieder ein konsistenter Zustand garantiert.

Für eine verbesserte Benutzererfahrung sollte eine Minimierung der Dauer angestrebt werden, während der sich das α-Doc in einem fachlich inkonsistenten Zwischenzustand befinden kann. Dafür sollte sichergestellt werden, dass alle Aktualisierungen, die zu einer in sich abgeschlossenen Interaktion des Benutzers gehören, in einer einzigen atomaren Nachricht an die übrigen Akteure propagiert werden. Auf diese Weise ist gewährleistet, dass umfangreichere Aktualisierungen auf der Empfängerseite direkt als Ganzes umgesetzt werden können.

Im Verlauf der prototypischen Implementierung wurde bereits eine deutliche Reduktion der Anzahl versandter Nachrichten durchgeführt. Es verbleiben jedoch noch Benutzungsszenarien im Bereich des Adornment Prototype Artifact und der Definition von Abhängigkeiten zwischen einzelnen α-Cards, bei denen bisher mehr einzelne Nachrichten als minimal erforderlich versendet werden.

Zur Lösung dieser Problematik wird die Einführung eines neuen Nachrichtentyps empfohlen, mit dem verschiedene einzelne Aktualisierungen gebündelt werden können. Dazu sind Anpassungen bei der Erzeugung von Ereignissen in den Modulen α-Editor und α-Properties erforderlich. Zudem muss die regelbasierte Wissensbibliothek um ent-

sprechende Regeln erweitert werden, welche die Verarbeitung des neu hinzugekommenen Nachrichtentyps ermöglichen.

11.2 Automatische Zusammenführung von Prozessartefakten

Wie bereits in Abschnitt 7.2.2.2 beschrieben, existieren verschiedene Strategien zur Auflösung von nebenläufigen Änderungskonflikten von Prozessartefakten. Die im Rahmen dieser Arbeit realisierte Methode des Zurücksetzens auf die letzte nicht konfliktbehaftete Version der jeweiligen α-Card ist prinzipiell für alle Arten von Prozessartefakten anwendbar. Obwohl im medizinischen Einsatz aufgrund der Struktur der abzubildenden Prozesse und der zur erwartenden Zuverlässigkeit der Transportkanäle nur in sehr seltenen Fällen überhaupt nebenläufige Änderungen auftreten, ist es wünschenswert, die Konfliktlösungsstrategie so gut wie möglich an die einzelnen Typen von Prozessartefakten anzupassen.

So ist es beispielsweise denkbar statt dem Zurücksetzen auf eine konfliktfreie Version bei XML-basierten Prozessartefakten eine automatische Zusammenführung anzubieten. Da das Schema der α-Card-Deskriptoren und des Payloads der Coordination-α-Cards bei jeder α-Doc-Instanz identisch ist, kann eine Strategie entwickelt werden, mit deren Hilfe konfligierende Versionen automatisch zu einer neuen gültigen Version zusammengeführt werden. Dazu müssen die einzelnen Elemente der Dateien paarweise miteinander verglichen werden, um ihren neuen Inhalt bestimmen zu können. Wird von allen Akteuren dasselbe Verfahren zur Zusammenführung angewandt und produziert dieses deterministische Ergebnisse, so kann wie bekannt auf Basis lokaler Informationen ein globaler Konflikt behoben werden.

11.3 Sichere Benutzerauthentifizierung

Für den praktischen Einsatz von α-Flow ist eine Implementierung einer sicheren Benutzerauthentifzierung erforderlich. Bislang wird der Benutzer beim ersten Öffnen des α-Doc nach seinen identifizierenden Daten und der seiner Adresse des elektronischen Kommunikationskanals für die Synchronisierung von Aktualisierungen gefragt. Hierbei kann jedoch nicht garantiert werden, dass es sich beim Benutzer tatsächlich um diejenige Person bzw. Institution handelt, die er vorgibt zu sein.

Um diese sicherheitstechnische Unzulänglichkeit zu überwinden, benötigt α-Flow Schnittstellen zur Integration in einen institutionsübergreifenden Verbund von Individuen und Institutionen im Gesundheitswesen. Dieser Verbund muss eine sichere Möglichkeit bereitstellen, die Echtheit des Benutzers verifizieren zu können. Denkbar wäre beispielsweise der Einsatz von Hardware-Tokens, die von einer Zertifizierungsstelle ausschließlich an berechtigte Akteure im Gesundheitswesen ausgegeben werden.

11.4 Persistierung von organisatorischen Informationen außerhalb des α-Doc

In der aktuellen Version des α-Flow-Gesamtsystems werden alle Informationen zur Benutzerauthentifizierung und zur Konfiguration des Workflows innerhalb des α-Doc abgelegt. Werden mehrere α-Doc-Instanzen verschiedener α-Episoden von einem Akteur auf demselben physikalischen Rechner genutzt, so müssen diese Informationen mehrfach eingegeben und persistiert werden.

Zur Verbesserung der Benutzererfahrung ist es daher denkbar, diese Daten außerhalb des α-Doc auf dem Rechner abzulegen. Welche technische Lösung hierfür zum Einsatz kommt, hängt maßgeblich von den beabsichtigten Zielplattformen von α-Flow ab.

Für den Einsatz unter Microsoft Windows bietet sich das Anlegen von individuellen Einträgen in der zentralen Registrierungsdatenbank des Systems an. Da diese jedoch unter Unix-basierten Betriebssystemen wie Linux oder OSX nicht vorhanden ist, könnte auf diesem Systemen ein spezieller Ordner im Heimat-Verzeichnis des Benutzers angelegt werden, in dem die Informationen zentral gespeichert werden.

11.5 Einbindung zusätzlicher Kommunikationsprotokolle

Die Architektur des Moduls α-Overnet wurde mit dem Blick auf die leichte Integration weiterer Kommunikationsprotokolle konzipiert. Es sind verschiedene Protokolle verfügbar, deren zukünftige Integration in die entworfene generische Architektur von α-Overnet zu evaluieren ist.

11.5.1 Alternative E-Mail-Transferprotokolle

Neben den für die prototypische Implementierung des fachlichen Lösungsansatzes verwendeten standardisierten Versionen des Internet Message Access Protocol (IMAP) und des Simple Message Transfer Protocol (SMTP) existieren weitere moderne Protokolle für den Umgang mit E-Mail-Nachrichten.

11.5.1.1 Push Extensions to the IMAP Protocol (P-IMAP)

Die von einem Konsortium um Oracle, Samsung, Sony-Ericsson und China Mobile entwickelten Push Extensions to the IMAP Protocol (P-IMAP) wurden im Jahr 2006 vorgestellt [MKC+06]. Sie enthalten zahlreiche Modifikationen und Verbesserungen des ursprünglichen Konzeptes von IMAP, um die Nutzung von E-Mails über mobile Endgeräte zu erleichtern.

So ist eine Möglichkeit zur Umsetzung ereignisbasierter Echtzeit-Benachrichtigungen, ähnlich dem IDLE-Command (vgl. Abschnitt 4.3.3), direkt in das Protokoll integriert. Es wird eine serverseitige Vorabfilterung von Nachrichten angeboten, sodass der Benutzer nur über für ihn bedeutsame Aktualisierungen informiert wird.

Zur Reduktion des zu übertragenden Transfervolumens ist es möglich, die zu übertragenden Daten und Steuerbefehle auf Seite des Servers zu komprimieren. Aufeinanderfolgende Kommandos des Protokolls werden in Makros zusammengefasst, um die Anzahl der zwischen Client und Server ausgetauschten Datenpakete zu minimieren.

Die permanent aufrechterhaltene Datenverbindung zwischen Client und Server kann jedoch nicht nur zur Nachrichtenübertragung vom Server zum Client genutzt werden. Unter Verwendung eines eigens entwickelten einfachen Protokolls ist es auch möglich, mittels P-IMAP auch E-Mails vom Client an den Server zu senden, um sie von dort aus an die vorgesehenen Empfänger zu verteilen.

P-IMAP ist derzeit nicht Teil eines anerkannten Industriestandards. Trotz interessanter Ansätze zur ereignisbasierten Echtzeit-Benachrichtigung und der effizienten Bandbreitennutzung verhindert das Fehlen einer Referenzimplementierung den zuverlässigen Einsatz für die Synchronisation in α-Flow. Sollten die Bemühungen um Standardisierung aber in Zukunft intensiviert werden, so stellt des Protokoll eine zu sinnvolle Alternative zu den bislang verwendeten Protokollen dar.

11.5.1.2 Simple Mail Access Protocol (SMAP)

Einen interessanten Ansatz für den Transport von E-Mail-Nachrichten realisiert das Simple Mail Access Protocol (SMAP) [Var]. Das Protokoll ist Teil der Implementierung des unter einer Open-Source-Lizenz frei verfügbaren „Courier Mail Server“[1].

Der Funktionsumfang von SMAP orientiert sich an dem von IMAP, die Syntax und Semantik der Steuerbefehle ähneln denen des Post Office Protocol und SMTP. Jeder SMAP-fähige Client bzw. Server ist durch einen Legacy-Modus vollständig kompatibel zu IMAP. In diesem IMAP-Modus können alle standardisierten Funktionen von IMAP verwendet werden, lediglich die SMAP-eigenen Funktionalitäten stehen nicht zur Verfügung.

Als Besonderheit bietet SMAP eine serverseitige Dekodierung von MIME-Nachrichtenteilen. Durch diese Dekodierung kann bei großen base64-kodierten Dateianhängen die durch diese Kodierung entstehende Erhöhung der Dateigröße vermieden werden. Vom auf diese Weise reduzierten Transfervolumen profitieren vor allem Nutzer mobiler Endgeräte mit langsamer Datenverbindung. Dazu tragen auch diverse vorgenommene Optimierungen bei, welche die Menge der ausgetauschten nachrichtenbezogenen Metainformationen zwischen Client und Server minimieren.

Weiterhin können mittels SMAP auch E-Mails versandt werden. Ähnlich zu IMAP legt der Clients Nachrichten in Verzeichnissen auf dem Server ab. Wird eine neue E-Mail abgelegt, so wird diese bei SMAP automatisch an die angegebenen Empfänger verteilt. Auf diese Weise muss jede E-Mail zum Versand genau ein einziges Mal an den Server übermittelt werden anstatt wie bei SMTP und IMAP jeweils sowohl an den SMTP- als auch an den IMAP-Server.

Die in SMAP vereinigten Konzepte stellen eine interessante Kombination der positiven Eigenschaften verschiedener auf dem Markt verfügbarer E-Mail-Protokolle dar. Da der „Courier Mail Server“ die derzeit einzige verfügbare Serverimplementierung des Protokolls darstellt, ist SMAP zum jetzigen Zeitpunkt noch nicht für den Einsatz in heterogenen Umgebungen, wie sie von α-Flow anvisiert werden, geeignet.

11.5.2 Weitere Kommunikationsprotokolle

Neben der im Rahmen der vorliegenden Arbeit umgesetzten SMTP/IMAP-basierten Implementierung bieten sich für α-Flow auch Protokolle an, die nicht auf dem Versand

1 Für weitere Informationen siehe `http://www.courier-mta.org/`.

klassischer E-Mail-Nachrichten basieren. So besitzt beispielsweise das Extensible Messaging and Presence Protocol (XMPP) [SA11a], [SA11b] verschiedene Vorteile gegenüber dem Versand von E-Mails. Zum einen ist es zuverlässiger als mit Hilfe der Delivery Status Notifications [Moo03] möglich, den Erhalt einer Nachricht für den Absender zu quittieren. Zum anderen kann aufgrund der Konzeption von XMPP als Instant Messaging-Protokoll mit einer noch schnelleren Benachrichtigung über neu vorliegende Nachrichten gerechnet werden. Derzeit ist die Verbreitungsdichte von XMPP-fähigen Servern noch nicht mit der von Mail-Servern zu vergleichen, für den zukünftigen Einsatz von α-Flow sollte das Protokoll jedoch näher evaluiert werden.

Ein weiteres auf die Eignung im Kontext von α-Flow zu untersuchendes Protokoll ist das Application Configuration Access Protocol (ACAP) [NM97]. Dieses Kommunikationsprotokoll bietet Mechanismen zur serverseitigen Speicherung von Schlüssel-Werte-Paaren. Es könnte somit für die verteilte Speicherung und Propagation von α-Adornments Verwendung finden.

12 Zusammenfassung

Interdisziplinäre und institutionsübergreifende Workflows zur Patientenbehandlung sind heute fester Bestandteil vieler Bereiche des Gesundheitswesens. Bei derartigen Prozessen ist in vielen Fällen die Anzahl der partizipierenden Parteien initial unbekannt und auch der exakte Behandlungsverlauf meist nicht vorhersehbar.

Um die daraus resultierende stetig wachsende Komplexität der Patientenbehandlung beherrschen zu können, sind neue Ansätze zur Koordinierung des Prozessverlaufs und der Zusammenarbeit zwischen den beteiligten Individuen und Institutionen erforderlich. Das Forschungsprojekt α-Flow bietet einen solchen Ansatz, bei dem die passiven medizinischen Dokumente um aktive Eigenschaften ergänzt und in einer elektronischen Fallakte gebündelt verwaltet werden.

Diese Fallakte koordiniert in ihrer Funktion als aktives Dokument den Informationsaustausch über neu gewonnene medizinische Erkenntnisse, die den Zustand des Patienten der jeweiligen Behandlungsepisode betreffen. Dazu werden die Informationen nicht mehr, wie bisher weit verbreitet, als papierbasierte Dokumente von Akteur zu Akteur transportiert, sondern auf elektronischem Wege propagiert. Dies erlaubt die unverzügliche Weitergabe neuer Erkenntnisse, um sicherzustellen, dass allen Beteiligten für ihren fachlichen Entscheidungsprozess über den weiteren Ablauf der Behandlung stets aktuelle Informationen zur Verfügung stehen.

Im Rahmen der vorliegenden Arbeit wurde auf konzeptioneller Ebene ein Lösungsansatz entwickelt, mit dem die Funktionalität des koordinierten Informationsaustausches in dokumenten-orientierten Prozessen umgesetzt werden kann. Zur Erarbeitung dieses Lösungsansatzes wurden, ausgehend von der Betrachtung eines möglichen Einsatzszenarios bei der Klassifikation von Brustkrebserkrankungen, die zu entwerfenden Komponenten des Fachkonzepts identifiziert.

Gefordert wurde ein Synchronisationsansatz zur Herstellung einer Ordnung auf den Versionen der ausgetauschten Prozessartefakte, der eine Erkennung nebenläufiger globaler Änderungskonflikte aufgrund lokal verfügbarer Informationen ermöglicht. Mit Hilfe logischer Zeitstempel wurde hierzu ein Verfahren entwickelt, mit dem die lokale Versionshistorie eines Akteurs nachträglich vervollständigt werden kann und die Erken-

nung globaler Konflikte durch vergleichende Analyse der lokal bekannten Zeitstempel umgesetzt wird.

Für den flexiblen Einsatz einer elektronischen Fallakte war es notwendig, die Divergenz zwischen Akteur, Adressinformationen und physischem Arbeitsplatz aufzulösen. Zu diesem Zweck wurde das erarbeitete Synchronisationskonzept erweitert und die Regeln zur Kommunikation zwischen den Akteuren angepasst. Dies erlaubt es den einzelnen Akteuren, die Fallakte eigenständig zu replizieren, um sie von mehreren Rechnern aus zu verwenden.

Eine weitere Komponente des fachlichen Lösungsansatzes ist das Protokoll zum Beitritt neuer Akteure zu einer Behandlungsepisode. Durch den Austausch von Informationen über den jeweils bekannten Status des Prozesses können sich neu beigetretene Akteure bei den übrigen Akteuren vorstellen und von diesen die aktuellen Informationen über den Stand der Behandlung erhalten. Dadurch ist gewährleistet, dass der Teilnehmerkreis dynamisch erweitert werden kann.

Um das entwickelte Fachkonzept in die existierende Systemarchitektur von α-Flow zu integrieren, wurden generische Java-Schnittstellen geschaffen, die unabhängig von einer konkreten Implementierung die Interaktion mit den übrigen Komponenten möglich machen. Die Schnittstellen zur Kommunikation mit anderen Akteuren wurden im Modul α-Overnet zusammengefasst. Für die Integration der Konzepte zum Beitritt neuer Akteure und zum Umgang mit der Versionshistorie wurden die entsprechenden Schnittstellen den Modulen α-VerVarStore und α-Properties hinzugefügt.

Auf Basis der generischen Schnittstellen wurde schließlich im Modul α-OffSync eine E-Mail-basierte Implementierung von α-Overnet angefertigt. Hierbei werden unter Verwendung von SMTP und IMAP Nachrichten zwischen den Akteuren ausgetauscht. Die Nachrichten liegen dabei im MIME-Format vor und werden mit Hilfe von OpenPGP signiert und verschlüsselt. Das entwickelte Synchronisationskonzept wurde durch Zurückgreifen auf das integrierte Versionsverwaltungssystem von α-Flow auf Grundlage der vorgestellten logischen Zeitstempel implementiert. Eine konkrete Umsetzung des Beitrittsprotokolls für neue Akteure rundet den Funktionsumfang der implementierten Softwarebausteine ab.

Die vorliegende Arbeit ermöglicht somit die zielgerichtete Koordinierung des Prozessverlaufs mit α-Flow und erlaubt den synchronisierten Austausch verteilter Änderungen. Damit wurde das Fundament für weitergehende Forschungen geschaffen, durch welche die Konzepte von α-Flow stetig erweitert werden können, um den Teilnehmern komplexer verteilter Prozesse optimale Grundlagen für eine effiziente Zusammenarbeit zu bieten.

Literaturverzeichnis

[AAB04] Almeida, J.B. ; Almeida, P.S. ; Baquero, C.: Bounded version vectors. In: *Distributed Computing* (2004), S. 102–116

[BC89] Beck, K. ; Cunningham, W.: A laboratory for teaching object oriented thinking. In: *SIGPLAN Not.* 24 (1989), September, S. 1–6

[BMH97] Bemmel, J. H. ; Musen, M. A. ; Helder, J. C.: *Handbook of medical informatics*. Bohn Stafleu Van Loghum, 1997

[BMR+96] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. 1. Auflage. Chichester : John Wiley & Sons, 1996

[CDF+07] Callas, J. ; Donnerhacke, L. ; Finney, H. ; Shaw, D. ; Thayer, R.: *OpenPGP Message Format*. RFC 4880 (Proposed Standard). `http://www.ietf.org/rfc/rfc4880.txt`. Version: November 2007 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[CDK02] Coulouris, G. ; Dollimore, J. ; Kindberg, T.: *Verteilte Systeme*. 3., überarbeitete Auflage. München : Pearson Studium, 2002

[Cer69] Cerf, V.G.: *ASCII format for network interchange*. RFC 20. `http://www.ietf.org/rfc/rfc20.txt`. Version: Oktober 1969 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[CHKS92] Ceri, S. ; Houtsma, M.A.W. ; Keller, A.M. ; Samarati, P.: The Case for Independent Updates. In: *Proceedings of the second Workshop on the Management of Replicated Data, 1992.* Monterey CA, 1992, S. 17–19

[CHKS95] Ceri, S. ; Houtsma, M.A.W. ; Keller, A.M. ; Samarati, P.: Independent Updates And Incremental Agreement in Replicated Databases. In: *Distributed and Parallel Databases* 3 (1995), S. 225–246

[CKD+01] Corrigan, J.M. ; Kohn, L.T. ; Donaldson, M.S. ; Maguire, S.K. ; Pike, K.C.: *Crossing the quality chasm: a new health system for the 21st century*. Washington, DC: National Academy Press, 2001

[Cri89] Cristian, Flaviu: Probabilistic clock synchronization. In: *Distributed Computing* 3 (1989), S. 146–158

[Cri03] Crispin, M.: *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501 (Proposed Standard). `http://www.ietf.org/rfc/rfc3501.txt`. Version: März 2003 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Cro82] Crocker, D.: *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. RFC 822 (Standard). `http://www.ietf.org/rfc/rfc822.txt`. Version: August 1982 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[DB03] Drummond, Lúcia M. ; Barbosa, Valmir C.: On reducing the complexity of matrix clocks. In: *Parallel Computing* 29 (2003), Nr. 7, S. 895 – 905

[FB96a] Freed, N. ; Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*. RFC 2049 (Draft Standard). `http://www.ietf.org/rfc/rfc2049.txt`. Version: November 1996 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[FB96b] Freed, N. ; Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045 (Draft Standard). `http://www.ietf.org/rfc/rfc2045.txt`. Version: November 1996 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[FB96c] Freed, N. ; Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046 (Draft Standard). `http://www.ietf.org/rfc/rfc2046.txt`. Version: November 1996 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Fid88] Fidge, C.J.: Timestamps in message-passing systems that preserve the partial ordering. In: *Proceedings of the 11th Australian Computer Science Conference* Bd. 10, 1988, S. 56–66

[FKP96] Freed, N. ; Klensin, J. ; Postel, J.: *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. RFC 2048 (Best Current Practice). `http://www.ietf.org/rfc/rfc2048.txt`. Version: November 1996 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[GHJV94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston : Addison-Wesley, 1994

[GMCF95] Galvin, J. ; Murphy, S. ; Crocker, S. ; Freed, N.: *Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted*. RFC 1847 (Proposed Standard). `http://www.ietf.org/rfc/rfc1847.txt`. Version: Oktober 1995 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Gol92] Golding, R.A.: *Weak-consistency group communication and membership*. Santa Cruz, University of California, Diss., 1992

[GZ89] Gusella, R. ; Zatti, S.: The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD. In: *Software Engineering, IEEE Transactions on* 15 (1989), Nr. 7, S. 847–853

[Had11] Hady, Scott A.: *Logical Unit Oriented Version Control System with Version Validity Support*, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 2011

[Ham05] Hammerschall, Ulrike: *Verteilte Systeme und Anwendungen*. München : Pearson Studium, 2005

[Han10] Hanisch, S.: *Konzeption und Implementierung einer Infrastruktur für aktive Dokumente*, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 2010

[KCD+99] Kohn, L.T. ; Corrigan, J.M. ; Donaldson, M.S. u. a.: *To Err Is Human: Building a Safer Health System*. Informa Pharma Science, 1999

[Kle08] Klensin, J.: *Simple Mail Transfer Protocol*. RFC 5321 (Draft Standard). `http://www.ietf.org/rfc/rfc5321.txt`. Version: Oktober 2008 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Lam78] Lamport, Leslie: Time, clocks, and the ordering of events in a distributed system. In: *Communications of the ACM* 21 (1978), Nr. 7, S. 558–565

[LBM+05] Lenz, R. ; Beyer, M. ; Meiler, C. ; Jablonski, S. ; Kuhn, K. A.: Informationsintegration in Gesundheitsversorgungsnetzen. In: *Informatik-Spektrum* 28 (2005), Nr. 2, S. 105–119

[Lei97] Leiba, B.: *IMAP4 IDLE command*. RFC 2177 (Proposed Standard). `http://www.ietf.org/rfc/rfc2177.txt`. Version: Juni 1997 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Len97] Lenz, Richard: *Adaptive Datenreplikation in verteilten Systemen*. Stuttgart [u.a.] : Teubner, 1997

[LMS05] Leach, P. ; Mealling, M. ; Salz, R.: *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122 (Proposed Standard). `http://www.ietf.org/rfc/rfc4122.txt`. Version: Juli 2005 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[MKC+06] Maes, S.H. ; Kuang, C. ; Cromwell, R. ; Chiu, E. ; Day, J. ; Ahad, R. ; Jeong, Wook-Hyun ; Rosell, Gustaf ; Sini, J. ; Son, Sung-Mu ; Xiahui, Fan ; Lijun, Zhao ; Bennett, D.: *Push Extensions to the IMAP Protocol (P-IMAP)*. `http://tools.ietf.org/html/draft-maes-lemonade-p-imap-12`. Version: 2006. – zuletzt abgerufen am 01.09.2011

[MM01] Mullet, Dianna ; Mullet, Kevin: *Mailmanagement mit IMAP*. Sebastopol CA : O'Reilly Media Inc., 2001

[MMBK10] Mills, D. ; Martin, J. ; Burbank, J. ; Kasch, W.: *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905 (Proposed Standard). `http://www.ietf.org/rfc/rfc5905.txt`. Version: Juni 2010 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Moo96] Moore, K.: *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. RFC 2047 (Draft Standard). `http://www.ietf.org/rfc/rfc2047.txt`. Version: November 1996 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Moo03] Moore, K.: *Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs).* RFC 3461 (Draft Standard). `http://www.ietf.org/rfc/rfc3461.txt`. Version: Januar 2003 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[MR96] Myers, J. ; Rose, M.: *Post Office Protocol - Version 3.* RFC 1939 (Standard). `http://www.ietf.org/rfc/rfc1939.txt`. Version: Mai 1996 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[NL09] Neumann, Christoph P. ; Lenz, Richard: alpha-Flow: A Document-based Approach to Inter-Institutional Process Support in Healthcare. In: *Proc of the 3rd Int'l Workshop on Process-oriented Information Systems in Healthcare (ProHealth '09) in conjunction with the 7th Int'l Conf on Business Process Management (BPM'09).* Ulm, Germany, September 2009

[NL10] Neumann, Christoph P. ; Lenz, Richard: The alpha-Flow Use-Case of Breast Cancer Treatment – Modeling Inter-Institutional Healthcare Workflows by Active Documents. In: *Proc of the 8th Int'l Workshop on Agent-based Computing for Enterprise Collaboration (ACEC) at the 19th Int'l Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2010).* Larissa, Greece, Juni 2010

[NL12] Neumann, Christoph P. ; Lenz, Richard: The alpha-Flow Approach to Inter-Institutional Process Support in Healthcare. In: *International Journal of Knowledge-Based Organizations (IJKBO)* 2 (2012), Nr. 3. – Accepted for publication

[NM97] Newman, C. ; Myers, J. G.: *ACAP – Application Configuration Access Protocol.* RFC 2244 (Proposed Standard). `http://www.ietf.org/rfc/rfc2244.txt`. Version: November 1997 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[NSWL11] Neumann, Christoph P. ; Schwab, Peter K. ; Wahl, Andreas M. ; Lenz, Richard: alpha-Adaptive: Evolutionary Workflow Metadata in Distributed Document-Oriented Process Management. In: *Proc of the 4th Int'l Workshop on Process-oriented Information Systems in Healthcare (ProHealth'11) in conjunction with the 9th Int'l Conf on Business Process Management (BPM'11).* Clermont-Ferrand, France, 2011

[Ora11a] Oracle Corporation: *How to Implement a Provider in the Java Cryptography Architecture*. `http://download.oracle.com/javase/6/docs/technotes/guides/security/crypto/HowToImplAProvider.html`. Version: 2011. – zuletzt aufgerufen am 01.09.2011

[Ora11b] Oracle Corporation (Hrsg.): *Java Cryptography Architecture (JCA) Reference Guide for Java Platform Standard Edition 6*. Redwood Shores: Oracle Corporation, 2011

[Peh10] Pehlianov, Hristiyan: *Konzeption und Implementierung einer Teilnehmerverfolgung für verteilte aktive Dokumente auf Basis Peer-to-Peer-basierter File Sharing Protokolle*, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, Bachelorarbeit, 2010

[PJPR+83] Parker Jr, D.S. ; Popek, G.J. ; Rudisin, G. ; Stoughton, A. ; Walker, B.J. ; Walton, E. ; Chow, J.M. ; Edwards, D. ; Kiser, S. ; Kline, C.: Detection of mutual inconsistency in distributed systems. In: *Software Engineering, IEEE Transactions on* (1983), Nr. 3, S. 240–247

[Pos80] Postel, J.: *User Datagram Protocol*. RFC 768 (Standard). `http://www.ietf.org/rfc/rfc768.txt`. Version: August 1980 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Ram09] Rampp, Florian D.: *Konzeption und Realisierung einer verteilten Adressdatenbank im Stil einer Publish/Subscribe-Architektur zum Austausch von Patientendaten zwischen autonomen medizinischen Informationssystemen*, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 2009

[Res01] Resnick, P.: *Internet Message Format*. RFC 2822 (Proposed Standard). `http://www.ietf.org/rfc/rfc2822.txt`. Version: April 2001 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Res08] Resnick, P.: *Internet Message Format*. RFC 5322 (Draft Standard). `http://www.ietf.org/rfc/rfc5322.txt`. Version: Oktober 2008 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[SA11a] Saint-Andre, P.: *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120 (Proposed Standard). `http://www.ietf.org/rfc/rfc6120.`

`txt`. Version: März 2011 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[SA11b] Saint-Andre, P.: *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. RFC 6121 (Proposed Standard). `http://www.ietf.org/rfc/rfc6121.txt`. Version: März 2011 (Request for Comments). – zuletzt abgerufen am 22.10.2011

[Sch11] Schwab, Peter K.: *Ein adaptives Attributmodell als Baustein einer Prozessunterstützung auf Basis von aktiven Dokumenten*, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 2011

[SK92] Singhal, M. ; Kshemkalyani, A.: An efficient implementation of vector clocks. In: *Information Processing Letters* 43 (1992), Nr. 1, S. 47–52

[SM94] Schwarz, R. ; Mattern, F.: Detecting causal relationships in distributed computations: In search of the holy grail. In: *Distributed computing* 7 (1994), Nr. 3, S. 149–174

[SPS11] Spitz, Stephan ; Pramateftakis, Michael ; Swoboda, Joachim: *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen*. 2., überarbeitete Auflage. Wiesbaden : Vieweg+Teubner, 2011

[Sun06] Sun Microsystems Inc. (Hrsg.): *JavaMail API Design Specification Version 1.4*. Santa Clara, CA: Sun Microsystems Inc., 2006

[TN11] Todorova, Aneliya ; Neumann, Christoph P.: alpha-Props: A Rule-Based Approach to 'Active Properties' for Document-Oriented Process Support in Inter-Institutional Environments. In: Porada, Ludger (Hrsg.) ; Gesellschaft für Informatik (Veranst.): *Lecture Notes in Informatics (LNI) Seminars 10 / Informatiktage 2011* Gesellschaft für Informatik, 2011. – ISBN 978–3–88579–444–8

[Tod10] Todorova, A.: *Konzeption und Implementierung eines leichtgewichtigen und autonomen Regel-basierten Systems als eine Realisierung von „Active Properties“ im Kontext von aktiven Dokumenten*, Lehrstuhl für Informatik 6 (Datenmanagement), Friedrich-Alexander-Universität Erlangen-Nürnberg, Diplomarbeit, 2010

[TRA99] Torres-Rojas, Francisco J. ; Ahamad, Mustaque: Plausible clocks: constant size logical clocks for distributed systems. In: *Distributed Computing* 12 (1999), S. 179–195

[TS03] Tanenbaum, Andrew S. ; Steen, Maarten van: *Verteilte Systeme*. 1.Auflage. München : Pearson Studium, 2003

[Var] Varshavchik, Sam: *Simple Mail Access Protocol, Version 1*. `http://www.courier-mta.org/cone/smap1.html`. – zuletzt abgerufen am 01.09.2011