Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3336

Wiederherstellung von Ereignisströmen in CEP-Systemen

Ruben Mayer

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dr. Boris Koldehofe
begonnen am:	4. April 2012
beendet am:	24. September 2012
CR-Klassifikation:	C.2.4, C.4

Abstract

In den letzten Jahren wurde in der Informationstechnologie das Paradigma des „Complex Event Processing“ entwickelt, mit dessen Hilfe aus Datenströmen durch Korrelationen von einfachen Ereignissen höherwertige Informationen abgeleitet werden können. Aufgrund der Anforderungen an die Skalierbarkeit, die durch Anwendungsgebiete wie das „Internet der Dinge“ an moderne CEP-Lösungen gestellt werden, ist eine verteilte Architektur vorteilhaft. Um die Ausfallsicherheit des Systems unter Einhaltung strenger Korrektheitsbedingungen zu garantieren, wird in dieser Arbeit ein Verfahren zur Wiederherstellung von Operatoren entwickelt, das einen niedrigeren Overhead verursacht als klassische Verfahren wie die redundante Auslegung von Operatoren oder die persistente Speicherung von Prozesszuständen in sogenannten Checkpoints. Dieses Verfahren wird dann durch die Möglichkeit, an beliebigen Stellen der Operatortopologie persistente Schichten einzuziehen, um ein Werkzeug zur Partitionierung der Topologie erweitert, womit die Belastung einzelner Operatoren verringert werden kann. Schließlich wird der Einfluss charakteristischer Gestaltungsparameter auf das Verfahren in einer Simulation evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

Durch die fortschreitende Entwicklung der Informations- und Kommunikationstechnologie wurde in den vergangenen Jahren die Möglichkeit einer dichten Vernetzung zwischen Informationssystemen und der realen Welt geschaffen. Durch die Verknüpfung von verschiedenartigen Sensoren [BKR11], Kameras [RHI+12] und mobilen Quellen [KORR12] wird so ein „Internet der Dinge“ [AIM10] geschaffen, in dem bestimmte Sachverhalte aus der realen Welt durch Computersysteme überwacht werden können. Dabei werden hohe Datenmengen an verschiedenen Orten produziert, die über Datenströme an andere Orte transferiert und dort miteinander korreliert werden [KKR10], um höherwertige Sachverhalte aus den einfachen Daten zu extrahieren [Luc01]. Ein System, das solche Korrelationen ermöglicht, wird auch als *„Complex Event Processing“*-System (CEP-System) bezeichnet [BK09, EB09]. Ein CEP-System fungiert als Middleware zwischen verschiedenen dezentralen Informationsquellen und den Konsumenten der aggregierten Informationen. Mögliche Einsatzgebiete liegen beispielsweise in der Logistik, dem Energiemanagement, der Finanzwirtschaft und in der Steuerung von Fertigungsprozessen [BK09].

Durch die unterschiedlichen Anwendungsgebiete werden vielfältige Anforderungen an CEP-Systeme gestellt. Es wird eine hohe Skalierbarkeit vorausgesetzt, die korrelierten Ereignisse müssen korrekt sein und das System muss als Ganzes robust gegen Ausfälle gestaltet werden. Vor allem die Skalierbarkeit und die Möglichkeit, Ereignisströme lokal nahe der Sensoren zu verarbeiten, spricht für ein verteiltes System. Doch wie kann man in einem solchen verteilten CEP-System unter Berücksichtigung der Skalierbarkeit die Ausfallsicherheit und Korrektheit garantieren?

Die bisherige Forschung zur Ausfallsicherheit von verteilten Systemen konzentriert sich vor allem auf zwei Ansätze zur Lösung dieses Problems: Zum einen kann durch redundante Auslegung der Ausfall von Verarbeitungsknoten (Operatoren) ausgeglichen werden [VKR11], zum anderen besteht die Möglichkeit, durch die persistente Speicherung von Zuständen diese nach einem Ausfall wiederherzustellen [EAWJ02]. Solche Verfahren sind zwar sicher und in der Erforschung recht fortgeschritten, doch die Skalierbarkeit für Berechnungen auf hochfrequenten Datenströmen ist nur begrenzt gegeben. Insbesondere entsteht durch die klassischen Lösungen ein hoher Overhead zur Laufzeit, was die redundante Kommunikation und Berechnungen oder die persistente Speicherung von sog. Checkpoints angeht. Das Ziel dieser Diplomarbeit ist es, ein Verfahren zu entwickeln, das die Ausfallsicherheit des Systems garantiert und gleichzeitig ohne redundante Berechnungen und ohne die Speicherung von Funktionszuständen in persistenten Datenspeichern auskommt, um den Overhead zur Laufzeit geringer zu halten als es mit den bisherigen Lösungen möglich ist.

Die wesentlichen Beiträge dieser Arbeit bestehen unter anderem aus einem Verarbeitungsmodell, mit dessen Hilfe die Zustandsinformationen von Operatoren möglichst klein gehalten werden. Zudem wird ein Verfahren entwickelt, wie diese Daten auf andere Operatoren zur Sicherung in ihrem volatilen Speicher verteilt werden können, sodass damit ein ausgefallener Operator in einem bestimmten Zustand wiederhergestellt werden kann. Dazu wird ein Algorithmus entworfen, der Ausfälle erkennt und die ausgefallenen Operatoren wiederherstellt. Das Verfahren wird durch die Möglichkeit erweitert, den Operatorgraphen durch persistente Schichten dynamisch zu partitionieren, um so die Belastung einzelner Operatoren durch gespeicherte Zustandsdaten zu reduzieren. Zum Abschluss wird das Verfahren in einer Simulation evaluiert, in welcher der Einfluss charakteristischer Parameter auf die Belastung der Operatoren untersucht wird.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2: Hier wird der Begriff des Complex Event Processing definiert und es werden verschiedene Architekturen für CEP-Lösungen vorgestellt.

Kapitel 3: In diesem Kapitel wird ein Systemmodell für verteiltes CEP eingeführt, das der weiteren Arbeit zugrunde liegt.

Kapitel 4: Die Zielstellung der Diplomarbeit wird konkret definiert und anhand eines Beispiels veranschaulicht. Insbesondere werden die Anforderungen an das CEP-System aufgestellt, die auch in definierten Fehlersituationen nicht verletzt werden dürfen.

Kapitel 5: Der bisherige Forschungsstand wird beschrieben und der eigene Lösungsansatz von diesem abgegrenzt.

Kapitel 6: Hier wird das in dieser Arbeit entwickelte Verfahren zur Wiederherstellung von Operatoren beschrieben. Dafür wird zunächst festgestellt, welche Zustandsinformationen ein Operator im Allgemeinen besitzt. Dann wird ein Ausführungsmodell eingeführt, das diese Informationen zu bestimmten Zeitpunkten minimieren kann. Schließlich wird beschrieben, wie die Zustandsinformationen verteilt werden und wie die Wiederherstellung ausgefallener Operator abläuft.

Kapitel 7: Problematiken bezüglich bestimmter Operatortopologien werden besprochen. Ein Werkzeug zur Partitionierung der Topologien und somit zur Verringerung der Lasten auf einzelnen Operatoren wird entwickelt.

Kapitel 8: In einer Simulation wird evaluiert, welchen Einfluss charakteristische Parameter auf die Belastung des Systems mit Zustandsinformationen haben.

Kapitel 9: Abschließend werden die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick auf mögliche Forschungsrichtungen gegeben, in welche die Arbeit fortgeführt werden kann.

2 Complex Event Processing - Eine Einführung

2.1 Was ist Complex Event Processing?

Complex Event Processing (CEP), oft auch Event Processing genannt, hat sich in den letzten Jahren als eigenes Gebiet in der Forschung und der Wirtschaft etabliert. Doch was steckt genau dahinter und wie grenzt es sich von anderen, ähnlichen Gebieten in der Informatik ab? Ausgehend von der Motivation aus dem ersten Kapitel, wollen wir CEP als einen Problemlösungsansatz betrachten: Die komplexen Aktivitäten in heutigen wirtschaftlichen Computersystemen setzen sich aus vielen einfacheren Aktivitäten zusammen, die wiederum direkt, z.B. über Sensoren oder Nachrichten innerhalb des Systems, detektiert werden können. Um aus den beobachteten Aktivitäten niedriger Komplexität höherwertige Einsichten zu gewinnen, muss in geringer Zeit eine hohe Datenmenge analysiert und verarbeitet werden. Dabei werden einfache Ereignisse zusammengefasst und nach Trends, Korrelationen und Mustern untersucht, deren Auftreten auf andere, komplexere Aktivitäten im System schließen lässt. Dies alles leistet ein CEP-System, welches an Ereignisquellen angeschlossen ist und kontinuierlich in Echtzeit die eingehenden Ereignisse weiterverarbeitet, um schließlich einem Endkunden (nur) die für ihn interessanten Aktivitäten durch Ausgabe entsprechender komplexer Ereignisse anzuzeigen.

Die Definition der Gesellschaft für Informatik fasst diesen Sachverhalt knapp und präzise zusammen:

> „CEP ist ein Sammelbegriff für Methoden, Techniken und Werkzeuge, um Ereignisse zu verarbeiten während sie passieren, also kontinuierlich und zeitnah. CEP leitet aus Ereignissen höheres, wertvolles Wissen in Form von sog. komplexen Ereignissen, d.h. Situationen die sich nur als Kombination mehrerer Ereignisse erkennen lassen, ab.“ (Informatik-Lexikon der Gesellschaft für Informatik [EB09])

Der Begriff „Complex Event Processing“ für eine solche Technologie wurde erstmals im Jahre 2002 von David Luckham in seinem Buch „The Power of Events“ [Luc01] eingeführt und von ihm maßgeblich geprägt. Wie die oben genannte Definition schließen lässt, manifestiert sich CEP allerdings in einer breit gefächerten Komposition unterschiedlichster Methoden, Techniken und Werkzeuge, die ihre Wurzeln in ebenso unterschiedlichen Forschungsgebieten haben [EB09].

2.1.1 Abgrenzung zu verwandten Gebieten

2.1.1.1 Stream Processing

Stream Processing ist eine Technologie, mit der CEP sehr nahe verwandt ist. Sie zeichnet sich dadurch aus, dass durch Queries auf Datenströme bestimmte Teilströme abgefragt werden können, vergleichbar mit Queries in einem Datenbanksystem. Eine solche Abfragesprache wurde beispielsweise mit CQL [ABW06] definiert. Stream Processing konzentriert sich dabei weniger auf die Verarbeitung von Datenströmen bestimmter Datentypen zu höherwertigen Datenströmen als auf die Möglichkeit, durch Queries bestimmte relevante Daten abzufragen. Die Technologien, die hinter Stream Processing stehen, sind allerdings für die Umsetzung von CEP-Lösungen relevant. Eine scharfe Abgrenzung der beiden Gebiete ist daher kaum möglich.

2.1.1.2 Technologien im Umfeld von CEP

Neben der Definition von CEP soll in diesem Abschnitt kurz erwähnt werden, welche Technologien in dieser Arbeit nicht in die Kategorie des Complex Event Processing fallen, obwohl sie in CEP-Systemen insgesamt durchaus eine Rolle spielen. Ein CEP-System als Ganzes leistet natürlich neben der bloßen Erkennung komplexer Situationen noch weitere Dienste: Insbesondere sollen Ereignisse in der Praxis nicht nur erkannt werden, sondern es soll direkt eine automatische Reaktion darauf erfolgen, die von den Ereigniskonsumenten ausgeht. Zu möglichen Reaktionen zählt das Absenden von Nachrichten in einer Messaging-Middleware, die Interaktion mit Geschäftsprozessen, der Aufruf von Programmen, und so weiter. Technologien, die der Reaktion auf den bloßen Erkenntnisgewinn durch CEP dienen, beispielsweise Visualisierungsverfahren für Ereignisse, reaktive Logikprogrammierung und Business Process Management, sind in dieser Arbeit nicht Teil von CEP. Ebensowenig fallen die Technologien auf der anderen Seite von CEP in dieses Feld: Sensortechnologien (bspw. RFID) sind zwar als Ereignisquellen elementar wichtig in CEP-Systemen, werden hier aber nicht als Teil der CEP-Technologie selbst angesehen. Vielmehr wird CEP als sog. Middleware betrachtet, die zwischen Informationsquellen und den an den Informationen interessierten Stellen Ereignisse verarbeitet und vermittelt.

2.2 Bestandteile von CEP-Verfahren

Nachdem im vorigen Abschnitt definiert wurde, was CEP leistet, stellt sich die Frage, wie genau die eigentliche Korrelation der Ereignisse funktioniert. Im Prinzip sind hier zwei verschiedene Ansätze möglich [EB09]: (1) Abfrage von a-priori bekannten Ereignismustern oder (2) Entdeckung bisher unbekannter Ereignismuster. Im ersten Fall werden die konkreten Ereignismuster, die eine komplexe Situation von Interesse anzeigen, direkt im CEP-System fest implementiert. Je nachdem wie das System aufgebaut ist, werden diese Ereignismuster

beim Deployment fest angelegt oder lassen sich von außen dynamisch durch spezielle Ereignisanfragesprachen konfigurieren, das CEP-System selbst manipuliert die Ereignismuster allerdings nicht. Im zweiten Fall werden Ereignismuster zur Laufzeit vom CEP-System selbst entwickelt und angewandt, beispielsweise durch Einsatz von Technologien wie maschinellem Lernen und Datamining auf den Ereignisströmen. Für den ersten Fall, die Abfrage von definierten Ereignismustern, gibt es bereits eine Reihe von Definitionssprachen und prototypische Implementierungen.

2.2.1 Definitionssprachen, Ausführungsmodelle und Prototypen

Mit *snoop* [CM94] wurde schon früh eine ausdrucksstarke Definitionssprache für Ereignismuster entwickelt. Weiteren Sprachen folgten [WKL+08, CM10], die sich meist auf die Unterstützung bestimmter Muster konzentrieren. Neben der reinen Definitionssprache benötigt ein CEP-System auch ein Verarbeitungs- bzw. Ausführungsmodell, das beschreibt, wie die in der Definitionssprache beschriebenen Korrelationen auf den Ereignisströmen umgesetzt werden. Diese Ausführungsmodelle entscheiden durch ihre spezielle Auslegung auf das Anwendungsgebiet über die Effizienz der Korrelationsberechnung. So gibt es Ausführungsmodelle, die speziell auf mobile CEP-Szenarien zugeschnitten sind und die Rekonfigurationen von Operatoren unterstützen [KORR12] oder solche, die für die Verarbeitung von Ereignisströmen aus RFID-Auslesungen optimiert sind [WDR06].

Es gibt zahlreiche Prototypen, die in der Forschung entwickelt wurden, beispielsweise Cordies [KKR10], SASE [WDR06], DHEP [SKPR10] uvm. Diese beinhalten jeweils eine spezifische Definitionssprache und ein Ausführungsmodell.

2.3 Aufbau von CEP-Systemen

2.3.1 Einordnung von CEP in die IT-Infrastruktur

Ein CEP-System kann in der IT-Architektur als Middleware zwischen Informationsquellen und -interessenten betrachtet werden. Die Ereignisquellen und -konsumenten werden durch Einbezug eines CEP-Systems entkoppelt [BK09], die Weiterverarbeitung, Filterung, usw. von Ereignissen wird dabei von der CEP-Middleware übernommen.

2.3.2 Zentrales CEP

Die einfachste und wohl auch intuitivste Architektur einer Systemkomponente wie einem CEP-System ist, es als eigenständiges, zentrales Programm auf einem Server laufen zu lassen. Alle Ereignisströme werden mit diesem zentralen Programm verbunden, in dem sämtliche Korrelationsregeln gespeichert sind und auf die eingehenden Ereignisströme angewandt werden. Dabei werden aus den eingehenden Ereignissen nach definierbaren Regeln direkt

komplexe Ereignisse erzeugt, die die Konsumenten weitergeleitet werden (vgl. Abb. 2.1). Ein Beispiel für eine kommerziell erhältliche zentrale CEP-Lösung ist „IBM Websphere Business Events“.

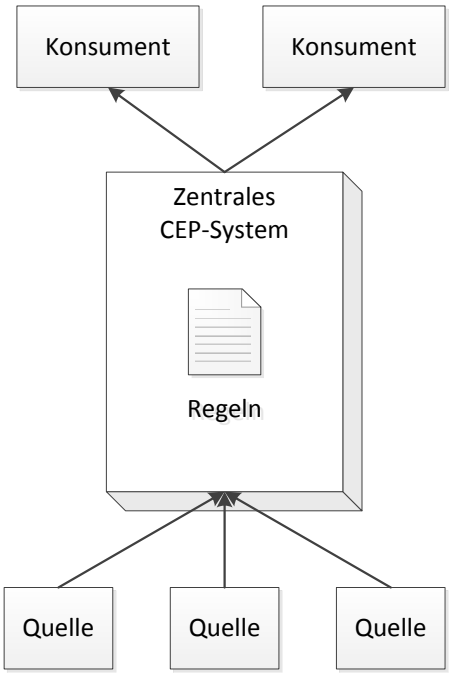

Abbildung 2.1: Ein zentrales CEP-System.

2.3.2.1 Vor- und Nachteile

Zu den Vorteilen eines zentralen CEP-Systems zählt, dass alle Ereignisströme nur mit einem zentralen System verbunden sind, sodass die Gesamtkomplexität des Systems in Grenzen gehalten wird. Damit ist ein solches System auch relativ einfach zu administrieren. Im Vergleich zu einem verteilten System ist ein zentrales einfacher zu programmieren und zu testen und daher kostengünstiger in der Entwicklung.

Nachteilig ist die begrenzte Skalierbarkeit eines zentralen Systems. Da die Ereignisse nicht immer nahe ihrer Quellen verarbeitet werden können, sondern alle zum zentralen System übertragen werden müssen, ergeben sich zweierlei Problematiken: Zum einen wird durch die Übertragung vieler Ereignisse das Netzwerk belastet und es kommt zu Verzögerungen in

der Verarbeitung, zum anderen sind in das CEP involvierte Parteien nicht autonom, sondern vom zentralen CEP-System abhängig. Insbesondere müssen sie Geschäftsregeln aufdecken, um sie im zentralen CEP-System integrieren zu können.

2.3.3 Verteiltes CEP

Ein verteiltes CEP-System ermöglicht die stufenweise Verarbeitung von Ereignisströmen. Anstatt dass aus einfachen Ereignissen direkt komplexe Ereignisse für die Ereigniskonsumenten gewonnen werden, wie es in einem zentralen CEP-System der Fall ist, werden Ereignisströme in Zwischenschritten miteinander korreliert, sodass nach und nach die gewünschten Ausgangsereignisse gebildet werden. Die Platzierung der Verarbeitungsknoten, die in dieser Arbeit *Operatoren* genannt werden, kann für den jeweiligen Anwendungsfall optimal angepasst werden [RDR10, SKR11]. Die Operatoren sind also untereinander vernetzt und bilden somit eine Topologie. Eine Beispieltopologie eines verteilten CEP-Systems befindet sich in Kapitel 3.1.

2.3.3.1 Vor- und Nachteile

Ein verteiltes CEP-System hat den Vorteil, dass durch die Verteilung der Ereignisverarbeitung auf verschiedene Operatoren die Skalierbarkeit des Systems verbessert wird. Die Verarbeitung von Ereignissen kann nahe ihrer Quelle geschehen, dadurch werden Kommunikationskosten und Latenzzeiten verringert. Zudem sind in das CEP involvierte Parteien in der Lage, eigene Operatoren zu realisieren und mit dem verteilten CEP-System zu verknüpfen. Dadurch können sie interne Geschäftsregeln geheim halten.

Nachteilig ist die hohe Systemkomplexität, wodurch der Administrationsaufwand und die Entwicklungskosten steigen.

2.3.4 Fazit

Für große Szenarien mit vielen hochfrequenten Ereignisquellen, die über große räumliche Distanzen verteilt sind, ist ein verteiltes CEP-System im Allgemeinen besser geeignet als ein zentrales. Diese Arbeit bezieht sich daher im Weiteren ausschließlich auf verteilte CEP-Systeme.

3 Verteiltes CEP - Ein Systemmodell

In diesem Abschnitt wird für verteilte CEP-Systeme eine Modellierung in einem sogenannten *Operatorenmodell* eingeführt. Es werden einige Annahmen und Abstraktionen eingeführt, die der weiteren Arbeit zugrunde liegen. Zudem werden auch schon einige Eigenschaften des Ausführungsmodells angeführt, mit dem das in dieser Arbeit entwickelte Wiederherstellungsverfahren für Operatoren arbeitet. Zweck des Kapitels ist die frühe Definition einiger Begriffe und Abstraktionen, mit deren Hilfe später die Aufgabenstellung und die Lösung derselben eindeutig beschrieben werden können.

3.1 Der Operatorgraph

Ein verteiltes CEP-System S kann durch einen gerichteten Operatorgraphen G(Ω, E) dargestellt werden, der sich aus einer Menge von Operatoren $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ zusammensetzt, die miteinander in einer unidirektionalen Verbindung stehen, was durch die Menge der Kanten $E \subseteq \Omega \times \Omega$ dargestellt wird. *Operatoren* (bezeichnet durch ω_i) verarbeiten eingehende Ereignisse (bezeichnet durch σ_i) und erzeugen neue Ereignisse. Dabei können sie eingehende Ereignisse nach vorgegebenen Regeln korrelieren. Die Gesamtfunktionalität des CEP-Systems wird von den Operatoren in Teilschritten realisiert, indem jeder Operator eine *Korrelationsfunktion* f über Ereignisse ausführt. *Ereignisse* sind die Daten, mit denen ein CEP-System arbeitet. Sie werden von Operatoren empfangen, verarbeitet und neu erzeugt. Schließlich bestehen zwischen den Operatoren noch *Kommunikationsverbindungen*, über die Operatoren Ereignisse übertragen können. Eine Reihe von solchen Übertragungen zwischen zwei Operatoren kann man auch als *Ereignisstrom* bezeichnen. Eingehende Ereignisströme werden als I_i bezeichnet, die Menge aller Eingangsströme in einem Operator ist I. Die Korrelationsfunktion f bildet die Menge von Eingangsströmen I in eine Menge von Ausgangsströmen O ab, d.h. die Korrelationsfunktion ist eine Abbildung $f : I \rightarrow O$. Die Operatoren bilden untereinander ein Netzwerk, auf dem Ereignisströme zwischen den Knoten verschickt werden. Ein Beispielgraph findet sich in Abbildung 3.1. Dort sind 5 Operatoren eingezeichnet, die ein Netzwerk bilden.

3.2 Ereignisse

Definition 3.1 (Ereignis)**.** *Ein Ereignis σ_i ist ein Daten-Objekt von einem bestimmten Ereignistypen. Es bekommt von seinem Produzenten einen unveränderlichen Zeitstempel $\tau(\sigma_i)$ und eine eindeutige, fortlaufende und unveränderliche Sequenznummer $\rho(\sigma_i)$ zugeteilt.*

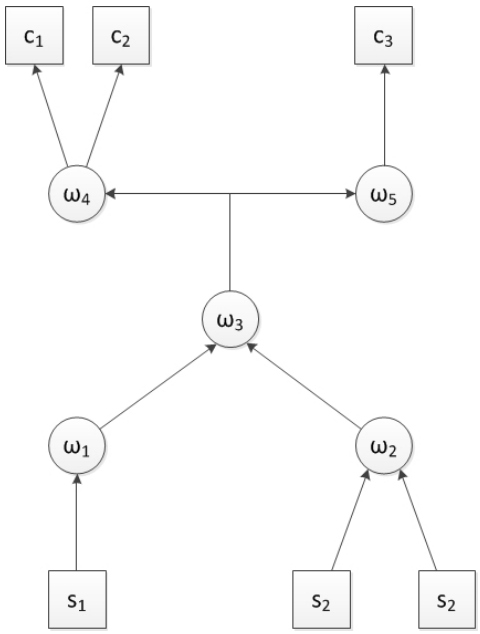

Abbildung 3.1: Ein CEP-System mit 5 Operatoren, 3 Ereignisquellen und 3 Konsumenten.

Ereignisse sind die Informationen bzw. Daten, die in einem CEP-System verarbeitet werden. Aus einer oder mehreren Ereignisquellen werden Ereignisströme erzeugt, die von dem CEP-System zu einem oder mehreren ausgehenden Ereignisströmen verarbeitet und an die Ereigniskonsumenten ausgeliefert werden. Auch innerhalb des CEP-Systems kommunizieren die einzelnen Operatoren über solche Ereignisströme. Ein Ereignis kann man sich als Datenobjekt vorstellen, in dem alle für die Verarbeitung wichtigen Informationen enthalten sind. In den folgenden Unterkapiteln werden diese Informationen erläutert.

3.2.1 Ereignistyp

Der *Ereignistyp* ist für jeden ausgehenden Ereignisstrom von einem Produzenten eindeutig festgelegt und wird als String kodiert. Durch die feste Kopplung von Ereignistyp und Produzenten kann jedes Ereignis anhand seines Typs eindeutig einem Produzenten zugeordnet werden.

3.2.2 Zeitstempel τ

In der Anwendung von Regeln zur Korrelation von Ereignissen spielt oft der Zeitpunkt eine wichtige Rolle, zu dem die Ereignisse aufgetreten sind. Die Beantwortung der Frage, wann eine Situation, die durch ein Ereignis signalisiert wird, genau eingetreten ist, hängt vom Kontext der Situation ab. Wir gehen aber davon aus, dass der Zeitpunkt des Auftretens von Situationen immer von den *Zeitpunkten der Ereignisse aus den synchronisierten Ereignisquellen*

abhängt und nicht vom Zeitpunkt des „Entdeckens“ der Situation durch einen Operator (die je nach Geschwindigkeit der Verarbeitung ohnehin schwanken kann). Daher werden die Zeitstempel erzeugter Ereignisse auf Basis der Zeitstempel der entsprechenden eingehenden Ereignisse erzeugt. Denkbar wäre ein Maximalwert oder Minimalwert der Zeitstempel aller korrelierten Ereignisse, welcher den Zeitstempel des entsprechend erzeugten Ereignisses bildet, oder auch die Angabe einer *Zeitspanne*. Der Zeitstempel ist somit unabhängig von der Echtzeit, zu der ein Ereignis von einem Operatoren erzeugt wird. Es kann dabei durchaus vorkommen, dass zwei Ereignisse desselben Typs mit demselben Zeitstempel auftreten, beispielsweise wenn ein Ereignis in der Korrelation mehrerer Ereignisse mitwirkt.

3.2.3 Sequenznummer ρ

Von seinem Produzenten erhält ein Ereignis σ_i eine **Sequenznummer** $\rho(\sigma_i)$. Sie ist unabhängig von einer physikalischen Uhr im Produzenten, es handelt sich vielmehr um eine Ordnung über die produzierten Ereignisse. Die Sequenznummern werden inkrementell vergeben, sodass folgende Invariante gilt:

Invariante 3.2.1. *Wenn ein Ereignis σ_i von einem Ereignistyp zeitlich vor einem anderen Ereignis σ_j vom selben Ereignistyp erzeugt wurde, dann gilt: $\rho(\sigma_i) < \rho(\sigma_j)$.*

Diese Invariante ist für das in dieser Arbeit entwickelte Verfahren wichtig: Die Ereignisse eines Typs können so in eine Reihenfolge gebracht werden und von einem Ereignis aus in „frühere“ und „spätere“ Ereignisse eingeteilt werden. Ereignisströme von einem bestimmten Ereignistypen sind also geordnet. Global über alle Ereignistypen hinweg ergibt sich hingegen durch die Sequenznummern alleine keine Ordnung.

3.2.4 Zusammenfassung: Inhalt eines Ereignisses

Ein Ereignis enthält also folgende Informationen:

- Typ des Ereignisses
- Zeitstempel vom Produzenten
- Sequenznummer vom Produzenten
- Payload: Daten des Ereignisses wie z.B. Attribute, die für die Weiterverarbeitung von Interesse sind

3.3 Operatoren

Definition 3.2 (Operator)**.** *Ein Operator ω_i ist ein Prozess, der auf einem physikalischen Host host(ω_i) läuft. Er empfängt eingehende Ereignisströme, sequenziert diese in einen globalen Eingangsstrom I und verarbeitet diesen anhand der Korrelationsfunktion $f : I \rightarrow O$ weiter zu einer Menge von ausgehenden Ereignisströmen O, die wiederum anderen Operatoren als Eingangsströme dienen können. Die so erzeugten Ereignisse sind von einem bestimmten Ereignistypen.*

3.3.1 Vernetzung von Operatoren

Operatoren sind untereinander über Kommunikationsverbindungen (siehe Abschnitt 3.4) vernetzt, über die Ereignisströme fließen. Ein ausgehender Ereignisstrom O_i eines Operators ω_i kann dabei einem anderen Operatoren ω_j als eingehender Ereignisstrom dienen. ω_j gilt dann als *Nachfolger* von ω_i und wird Teil der Menge der Nachfolger, die durch die Bezeichnung *succ(ω_i)* gekennzeichnet ist. Anders herum ist ω_j dann *Vorgänger* von ω_i und ist Teil der Menge *pred(ω_i)*. Über die Nachfolger und Vorgänger eines Operatoren ω_i lässt sich jeweils eine transitive Hülle bilden, die als *succ*(ω_i)* bzw. *pred*(ω_i)* bezeichnet wird.

3.3.2 Fehlermodell von Operatoren

Operatoren werden auf miteinander verbundenen Netzwerkknoten ausgeführt, die Hosts genannt werden. Der Host eines Operators ω_i wird als *host(ω_i)* bezeichnet. Ein Host kann dabei mehrere Operatoren beinhalten. Ausfälle von Hosts laufen nach dem *Crash-Stop-Modell* ab: Dabei stoppt die Ausführung und der Host verliert sämtliche nicht-persistenten Daten über den Zustand seiner Operatoren [SS83]. Andere Operatoren werden über den Absturz nicht direkt informiert, sondern müssen sich die Information selbst beschaffen, beispielsweise über die Kontrolle von Heartbeats.

Das Verfahren in dieser Arbeit würde auch mit einem *Crash-Recovery-Fehlermodell* funktionieren, das ja im Prinzip eine Erweiterung von Crash-Stop darstellt. Anstatt dass ein ausgefallener Host durch einen anderen ersetzt wird, könnte dann darauf gewartet werden, dass der Host selbstständig wieder seine Arbeit aufnimmt, worauf er dann in einen definierten Zustand versetzt werden müsste. Da Crash-Stop aber ein allgemeineres Modell ist und viele Ausfallursachen abdeckt, wird in dieser Arbeit mit diesem Modell gearbeitet.

3.3.3 Sequenzierung der Eingangsströme

Ein Operator ω_i kann im Operatorgraphen mehrere Vorgänger-Operatoren, d.h. auch mehrere eingehende Ereignisströme, besitzen. Operatoren verfügen untereinander über keine Zeitsynchronisierung, sie laufen asynchron. Um Ereignisse aus mehreren Strömen in einer Korrelationsfunktion zu verarbeiten, werden die Ströme in einen einzigen Eingangsstrom I, der die Menge der einzelnen Eingangsströme I_i zusammenfasst, gebracht. I fasst die

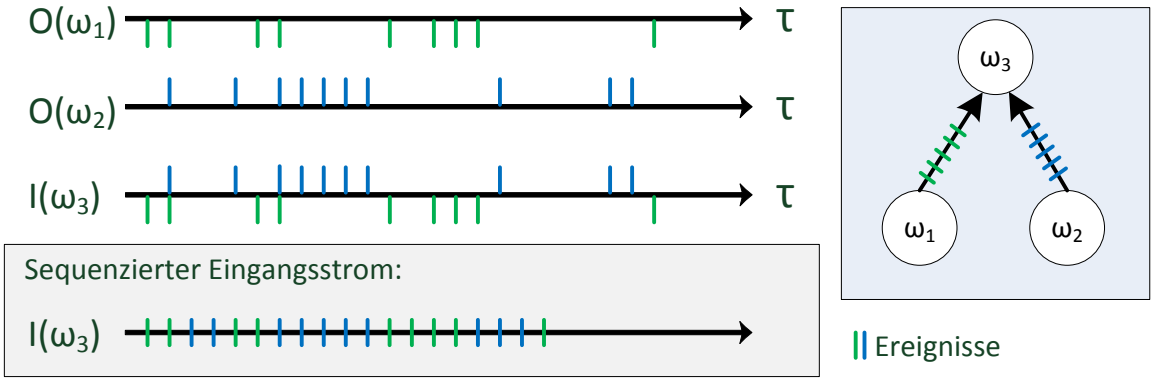

Abbildung 3.2: Sequenzierung zweier Ströme zum Eingangsstrom I.

einzelnen Ströme zusammen und ordnet sie gegeneinander, sodass Ereignisse verschiedener Eingangsströme gegeneinander eine eindeutige Positionierung bzw. Ordnung besitzen. Man spricht hier auch vom *Sequenzieren* der eingehenden Ereignisse, die anschließend vom Operatoren verarbeitet werden können. Die Ereignisse werden nach ihrem *Zeitstempel* angeordnet, bei gleichem Zeitstempel entscheidet der *Ereignistyp* und als drittrangiges Kriterium die *Sequenznummer*. In Abb. 3.2 ist ein Szenario dargestellt, das eine zweistufige Sequenzierung zweier einzelner Eingangsströme in einen gemeinsamen Eingangsstrom I zeigt: Die beiden Ausgangsströme der Vorgängeroperatoren werden dabei zunächst in eine Sequenz mit aufsteigenden Zeitstempeln gebracht, wobei auch verschiedene Ereignisse mit dem gleichen Zeitstempel auftreten können. Daher wird nach der ersten Sortierung nach dem primären Kriterium (Zeitstempel) eine weitere Sortierung nach dem sekundären Kriterium (Herkunft bzw. Typ des Ereignisses) durchgeführt, in dem Beispiel werden bei gleichem Zeitstempel Ereignisse aus ω_1 vor Ereignissen aus ω_2 einsortiert. Wäre entgegen des Beispielfalls auch dann noch keine eindeutige Sequenz erreicht, würde eine Sortierung nach dem tertiären Kriterium (Sequenznummer) eine eindeutige Lösung ermöglichen. Die primäre Anordnung nach Zeitstempel kommt dem häufigen Anwendungsfall entgegen, dass ein Operator Ereignisse über bestimmte Zeiträume korreliert, um zeitabhängige Situationen zu erkennen. Die Korrelationsfunktion kann in einem nach Zeitstempeln geordneten Strom solche Korrelationen viel einfacher realisieren als bei einer Sortierung des Stroms nach anderen Kriterien. Wird ein Ereignis in der Sequenzierung an einer bestimmten Stelle eingeordnet, ist durch das Ordnungsverfahren sichergestellt, dass nachfolgende Ereignisse im Eingangsstrom keinen niedrigeren Zeitstempel haben. Dadurch kann die Verarbeitung des Eingangsstroms I effizient gestaltet werden.

Die Sequenzierung ist also Teil des Ausführungsmodells der Operatoren und dient der Verarbeitungseffizienz.

3.3.3.1 Definition von Relationen über Ereignisse im globalen Eingangsstrom I

Folgende Relationen über zwei Ereignisse σ_i und σ_j werden definiert:

$\sigma_i > \sigma_j$: σ_i liegt in I *hinter* σ_j.

$\sigma_i < \sigma_j$: σ_i liegt in I *vor* σ_j.

$\sigma_i = \sigma_j$: σ_i *ist identisch* mit σ_j.

$\sigma_i \geq \sigma_j$: σ_i liegt in I *hinter* σ_j **oder** *ist identisch* mit σ_j.

$\sigma_i \leq \sigma_j$: σ_i liegt in I *vor* σ_j **oder** *ist identisch* mit σ_j.

3.3.4 Konsum von Ereignissen

Ein eingehendes Ereignis kann von einem Operator mehrfach zur Berechnung eines neuen Ereignisses herangezogen werden. An einem bestimmten Punkt kann der Operator allerdings ein Ereignis nach seiner Verarbeitung „aus dem Rennen nehmen'", d.h. aus dem Eingangsstrom löschen, sodass es in künftigen Berechnungen keine Rolle mehr spielt. Wird ein Ereignis in einer Berechnung genutzt und anschließend gelöscht, spricht man davon, dass das Ereignis *konsumiert* wurde. Wird ein Ereignis nach seiner Nutzung nicht direkt konsumiert, steht es für weitere Operationen weiterhin im Eingangsstrom zur Verfügung, bis es schließlich doch konsumiert wird oder der Operator es verwirft.

Die Konsumierung von Ereignissen kann ein wichtiger Bestandteil von CEP-Systemen sein und wurde beispielsweise in der Ereignisverarbeitungssprache **snoop** von S. Chakravarthy und D. Mishra [CM94] thematisiert. Dort wird diese Thematik durch die *Parameterkontexte* behandelt, die eindeutige Korrelationsvorschriften bei einem nicht eindeutigen Eingangsstrom ermöglichen. Hintergrund dieser Kontexte ist, dass Korrelationsvorschriften der Art „Korreliere immer ein Ereignis des Typs A mit einem Ereignis des Typs B" manchmal alleine noch keine hinreichend eindeutige Verarbeitung ermöglichen. Wenn nun beispielsweise im Eingangsstrom mehrere Ereignisse beider Typen vorliegen, muss der Operator eindeutig entscheiden können, welche spezifischen Ereignisse aus der Menge der potentiellen Korrelationskandidaten er zur Berechnung der Ausgangsereignisse heranzieht. In snoop wurden dafür verschiedene Möglichkeiten definiert, die in Tabelle 3.1 erläutert und in Bezug zu den dafür notwendigen Konsumoperationen gesetzt werden.

Um alle Parameterkontexte aus snoop unterstützen zu können, ist für die Operatoren also die Möglichkeit des gezielten Ereigniskonsums unerlässlich. Es muss möglich sein, manche Ereignisse mehrmals zu verarbeiten, während andere schon nach der ersten Verarbeitung entfernt werden. Die Implikationen, die sich aus dem Ereigniskonsum für das Wiederherstellungsverfahren ergeben, das in dieser Arbeit entwickelt wird, werden in Kapitel 6.2.1 ausführlich analysiert.

Name	Beschreibung	Konsumoperationen
Recent	Es werden immer die *neuesten* möglichen Ereignisse zur Korrelation herangezogen, sobald eine Korrelation möglich ist. Nach der Korrelation werden die daran teilnehmenden Ereignisse konsumiert.	Konsumiert werden die neusten Ereignisse, die zu einer Korrelation geführt haben.
Chronicle	Ereignisse werden nach ihrem Auftreten in *chronologischer* Reihenfolge zur Korrelation herangezogen. Das bedeutet, dass immer die *ältesten* verfügbaren Ereignisse korreliert werden. Nach der Korrelation werden die daran teilnehmenden Ereignisse konsumiert.	Konsumiert werden die ältesten Ereignisse, die zu einer Korrelation geführt haben.
Continuous	Aus dem Eingangsstrom werden *kontinuierlich* Korrelationen durchgeführt, und zwar mit jedem Ereignis, das als erstes Ereignis einer Korrelation in Frage kommt und den ältesten nachfolgenden weiteren für die Korrelation relevanten Ereignissen. Jedes Ereignis kann nur eine Korrelation starten, daher wird das Startereignis konsumiert.	Konsumiert wird das Startereignis einer Korrelation.
Cumulative	Alle Ereignisse zwischen dem ersten Ereignis, das eine Korrelation starten kann, und dem ersten Ereignis, mit dem die Korrelation abgeschlossen wird, werden in die Korrelation mit einbezogen. Nach der Korrelation werden alle daran beteiligten Ereignisse konsumiert.	Konsumiert werden alle Ereignisse zwischen dem Startereignis und dem letzten Ereignis in einer Korrelation.

Tabelle 3.1: Parameterkontexte in snoop [CM94]

3.4 Kommunikationsverbindungen

Um die Betrachtung des Operatorenmodells von der Problematik möglicher Ausfälle der Kommunikationsverbindungen zu entkoppeln, werden für diese Verbindungen bestimmte Eigenschaften vorausgesetzt. Hinweise zur Implementierung der eingeführten Abstraktionen finden sich in der Literatur [GR06].

Eigenschaft 3.4.1 (Keine Erzeugung)**.** *Wenn ein Operator ω_2 ein Ereignis σ empfängt, dann hat ein anderer Operator ω_1 oder eine Ereignisquelle q dieses Ereignis erzeugt.*

Eigenschaft 3.4.2 (Keine Duplikation)**.** *Jedes übertragene Ereignis σ, das von einem Operator ω gesendet wird, wird maximal **einmal** empfangen.*

Eigenschaft 3.4.3 (Best-Effort)**.** *Wenn Operator ω_1 an Operator ω_2 ein Ereignis σ sendet, und keiner der Operatoren einen Fehler vermutet, dann empfängt ω_2 σ **letztendlich**.*

Dies sind die Eigenschaften eines sog. Best-Effort-Kommunikationskanals (vgl. [GO96]). Zusätzlich wird noch folgende Eigenschaft benötigt:

Eigenschaft 3.4.4 (Erhaltung der Reihenfolge)**.** *Bei der Übertragung mehrerer Ereignisse zwischen zwei Operatoren empfängt der Empfänger die Ereignisse in der Reihenfolge, in denen sie vom Sender versendet wurden.*

Die Eigenschaften der Kommunikationskanäle werden im Folgenden erläutert:

- Eigenschaft 3.4.1 besagt, dass die Kommunikationskanäle keine Ereignisse an den Empfänger übermitteln dürfen, die nicht vom Sender selbst erzeugt wurden.
- Eigenschaft 3.4.2 besagt, dass die Kommunikationskanäle Ereignisse bei der Übertragung nicht duplizieren dürfen. Die Frage, ob Operatoren Ereignisse mehrfach erzeugen, wird von dieser Eigenschaft nicht berührt.
- Eigenschaft 3.4.3 besagt, dass jede Nachricht letztendlich korrekt übertragen wird, solange weder der Sender noch der Empfänger einen Fehler im Kommunikationspartner oder -link vermutet. Die Frage der Fehlerdetektion, d.h. wie die Operatoren den Ausfall anderer Operatoren oder der Verbindung zu diesen detektieren, wird in Kapitel 6.5.3.2 untersucht.
- Eigenschaft 3.4.4 besagt, dass der Kommunikationskanal Ereignisse in der Reihenfolge beim Empfänger abliefert, in der sie der Sender versendet hat.

3.5 Ereignisquellen

Ereignisquellen sind Systemkomponenten, die eine Schnittstelle zu der Welt außerhalb des CEP-Systems realisieren. Zu einer solchen „Umwelt“ können zum Beispiel Sensoren gehören oder auch Computersysteme, die Signale an die Ereignisverarbeitung weiterleiten. Die aus der Umgebung eingehenden Informationen werden zu Ereignissen im Sinne des CEP-Systems weiterverarbeitet, d.h. die für die Verarbeitung in den Operatoren notwendigen Informationen werden hinzugefügt und das Ereignis wird in ein für die Operatoren lesbares Format gebracht. In diesem Sinne sind Ereignisquellen mit den „Event Adapters“ aus dem in D. Luckhams Buch *„The Power of Events [...]“* [Luco1] eingeführten CEP-System vergleichbar.

Als besonderes Merkmal besitzen die Ereignisquellen im Gegensatz zu den Operatoren *untereinander synchronisierte Systemuhren*, damit sie den Ereignissen *Zeitstempel* geben können, welche die synchronisierte Realzeit der Ereignisse wiederspiegeln. Die von Ereignisquellen erzeugten Ereignisse sind die einfachsten Ereignisse im CEP-System und werden deshalb auch *Basisereignisse* genannt. Die Ereignisse werden in einem *Ereignislog* persistent gespeichert, bis sie nicht mehr benötigt werden. Außerdem können Ereignisquellen auch andere notwendigen Daten persistent speichern. Ereignisquellen müssen so implementiert werden, dass sie für die Anforderungen des Gesamtsystems ausreichend ausfallsicher sind, entweder

durch eine redundante Auslegung oder durch den Einsatz von geeigneten Wiederherstellungsverfahren. Sie können vom CEP-System nicht wiederhergestellt werden und dürfen daher keine Daten verlieren, die nicht vorher explizit zur Löschung freigegeben wurden. Die Implementierung dieser Abstraktion wird in dieser Arbeit nicht thematisiert, eine Lösung über robuste, persistente Speichersysteme wie z.B. „Redundant Arrays of Independent Disks“ (RAIDs) ist dafür denkbar.

3.6 Ereigniskonsumenten

Ereigniskonsumenten sind Adapter zwischen den Operatoren, deren Ausgangsströme das CEP-System verlassen, und den an den Ereignissen interessierten Stellen in der Welt außerhalb des CEP-Systems. Ebenso wie die Ereignisquellen müssen sie ständig verfügbar sein, müssen aber keine Informationen aus den Operatoren persistent speichern können. Die genaue Implementierung dieser Abstraktion wird ebenso wie bei den Ereignisquellen in dieser Arbeit nicht weiter thematisiert, eine Lösung über redundante oder wiederherstellbare Prozesse ist dafür denkbar.

3.7 Nutzung von Kontrollereignissen zur Effizienzsteigerung

Um lange Wartezeiten bei Berechnungen und Sequenzierungen und zu vermeiden, können Quellen von Zeit zu Zeit Kontrollereignisse versenden, die entsprechend weiterverarbeitet werden können. Auf diese Weise schreitet die Gesamtverarbeitung fort und keine Stelle im System wartet vergeblich auf Ereignisse, die niemals ankommen werden.

3.8 Zeit und Asynchronität - Analyse der Implikationen

3.8.1 Synchrone Quellen

Nur die Ereignisquellen verfügen über hinreichend synchronisierte Systemuhren, um tatsächliche Echtzeitstempel für die Basisereignisse vergeben zu können. Mithilfe der Zeitstempel aus den Quellen wird in höheren Verarbeitungsebenen der Bezug der korrelierten Ereignisse zur Echtzeit einer eingetretenen Situation hergestellt.

3.8.2 Asynchrone Operatoren

Für die Operatoren werden keine Annahmen über zeitliche Abläufe getroffen, was Berechnungen und Kommunikationsverzögerungen angeht. Dadurch hat das eingeführte Systemmodell eine hohe Abdeckung bezüglich realer Systeme. Synchronität in den Operatoren wird für das Wiederherstellungsverfahren weder vorausgesetzt noch benötigt.

4 Zielstellung

4.1 Definition der Zielstellung

Diese Arbeit adressiert große Szenarien in verteilten CEP-Systemen, die hohe Anforderungen an die Skalierbarkeit, die Verfügbarkeit, die Latenz, die Korrektheit und die Ausfallsicherheit stellen. Es wird ein Modell für ein hochskalierbares, verteiltes CEP-System entworfen, das durch ein Wiederherstellungsverfahren für Ereignisströme gegenüber gutartigen Fehlern in den Operatoren robust ist. Insbesondere dürfen beim *gleichzeitigen* Ausfall von einer festgelegten Anzahl F an Operatoren die Anforderungen (siehe Abschnitt 4.2) an das CEP-System nicht verletzt werden. Das Verfahren soll ohne persistente Checkpoints der Operatorzustände und ohne redundante Berechnungen auskommen, um den Overhead zur Laufzeit gering zu halten. Durch die Einführung von persistenten Schichten soll vielmehr ein dynamisch einsetzbares Werkzeug geschaffen werden, um Operatortopologien zu partitionieren und so die Belastung einzelner Operatoren mit Zustandsdaten zu verringern. Das Wiederherstellungsverfahren wird erläutert und anhand eines konkreten Szenarios in einer Simulation evaluiert.

4.2 Anforderungen an das CEP-System

Aus den Anwendungsfällen lassen sich Anforderungen ableiten, die im Allgemeinen an CEP-Systeme gestellt werden [VKR11, CDMRV10, SSMW10]. Ein CEP-System arbeitet dann zuverlässig, wenn es die gestellten Anforderungen unter bestimmten Bedingungen, d.h. auch in definierten Fehlersituationen, erfüllt.

4.2.1 Hohe Skalierbarkeit

In den typischen Anwendungsgebieten von CEP fallen mitunter hohe Datenmengen an, die vom CEP-System verarbeitet werden müssen. Die Vernetzung von Gegenständen im „Internet der Dinge“, beispielsweise durch Einsatz von RFID-Chips, produziert eine regelrechte Datenflut, die von CEP-Systemen bewältigt werden muss [SSMW10]. Daher muss das System skalierbar sein, d.h. es muss ohne Qualitätseinschränkungen mit steigenden Lasten zurechtkommen. Insbesondere darf die steigende Last nicht zu einer Erhöhung der Latenzzeiten oder zu einer Einschränkung der Korrektheit der Ergebnisse führen.

4.2.2 Geringe Latenzzeiten

Im Gegensatz zu Techniken, die zur Analyse von bestehenden Datensätzen relativ lange **nach** dem eigentlichen Auftreten der entsprechenden Vorgänge dienen, bspw. klassisches Data-Mining und Data-Warehouses, wird mit der Anwendung von CEP eine **sofortige** Analyse der Ereignisse in Echtzeit verfolgt. Je schneller eine Situation erkannt wird, desto besser. Werden aufgetretene Situationen zu spät erkannt, kann die Erkenntnis darüber sogar wertlos werden. Im Business Activity Monitoring (BAM) beispielsweise können CEP-Systeme zur Echtzeit-Erkennung von Situationen im Geschäftsprozess benutzt werden, beispielsweise zur Ermittlung von Key-Performance-Indikatoren (KPIs) und Überwachung von Service-Level-Agreements (SLAs) [CDMRV10]. Es dürfen also keine großen Verzögerungen (Latenzen) bis zur Entdeckung einer Situation auftreten.

4.2.3 Korrektheit der Ergebnisse

Die Anforderungen an die Korrektheit der Ergebnisse (d.h. der detektierten Situationen) eines CEP-Systems können je nach Anwendungsfall voneinander abweichen [OB10]. Anforderungen an die Korrektheit der Situationserkennung von CEP-Systemen sind in dieser Arbeit:

- Keine falschen Ereignisse, d.h. jedem vom CEP-System an einen Konsumenten abgegebenen Ereignis muss auch eine tatsächlich geschehene Situation zugrunde liegen.
- Keine verlorenen Ereignisse, d.h. jede aufgetretene Situation von Interesse wird auch tatsächlich durch ein entsprechendes Ereignis den Konsumenten mitgeteilt. Auch beim gleichzeitigen Ausfall von F Operatoren gehen keine für die Situationserkennung relevanten Informationen verloren.
- Erhaltung der Reihenfolge der Ereignisse, d.h. auch durch den Ausfall und die Wiederherstellung von F Operatoren darf die Reihenfolge der erkannten Situationen nicht durcheinandergeraten.

Strenge Korrektheitsanforderungen sind insbesondere dort gestellt, wo schon ein kleiner Fehler in der Ereignisverarbeitung zu großen Auswirkungen führen kann, zum Beispiel in der Produktionssteuerung einer Fabrik [VKR11]. Ein detailliertes Beispielszenario, das solche hohen Korrektheitsanforderungen an das CEP-System begründet, wird in Abschnitt 4.3 vorgestellt.

4.2.3.1 Abgrenzung zu weicheren Korrektheitsanforderungen

Manchmal kann es ausreichend sein, bestimmte Mindestanforderungen zu stellen und einige nicht korrekt erkannte Situationen zu tolerieren oder außerhalb des CEP-Systems manuell zu behandeln [OB10]. Insbesondere kann so eine Verbesserung der Performanz des Systems erreicht werden. Beispiele: (i) Im Tracking von RFID-Daten können zeitweise Ausfälle verkraftet werden, indem entsprechende Situationen manuell nachgeprüft werden.

(ii) Beim Berechnen von Durchschnittstemperaturen kann ein Konsument unter Umständen damit leben, dass über bestimmte Zeiträume hinweg einige Sensoren vom CEP-System nicht korrekt berücksichtigt werden, so lange ein genügend großer Teil der Temperaturdaten in die Gesamtberechnung eingeflossen ist.

Diese Arbeit konzentriert sich allerdings auf eine Lösung, die hohe Korrektheitsanforderungen abdeckt.

4.2.4 Ausfallsicherheit

CEP-Systeme sollen im Allgemeinen *hochverfügbar* sein. Auch beim gleichzeitigen Ausfall von F Operatoren soll das Gesamtsystem weiterhin verfügbar sein und korrekt arbeiten und nach außen hin keine Abweichungen im Verhalten zeigen. Ereignisse dürfen weder *verloren* gehen noch *verändert* werden, und auch die Latenzanforderungen müssen beim Ausfall mehrerer Operatoren weiterhin eingehalten werden.

4.3 Beispielszenario: Logistik in einem großen Seehafen

Zur Steuerung und Überwachung der Logistik in einem großen Seehafen soll ein verteiltes CEP-System eingesetzt werden. Eine Hafenlogistik bietet von Natur aus ein verteiltes Szenario: Es gibt viele verschiedene Docks und Lagerhallen, an denen täglich tausende Container von dutzenden großen Containerschiffen umgeschlagen und teilweise auch zwischengelagert werden. Jeder Container ist mit RFID-Chips ausgestattet und überall auf dem Hafengelände sind Sensoren installiert, die die Position der Container ständig ermitteln. Auch innerhalb eines Containers fallen Sensordaten an, beispielsweise über das Klima innerhalb des Containers (Temperatur, Luftfeuchtigkeit, etc.) und den Zustand der Ladung. Insgesamt fallen in jeder Sekunde viele tausend Ereignisse aus den Sensoren an, die möglichst lokal in Operatoren weiterverarbeitet werden, um Situationen wie z.B. „Container X ist in Lagerhalle Y angekommen“, „Kühlaggregat in Container X ausgefallen, Temperatur erreicht kritischen Wert“, etc. zu erkennen. Zudem ist die Hafenlogistik nur ein Teil der gesamten Logistikkette, die sich über verschiedene Transportwege erstrecken kann, beispielsweise bei einer Verladung auf LKWs oder Güterzüge.

4.3.1 Begründung der harten Korrektheitsanforderungen

Die Anforderungen aus Abschnitt 4.2.3 werden für das vorliegende Szenario folgendermaßen begründet:

1.) Keine falschen Ereignisse: Falsche Ereignisse könnten zu falschen Warnmeldungen oder auch falschen Abrechnungen mit beteiligten Dienstleistern führen.

2.) Keine verlorenen Ereignisse: Wenn beispielsweise eine Positionsaktualisierung eines Containers verloren geht, könnte es schwer werden, diesen auf dem riesigen Hafengelände wiederzufinden.

3.) Erhaltung der Reihenfolge: Wäre die Reihenfolge verändert, könnten falsche Schlüsse aus den Ereignissen gezogen werden, beispielsweise bei zwei aufeinanderfolgenden Positionsaktualisierungen wäre der daraus extrahierte Weg eines Containers falsch.

5 Abgrenzung

Um die Anforderungen zu erfüllen, die in Kapitel 4.2 insbesondere an die Ausfallsicherheit und Korrektheit verteilter CEP-Systeme gestellt wurden, sind verschiedene Lösungsansätze denkbar. In diesem Kapitel werden die grundsätzlichen Möglichkeiten untersucht, wie CEP-Systeme ausfallsicher gestaltet werden können, und der Lösungsraum wird bezüglich der weiteren Anforderung aus Kapitel 4.2 abgegrenzt.

Um die Ausfallsicherheit zu steigern, kann man prinzipiell verschiedene Methoden anwenden und miteinander kombinieren. Zu den wichtigsten Methoden gehört, die Ausfallsicherheit der einzelnen Komponenten zu verbessern, ausgefallene Komponenten wiederherzustellen und die Komponenten redundant anzulegen [RH85]. In der Vergangenheit haben sich zur Steigerung der Ausfallsicherheit verteilter Systeme vor allem 2 Ansätze durchgesetzt: (1) Aktive Replikation von redundanten Prozessen [VKR11] und (2) Rollback-Recovery ausgefallener Prozesse anhand persistent gespeicherter Zustandsdaten [EAWJ02, SM11]. Im Folgenden werden beide Ansätze vorgestellt und auf ihre Tauglichkeit bezüglich der Anforderungen aus Kapitel 4.2 untersucht.

5.1 Redundante Prozesse

Im Entwurf eines verteilten CEP-Systems kann man dieses redundant auslegen, sodass einige an der verteilten Berechnung teilnehmenden Operatoren abstürzen können und das System als Ganzes weiterhin fehlerfrei arbeitet. In der Praxis erreicht man ein solches Verhalten durch das Vorhalten von Operator-Replikaten [VKR11]. Während ein Operator im Normalbetrieb läuft, muss er F Replikate vorhalten, die im Falle seines Ausfalls diesen Operatoren ersetzen können, sodass die Berechnung fehlerfrei weiterläuft (vgl. Abb. 5.1). Die Replikate müssen immer auf dem Stand des „Original-Operators“ gehalten werden, was durch die Replikation der eingehenden Ereignisse an alle Replikate realisiert wird (*active standby*). Da die Replikate dieselben Ereignisse mit denselben Algorithmen verarbeiten wie der Original-Operator, wird der Zustand der Replikate immer aktuell gehalten. Fällt der Original-Operator aus, kann eines der Replikate direkt die Arbeit übernehmen. Diesen Ansatz kann man noch verfeinern, indem man die Versendung von ausgehenden Ereignissen so koordiniert, dass nur einer der redundanten Operatoren diese Ereignisse an die Nachfolger weiterleitet [VKR11].

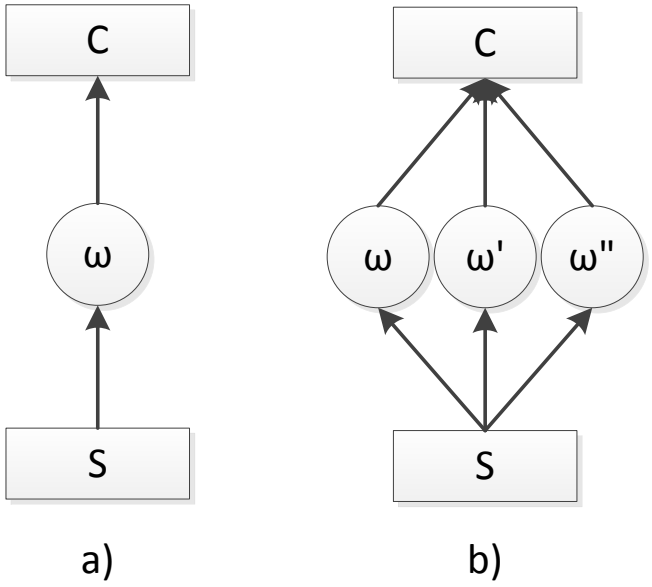

Abbildung 5.1: Active Standby: Links ein System ohne Replikate, rechts ein System mit redundanter Auslegung [VKR11].

5.1.1 Bewertung der Lösung

Durch die redundanten Operatoren wird die Latenzzeit des Gesamtsystems nicht erhöht. Die Ergebnisse sind auch bei Operatorausfällen korrekt [VKR11]. Die Ausfallsicherheit bei F gleichzeitigen Ausfällen ist dann gegeben, wenn jeder Operator F Replikate bereitstellt. Eine solche redundante Lösung kann also die Anforderungen nach geringen Latenzzeiten, Korrektheit der Ergebnisse und der Ausfallsicherheit erfüllen.

Die Frage nach der Skalierbarkeit gestaltet sich schwieriger. Für eine hohe Ausfallsicherheit in großen Operatortopologien müssen sehr viele aktive Replikate vorgehalten werden. Während ein Operator hingegen fehlerfrei läuft, verschwenden die Replikate Ressourcen, da sie in dieser Zeit vergeblich arbeiten. Die Lösung ist also zumindest sehr *kostenintensiv*. Ein Vorhalten von genügend Replikaten für *extreme Ausfallsituationen* ist kaum wirtschaftlich durchzuführen. Eine Lösung mit weniger Overhead im Normalbetrieb, d.h. in der Zeit, in der keine Fehler auftreten, wäre unter der Annahme begrenzter Ressourcen besser skalierbar.

5.2 Rollback-Recovery mit persistentem Zustandsspeicher

Anstatt redundante Operatoren vorzuhalten, die einen Ausfall kompensieren können, kann man auch den Versuch unternehmen, einen Operator nach dessen Absturz wiederherzustellen. In CEP-Systemen gibt es zwei Möglichkeiten, wie ein Operator mit eingehenden Ereignissen verfährt. Entweder er betrachtet jedes Ereignis einzeln und bearbeitet es ausschließlich bezüglich der Informationen, die dieses einzelne Ereignis beinhaltet (was die Möglichkeiten der Ereignisverarbeitung aber extrem beschneidet). In diesem Fall ist der Operator *zustandslos*, die Verarbeitung eines Ereignisses steht für sich alleine und ist nicht von anderen, vorher oder nachher auftretenden Ereignissen abhängig. Oder er aggregiert Informationen aus mehreren eingehenden Ereignissen, die er zusammenfasst und in ausgehende Ereignisse weiterverarbeitet. Dann baut der Operator einen *internen Zustand* auf, der Informationen über schon ausgeführte Verarbeitungsschritte beinhaltet und das weitere Verhalten beim Eingang neuer Ereignisse bestimmt. Solch ein interner Zustand muss, um den mit einem Ausfall einhergehenden Verlust der volatil gespeicherten Daten zu überstehen, in irgendeiner Form persistent gesichert werden, damit der Operator wiederhergestellt werden kann. Um den internen Zustand eines Operators zu sichern, gibt es zwei mögliche Verfahren [EAWJ02]: *Checkpoints* und *Logs*.

Checkpoints [KBG08] sind Speicherauszüge von Operatoren, die entweder in regelmäßigen Abständen oder durch ein übergeordnetes Protokoll koordiniert angefertigt werden. Diese enthalten den internen Zustand des Operators zum Zeitpunkt des Anlegens des Checkpoints. Im Falle einer Wiederherstellung wird der Zustand, der im Checkpoint gespeichert ist, in den Operator geladen und die Arbeit wird an dieser Stelle fortgesetzt. Zustandsinformationen, die zwischen dem Anlegen eines Checkpoints und dem Absturz des Operators angefallen sind, gehen verloren.

Logs speichern zunächst alle in einen Operator eingehenden Ereignisse. Falls ein Operator ausfällt und wiederhergestellt werden muss, werden die gespeicherten Ereignisse aus dem Log geladen und in den neu gestarteten Operatoren eingespeist. Die Verarbeitung wird sozusagen *zurückgerollt* und neu durchgeführt, was schließlich darin mündet, dass der Zustand vor dem Absturz wiederhergestellt wird. Allerdings können durch die mehrfache Verarbeitung von eingehenden Ereignissen Duplikate produziert werden, die von nachfolgenden Operatoren herausgefiltert werden müssen. Es ist sinnvoll, die beiden Verfahren der Logs und der Checkpoints miteinander zu kombinieren. So wird von Zeit zu Zeit ein Checkpoint gespeichert und die Zeit zwischen zwei Checkpoints wird durch Logs abgebildet. Zur Wiederherstellung wird dann der letzte Checkpoint geladen und alle Logs ab diesem Zeitpunkt werden erneut eingespeist, damit nicht die gesamte Verarbeitung seit dem initialen Start des Operators wiederholt werden muss. Die Logs sollten zudem regelmäßig „zurechtgeschnitten“ (Pruning) werden, d.h. nicht mehr benötigte Logs werden gelöscht, damit der Speicher entlastet wird.

5.2.1 Bewertung der Lösung

Die Korrektheit der Ergebnisse und die Ausfallsicherheit bei F gleichzeitigen Ausfällen sind mit diesem Verfahren umsetzbar: Zum einen kann ein Operator durch die Checkpoints und Logs in genau den Zustand versetzt werden, den er zum Zeitpunkt seines Ausfalls hatte, sodass die produzierten Ereignisse nicht vom Normalbetrieb abweichen. Zum anderen sind die Operatoren durch die lokale Speicherung ihrer Zustandsinformationen untereinander nicht auf Hilfe bei der Wiederherstellung angewiesen, sodass auch mehrere gleichzeitige Ausfälle auf die Wiederherstellung des einzelnen Operators keinen Einfluss haben.

Das beschriebene Rollback-Recovery-Verfahren ist auf einen persistenten Speicher angewiesen, der die Zustandsdaten (Checkpoints und Logs) sicher speichert, sodass der Operator nach seinem Neustart auf den letzten bekannten Zustand initialisiert werden kann. Eine persistente Speicherung ist im Sinne des Ressourcenverbrauchs teuer, insbesondere in CEP-Systemen, die sehr viele Ereignisse in kurzer Zeit verarbeiten müssen. Zudem benötigt das Anlegen eines Speicherabbilds, d.h. eines Checkpoints, Prozessorzeit, was zu Verzögerungen in der Verarbeitung der Ereignisströme führen kann. Die Skalierbarkeit und die Latenzzeiten können bei einem solchen System also schnell zu kritischen Punkten werden.

5.3 Schlussfolgerungen

Wie sich in der Untersuchung klassischer Lösungen zur Steigerung der Ausfallsicherheit gezeigt hat, sind die Ansätze zwar geeignet, einen Operatoren korrekt wiederherzustellen, jedoch nur zum Preis eines hohen Overhead während des Normalbetriebs. Bei hohen Lasten und großen Topologien sind die Verfahren zumindest sehr teuer, was bei begrenzten Ressourcen zu Problemen mit der Skalierbarkeit führt. Gesucht ist ein Verfahren, das ohne redundante Operatoren und ohne persistente Zustandsspeicherungen in den Operatoren auskommt. Ein solches Verfahren soll im Folgenden entwickelt werden.

6 Problemlösung

In Kapitel 3 wurde ein Systemmodell für verteilte CEP-Systeme eingeführt und es wurden einige Eigenschaften des eigentlichen Ausführungsmodells festgelegt, um anhand der darin definierten Begrifflichkeiten die Aufgabenstellung dieser Diplomarbeit eindeutig beschreiben zu können. In dem nun folgenden Kapitel wird nun zunächst das Ausführungsmodell, das dem Wiederherstellungsverfahren zugrunde liegt, vollständig beschreiben. Darauf aufbauend wird das Wiederherstellungsverfahren für ausgefallene Operatoren entwickelt, indem zunächst der Zustand eines Operators beschrieben und das Vorgehen bei der Verteilung dieser Informationen im CEP-System Schritt für Schritt herausgebildet wird. Anschließend werden Algorithmen zur Fehlerdetektion, zur Wiederherstellung der Operatortopologie und zur Wiederherstellung des Zustands von Operatoren entworfen. All dies macht dann zusammengenommen das Verfahren zur Wiederherstellung von Operatoren aus und bildet damit den Hauptteil dieser Diplomarbeit.

6.1 Fensterbasiertes Ausführungsmodell von Operatoren

Wie in der Definition für Operatoren festgelegt, hat ein Operator ω_i einen Eingangsstrom I, der durch die Sequenzierung nach Zeitstempeln τ geordnet und mit inkrementellen Sequenznummern ρ ausgestattet ist. Auf einem solchen Strom lässt sich ein Teilstrom S[σ_{first}, σ_{last}] zwischen zwei Ereignissen σ_{first} und σ_{last} definieren. Ein Operator implementiert eine Korrelationsfunktion f, die ein Fenster S auf dem Eingangsstrom I jeweils auf ein Ausgangsereignis im Ausgangsstrom O abbildet. Das heißt, die Korrelationsfunktion wird kontinuierlich hintereinander ausgeführt und arbeitet dabei in jeder Ausführung auf genau einem solchen Fenster S. Die aufeinander folgenden Fenster können dabei überlappen, d.h. bestimmte Ereignisse können mehrfach als Eingabe für Ausführungen der Korrelationsfunktion dienen. Es gibt aber immer nur ein *aktuelles Korrelationsfenster* w zu einem bestimmten Zeitpunkt in einem bestimmten Operator. Ein solches Ausführungsmodell ist sehr ausdrucksstark, d.h. es kann eine große Bandbreite von möglichen CEP-Funktionalitäten in den Operatoren abgebildet werden (vgl. Abschnitt 6.1.3).

6.1.1 Verwaltung von Korrelationsfenstern

6.1.1.1 Das Prädikat P_{first}

Bei der Initialisierung eines Operators muss auf dem Eingangsstrom I auf einem Ereignis das erste Korrelationsfenster geöffnet werden. Welches der Ereignisse dafür genutzt wird, wird

im Prädikat P_{first} beschrieben. Das Prädikat wird bezüglich jedes eingehenden Ereignisses σ *kontextfrei* ausgewertet, d.h. nur für sich und ohne Einbeziehung von Informationen aus anderen Ereignissen. Beim ersten Ereignis, auf welchem das Prädikat als `true` ausgewertet wird, wird das erste Korrelationsfenster geöffnet.

Beispiel Der Eingangsstrom eines Operators setzt sich aus 3 verschiedenen Ereignistypen zusammen: A, B und C. Der Operator korreliert jeweils eine Gruppe aus Ereignissen aller 3 Typen A;B;C in dieser Reihenfolge zu einem neuen Ereignis D. Bei dem aus 3 Eingangsströmen sequenzierten globalen Eingangsstrom

B B C ***A*** A C C ***B*** B ***C*** C

beispielsweise würden die kursivgedruckten Ereignisse zur Erzeugung eines neuen Ereignisses führen. Das Prädikat P_{first} wäre für das erste Ereignis des Typs A `wahr` und würde zur Öffnung des ersten Korrelationsfensters führen.

6.1.1.2 Das Prädikat P_{last}

Nachdem ein Korrelationsfenster geöffnet wurde und Ereignisse auf dem Eingangsstrom I innerhalb des offenen Fensters von der Korrelationsfunktion zur Erzeugung eines ausgehenden Ereignisses genutzt wurden, wird das Fenster nach Abschluss der dazu nötigen Berechnungen wieder geschlossen. Der Abschluss des Fensters markiert das Ende der Berechnung und muss im Falle der wiederholten Verarbeitung desselben Fensters zum selben Ergebnis führen. Um das letzte Ereignis des Fensters zu ermitteln, wird jedes Ereignis ab dem Startereignis bezüglich des Prädikats P_{last} ausgewertet. In die Auswertung des Prädikats können Informationen aus sämtlichen Ereignissen innerhalb des aktuellen Fensters einfließen, nicht jedoch aus Ereignissen, die außerhalb des Fensters liegen.

Beispiel Im Beispiel in Abschnitt 6.1.1.1 wäre das Prädikat P_{last} so definiert, dass das aktuelle Fenster geschlossen wird, sobald auf das Startereignis des Typs A schließlich ein weiteres Ereignis des Typs B und daraufhin ein weiteres Ereignis C gefolgt ist, wobei die entsprechenden Ereignisse nicht direkt aufeinander folgen müssen. Beim ersten Ereignis des Typs C, welches diese Bedingungen erfüllt, würde das aktuelle Korrelationsfenster geschlossen und das ausgehende Ereignis D erzeugt. Die Ereignisse, die in die Erzeugung von D eingeflossen sind, würden konsumiert, sodass sie in weiteren Korrelationen nicht mehr berücksichtigt werden.

6.1.1.3 Die Funktion *next(*σ, P_{first}*)*

Nachdem das aktuelle Korrelationsfenster geschlossen wurde, muss das Startereignis des nächsten Korrelationsfensters ermittelt werden. Dies geschieht im Operator mithilfe der Funktion *next(*σ, P_{first}*)*. Die Funktion sucht im Eingangsstrom ab dem Ereignis σ nach dem Ereignis, mit dem das nächste Korrelationsfenster beginnt, d.h. welches das Prädikat P_{first} erfüllt. Wichtig ist vor allem, dass die Berechnung reproduzierbar ist, d.h. bei einer Wiederholung der Berechnung des vorangegangenen Fensters auf demselben Eingangsstrom muss auch die Funktion *next(*σ, P_{first}*)* dasselbe Ereignis als nächstes Startereignis bestimmen. Die Funktion *next(*σ, P_{first}*)* ist wie folgt definiert:

$$next(\sigma, P_{first}) = \begin{cases} \sigma' \geq \sigma \wedge P_{first}(\sigma') : min(\sigma') \\ \perp \text{ wenn ein solches Ereignis nicht existiert} \end{cases}$$

Die Vergleichsrelationen wurden bereits in Kapitel 3.3.3.1 eingeführt und beziehen sich auf die Position eines Ereignisses im Eingangsstrom, die Bedingung min(σ') bedeutet, dass es kein den Bedingungen entsprechendes Ereignis geben darf, das im Strom weiter vorne platziert ist als σ' (σ' ist also minimal). Entsprechend ist σ' das früheste Ereignis im Strom, auf das die Bedingungen $\sigma' \geq \sigma$ und $P_{first}(\sigma')$ zutreffen. Im o.g. Beispiel wäre nach der Verarbeitung des ersten Korrelationsfensters das nächste Ereignis des Typs A ein mögliches Startereignis für die nächste Korrelation einer Gruppe aus A;B;C. Der Operator würde die Funktion *next(A,* P_{first}*)* aufrufen und das zweite Ereignis von Typ A als nächstes Startereignis ermitteln.

Die Parameter der Funktion bestimmt der Operator entsprechend der in ihm implementierten Semantiken. Durch den Parameter σ und die Prädikate P_{first} und P_{last} kann er bestimmen, wie sich das Korrelationsfenster auf dem Eingangsstrom fortbewegt. Dabei ist zu beachten, dass sich das Startereignis zwischen zwei aufeinanderfolgenden Fenstern nicht nach hinten im Strom verschieben darf, d.h. $\sigma \geq \sigma_{lastStart}$ mit $\sigma_{lastStart}$ als Startereignis des letzten Korrelationsfensters.

6.1.2 Ereigniskonsum

Im Ausführungsmodell darf ein Operator auf einem Eingangsstrom in jedem Korrelationsfenster beliebig viele Ereignisse konsumieren, d.h. verarbeiten *und aus dem Eingangsstrom löschen* (vgl. dazu Kapitel 3.3.4).

6.1.3 Ausdrucksstärke des Ausführungsmodells

Nachdem nun ein fensterbasiertes Ausführungsmodell für Operatoren eingeführt wurde, stellt sich die Frage, wie ausdrucksstark oder mächtig dieses Modell ist. Anders formuliert: Inwiefern lassen sich Klassen von Situationserkennungen mit diesem Modell tatsächlich umsetzen und wo liegen die Grenzen des Modells? Welche Erkennungsmuster müssen überhaupt unterstützt werden? Um dieser Frage auf den Grund zu gehen, wird in dieser

Arbeit ein „Klassiker“ im Forschungsfeld der Ereignisverarbeitung betrachtet: Die Ereignisverarbeitungssprache *snoop* [CM94], die bereits im Systemmodell in Kapitel 3.3.4 erläutert wurde. Im Folgenden werden diese Parameterkontexte genutzt, um das fensterbasierte Ausführungsmodell bezüglich seiner Ausdrucksstärke zu bewerten. Je mehr verschiedene Arten der Ereignisauswahl mit dem Ausführungsmodell implementierbar sind, desto ausdrucksstärker ist das Modell. Für jeden Parameterkontext aus snoop wird gezeigt, wie er durch das Ausführungsmodell umgesetzt werden kann.

Recent : Es werden immer die neuesten möglichen Ereignisse zur Korrelation herangezogen, sobald eine Korrelation möglich ist. Nach der Korrelation werden die daran teilnehmenden Ereignisse konsumiert.

Abbildungsvorschrift:

1. Öffne ein Korrelationsfenster beim ersten Ereignis σ_{first}, das das Startereignis einer Korrelation sein könnte.
2. Lese so lange Ereignisse ein, bis eine Korrelation möglich ist.
3. Schließe dann das Korrelationsfenster.
4. Ziehe die jeweils neuesten nach der Korrelationsvorschrift in Frage kommenden Ereignisse zur Korrelation heran und konsumiere diese danach.
5. Öffne das nächste Korrelationsfenster beim nächsten auf σ_{first} folgenden Ereignis, das das Startereignis einer Korrelation sein könnte.
6. Beginne wieder bei Schritt 2.

Chronicle : Ereignisse werden nach ihrem Auftreten in chronologischer Reihenfolge zur Korrelation herangezogen. Das bedeutet, dass immer die ältesten verfügbaren Ereignisse korreliert werden. Nach der Korrelation werden die daran teilnehmenden Ereignisse konsumiert.

Abbildungsvorschrift:

1. Öffne ein Korrelationsfenster beim ersten Ereignis σ_{first}, das das Startereignis einer Korrelation sein könnte.
2. Lese so lange Ereignisse ein, bis eine Korrelation möglich ist.
3. Schließe dann das Korrelationsfenster.
4. Ziehe die jeweils ältesten nach der Korrelationsvorschrift in Frage kommenden Ereignisse zur Korrelation heran und konsumiere diese danach.
5. Öffne das nächste Korrelationsfenster beim nächsten auf σ_{first} folgenden Ereignis, das das Startereignis einer Korrelation sein könnte.
6. Beginne wieder bei Schritt 2.

Continous : Auf dem Eingangsstrom werden kontinuierlich Korrelationen durchgeführt, und zwar mit jedem Ereignis, das als erstes Ereignis einer Korrelation in Frage kommt und den ältesten nachfolgenden weiteren für die Korrelation relevanten Ereignissen. Jedes Ereignis kann nur eine Korrelation starten, daher wird das Startereignis konsumiert.

Abbildungsvorschrift:

1. Öffne ein Korrelationsfenster beim ersten Ereignis σ_{first}, das das Startereignis einer Korrelation sein könnte.
2. Lese so lange Ereignisse ein, bis eine Korrelation möglich ist.
3. Schließe dann das Korrelationsfenster.
4. Ziehe neben σ_{first} die jeweils ältesten nach der Korrelationsvorschrift in Frage kommenden Ereignisse zur Korrelation heran und konsumiere dann nur σ_{first}.
5. Öffne das nächste Korrelationsfenster beim nächsten auf σ_{first} folgenden Ereignis, das das Startereignis einer Korrelation sein könnte.
6. Beginne wieder bei Schritt 2.

Cumulative : Alle Ereignisse zwischen dem ersten Ereignis, das eine Korrelation starten kann, und dem ersten Ereignis, mit dem die Korrelation abgeschlossen wird, werden in die Korrelation mit einbezogen. Nach der Korrelation werden alle daran beteiligten Ereignisse konsumiert.

Abbildungsvorschrift:

1. Öffne ein Korrelationsfenster beim ersten Ereignis σ_{first}, das das Startereignis einer Korrelation sein könnte.
2. Lese so lange Ereignisse ein, bis eine Korrelation möglich ist.
3. Schließe dann das Korrelationsfenster.
4. Ziehe alle Ereignisse des Korrelationsfensters zur Korrelation heran.
5. Der Konsum des kompletten Fensters kann vermieden werden: Öffne das nächste Korrelationsfenster beim nächsten auf das Endereignis folgenden potentiellen Startereignis.
6. Beginne wieder bei Schritt 2.

Damit können alle snoop-Parameterkontexte in dem Ausführungsmodell abgebildet werden. Das Modell ist also mächtig genug, um in snoop definierte Korrelationen zu implementieren. Beispiele zu den Parameterkontexten werden in der Veröffentlichung von S. Chakravarthy und D. Mishra [CM94] gegeben und werden daher an dieser Stelle nicht detailliert ausgeführt.

6.2 Reproduzierbarkeit der Ereigniserzeugung

Die Idee, die hinter dem Ausführungsmodell und der Wiederherstellung von Ereignisströmen steckt, lässt sich folgendermaßen zusammenfassen: Wenn ein Operator ausfällt, muss er in seinem Zustand wiederhergestellt werden. Ohne persistente Sicherung durch Checkpoints stehen die nötigen Informationen allerdings nicht direkt zur Verfügung, sondern müssen aus den Ausgangsströmen der Vorgängeroperatoren im Operatorgraphen wiederhergestellt werden. Die Vorgängeroperatoren halten dazu einen Teil ihrer Ausgangsströme zur wiederholten Verarbeitung im neu gestarteten Operator bereit. Wenn es gelingt, aus diesen Teilstücken als Eingangsstrom den ursprünglichen Ausgangsstrom des Operators wiederherzustellen und fehlerlos an der Stelle fortzusetzen, an der der Operator ausgefallen ist, kann er so weiterarbeiten, als wäre er nie ausgefallen und hätte nie alle volatilen Zustandsinformationen verloren. Insbesondere kann so der Zustand des CEP-Systems auch nach dem Ausfall mehrerer Operatoren in Reihe wiederhergestellt werden: Die Ereignisströme werden sukzessive von Vorgängern zu Nachfolgern reproduziert, bis alle ausgefallenen Operatoren wiederhergestellt sind. Durch die reproduzierbare Öffnung und Schließung der Korrelationsfenster und der ebenfalls reproduzierbaren Berechnung des ausgehenden Ereignisses in einem Korrelationsfenster wird sichergestellt, dass auf demselben Eingangsstrom ein Operator immer denselben Ausgangsstrom berechnet. Bisher sind die Fenster jedoch noch voneinander abhängig: Die Berechnung des Startereignisses des Folgefensters geschieht innerhalb des Vorgängerfensters. Außerdem kann die Konsumierung von Ereignissen den Eingangsstrom und damit auch die Berechnungen in folgenden Fenstern beeinflussen. Diese Abhängigkeiten müssen aufgelöst werden, damit zur Wiederherstellung eines Teilstroms von Ausgangsereignissen nicht der gesamte Eingangsstrom I neu bearbeitet werden muss.

Die Wiederherstellung der Startereignisse ist dabei der einfachere Teil: Ein Teilstrom T, der mit einem Ereignis σ_s beginnt, kann reproduziert werden, wenn die wiederholte Verarbeitung im Startereignis des Korrelationsfensters w_s beginnt, in dem σ_s ursprünglich erzeugt wurde, d.h. ein eindeutiger Verweis auf dieses Startereignisses muss gespeichert werden. Die Verarbeitung muss dabei auf einem Eingangsstrom geschehen, der mit dem Eingangsstrom zum Zeitpunkt der ersten Ausführung von w_s identisch ist.

Der Effekt von Konsumoperationen auf die Reproduzierbarkeit von Ereignisströmen wird im folgenden Abschnitt untersucht. In Abschnitt 6.2.2 werden die hier aufgeführten Eigenschaften des Ausführungsmodells formalisiert zusammengefasst.

6.2.1 Wiederherstellung des Eingangsstroms bei Konsumoperationen

Wenn ein Konsum von Ereignissen (wie in Kapitel 3.3.4 eingeführt) in einem Korrelationsfenster stattfindet, der die Ereignisse in einem darauf folgenden Korrelationsfenster beeinflusst, d.h. wenn Ereignisse konsumiert werden, die im Eingangsstrom *hinter* dem Startereignis eines Folgefensters liegen, dann entsteht dadurch eine Abhängigkeit zwischen dem Fenster, in dem Ereignisse konsumiert wurden und den betroffenen Folgefenstern. Wie in Abschnitt

6.2 erläutert wurde, muss jede Abhängigkeit zwischen den Fenstern aufgelöst werden, damit ein effizientes Rollback-Recovery-Verfahren zur Wiederherstellung von Ereignisströmen erreicht werden kann.

Um die Erzeugung eines Ereignisses σ_o in einem Operator ω zu wiederholen, muss die Korrelationsfunktion f auf dasselbe Fenster w_o von Ereignissen aus demselben Eingangsstrom I angewendet werden. Das heißt, dass sowohl das Startereignis σ_s und das Endereignis σ_e bei der Wiederholung mit der ursprünglichen Ausführung von f übereinstimmen müssen, als auch alle Ereignisse, die zwischen σ_s und σ_e liegen. Wenn in einem Vorgängerfenster w_p, das mit w_o überlappt, Ereignisse zwischen σ_s und σ_e aus dem Eingangsstrom I konsumiert wurden, muss bei einer Wiederholung von w_o zuvor dieser Konsum von Ereignissen ebenfalls wiederholt werden, damit das Ergebnis der erneuten Anwendung von f auf das Fenster mit dem Ergebnis der ursprünglichen Anwendung übereinstimmt.

Beispiel Ein Operator hat einen Eingangsstrom I, der sich aus Ereignissen der Typen A, B und C zusammensetzt. Es soll jeweils eine (nicht notwendigerweise zusammenhängende) Sequenz aus A, B und C detektiert und konsumiert werden und für jede detektierte Sequenz ein Ausgangsereignis von Typ D erzeugt werden. Dabei werden jeweils die ältesten Vorkommen der jeweiligen Ereignistypen miteinander korreliert (*Chronicle*-Parameterkontext aus snoop [CM94]). Beim Eingangsstrom I

$$B_1\ B_2\ C_3\ |A_4\ A_5\ C_6\ C_7\ B_8\ B_9\ C_{10}|\ C_{11}\ ...$$

(die tiefgestellten Zahlen sind ein Index zur eindeutigen Referenzierung eines Ereignisses in der Beschreibung dieses Beispiels) würden die Ereignisse 4, 8 und 10 im Fenster w_p (das sich von Ereignis 4 bis Ereignis 10 streckt) zu einem Ereignis D_p verarbeitet und konsumiert. Der Eingangsstrom I würde dadurch verändert und sähe danach so aus:

$$B_1\ B_2\ C_3\ \_\_\ |A_5\ C_6\ C_7\ \_\_\ B_9\ \_\_\ C_{11}|\ ...$$

Das darauffolgende Korrelationsfenster w_o geht von Ereignis 5 bis Ereignis 11 und verarbeitet und konsumiert die Ereignisse 5, 9 und 11 zum ausgehenden Ereignis D_o. Was würde passieren, wenn nach einer Wiederherstellung des Operators die Verarbeitung des ursprünglichen Eingangsstroms I ab Ereignis 5 wiederholt würde, also ohne erneute Ausführung der Konsumoperationen des Korrelationsfensters w_p?

$$|A_5\ C_6\ C_7\ B_8\ B_9\ C_{10}|\ C_{11}\ ...$$

Das Fenster w_o, das bei Ereignis 5 startet, würde sich bis Ereignis 10 ziehen, und die Ereignisse 5, 8 und 10 würden zu einem Ausgangsereignis D_o' verarbeitet, welches vom ursprünglichen Ereignis D_o abweicht. Auf diese Weise ließe sich der Teil des Ausgangsstroms O ab Ereignis D_o aus dem ursprünglichen Eingangsstrom I nicht wiederherstellen. Wenn nur der ursprüngliche Eingangsstrom erhalten ist, muss zur Wiederherstellung eines Teils des Ausgangsstroms

der gesamte Eingangsstrom neu verarbeitet werden. Dies ist in der Praxis natürlich keine Option. Besser ist es, den Zustand des Eingangsstroms mit seinen Veränderungen durch die Konsumoperationen zu sichern und im Falle einer Wiederherstellung des Operators diesem zur Verfügung zu stellen. Wie genau diese Sicherung geschieht, wird im Abschnitt 6.4 über den Ablauf der Ereignisverarbeitung im Normalbetrieb näher betrachtet.

6.2.2 Eigenschaften des Ausführungsmodells bezüglich der Zustände von Operatoren

Durch die kontextfreie Definition des Prädikats P_{first} wird sichergestellt, dass ein Ereignis unabhängig von anderen Ereignissen bezüglich dieses Prädikats ausgewertet wird. Das Prädikat P_{last} berücksichtigt nur Ereignisse des aktuellen Korrelationsfensters, d.h. auch dieses Prädikat bringt keine Abhängigkeit zwischen zwei aufeinanderfolgenden Fenstern mit sich. Abgesehen von den Konsumoperationen, die in Abschnitt 6.2.1 untersucht wurden, sind die Fenster also unabhängig voneinander und es gilt folgender Satz:

Satz 6.2.1 (Zustandslosigkeit zwischen zwei aufeinanderfolgenden Fenstern)**.** *Zwischen der Ausführung zweier aufeinanderfolgender Korrelationsfenster muss abgesehen von den Konsumoperationen keine Zustandsinformation gespeichert werden.*

Dies führt direkt zu folgendem Satz:

Satz 6.2.2 (Ereigniserzeugung nur anhand des aktuellen Fensters)**.** *Ausgehende Ereignisse werden nur anhand der Ereignisse im aktuellen Fenster erzeugt. Ereignisse, die vor der Öffnung des aktuellen Fensters liegen, beeinflussen die Ereigniserzeugung nicht.*

Dies führt zur folgenden Schlussfolgerung:

Schlussfolgerung 6.2.1 (Wiederherstellung von Ereignisströmen)**.** *Um ein ausgehendes Ereignis σ_o in einem Operator ω wiederherzustellen, müssen im Operator ausschließlich die Ereignisse vorliegen, die in dem Korrelationsfenster liegen, das bei der Erzeugung von σ_o das aktuelle Korrelationsfenster gewesen ist.*

Dies ist die entscheidende Eigenschaft für das folgende Wiederherstellungsverfahren und somit die Motivation zur Einführung des fensterbasierten Ausführungsmodells der Operatoren.

6.2.3 Zeitstempel von Ereignissen

In Kapitel 3.2.2 wurde die genaue Gestaltung der Zeitstempel recht offen gelassen. Für das Ausführungsmodell in dieser Arbeit ist es im Allgemeinen auch unerheblich, was genau der Zeitstempel ausdrückt, solange folgende Eigenschaften gelten:

Eigenschaft 6.2.1 (Reproduzierbare Berechnung der Zeitstempel)**.** *Zeitstempel müssen unabhängig vom Zeitpunkt der Erzeugung eines Ereignisses in einem Operator sein, sodass sie im Fall einer wiederholten Erzeugung nicht vom ursprünglichen Wert abweichen.*

Eigenschaft 6.2.2 (Aufsteigende Zeitstempel im Ausgangsstrom)**.** *Der Ausgangsstrom O in einem Operator muss gleichbleibende oder aufsteigende Zeitstempel besitzen.*

Eigenschaft 6.2.3 (Zeitstempel von Startereignissen)**.** *Bei zwei im Ausgangsstrom aufeinanderfolgenden, in einem Operatoren erzeugten Ausgangsereignissen σ_i und σ_{i+1} gilt: Das Startereignis des Korrelationsfensters zur Erzeugung von σ_i hat keinen größeren Zeitstempel als das entsprechende Startereignis von σ_{i+1}.*

6.2.3.1 Begründung der Notwendigkeit der Eigenschaften

Eigenschaft 6.2.1 ist notwendig: Im Falle einer wiederholten Erzeugung, d.h. wenn ein Ereignisstrom wiederhergestellt werden soll, ist die Echtzeit der Ereigniserzeugung natürlich eine andere als bei der ursprünglichen Erzeugung der Ereignisse. Ereignisse, die aus wiederhergestellten Ereignisströmen kommen, können dadurch die Verarbeitung in nachfolgenden Operatoren durcheinanderbringen, indem sie ursprünglich später aufgetretene Ereignisse „überholen“ und so die Ergebnisse vom Normalbetrieb abweichen.

Eigenschaft 6.2.2 ist notwendig: Da der Eingangsstrom eines Nachfolgeoperators nach Zeitstempeln aufsteigend geordnet ist, muss zur Wiederherstellung eines Teils dieses Eingangsstrom nur ein zeitlich begrenzter Teil der Ausgangsströme der Vorgängeroperatoren vorgehalten werden. Diese Begrenzung lässt sich effizient implementieren, wenn auch die Ausgangsströme nach Zeitstempeln aufsteigend geordnet sind. Details zur Wiederherstellung von Ereignisströmen werden in Abschnitt 6.5 erläutert. Um im in Kapitel 6.1 beschriebenen Ausführungsmodell einen Ausgangsstrom mit aufsteigenden Zeitstempeln zu generieren, muss die Berechnung der Zeitstempel bei der Erzeugung von Ereignissen in aufeinanderfolgenden Korrelationsfenstern zu aufsteigenden oder gleichbleibenden, aber keinesfalls zu absteigenden Zeitstempeln führen.

Eigenschaft 6.2.3 ist notwendig: Die Wiederherstellung von ausgehenden Ereignissen in einem ausgefallenen Operator soll dadurch möglich werden, dass die eingehenden Ereignisströme neu eingelesen und verarbeitet werden. Wenn für ein Ereignis der Punkt in den eingehenden Strömen definiert ist, ab dem diese neu verarbeitet werden müssen, dürfen nicht für ein späteres Ereignis plötzlich frühere Punkte in den Eingangsströmen notwendig sein.

Bemerkung: Eine Umsortierung erzeugter Ereignisse zur Herstellung von Eigenschaft 6.2.2 kommt also nicht in Frage, da dadurch Eigenschaft 6.2.3 verletzt werden kann.

6.2.3.2 Mögliche Zeitstempeldefinition

Ohne Beschränkung der Allgemeinheit wird im Folgenden eine Implementierung von Zeitstempeln und einer Ordnung darüber eingeführt, die für verteilte CEP-Systeme sinnvoll ist und für die Evaluation als Beispielimplementierung genutzt wird.

Definition 6.1 (Zeitstempel)**.** *Ein Zeitstempel τ eines komplexen Ereignisses σ ist ein Zeitraum, der sich zwischen zwei Zeitpunkten t_{first} und t_{last} aufspannt. t_{first} ist der Zeitpunkt, zu dem das erste einfache Ereignis in einer Ereignisquelle aufgetreten ist, das in die Korrelation von σ eingeflossen ist. Entsprechend ist t_{last} der Zeitpunkt, zu dem das letzte einfache Ereignis in einer Ereignisquelle aufgetreten ist, das in die Korrelation von σ eingeflossen ist.*

Damit ist t_{last} der Zeitpunkt des eigentlichen Auftretens der durch das komplexe Ereignis angezeigten Situation und t_{first} der Zeitpunkt des Beginns der Situationserkennung. In unserem Ausführungsmodell für Operatoren (siehe Kapitel 6.1) werden Ereignisse sequentiell auf einem Ereignisstrom korreliert, wobei ein Korrelationsfenster die Ereignisse bestimmt, die in der Korrelation eines neuen Ereignisses berücksichtigt werden. In diesem Sinne berechnet sich der Zeitstempel des neu erzeugten Ereignisses aus dem kleinsten t_{first} und dem größten t_{last} aller Ereignisse im entsprechenden Korrelationsfenster. Da die Startereignisse von Korrelationsfenstern im sequenzierten Eingangsstrom sich nur nach hinten bewegen können, die Endereignisse aber unabhängig von den Endereignissen anderer Fenster bestimmt werden (also nach vorne und hinten schwanken können), gibt es nur eine sinnvolle Ordnung über die Zeitstempel:

Definition 6.2 (Ordnung über Zeitstempel)**.** *Zeitstempel werden über die Startzeitpunkte t_{first} geordnet. Das heißt, für zwei Zeitstempel τ_1 und τ_2 gilt:*
$\tau_1 > \tau_2 \Leftrightarrow t_{first_1} > t_{first_2}$

Bei Gleichheit von t_{first} gilt weder $\tau_1 > \tau_2$ noch $\tau_1 < \tau_2$, sondern $\tau_1 \doteq \tau_2$.

Mit dieser Ordnung ist ein Ausgangsstrom, der in einem Operator durch das fensterbasierte Ausführungsmodell erzeugt wird, automatisch nach Zeitstempeln aufsteigend geordnet, da die Startereignisse von Korrelationsfenstern immer zeitlich aufsteigend sind: Eigenschaft 6.2.2 wird also erfüllt. Durch die Berechnung der Zeitstempel aus den Zeitstempeln der korrelierten Ereignisse ist auch Eigenschaft 6.2.1 erfüllt. Die Erfüllung von Eigenschaft 6.2.3 ergibt sich aus der Erfüllung von Eigenschaft 6.2.2 in den beiden Eingangsströmen und der Tatsache, dass Startereignisse von Korrelationsfenstern sich nur nach hinten bewegen können (vgl. Abschnitt 6.1.1.3).

6.3 Voraussetzungen zur Wiederherstellung ausgefallener Operatoren

Im CEP-System werden Ereignisse in den Operatoren miteinander korreliert und weiterverarbeitet, bis sie schließlich an einen oder mehrere Konsumenten ausgeliefert werden.

Ein Ereignis, das nach möglicherweise mehrstufiger Korrelation schließlich an einen Konsumenten ausgeliefert wird, hat in der Zwischenzeit eine Menge an Informationen aus „niederwertigen“ Ereignissen in sich aufgenommen. Letztendlich lässt sich der Informationsgehalt eines an einen Konsumenten ausgelieferten Ereignisses schrittweise auf eine Menge von „Zwischen-Ereignissen“ und schließlich vollständig auf „primitive“ Ereignisse aus den Ereignisquellen herunterbrechen. Im Folgenden bezeichnen wir ein ausgehendes Ereignis, das an einen Konsumenten ausgeliefert werden kann, als $\sigma_{consumer}$. In jedem der Zwischenschritte, die in der Verarbeitung von Ereignissen aus den Quellen bis zu der Auslieferung an die Konsumenten liegen, wird zumindest indirekt an einem oder mehreren $\sigma_{consumer}$ gearbeitet. Jeder Operator, der einen dieser Zwischenschritte ausführt, hat zu jedem Zeitpunkt einen ganz bestimmten Zustand bezüglich der Erzeugung der $\sigma_{consumer}$-Ereignisse, auch wenn er nicht direkt ein solches Ereignis erzeugt. Die eigentlichen $\sigma_{consumer}$-Ereignisse werden unmittelbar nur von Operatoren erzeugt, die direkte Vorgänger von Konsumenten im Operatorgraphen sind. Die Information, welche Zwischenergebnisse (und schließlich, welche einfachen Ereignisse aus den Ereignisquellen) zur Korrelation der sich aktuell im Aufbau befindlichen $\sigma_{consumer}$-Ereignisse eingebunden wurden, muss stromabwärts von den letzten Operatoren zu den ersten Operatoren weitergegeben werden. Dazu dienen die Bestätigungsnachrichten, die hier kurz auch als ACK gekennzeichnet sind. Jedes Ereignis, das direkt für den Bau eines $\sigma_{consumer}$-Ereignisses, d.h. auf der obersten Verarbeitungsebene, gebraucht wird, muss solange verfügbar oder zumindest wiederherstellbar sein, bis das entsprechende Ereignis sicher an den Konsumenten ausgeliefert wurde und es damit nicht mehr benötigt wird. Diese oberste Verarbeitungsebene nennen wir im Folgenden Ebene 0. Wenn wir nun Ereignisse aus Ebene 0 wiederherstellen müssen, benötigen wir dafür Ereignisse aus Ebene 1, für Ereignisse aus Ebene 1 benötigen wir Ereignisse aus Ebene 2, und so weiter.

6.3.1 Das Wiederherstellungsproblem

An das Wiederherstellungsverfahren ist die Anforderung gestellt, dass beim gleichzeitigen Ausfall von F Operatoren eine Stabilisierung der Ereignisströme erreicht werden kann, die dazu führt, dass in der Ebene der Konsumenten (sozusagen Ebene -1) die eintreffenden Ereignisse in keiner Weise vom Normalbetrieb ohne Ausfälle abweichen. Im schlechtesten Fall im Sinne der Wiederherstellung von Ereignisströmen fallen F Operatoren in F zusammenhängenden Verarbeitungsebenen aus, d.h. die ausgefallenen Operatoren stehen alle in einer strikten Reihe von Vorgänger- und Nachfolgerbeziehungen. Um den Operator der obersten ausgefallenen Ebene wiederherzustellen, benötigt man Ereignisse aus der zweithöchsten Ebene. Wenn diese Ebene durch einen Ausfall ebenfalls weggebrochen ist, benötigt man Ereignisse aus der dritthöchsten Ebene, um zunächst die zweithöchste Ebene wiederherzustellen, und so weiter. Jede Ebene muss in der Form wiederhergestellt werden, dass für die höherliegenden Ebenen der Ausfall der Ebene nicht ersichtlich ist. Die Wiederherstellung einer Ebene bzw. der Operatoren einer Ebene nach einem Ausfall ist dann gegeben, wenn alle Ereignisse im Ausgangsstrom wiederhergestellt sind, die für ein noch unvollständiges Ereignis $\sigma_{consumer}$ benötigt werden, und der Operator mit dem letzten unvollständig berechneten Korrelationsfenster die Berechnung fortsetzt. Wenn dies gegeben ist, ist für die nächsthöhere

Ebene der Ausfall transparent, da die wiederhergestellten Operatoren ihre Arbeit an genau der Stelle fortsetzen, an der sie zuvor ausgefallen sind.

6.3.1.1 Definition des Problems

Ein Operator ist dann wiederherstellbar, wenn nach dem Verlust seiner volatiler Zustandsinformationen diese durch Informationen wiederhergestellt werden können, die in anderen Stellen im CEP-System gespeichert sind. Der Zustand eines Operators ist dabei wie folgt definiert:

Definition 6.3 (Zustand eines Operators)**.** *Der Zustand eines Operators ω zu einem bestimmten Zeitpunkt zeichnet sich aus durch:*

- *Sein Ausgangslog L(ω) von ausgehenden Ereignissen*
- *Seinen Eingangsstrom I (inklusive der konsumierten Ereignisse zum aktuellen Zeitpunkt)*
- *Dem aktuellen Berechnungszustand der Korrelationsfunktion f*

Die Herausforderung besteht nun darin, diese Zustandsinformationen möglichst klein zu halten. Das größte Problem besteht dabei im aktuellen Stand der Berechnung der Korrelationsfunktion f: Um den genauen Zustand der Berechnung festzuhalten, wäre im Allgemeinen ein teilweises Speicherabbild unerlässlich, um alle Variablen, den Control Stack, etc. zu sichern, wie es in vielen Rollback-Recovery-Verfahren ja auch durchgeführt wird. An dieser Stelle kommt das fensterbasierte Ausführungsmodell der Operatoren ins Spiel. Immer zu Beginn eines Korrelationsfensters ist der Berechnungszustand von f nämlich *leer* und man muss all die Zustandsinformationen, die in der Ausführung einer beliebigen Routine anfallen können, nicht speichern. Stattdessen genügt es, das Startereignis auf dem Eingangsstrom zu speichern, die Korrelationsfunktion (die ja persistent als Routine im Operator gespeichert ist) wird dann auf den Ereignissen des Eingangsstrom ab dem gesicherten Punkt ausgeführt. Zu bestimmten Zeiten wird jeder Operator seinen Zustand in einem sog. *Sicherungspunkt* festhalten und diesen Sicherungspunkt dann an andere Operatoren übertragen. Ein Sicherungspunkt ist wie folgt definiert:

Definition 6.4 (Sicherungspunkt)**.** *Ein Sicherungspunkt eines Operators enthält alle Informationen, um mithilfe derselben Eingangsströme einen Operator in genau den Zustand zu bringen, den er zum für die Erstellung des Sicherungspunktes relevanten Zeitpunkt hatte. Der relevante Zeitpunkt wird auf Basis der Bestätigungsnachrichten der Vorgängeroperatoren bestimmt.*

Was genau der Sicherungspunkt beinhaltet, wird später in Abschnitt 6.4 untersucht. Jedenfalls muss er genügend Informationen bereitstellen, um aus einer Reihe von Eingangsströmen, die eventuell auch an verschiedenen Sicherungspunkten starten können, das richtige Korrelationsfenster zu öffnen und die Korrelationsfunktion f darauf auszuführen. Zudem müssen diese Eingangsströme auch noch durch die Wiederholung der Konsumoperationen auf den Stand gebracht werden, der zum Zeitpunkt der Öffnung des ursprünglichen Korrelationsfensters gültig war. Liegen diese Informationen und die ursprünglichen Eingangsströme (ab

den relevanten ersten Ereignissen) vor, kann ein Operator wiederhergestellt werden. Die Wiederherstellung ist wie folgt definiert:

Definition 6.5 (Wiederherstellung eines Operators)**.** *Ein Operator ist nach dem Verlust seiner volatilen Zustandsinformationen genau dann wiederhergestellt, wenn sein Zustand dem letzten durch einen Sicherungspunkt bestätigten Zustand vor dem Ausfall des Operators entspricht.*

Der Operator wird also bezüglich eines Sicherungspunktes wiederhergestellt. Im Idealfall geschieht dies bezüglich des aktuellsten Sicherungspunktes, insofern dieser auch an alle relevanten Vorgängeroperatoren übertragen wurde. Welche Vorgängeroperatoren für welchen Fehlerfall den Sicherungspunkt speichern müssen, wird später in Abschnitt 6.4 genauer untersucht.

6.3.2 Invarianten für die Problemlösung

Bevor eine Lösung des Wiederherstellungsproblems Schritt für Schritt eingeführt wird, werden in diesem Abschnitt zwei Invarianten für das CEP-System aufgeführt. Diese Invarianten müssen jederzeit gültig sein. Anhand der Invarianten wird dann in Abschnitt 6.3.2.1 die Korrektheit des Systems bewiesen, und später in Abschnitt 6.5.4 die Gültigkeit der Invarianten im Wiederherstellungsverfahren bewiesen. Die Invarianten sind also eine Abstraktion, die in der Analyse des Systems hilfreich ist.

Invariante 6.3.1 (Wiederherstellbarkeit von Operatoren)**.** *Im CEP-System müssen jederzeit alle notwendigen Informationen vorhanden sein, um in der Fehlersituation „Ausfall von F Operatoren" jeden Operator bezüglich eines Sicherungspunkts wiederherstellen zu können.*

Diese Invariante besagt, dass jeder Operator in der definierten Fehlersituation „Ausfall von F Operatoren" wiederherstellbar sein muss. Das impliziert, dass die zur Wiederherstellung notwendigen Information genügend oft repliziert sein müssen, damit auch ein zeitgleicher Ausfall von F Operatoren die Informationen nicht aus dem System entfernen kann.

Invariante 6.3.2 (Zeitpunkte von Sicherungspunkten)**.** *Sicherungspunkte müssen so gewählt werden, dass sie sich auf Zeitpunkte beziehen, zu denen es der Zustand der Operatoren erlaubt, alle noch nicht von allen Konsumenten empfangenen $\sigma_{consumer}$-Ereignisse zu erzeugen.*

Diese Invariante besagt, dass jedes $\sigma_{consumer}$-Ereignis auch beim gleichzeitigen Ausfall von F Operatoren erzeugbar bleiben muss.

6.3.2.1 Beweis der Einhaltung der Korrektheitsanforderungen

Wenn die Invarianten 6.3.1 und 6.3.2 gelten, hat das folgende Auswirkungen:

- Durch Ausfälle können keine Ereignisse erzeugt werden, die es nicht auch ohne die Ausfälle gegeben hätte (falsche Ereignisse), denn Operatoren erzeugen nach der Wiederherstellung genau dieselben Ausgangsströme, die sie auch ohne Ausfall erzeugt hätten. Dies folgt aus Invariante 6.3.1 und der Definition von Sicherungspunkten in Abschnitt 6.3.1.1.
- Dass durch Ausfälle keine Ereignisse verlorengehen können, folgt direkt aus Invariante 6.3.2.
- Die Erhaltung der Reihenfolge der Ereignisse folgt aus der Tatsache, dass Ereignisse nicht umsortiert werden können und die Kommunikationskanäle Ereignisse in der Reihenfolge der Versendung beim Empfänger ausliefern (vgl. Kapitel 3.4).

Das System läuft also bei Einhaltung der Invarianten korrekt nach den Korrektheitsanforderungen aus Kapitel 4.2.3.

6.4 Verwaltung von Zustandsinformationen im Normalbetrieb

Der Normalbetrieb ist der Betrieb des verteilten CEP-Systems, wenn keine Operatoren ausfallen. Ausgehend vom Normalbetrieb wird dann die Reaktion des Systems auf Ausfälle von Operatoren untersucht. Im Normalbetrieb werden Informationen gesammelt, verteilt und verwaltet, die für eine Wiederherstellung von Operatoren benötigt werden. Jeder Operator führt einen Algorithmus zur Verwaltung von Logs und Sicherungspunkten aus, die er in seinem *volatilen Speicher* sichert. Das heißt insbesondere, dass ein Operator *keine Statusinformationen persistent speichert*, was den wesentlichen Unterschied zu klassischen Rollback-Recovery-Verfahren ausmacht (vgl. dazu Kapitel 5). Zur Verwaltung dieser Informationen tauschen sich die Operatoren gegenseitig Statusnachrichten aus. Diese sind nicht Teil der eigentlichen Ereignisverarbeitung, vielmehr bilden die Operatoren untereinander ein von der verteilten Ereignisverarbeitung getrenntes Netz zum Austausch von Statusinformationen. Statusnachrichten tauchen also beispielsweise nicht in der Queue von eingehenden Ereignissen auf, sie können direkt zwischen Operatoren ausgetauscht werden und werden auch mit höherer Priorität als die „normale“ Ereignisverarbeitung verarbeitet.

6.4.1 Einfaches System: Quelle, Operator, Konsument

6.4.1.1 Ereignisströme

In Abb. 6.1 ist das einfachste verteilte CEP-System im Sinne des Systemmodells in dieser Arbeit dargestellt. Eine Ereignisquelle S ist mit einem Operator ω verbunden, der wiederum mit einem Ereigniskonsumenten C verbunden ist. Von S geht ein Ereignisstrom von einfachen Ereignissen des Typs σ_p nach ω, wo die Ereignisse miteinander korreliert werden. Als Ergebnis dieser Verarbeitung von Ereignissen des Typs σ_p erzeugt ω ausgehende Ereignisse des Typs σ_c. Diese ausgehenden Ereignisse werden in einer Sequenz im Ausgangsstrom angeordnet und inkrementell mit Sequenznummern ρ_c versehen. Schließlich werden sie an

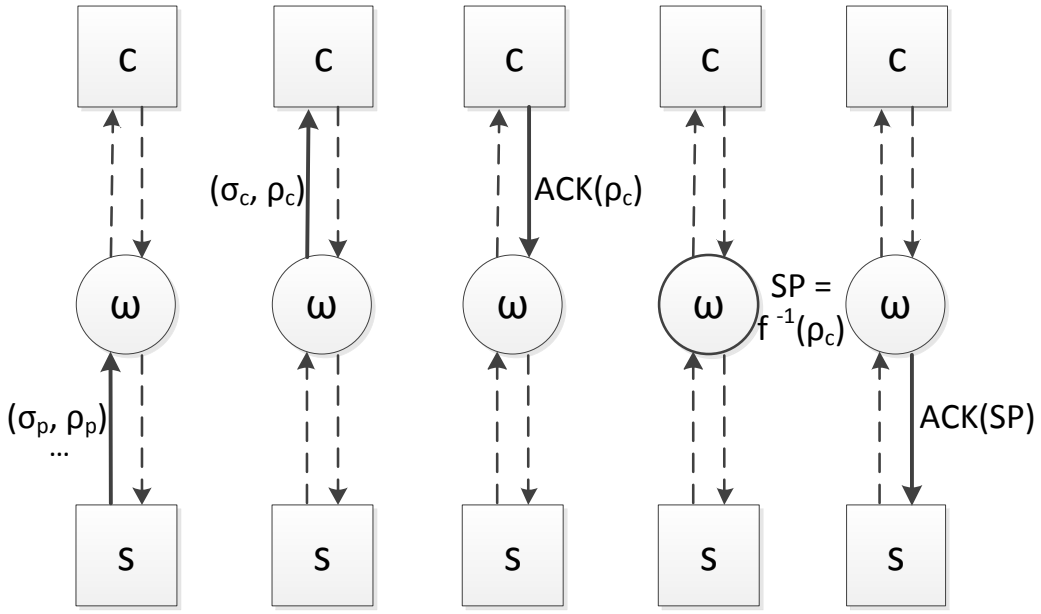

Abbildung 6.1: Versendung und Bestätigung von Ereignissen im einfachsten CEP-System.

den Ereigniskonsumenten C ausgeliefert. Wie an diesem Vorgang zu erkennen ist, fließen die Ereignisse immer von der Ereignisquelle über den Operator zum Ereigniskonsumenten. Wir nennen diese Fließrichtung der Ereignisströme „stromabwärts“.

6.4.1.2 Bestätigung durch den Konsumenten

Sobald Ereignisse des Typs σ_c den Konsumenten erreichen, kann dieser den Empfang beim Operator bestätigen. Dabei muss zur Lastminderung nicht jedes Ereignis einzeln bestätigt werden, es ist auch denkbar, dass nur in bestimmten Intervallen Ereignisse bestätigt werden und durch eine solche Bestätigung alle Vorgängerereignisse im Ereignisstrom mit bestätigt werden. Durch die Eigenschaften der Abstraktion der „Best-Effort-Verbindungen“ ist sichergestellt, dass diese Annahme getroffen werden kann (vgl. Kapitel 3.4). Die Bestätigung des Konsumenten ACK(ρ_c) enthält die Sequenznummer ρ_c des bestätigten Ereignisses.

Wenn ein Ereignis vom Konsumenten bestätigt wurde, heißt das, dass dieses Ereignis im CEP-System nicht mehr benötigt wird. Es wurde erfolgreich ausgeliefert und hat damit den Zuständigkeitsbereich des CEP-Systems verlassen. Das bestätigte Ereignis muss also weder im Ausgangslog von ω vorgehalten werden, noch müssen an irgendeiner anderen Stelle Ereignisse vorgehalten werden, die irgendeine Vorstufe zur erneuten Erzeugung des Ereignisses darstellen. Diese Information muss *im gesamten CEP-System* verteilt werden, und zwar „stromaufwärts“ vom Konsumenten bis zur Quelle. Dazu dienen die Bestätigungsnachrichten. Der Konsument bestätigt wie erwähnt den Empfang direkt bei ω über eine Nachricht

ACK mit der Sequenznummer des bestätigten Ereignisses. Bei ω muss diese Bestätigung für die Vorgänger im Operatorgraphen in Sequenznummern der entsprechenden Ströme, die ω als Eingangsströme zur Erzeugung des bestätigten Ausgangsereignisses gedient hatten, umgerechnet werden. Dazu ruft ω die *Umkehrfunktion* f^{-1} auf.

6.4.1.3 Umkehrfunktion f^{-1}

Um die Umkehrfunktion f^{-1} zu verstehen, muss man zunächst die Korrelationsfunktion f betrachten. Die Funktion $f\colon \mathrm{I} \rightarrow \mathrm{O}$ bildet eine Menge von Ereignissen (genauer: eine durch ein Korrelationsfenster begrenzte Menge von Ereignissen) aus dem Eingangsstrom auf ein ausgehendes Ereignis ab. Sie wird auf aufeinanderfolgenden Korrelationsfenstern ausgeführt und erzeugt so eine Menge von Ausgangsereignissen, die im Ausgangsstrom O sequenziert werden. Die Umkehrfunktion bildet mathematisch gesehen entsprechend ein ausgehendes Ereignis auf „sein“ Fenster im Eingangsstrom ab, also $f^{-1}\colon \mathrm{O} \rightarrow \mathrm{I}$. Wenn ein Operator eine Sequenznummer eines ausgehenden Ereignisses σ_o bestätigt bekommt, kann er mithilfe dieser Sequenznummer in seinem Ausgangslog $\mathrm{L}(\omega)$ das entsprechende Ereignis ermitteln. Wären in diesem Ereignis alle Ereignisse des Korrelationsfensters gespeichert, könnte $f^{-1}(\sigma_o)$ anhand dieser Informationen in σ_o ermittelt werden. Doch im Prinzip ist es unnötig, tatsächlich das gesamte Korrelationsfenster von σ_o zu rekonstruieren. Wirklich von Belang ist das *Startereignis* σ_s des Fensters, denn es ist sicher, dass *alle Ereignisse, die im Eingangsstrom vor dem Startereignis des Korrelationsfensters eines bestätigten Ereignisses liegen, nicht mehr benötigt werden*. Eigentlich werden auch alle Ereignisse *bis zum Startereignis des* ***nächsten*** *Korrelationsfensters* nicht mehr benötigt, doch es ist nicht immer möglich, dieses nächste Startereignis zu bestimmen. Daher bleibt, um ein immer gleiches Verhalten des Systems zu garantieren, nur die Freigabe aller Ereignisse vor dem Startereignis des Korrelationsfensters. Die Umkehrfunktion f^{-1} kann vereinfacht werden zu einer Funktion $f^{-1}\colon \sigma \rightarrow \sigma$, die ein Ereignis σ_o des Ausgangsstroms auf das Ereignis σ_s im Eingangsstrom abbildet, das das Startereignis des Korrelationsfensters gewesen ist, das zur Erzeugung von σ_o führte. Wird der Ereignistyp und die Sequenznummer von σ_s direkt in σ_o gespeichert, kann anhand dieser Informationen im Eingangsstrom I das entsprechende Ereignis ermittelt und so die Funktion f^{-1} im Operator implementiert werden. Wenn ein Operator mehr als einen Eingangsstrom hat, ist die Ermittlung von f^{-1} etwas komplexer. Für den Moment genügt aber die Ermittlung dieses einen Ereignisses σ_s aus dem Eingangsstrom.

Die Sequenznummer von σ_s wird zum Sicherungspunkt von ω hinzugefügt.

6.4.1.4 Konsumoperationen

Zusätzlich zu den Informationen über das Korrelationsfenster muss auch der Zustand des Eingangsstroms zur Zeit des Öffnens des Korrelationsfensters gespeichert werden, wenn im Ausführungsmodell der Operatoren Konsumoperationen auf dem Eingangsstrom erlaubt sind. Durch die reproduzierbare Sequenzierung der Eingangsströme (vgl. Kapitel 3.3.3) im Operator kann der Eingangsstrom grundsätzlich über die Ausgangslogs $\mathrm{L}(\omega_{predecessors})$

der Vorgängerknoten wiederhergestellt werden. Durch Konsumoperationen kann dieser ursprüngliche Eingangsstrom aber bis zum Erreichen des Korrelationsfensters verändert worden sein. Daher müssen alle Konsumoperationen, die in jedem Korrelationsfenster durchgeführt wurden, in einer Tabelle gespeichert werden (Tabelle 6.1).

out_id	consumed_events
1	A1, A3, etc...
2	A2, A5, etc...

Tabelle 6.1: Tabelle der Konsumoperationen

Die Spalte `out_id` beinhaltet aufsteigend die Sequenznummer des Korrelationsfensters (diese stimmt mit der Sequenznummer des darin erzeugten Ereignisses überein). Die Spalte `consumed_events` listet die Ereignisse auf, die im entsprechenden Korrelationsfenster konsumiert wurden. In Sicherungspunkt fügt ω sämtliche Konsumoperationen bis einschließlich der Zeile mit der Sequenznummer von $w_{recover}$ ein.

6.4.1.5 Übertragung des Sicherungspunkts

Nachdem die Informationen für den Sicherungspunkt gesammelt wurden, wird dieser an den Vorgänger von ω, in diesem Fall die Ereignisquelle S, übertragen werden. In Abb. 6.1 wird dies durch die Nachricht ACK(SP) ausgeführt, die stromabwärts vom Operator zur Quelle gesendet wird. Der Sicherungspunkt führt in der Ereignisquelle S zu zweierlei Aktionen: Zum einen wird das Log der Ereignisse bereinigt, indem alle Ereignisse, die im Ausgangsstrom vor der Sequenznummer im Sicherungspunkt liegen, gelöscht werden. Zum anderen werden die Informationen des Sicherungspunktes selbst in S gespeichert, damit sie im Falle einer Wiederherstellung von ω zur Verfügung stehen. Wenn die ausgehenden Ereignisse ab σ_s und alle Konsumoperationen bis zum Zeitpunkt der Ausführung des Korrelationsfensters, das mit σ_s startet und zur Erzeugung von σ_o führt, zur Verfügung stehen, kann ω genau in den Zustand versetzt werden, der zu der Zeit vor der Berechnung von σ_o gültig war: ω kann bezüglich des Sicherungspunkts wiederhergestellt werden.

6.4.2 Mehrere Konsumenten

Um das einfachste CEP-System zu erweitern, wird im Folgenden der Fall betrachtet, dass ein Operator mit mehreren Konsumenten verbunden ist. In Abb. 6.2 existiert wieder eine Ereignisquelle S, die mit einem Operatoren ω verbunden ist. Dieser ist nun allerdings mit n Konsumenten C_1 bis C_n verbunden, die beide jeweils bestimmte Ereignisse konsumieren, die ω produziert. Es stellt sich nun die Frage, was sich durch das Vorhandensein mehrerer Operatoren für die Bestätigung von Ereignissen und das Erzeugen der Sicherungspunkt verändert.

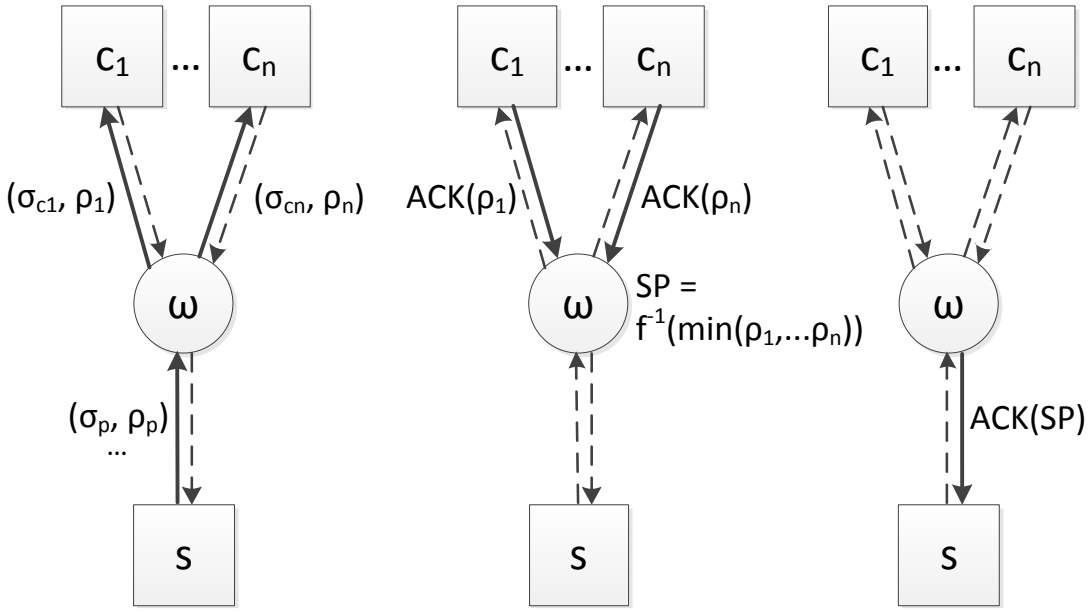

Abbildung 6.2: Versendung und Bestätigung von Ereignissen in einem CEP-System mit mehreren Konsumenten.

6.4.2.1 Ereignisströme

An den Ereignisströmen ändert sich nur wenig. Der Eingangsstrom aus der Ereignisquelle wird weiterhin von ω zu einem Ausgangsstrom weiterverarbeitet. Die Ereignisse des Ausgangsstroms werden an die Konsumenten versendet. Es kann sich dabei um immer exakt dieselben Ereignisse handeln oder es kann sein, dass ω manche Ereignisse nur an einen der Konsumenten versendet. Jedoch ist zu beachten, dass ω nach den Regeln des Systemmodells trotzdem nur einen sequenzierten Ausgangsstrom O besitzt, unabhängig davon, ob alle Konsumenten an allen Ereignissen Interesse haben. Ein solches Verhalten ließe sich beispielsweise durch Publish-Subscribe-Channels implementieren, in dem sich die Konsumenten registrieren. Wir gehen aber von dem allgemeinsten Fall aus, dass jeder Operator an jedem Ereignis Interesse hat, da dadurch alle anderen Fälle mit abgedeckt werden.

6.4.2.2 Mehrere Konsumenten bestätigen zur selben Zeit verschiedene Ereignisse

Der interessante Teil der Untersuchung beginnt, wenn die Konsumenten parallel Bestätigungsnachrichten an ω zurückgeben. Die Bestätigungsnachrichten unterscheiden sich nicht von denen im Fall eines einzelnen Konsumenten: Jeder Konsument bestätigt in regelmäßigen

Abständen den Empfang des letzten Ereignisses durch eine Nachricht, die die Sequenznummer des bestätigten Ereignisses enthält. Es kann nun jedoch vorkommen, dass der Operator ω zur selben Zeit verschiedene Bestätigungsnachrichten von den verschiedenen Konsumenten erhält: Einer der Konsumenten bestätigt das Ereignis mit der Sequenznummer ρ_1, der andere das Ereignis mit der Sequenznummer ρ_2. Auf Basis welcher Bestätigung kann ω die Umkehrfunktion f^{-1} aufrufen, den Sicherungspunkt bestimmen und diesen an die Ereignisquelle weitergeben?

Wenn in ω verschiedene Bestätigungsnachrichten vorliegen, muss unter den Konsumenten immer die Bestätigung mit der *niedrigsten* Sequenznummer als die aktuell gültige Bestätigung betrachtet werden. Wird eine andere Bestätigung herangezogen und ω wird wiederhergestellt, können Ereignisse irreversibel verloren gegangen sein, die von *manchen* der Konsumenten noch nicht bestätigt wurden. Erst, wenn jeder Konsument ein bestimmten Ereignis bestätigt hat, darf es wirklich freigegeben werden. Dies wird durch das Heranziehen der niedrigsten aller aktuellen Bestätigungen aller Konsumenten sichergestellt. Doch wie kann es sein, dass manche Konsumenten ein Ereignis erhalten und bestätigt haben, das andere Konsumenten noch nicht empfangen konnten? Dieser Fall kann eintreten, wenn ω ein Ereignis erzeugt und an einige Konsumenten versenden kann, während die Kommunikationskanäle zu anderen der Konsumenten lange Latenzzeiten haben. Während der Versand des Ereignisses zu diesen Konsumenten noch stockt, d.h. beispielsweise ω macht weitere Übertragungsversuche, haben die anderen das Ereignis bereits bestätigt. Wenn nun ω abstürzt, bevor es den Versand zu allen Konsumenten durchgesetzt hat, aber die Bestätigung schon weitergegeben hat, können die Konsumenten, bei denen der Versand nicht fertiggestellt werden konnte, dieses Ereignis nicht mehr erhalten. Daher kann ein Ereignis nur als bestätigt gelten, wenn es von allen Konsumenten bestätigt wurde. Immer, wenn eine Bestätigungsnachricht von einem der Konsumenten eingeht, überprüft der Operator, ob sich dadurch der aktuelle Sicherungspunkt ändert. Falls ja, wird der Sicherungspunkt wie im obigen Abschnitt beschrieben bestimmt und an die Ereignisquelle weitergeleitet.

Ansonsten ändert sich an dem Verfahren gegenüber dem Fall mit nur einem Konsumenten nichts.

6.4.3 Mehrere Quellen

Im Folgenden wird das System wiederum erweitert: Nun gibt es n Ereignisquellen statt nur einer einzigen. Wie in Abb. 6.3 dargestellt, sind die Ereignisquellen S_1 bis S_n mit dem Operatoren ω verbunden, der wiederum mit den Konsumenten C_1 bis C_n verbunden ist.

6.4.3.1 Ereignisströme

An der Verarbeitung der Ereignisströme ändert sich nur wenig. Da ω nun mehrere Eingangsströme hat, müssen diese durch ein eindeutiges, reproduzierbares Verfahren in einen einzigen

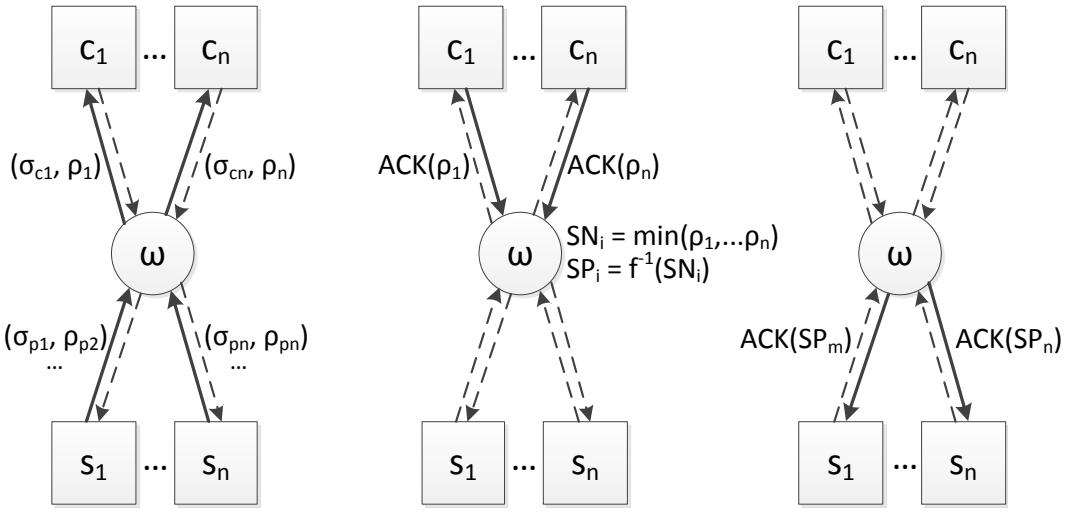

Abbildung 6.3: Versendung und Bestätigung von Ereignissen in einem CEP-System mit mehreren Quellen.

Eingangsstrom sequenziert werden. Auf diesem Strom führt ω wieder die Korrelationsfunktion aus und erzeugt einen Ausgangsstrom, den er an die Konsumenten weiterleitet.

6.4.3.2 Verschiedene Ereignisquellen mit verschiedenen Versionen des Sicherungspunkts

Im Vergleich zum vorherigen Fall mit nur einer Ereignisquelle tritt nun folgendes Problem auf:

ω hat zu verschiedenen Zeitpunkten verschiedene Sicherungspunkte. Wenn die Weitergabe des Sicherungspunktes an eine der Ereignisquellen erfolgreich verlaufen ist und dann ω ausfällt, bevor der Sicherungspunkt auch an die anderen Ereignisquellen weitergegeben werden konnte, besitzen die verschiedenen Quellen einen unterschiedlich aktuellen Sicherungspunkt. Die Frage ist nun, an welchem Punkt die Verarbeitung der Eingangsströme bei der Wiederherstellung von ω aufgenommen wird. Durch den Empfang einer Bestätigungsnachricht führt eine Ereignisquelle irreversible Aktionen bezüglich des empfangenen Sicherungspunktes aus, sodass sie nie wieder an der Wiederherstellung eines älteren Sicherungspunktes teilnehmen kann. Es muss daher ω zum *aktuellsten* Sicherungspunkt wiederhergestellt werden. Zudem muss ein Sicherungspunkt genügend Informationen enthalten, dass bei der Wiederherstellung von ω auch Eingangsströme mit *beliebigen Ereignisfolgen vor dem Startereignis des ersten Korrelationsfensters* kein Problem darstellen. Diese „beliebigen“

Ereignisfolgen (sie sind nicht völlig beliebig, aber in ihrem genauen Umfang eben doch unbekannt) können durch veraltete Sicherungspunkte in manchen Vorgängern entstehen, die zu einem nicht vorhersehbaren Eingangsstrom in ω führen. Erst ab dem Ereignis σ_s im aktuellsten Sicherungspunkt ist der Eingangsstrom von ω tatsächlich wieder vollständig und reproduzierbar hergestellt. Welcher der verschiedenen Sicherungspunkte der Vorgänger von ω der aktuellste ist, wird auf dieselbe Weise ermittelt, in der die Sequenzierung von Ereignissen aus den verschiedenen eingehenden Ereignisströmen in den Eingangsstrom I bestimmt wird. Werden die Ereignisse σ_s aus den Vorgängern in eine Sequenz gebracht, ist das letzte Ereignis in dieser Sequenz das Startereignis des aktuellsten Sicherungspunkts.

6.4.3.3 Umkehrfunktion f^{-1}

Die Berechnung der Umkehrfunktion f^{-1}: O $\rightarrow$ I, die eine Abbildung des bestätigten Ausgangsereignis auf das Startereignis σ_s des entsprechenden Korrelationsfensters im Eingangsstrom ist, wird im Falle mehrerer Vorgänger von ω etwas abgeändert. Während bei nur einem Vorgänger die Sequenznummer von σ_s zur eindeutigen Identifizierung ausreicht, muss im Falle unterschiedlicher Vorgänger für jeden dieser Vorgänger eine eigene Sequenznummer bestimmt und einem Vektor gespeichert werden. Diese bestimmt jeweils, ab welchem Ausgangsereignis ein Vorgänger seinen Ausgangsstrom zur Wiederherstellung an ω versenden muss. Die Umkehrfunktion bestimmt also nicht mehr nur das Startereignis des Korrelationsfensters, sondern die ersten Ereignisse aller einzelnen Eingangsströme I_i in diesem Korrelationsfenster. Diese Information muss aus dem bestätigten Ereignis im Ausgangslog extrahiert werden. Daher muss die Information in einem Ausgangsereignis σ_o nach Abschnitt 6.4.1.3 um einen Vektor von Startereignissen erweitert werden.

Ansonsten ändert sich im Vergleich zum System mit nur einer Ereignisquelle nichts.

6.4.4 Mehrere sequentiell abhängige Operatoren

Bisher wurden CEP-Systeme mit nur einem Operatoren betrachtet. Nun wird das Modell erweitert, indem zwischen den Ereignisquellen und den Ereigniskonsumenten mehrere sequentiell abhängige Operatoren die Ereignisse in verschiedenen Zwischenschritten weiterverarbeiten. In Abb. 6.4 ist eine Reihe von Ereignisquellen S_1 bis S_n mit einem Operator ω_k verbunden, der in einer Sequenz von miteinander verbundenen Operatoren von ω_{k-1} bis ω_0 steht. ω_0 ist dann mit einer Reihe von Ereigniskonsumenten verbunden, C_1 bis C_n.

6.4.4.1 Ereignisströme

Die Ereignisse aus den Quellen werden im Operatoren ω_k zu anderen Ereignissen korreliert, die einen Zwischenschritt zwischen den einfachen Ereignissen aus den Quellen und den komplexen Ereignissen, die an die Konsumenten ausgeliefert werden, darstellen. Diese Ereignisse werden wiederum von anderen Operatoren weiterverarbeitet, jeder Operator implementiert

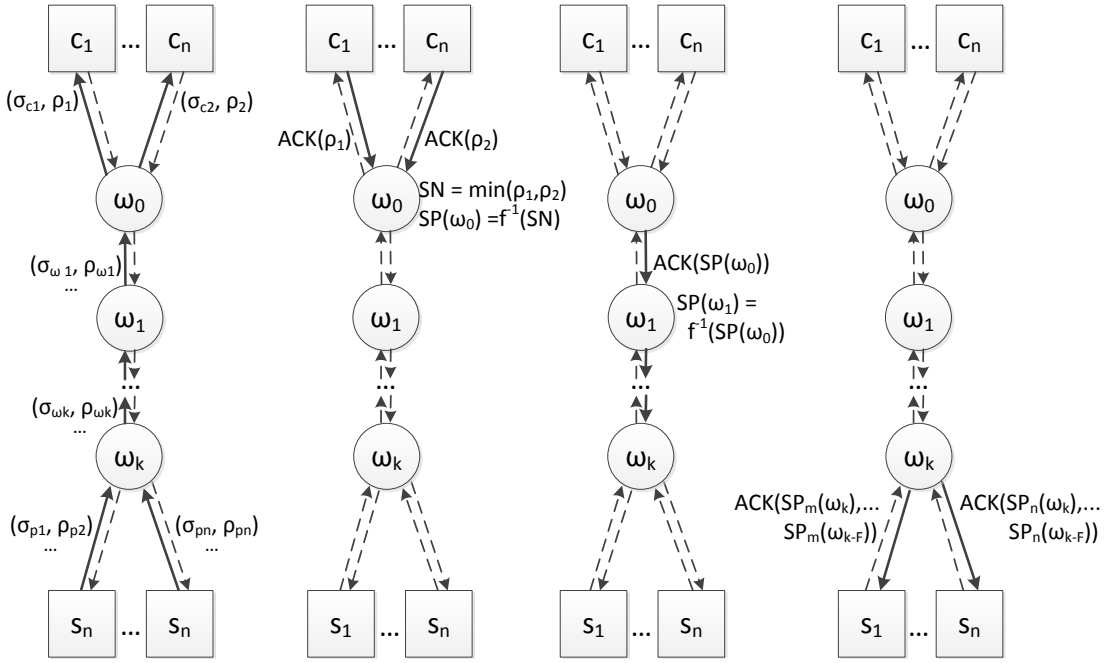

Abbildung 6.4: Versendung und Bestätigung von Ereignissen in einem CEP-System mit mehreren sequentiell abhängigen Operatoren.

einen Verarbeitungsschritt. Schließlich führt ω_0 in der letzten Verarbeitungsebene die letzte Korrelation durch, als deren Ergebnis das „Endprodukt“ von komplexen Ereignissen entsteht, die an die Konsumenten ausgeliefert werden und die die vom CEP-System erkannten Situationen von Interesse anzeigen.

6.4.4.2 Verschiedene Operatoren mit jeweils eigenen Sicherungspunkten

In Systemen mit mehreren voneinander abhängigen Operatoren hat jeder Operator zu jedem Zeitpunkt einen eigenen, bestimmten Zustand. Zu bestimmten Zeitpunkten sichert jeder Operator diesen Zustand in einem Sicherungspunkt. Der Zeitpunkt der Sicherung wird von den Bestätigungsnachrichten der Ereigniskonsumenten angestoßen. Immer, wenn ein Ereigniskonsument bei seinem Vorgängeroperatoren den Empfang eines Ereignisses oder mehrerer Ereignisse bestätigt, breitet sich diese Bestätigungsnachricht wellenförmig über die jeweiligen Vorgänger im Operatorgraphen bis hin zu den Ereignisquellen aus. Ein bestätigtes Ereignis der höchsten Komplexität führt i.A. dazu, dass Ereignisse aller Zwischenstufen bestätigt werden können und sich diese Bestätigung schließlich bis zu den einfachsten Ereignissen aus den Quellen fortsetzt.

Immer, wenn ein Operator von seinen Vorgängern eine Bestätigung bezüglich einiger der Ereignisse aus seinem Ausgangsstrom erhalten hat, aktualisiert er seinen Sicherungspunkt in der Weise, wie es in den vorherigen Abschnitten beschrieben wurde: Bei mehreren

Nachfolgern gilt die niedrigste aktuelle Bestätigung, die Umkehrfunktion wird aufgerufen, das Startereignis σ_s des entsprechenden Korrelationsfensters $w_{recover}$ und der zu dem Zeitpunkt gültige Stand der Konsumtabelle werden im Sicherungspunkt gespeichert. Während in den bisher betrachteten Systemen mit nur einem Operator dieser Sicherungspunkt direkt an die Ereignisquellen übertragen und dort gesichert wurde, gestaltet sich die Situation bei mehreren abhängigen Operatoren etwas anders.

Um die Bestimmung und Übertragung der Sicherungspunkte genauer zu untersuchen, folgen wir dem Verlauf der „Bestätigungswelle“ von den Konsumenten bis zu den Quellen. Die Bestätigungen der Konsumenten enthalten nur eine Sequenznummer, nämlich die des letzten bestätigten Ereignisses. Nennen wir den Operatoren, der eine solche Bestätigung bekommt, ω_0 und nehmen an, dass die Bestätigung zur Aktualisierung des Sicherungspunktes führt. ω_0 überträgt seinen neuen Sicherungspunkt SP[ω_0] mit Hilfe einer Bestätigungsnachricht ACK(SP[ω_0]) an seinen Vorgänger ω_1. Im übertragenen Sicherungspunkt findet sich die Information, ab welchem Ereignis im Ausgangsstrom von ω_1 dieser im Fall einer Wiederherstellung von ω_0 neu übertragen werden muss. Alle Ereignisse, die davor in der Sequenz im Ausgangsstrom davor liegen, können also freigegeben werden. ω_1 wendet auf das Ereignis aus SP[ω_0] die Umkehrfunktion an, das Ergebnis ist das Ereignis σ_s aus dem Eingangsstrom von ω_1, von dem aus alle älteren Ereignisse in diesem Eingangsstrom freigegeben werden können. ω_1 bildet nun einen Sicherungspunkt SP[ω_1] bezüglich des Zeitpunkts, an dem das Korrelationsfenster, das zur Erzeugung des in SP[ω_0] bestätigten Ereignisses führte, geöffnet wurde. Dieser Sicherungspunkt wird mit einer Bestätigungsnachricht an den Vorgänger von ω_1 übertragen, welcher wiederum seinen eigenen Sicherungspunkt auf dieselbe Weise aktualisiert und an seine Vorgänger überträgt, bis schließlich die Bestätigung und Aktualisierung der Sicherungspunkte in jedem Pfad bis zu den Ereignisquellen fortgeführt worden ist.

6.4.4.3 Redundante Speicherung der Sicherungspunkte

Das Ziel des Wiederherstellungsverfahrens ist, wie in Kapitel 4 beschrieben, die Wiederherstellung des Systems bei einem gleichzeitigen Ausfall von maximal F Operatoren. Das heißt, dass in einer Sequenz von F+1 abhängigen Operatoren immer mindestens 1 Operator korrekt läuft. Im schlechtesten Fall muss von diesem Operator, nennen wir ihn ω_k, ausgehend eine Reihe von F Operatoren, ω_{k-1} bis ω_{k-F}, wiederhergestellt werden. ω_k hat den Sicherungspunkt SP[ω_{k-1}] gespeichert und kann anhand der darin enthaltenen Informationen und dem Ausgangslog L(ω_k) seinen Nachfolger ω_{k-1} wiederherstellen. ω_{k-1} kann zwar sein Ausgangslog L(ω_{k-1}) wiederherstellen, doch fehlt nach der Wiederherstellung der Sicherungspunkt SP[ω_{k-2}], ohne den ω_{k-2} nicht wiedergestellt werden kann. Daher muss ω_k auch den Sicherungspunkt SP[ω_{k-2}] speichern und bei der Wiederherstellung von ω_{k-1} an diesen übertragen. Genauso verhält es sich mit den Operatoren ω_{k-3} bis ω_{k-F}: Um eine Reihe von F Operatoren wiederherstellen zu können, muss in ω_k eine Liste von F Sicherungspunkten SP[ω_{k-1}] bis SP[ω_{k-N}] gespeichert sein.

Aus dieser Beobachtung folgt, dass ein Operator nicht nur die Sicherungspunkte seiner direkten Nachfolger speichert, sondern eine Liste von Sicherungspunkten der folgenden

F Operatoren. Immer, wenn er eine Bestätigungsnachricht von einem seiner Nachfolger empfängt, aktualisiert er den eigenen Sicherungspunkt und fügt ihn an erster Stelle in eine Liste von Sicherungspunkten ein. Falls daraufhin mehr als F Sicherungspunkte in der Liste sind, entfernt er den letzten Eintrag aus der Liste und schickt sie weiter an seine Vorgänger. Die Liste der Sicherungspunkte in Operator ω_i enthält also die Sicherungspunkte ω_i bis maximal ω_{i-F}, sofern es so viele Operatoren gibt, die auf ω_i folgen. Der Vorgänger von ω_i, ω_{i+1}, erhält diese Liste, berechnet anhand des Sicherungspunkts SP[ω_i] den eigenen Sicherungspunkt SP[ω_{i+1}], fügt diesen in die Liste ein, entfernt SP[ω_{i-F}] und schickt eine Bestätigungsnachricht mit der Liste an seinen Vorgänger ω_{i+2}. Dieses Verfahren wird so lange fortgesetzt, bis die Ereignisquellen von den Bestätigungsnachrichten erreicht werden.

6.4.5 Operator mit mehreren Vorgängern

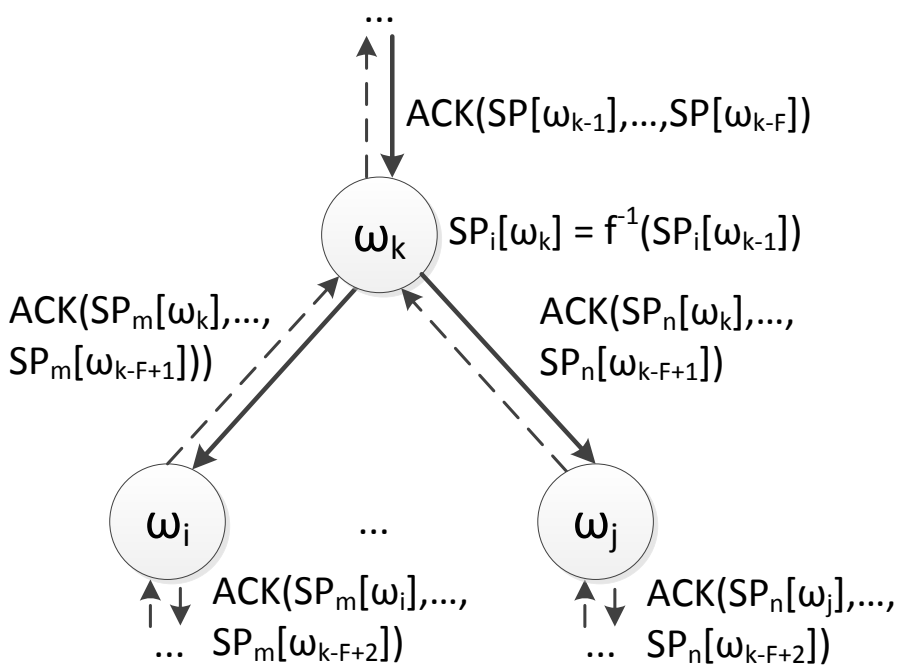

Abbildung 6.5: Bestätigungsnachrichten in einem Ausschnitt aus einem CEP-System mit einem Operator, der mehrere Vorgänger hat.

In Abb. 6.5 ist ein Ausschnitt aus einem CEP-System dargestellt, in dem ein Operator, ω_k, mehrere Vorgänger hat. Mehrere Vorgänger zu haben bedeutet für ω_k, dass zu seiner Wiederherstellung die Ausgangsströme aus sämtlichen Vorgängern notwendig sind. Daher bestätigt ω_k seinen eigenen Sicherungspunkt bei allen seinen Vorgängern. Die Sicherungspunkte der auf ω_k folgenden Operatoren müssen, wie in Abschnitt 6.4.4 begründet wird, redundant in den Vorgängeroperatoren von ω_k gespeichert werden, sodass jede Kette von F aufeinanderfolgenden Operatoren wiederherstellbar ist. Daher versendet ω_k neben dem eigenen Sicherungspunkt auch die Sicherungspunkte der F-1 auf ω_k folgenden Operatoren

an seine Vorgänger.

Es ist zu beachten, dass zwar die Vorgänger von ω_k verschieden aktuelle Sicherungspunkte von ω_k und seinen Nachfolgern besitzen können, die *Listen der Sicherungspunkte* an sich aber immer *in sich konsistent* sind. Eine konsistente Liste von Sicherungspunkten bedeutet, dass der Sicherungspunkt eines Nachfolgers stets bezüglich eines Zeitpunkts erstellt wurde, der zeitlich später als der entsprechende Zeitpunkt des Sicherungspunkts des Vorgängers liegt. Wäre es anders, d.h. ein Operator soll einen Nachfolger in seinem Zustand zu einem früheren Zeitpunkt wiederherstellen als der Zeitpunkt des eigenen Zustands hergibt, könnte dieser die Wiederherstellung nicht durchführen, da er dafür ältere Ereignisse aus seinem Ausgangsstrom benötigte. Die Konsistenz der Listen ist durch die Eigenschaften der Umkehrfunktion gegeben: Das Ergebnis der Umkehrfunktion ist stets ein Ereignis, das in der ursprünglichen Ausführung der Korrelationsfunktion ein Eingangsereignis war. Wenn aus einem Ereignis ein anderes Ereignis erzeugt wird, muss ersteres zuerst existiert haben. Daher bezieht sich der Sicherungspunkt, der anhand der Umkehrfunktion berechnet wurde, immer auf einen früheren Zeitpunkt.

Damit diese Konsistenz erhalten bleibt, ist die Speicherung der Listen in allen Vorgängern notwendig. Wäre die Liste beispielsweise im CEP-System in Abb. 6.5 nur in ω_i gespeichert und hätte ω_j einen aktuelleren Sicherungspunkt $\mathrm{SP}_n[\omega_k]$ erhalten als ω_i, der $\mathrm{SP}_m[\omega_k]$ speichert, könnte zwar ω_k zum Sicherungspunkt $\mathrm{SP}_n[\omega_k]$ wiederhergestellt werden, doch die Nachfolgeroperatoren müssten mit den älteren Sicherungspunkten aus den Listen von ω_i wiederhergestellt werden. Es könnte nicht garantiert werden, dass diese Sicherungspunkte nicht auf ältere Ereignisse zugreifen müssen, als sie in $\mathrm{SP}_n[\omega_k]$ vorhanden sind. Durch das „Mischen“ der Sicherungspunkte wäre also die Konsistenz der Liste nicht mehr zu gewährleisten.

6.4.6 Operator mit mehreren Nachfolgern

Um sämtliche möglichen Topologien der Operatorgraphen abzudecken, fehlt noch ein Fall: Ein Operator, der mit mehreren Nachfolgeroperatoren verbunden ist. Abb. 6.6 zeigt einen Ausschnitt aus einem solchen CEP-System. Operator ω_k muss in der Lage sein, sowohl sämtliche Nachfolger als auch deren Nachfolger bis zu F-1 Ebenen weit wiederherzustellen. Das heißt, dass ω_k alle Sicherungspunkte seiner Nachfolger und deren Nachfolger der nächsten F-1 Ebenen in seinem Vorgänger speichern muss. An dieser Stelle wird die bisher verwendete *Liste* von Sicherungspunkten durch einen *Baum* ersetzt. Man kann es auch so formulieren, dass in einem CEP-System, in dem Operatoren niemals mehrere Nachfolger haben, der Baum von Sicherungspunkten so flach ist, dass es sich dabei um eine Liste handelt. Der vor uns liegende Fall ist also allgemeiner als die Spezialfälle, die wir zuvor betrachtet haben.

Ein Operator muss stets einen *Baum von Sicherungspunkten*, einen „Sicherungsbaum“ speichern. Dieser Baum hat den Sicherungspunkt des Operators selbst als Wurzel und verläuft in Ereignisflussrichtung entsprechend der Topologie des Operatorgraphen bis zur Tiefe

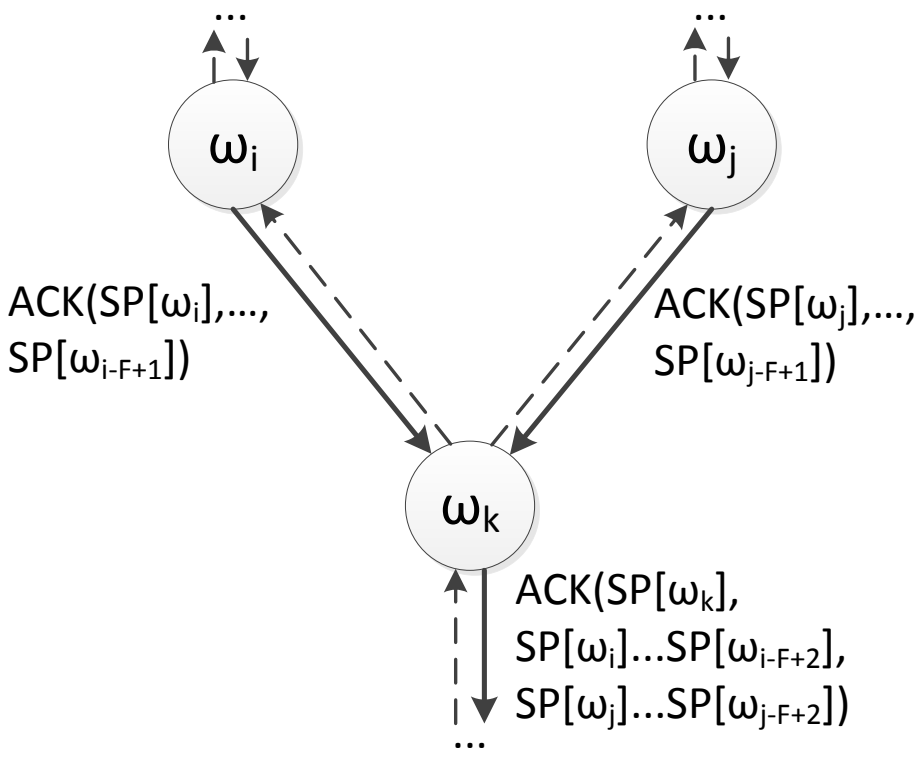

Abbildung 6.6: Bestätigungsnachrichten in einem Ausschnitt aus einem CEP-System mit einem Operator, der mehrere Nachfolger hat.

F. Wenn ein Operator von einem Nachfolger einen Sicherungsbaum erhält, aktualisiert er anhand des Sicherungspunkts des direkten Vorgängers seinen eigenen Sicherungspunkt und baut an diesen als Wurzel die jeweils aktuellsten Sicherungsbäume all seiner Nachfolger an. Diesen neuen Sicherungsbaum beschneidet er so, dass alle Knoten in einer Tiefe größer als F entfernt werden und sendet ihn anschließend an seine Vorgänger.

Das Konzept der Sicherungsbäume ist also die konsequente Verallgemeinerung des bisherigen Ansatzes, die es ermöglicht, eine „verzweigte“ Topologie der Tiefe F von einem Operator aus wiederherzustellen.

6.4.7 Zusammenfassung: Verwaltung von Logs und Sicherungspunkten

Wenn man alle gesammelten Informationen zusammenfasst, kommt man zu einem allgemeinen Ablauf der Verwaltung von Sicherungspunkten und Logs, der für alle erlaubten Topologien des Operatorgraphen geeignet ist.

Algorithmus 6.1 Operator: handleAcknowledgement

```
procedure HANDLEACKNOWLEDGEMENT(Acknowledgement ack)
    UPDATESTOREDACK(ack,ack.producer)        // aktualisiere ACK des Nachfolgers
    if oldest stored ACK has changed then
        UPDATEOWNSAVEPOINT                   // durch Aufruf von f^-1
        PRUNELOGS                            // Events und Konsumoperationen
        BUILDNEWSAVEPOINTTREE                // mit eigenem Sicherungspkt. als Wurzel
        SENDACK(predecessors)
    end if
end procedure
```

6.4.7.1 Inhalt eines Sicherungspunktes

Ein Sicherungspunkt SP[ω] eines Operators ω enthält:

1. Einen Vektor von Sequenznummern. Für jeden Operator aus `pred(`ω`)` enthält der Vektor eine Sequenznummer.
2. Die Sequenznummer des nächsten von ω zu erzeugenden Ereignisses.
3. Die Tabelle der Konsumoperationen zum Wiederherstellungszeitpunkt.

6.4.7.2 Empfang von Bestätigungsnachrichten mit Sicherungsbäumen

Ein Operator empfängt von seinen Nachfolgern Bestätigungsnachrichten, die einen Baum von Sicherungspunkten enthalten. Diese Bestätigungsnachrichten werden von den Konsumenten angestoßen und werden stromabwärts von Operator zu Operator weiterverarbeitet, bis sie schließlich in den Ereignisquellen ankommen. Wenn ein Operator ω von einem seiner Nachfolger ω_s eine Bestätigungsnachricht empfängt, führt er Algorithmus 6.1 aus. Im Folgenden wird der Algorithmus erklärt:

1. Der Operator aktualisiert das gespeicherte ACK bezüglich seines Nachfolgers.
2. Der überprüft, ob sich dadurch das älteste gespeicherte ACK ändert. Falls ja, aktualisiert er seinen eigenen Sicherungspunkt, indem er die Umkehrfunktion auf das ihn betreffende Ereignis in SP[ω_s] aufruft.
3. Er kürzt sein Ausgangslog und bereinigt seine Tabelle der Konsumoperationen, falls er seinen eigenen Sicherungspunkt in diesem Durchgang aktualisiert hat.
4. Er fügt den eigenen (evtl. aktualisierten) Sicherungspunkt als Wurzel in einen Sicherungsbaum ein. Die Kinder sind die aktuellen Sicherungsbäume seiner Nachfolger. Der Baum wird auf eine Tiefe von F gekürzt.
5. Er versendet den neuen Sicherungsbaum an alle seine Vorgänger.

Bereinigung der eigenen Logs Das Ausgangslog L(ω) eines Operators ω wird immer so gekürzt, dass alle Ereignisse gelöscht werden, die vor der niedrigsten ihn betreffenden Sequenznummer bezüglich aller seiner Nachfolger liegen. Diese Ereignisse sind von allen Nachfolgern bestätigt und müssen nicht mehr im Speicher für den Fall einer Wiederherstellung eines Nachfolgers vorgehalten werden. Das Ausgangslog und seine Kürzung betreffen also die direkten Nachfolger und deren Wiederherstellung.

Die Tabelle der Konsumoperationen wird immer so gekürzt, dass die Informationen über Konsumoperationen gelöscht werden, die vor dem Startereignis des nächsten wiederherzustellenden Fensters liegen. Diese Konsumoperationen werden im Wiederherstellungsfall nicht mehr benötigt, da sie die Berechnung der Korrelationsfunktion nicht beeinflussen. Das Startereignis des nächsten wiederherzustellenden Fensters ist genau das Ereignis σ_s, welches die Umkehrfunktion zurückgibt. Die Konsumoperationen und deren Kürzung betreffen also die Wiederherstellung des Operators selbst.

6.5 Wiederherstellung im Fehlerfall

6.5.1 Anforderungen an die Wiederherstellung

Wenn ein oder mehrere Operatoren ausfallen, muss das CEP-System in zweierlei Hinsicht wiederhergestellt werden. Zum einen muss die Topologie des Operatorgraphen wiederhergestellt werden: Die ausgefallenen Operatoren werden durch neue ersetzt und die Verbindungen zwischen den Operatoren müssen neu aufgebaut werden. Letztendlich muss sich die Topologie so stabilisieren, dass sie der Topologie vor dem Ausfall entspricht. Zum anderen müssen die Ereignisströme der ausgefallenen Operatoren wiederhergestellt werden. Wenn ein Operator ausfällt, verliert er sämtliche volatilen Zustandsinformationen, wie z.B. das Ausgangslog, den empfangenen Eingangsstrom, die laufende Berechnung der Korrelationsfunktion und den gespeicherten Sicherungsbaum. Alle diese Informationen müssen wiederhergestellt werden, um schließlich die Ströme von Ereignissen zwischen den Operatoren wiederherzustellen. Es darf kein unbestätigtes Ereignis durch den Verlust der Zustandsinformationen irreversibel verlorengehen. Die Ereignisströme müssen sich also insofern stabilisieren, dass die Ereignisse, die an die Konsumenten ausgeliefert werden, sich nicht von dem Fall unterscheiden, in dem kein Operator ausgefallen ist.

Im Folgenden wird in einem zweistufigen Verfahren untersucht, wie das Wiederherstellungsverfahren aussehen muss, um beide Anforderungen zu erfüllen. Dazu wird zunächst angenommen, dass die Topologie immer stabil bleibt, um ein Verfahren zur Stabilisierung der Ereignisströme zu entwickeln. Schließlich wird ein Stabilisierungsverfahren für die Topologie eingeführt, und somit gezeigt, dass auch die erste Wiederherstellungsanforderung abgedeckt wird.

Algorithmus 6.2 Operator: recover

```
procedure RECOVER
    SEND RECOVERYREQUEST TO ALL PREDECESSORS  // aktualisiere ACK des Nachfolgers
    wait for answers...
    if all predecessors answered then
        RecoveryInformation ri = HIGHESTNEXTSEQNO(allAnswers)          // nutze die
                // Informationen aus der Antwort, die die höchste nächste Seq.nr. angibt
        EXTRACTOWNSAVEPOINTTREE(ri)
        SEQUENCEINCOMINGEVENTSTREAMS(ownSavepoint)
        OPENCORRELATIONWINDOW(incomingStream)
        CONSUMEEVENTS(consumptionTable)
    end if
    SEND RECOVERYNOTIFICATION TO ALL SUCCESSORS
end procedure
```

6.5.2 Wiederherstellung der Ereignisströme

Wir nehmen zunächst also an, dass ein ausgefallener Operator sich insofern erholt, als dass er nach dem Ausfall automatisch neu gestartet wird. Alle volatilen Informationen sind verloren, doch der Operator taucht unter derselben Adresse im Netzwerk wieder auf und startet eine Recovery-Prozedur. Nehmen wir weiter an, dass der Operator seine direkten Vorgänger im Operatorgraphen kennt und diese Informationen auch nach dem Neustart noch vorhanden ist.

6.5.2.1 Die Recovery-Prozedur

Sobald ein Operator ω neu gestartet wurde, ruft er eine Initialisierungsroutine auf. Diese Routine sendet einen RECOVERYREQUEST an alle direkten Vorgänger des Operators. Wenn ein Operator ω_p einen RECOVERYREQUEST empfängt, sendet er anhand des gespeicherten Sicherungspunkts SP[ω] folgende Informationen an seinen Nachfolger ω:

- Die Sequenznummer des nächsten von ω zu erzeugenden Ereignisses.
- Die Tabelle der Konsumoperationen in ω zum Wiederherstellungszeitpunkt.
- Der Vektor von Sequenznummern aller direkten Vorgänger von ω.
- Alle Ereignisse aus dem Ausgangslog L(ω_p) ab der ω_p betreffenden Sequenznummer.
- Der Sicherungsbaum von ω_p zur Wiederherstellung nachfolgender Operatoren.

Sobald ω von allen Vorgängern auf den RECOVERYREQUEST Antwort erhalten hat, kann der Zustand zum aktuellsten Sicherungspunkt wiederhergestellt werden. Dazu geht ω nach Algorithmus 6.2 vor, der im Folgenden erläutert wird:

1. Aus allen Antworten wird die Antwort mit dem höchsten Wert für die Sequenznummer des nächsten von ω zu erzeugenden Ereignisses ausgewählt.
2. Aus dieser Antwort wird der eigene Sicherungsbaum wiederhergestellt, indem der Teilbaum mit ω als Wurzelknoten aus dem Sicherungsbaum von ω_p ausgeschnitten wird.
3. Anhand des Vektors von Sequenznummern wird aus den eingehenden Ereignisströmen aus den Vorgängern für jeden dieser Ströme das Ereignis ermittelt, ab dem die Ströme in den Eingangsstrom I sequenziert werden. Alle eingehenden Ereignisse mit einer kleineren Sequenznummer als die entsprechende Sequenznummer im Vektor werden verworfen.
4. Die Konsumoperationen werden auf den sequenzierten Eingangsstrom I angewandt.
5. Ein Korrelationsfenster wird geöffnet, als Startereignis dient das erste Ereignis aus dem sequenzierten Eingangsstrom.

Wenn mehrere Operatoren in Reihe ausgefallen sind, werden sie von unten nach oben sukzessive wiederhergestellt. Sobald ein Operator wiederhergestellt ist, sendet er an seine Nachfolger eine RecoveryNotification. Er kennt die Nachfolger aus den empfangenen Sicherungsbäumen. Ist der Nachfolger selbst ausgefallen gewesen und befindet sich gerade in der Wiederherstellungsphase, in der er auf Antwort von allen Vorgängern wartet, kann er nun seinen RecoveryRequest an den gerade wiederhergestellten Vorgänger wiederholen. Auf diese Weise wird letztendlich jeder Operator wiederhergestellt.

Es ist zu beachten, dass die Sicherungsbäume sich bei einer Wiederherstellung um eine Ebene verkürzen. Bei einer sequentiellen Wiederherstellung von N aufeinander folgenden Operatoren kann der Sicherungsbaum des letzten wiederhergestellten Operators folglich völlig leer sein, was bedeutet, dass kein weiterer Operator in dieser Reihe wiederhergestellt werden kann. Die Informationen aus den Sicherungsbäumen reichen also gerade dazu aus, F Operatoren in Reihe wiederherzustellen, und keinen weiteren. In dem Sinne ist ein Operator an dieser Stelle noch nicht vollständig wiederhergestellt, da er einen Ausfall weiterer F Operatoren in Folge nicht direkt kompensieren kann. Allerdings kann der Operator schon mit der Ereignisverarbeitung fortfahren, d.h. die Ereignisverarbeitung ist bereits wiederhergestellt, die Recovery-Fähigkeit aber noch nicht. Um einen Operator vollständig wiederherzustellen, bedarf es der Wiederherstellung des kompletten Sicherungsbaumes. Dies geschieht durch Antwort auf die RecoveryNotification: Ist der Nachfolger, der sie erhält, nicht selbst in einer Wiederherstellungsphase, sondern bereits komplett wiederhergestellt, sendet er an seinen nur teilweise wiederhergestellten Vorgänger seinen aktuellen Sicherungspunkt, damit dieser den Sicherungsbaum wieder komplettieren kann. Ein Operator ist komplett wiederhergestellt, wenn er von allen seinen Nachfolgern den Sicherungsbaum empfangen und in seinen eigenen Sicherungsbaum integriert hat. Dies führt im Normalbetrieb zur sukzessiven Weitergabe des Sicherungsbaumes, bis alle Operatoren vollständig wiederhergestellt sind.

6.5.3 Kontrolle und Anpassung der Topologie

Die bisherige Annahme des „automatischen“ Neustarts eines ausgefallenen Operators lässt sich anhand der Abstraktion einer zentralen Fehlerdetektor-Komponente [Rey05, FGK11, CT96, GR06] im CEP-System veranschaulichen. Ein solcher zentraler Detektor könnte bestimmen, wann ein Operator ausgefallen ist und seine Recovery-Prozedur anstoßen. Doch wie genau arbeitet der Fehlerdetektor?

6.5.3.1 Eigenschaften des Fehlerdetektors

Der Fehlerdetektor ist eine zentrale Komponente, die auf einem Fail-Recovery-System läuft. Das heißt, wenn der Fehlerdetektor abstürzt wird er neugestartet und kann seine Arbeit an der Stelle fortsetzen, an der er zuvor abgestürzt war. Das System wird letztendlich immer lebendig sein, d.h. nach einer unbekannten Zahl von Abstürzen und Wiederherstellungen läuft der Fehlerdetektor stabil. Zu beachten ist dabei, dass ein solcher Fehlerdetektor eventuell mit einem persistenten Speicher ausgestattet sein kann, aber die Ereignisströme und Operatoren in der eigentlichen Ereignisverarbeitung weiterhin ohne persistenten Speicher arbeiten.

6.5.3.2 Fehlererkennung

Um festzustellen, ob ein spezifischer Operator korrekt arbeitet oder ob er ausgefallen ist, gibt es grundsätzlich 2 Möglichkeiten (vgl. [GR06, S. 45ff]): Die erste Möglichkeit ist von einem *synchronen System* auszugehen, in dem folgende Eigenschaften gelten: Synchrone Berechnung (d.h. nach oben begrenzte Berechnungszeit), synchrone Kommunikation (d.h. nach oben begrenzte Kommunikationsverzögerungen) und synchrone physikalische Uhren. In einem solchen System wäre es recht einfach, einen Operator innerhalb einer festen Zeitgrenze als fehlerhaft bzw. ausgefallen zu erkennen, beispielsweise indem jeder Operator dem Fehlerdetektor periodisch Heartbeat-Nachrichten zuschickt. Diese müssen durch die festen Zeitgrenzen für Berechnung und Nachrichtenübertragung innerhalb einer bestimmten Zeit ankommen, wenn der Operator korrekt arbeitet. Wenn die Zeitgrenzen immer eingehalten werden, wird ein fehlerhafter Operator schließlich innerhalb einer berechenbaren Zeit vom Fehlerdetektor als solcher erkannt. Dieses Modell hat allerdings hauptsächlich einen Nachteil: Die Abdeckung des Systems. In realen CEP-Szenarien kann man nicht einfach von einer solchen Synchronität ausgehen, insbesondere wenn das System hochskalierbar ist und beispielsweise die Operatoren über das Internet miteinander verbunden sind und sehr komplexe Berechnungen ausführen müssen. Das Problem der schwankenden Antwortzeiten wird auch als „Jitter“ bezeichnet. Man müsste zumindest die Zeitschranken für den Jitter so hoch ansetzen, dass man mit einer sehr hohen Wahrscheinlichkeit davon ausgehen kann, dass sie von einem korrekten Prozess nicht verletzt werden, d.h. also von den *Worst-Case*-Fällen ausgehen. Dies kann das System aber ineffizient machen, da zwischen dem Auftritt des Fehlers und seiner Erkennung eine lange Zeit vergehen kann. In dieselbe Richtung geht auch die Idee der *teilsynchronen Systeme*: Die zeitlichen Annahmen gelten darin **letztendlich**, d.h.

nach einer beschränkt langen Zeit, in der sie verletzt werden können. Es ist damit ebenso ein synchrones System in dem Sinne, dass zeitliche Annahmen gemacht werden, auf denen die weiteren Algorithmen basieren.

Eine Alternative zum synchronen Systemmodell ist ein asynchrones Systemmodell ohne solche festen Synchronitätsvorgaben, wie es in unserem Systemmodell angenommen wird. Das heißt, ein Operator kann für einen Berechnungsschritt eine *beliebig lange*, aber endliche Zeit benötigen, Ereignisse und Nachrichten zwischen 2 Operatoren können beliebig lange unterwegs sein, bis sie schließlich vom Empfänger empfangen werden, und die Uhren in den Operatoren sind in keiner Weise synchronisiert, d.h. weder haben sie ursprünglich dieselbe Zeit noch laufen sie in derselben Geschwindigkeit. Zunächst kann ein Fehlerdetektor A über einen Operatoren B in einem asynchronen System nicht eindeutig und fehlerfrei feststellen, ob dieser abgestürzt ist oder noch korrekt läuft. Wartet A beispielsweise auf eine Heartbeat-Nachricht von B, gibt es zwei Möglichkeiten: B ist abgestürzt oder er läuft korrekt. Wie lange soll A nun auf den Heartbeat warten? Wenn B abgestürzt ist, wartet A unendlich lange und das System als Ganzes ist nicht mehr *lebendig* (Liveness-Property, vgl. [Lam77]). Wenn B nicht abgestürzt ist, aber sich der Heartbeat trotzdem sehr lange verzögert, beispielsweise weil der Kommunikationskanal überlastet ist, und A irgendwann fälschlicherweise annimmt, dass B wohl abgestürzt sein muss und daraufhin eine nicht reversible Aktion ausführt, läuft das System vielleicht nicht *sicher* (Safety-Property, vgl. [Lam77]). Was ist also zu tun?

Der Lösungsansatz in dieser Arbeit liegt darin, die *Sicherheit* des Systems zu garantieren, auch wenn der Fehlerdetektor eine falsche Entscheidung bezüglich eines Operators trifft. Dies kann dann realisiert werden, wenn der Fehlerdetektor die Ausführung irreversibler Aktionen vermeidet. Zudem muss ein Kompromiss zwischen der Korrektheit des Fehlerdetektors und der Reaktionszeit des Systems auf Ausfälle gefunden werden, indem eine flexible Zeitschranke für die Verzögerung der Heartbeat-Nachrichten definiert wird (vgl. Abschnitt 6.5.3.4).

6.5.3.3 Sichere Wiederherstellung der Topologie bei unsicherer Fehlererkennung

Um die Sicherheit des Systems zu garantieren, auch wenn der Fehlerdetektor eine falsche Entscheidung bezüglich des Zustands (`lebendig` oder `ausgefallen`) eines Operators getroffen hat, muss man zunächst untersuchen, welche falschen Entscheidungen dabei in Frage kommen. Dabei soll im Folgenden der Befund `ausgefallen` als *positiver* Befund bezeichnet werden, entsprechend ist der Befund `lebendig` ein *negativer* Befund. Falsche Befunde können *falsch-negativ* oder *falsch-positiv* sein.

Falsch-negative Befunde bedeuten, dass der Fehlerdetektor einen ausgefallenen Operator dauerhaft als `lebendig` einschätzt. Im Fall einer Heartbeat-Überprüfung, wie sie im dieser Arbeit zugrundeliegenden Modell angewandt wird, kann ein solcher falsch-negativer Befund nicht auftreten, denn ein ausgefallener Operator kann keine Heartbeat-Nachrichten mehr erzeugen und somit wird er letztendlich vom Fehlerdetektor als ausgefallen eingeschätzt.

Falsch-positive Befunde bedeuten, dass der Fehlerdetektor einen Operator (evtl. nur kurzzeitig) als `ausgefallen` einschätzt, obwohl der Operator eigentlich lebendig ist. Dies kann vorkommen, wenn der Fehlerdetektor eine zu geringe Zeitschranke für die Wartezeit auf eine Heartbeat-Nachricht hat oder sich die Nachricht bspw. wegen Überlastung der Verbindung außergewöhnlich lange verzögert. Es gibt aufgrund des asynchronen Systems für den Fehlerdetektor keine Möglichkeit, einen falsch-positiven Befund innerhalb einer festgelegten Zeitspanne von einem korrekten Befund zu unterscheiden. Letztendlich wird ein lebendiger Operator seine Heartbeat-Nachrichten immer an den Fehlerdetektor senden, doch wenn der Heartbeat ausfällt, kann der Fehlerdetektor nicht erkennen, ob der Operator wirklich ausgefallen ist. Er nimmt nach Ablauf einer Zeitschranke an, dass der Operator ausgefallen ist, kann nun aber weiterhin nicht unterscheiden, ob dieser Befund korrekt oder falsch-positiv ist: Die Fehlererkennung ist *unsicher*.

Die Problematik der unsicheren Fehlererkennung liegt darin, dass ein Fehlerdetektor im schlechtesten Fall theoretisch *alle Operatoren* als ausgefallen einschätzt und deren Wiederherstellung anstößt. Anders gesagt, es gibt in einem asynchronen System trotz sorgfältigster Auswahl der Zeitschranken *keine Höchstzahl von als ausgefallen eingeschätzten Operatoren* (die geringer ist als die Zahl der Operatoren insgesamt). Ganz im Gegensatz dazu steht die Annahme, dass im Gesamtsystem *maximal F (sequentielle) Operatoren gleichzeitig tatsächlich ausfallen* dürfen, um das System trotzdem wiederherstellen zu können. Damit wird klar, dass der Fehlerdetektor nicht ohne weiteres jeden als ausgefallen eingeschätzten Operator sofort durch einen neugestarteten und im Sinne eines Sicherungspunktes wiederhergestellten Operatoren ersetzen darf. Stattdessen setzt das Wiederherstellungsverfahren auf eine Doppelstrategie: Wenn der Fehlerdetektor einen Operator verdächtigt, wird ein Ersatzoperator initialisiert und auf den letzten Sicherungspunkt des verdächtigten Operators wiederhergestellt. Der Operator und sein Ersatz laufen nun zunächst parallel zueinander: Die Vorgänger senden ihre Ereignisse an beide Operatoren und bekommen von beiden Operatoren Bestätigungsnachrichten, ebenso verhält es sich mit den erzeugten Ereignissen, die von beiden Operatoren an die Nachfolger weitergegeben werden. Da der Ersatzoperator auf einen (früheren) Zustand des verdächtigten Operators wiederhergestellt wurde, erzeugt er exakt denselben Ausgangsstrom, sodass die Vorgänger und Nachfolger von ihm dieselben Ereignisse und Bestätigungsnachrichten bekommen. Ist der verdächtigte Operator noch lebendig, kommen bei den Nachfolgern Ereignisse doppelt an, die durch die Sequenznummern recht einfach herausgefiltert werden können. Zudem kommen die Bestätigungsnachrichten bei den Vorgängern ebenfalls doppelt an, doch da nur die jeweils aktuellste Bestätigung zur Veränderung des Sicherungsbaumes führt, werden solche Duplikate von den Vorgängern ebenfalls ignoriert. Ein solcher „Doppelbetrieb“ ist für die Korrektheit der Ereignisverarbeitung folglich mit keinen Einschränkungen verbunden, das Verfahren funktioniert weiterhin, ohne zu Fehlern bei den Konsumenten zu führen.

Der Doppelbetrieb sollte aus Gründen der Effizienz und der Skalierbarkeit des Systems schnellstmöglich wieder enden, zumal bei einem vermuteten Ausfall des Ersatzoperators ein Dreifachbetrieb, Vierfachbetrieb, etc. entstehen kann. Um einen Doppelbetrieb zu beenden, muss einer der redundanten Operatoren durch den Fehlerdetektor entfernt werden: Entweder der Originaloperator oder der Ersatzoperator. Der Ersatzoperator kann sofort entfernt

werden, wenn sich herausstellt, dass der Originaloperator entgegen der ursprünglichen Annahmen doch nicht ausgefallen, sondern lebendig ist. Der Originaloperator kann erst dann entfernt werden, wenn der Ersatzoperator vollständig wiederhergestellt ist, d.h. nicht nur im Sinne der Ereignisverarbeitung, sondern auch im Sinne der Wiederherstellungsfähigkeit nachfolgender Operatoren (vgl. dazu Abschnitt 6.5.2). Zudem muss durch den Ersatzoperator ein *Fortschritt* in der Ereignisverarbeitung erzielt worden sein, bevor der Originaloperator entfernt wird, damit die *Lebendigkeit* des Systems *in einem globalen Kontext* sichergestellt werden kann. Solch ein Fortschritt ist dann gesichert, wenn der Ersatzoperator eine Bestätigungsnachricht von einem seiner Nachfolger erhält, die eine höhere Sequenznummer bestätigt als dieser Nachfolger zuvor beim Originaloperator bestätigt hatte. Ob der Originaloperator tatsächlich ausgefallen war, lässt sich nicht feststellen, er wird im Zweifelsfall nach der Installation und vollständigen Wiederherstellung des Ersatzoperators einfach vom System entfernt. Ein Operator wird entfernt, indem allen mit ihm verbundenen Operatoren der Befehl gegeben wird, die Verbindungen mit dem Operator zu entfernen. Anschließend kann der Host die Ressourcen anderweitig vergeben.

Allgemein betrachtet kann es vorkommen, dass der Ersatzoperator selbst ebenfalls direkt nach seiner Initialisierung ausfällt oder vom Fehlerdetektor als ausgefallen eingeschätzt wird, noch bevor er wirklich Ereignisse erzeugt und einen wie oben definierten Fortschritt im Gesamtsystem verursacht. Dann wird ein neuer Ersatzoperator eingerichtet und wenn dieser eine Zeitlang keinen Heartbeat an den Fehlerdetektor ausliefert, ein weiterer, und so weiter. Nach Definition der Ausfallcharakteristik dürfen im gesamten System maximal F Operatoren tatsächlich ausfallen, um Garantien für eine Wiederherstellung des Systems zu erhalten, d.h. wenn mehr Operatoren gleichzeitig ausfallen, kann das System in einem undefinierten Zustand geraten und weder Sicherheit noch Lebendigkeit werden garantiert. Das bedeutet, dass wenn schließlich eine Gruppe von F Ersatzoperatoren gestartet wurde, *mindestens einer* der Operatoren (oder der Originaloperator) tatsächlich lebendig sein muss, ganz egal wie der Fehlerdetektor die Lebendigkeit einschätzt. Vielleicht hat er sämtliche Zeitschranken für die Kommunikationsverzögerungen und den Jitter zu niedrig angesetzt und schätzt damit zunächst generell jeden Operator, auch die gerade erst initialisierten, als ausgefallen ein. Dieser eine Operator wird schließlich irgendwann den oben definierten Fortschritt machen und diesen an den Fehlerdetektor melden. Erst dann entscheidet sich der Fehlerdetektor dafür, jenen Operator als einzigen unter seinen „Klonen“ in der neuen Topologie zu erhalten und entfernt alle anderen Operatoren. Dadurch, dass der Fehlerdetektor auf den Fortschritt gewartet hat, bevor er so handelte, konnte er zum einen sicherstellen, dass er nur einen tatsächlich lebendigen Operator in die neue, stabilisierte Topologie aufnimmt und zum anderen, dass ein global sichtbarer Fortschritt stattfindet. Selbst wenn der so ermittelte Operator direkt danach wieder ausfällt oder als ausgefallen eingeschätzt wird und der Fehlerdetektor schließlich eine neue Gruppe von Ersatzoperatoren nach und nach initialisiert, ist durch den Fortschritt in jeder Durchführung dieses Stabilisierungsverfahrens ein Gesamtfortschritt und damit die Lebendigkeit des Systems sichergestellt.

Was könnte passieren, wenn der Fehlerdetektor nicht einen Fortschritt in dem Operator abwartet, der letztendlich als einziger in der Topologie verbleibt, sondern sofort nach der Initialisierung und vollständigen Wiederherstellung eines Ersatzoperators alle anderen

Operatoren aus dem System entfernt? Nehmen wir an, einer der Operatoren, ω, ist genau der *eine* Operator aus den maximal F Operatoren mit derselben Korrelationsfunktion, der für die Zeit der Stabilisierungsphase der Topologie durchgehend lebendig ist. ω könnte direkt nach seiner (wenn es sich nicht um den Originaloperatoren handelt) Initialisierung und Zustandswiederherstellung vom Fehlerdetektoren als ausgefallen eingeschätzt werden, weil die Heartbeat-Nachricht sich verzögert. Ein anderer Operator - nehmen wir an, es handelt sich dabei bereits um den F-ten Ersatzoperator ω_F - wird daraufhin initialisiert und wiederhergestellt, und ersetzt ω, d.h. der eigentlich lebendige Operator wird aus dem System entfernt. Da ω lebendig ist, darf ω_F direkt nach seiner Wiederherstellung (und der Bestätigung dieser beim Fehlerdetektor) ausfallen. Das Problem ist nun, dass der einzige wirklich durchgängig lebendige Operator fälschlicherweise verworfen wurde, während ein nur kurzzeitig lebendiger Operator, der keinen wahren Fortschritt für das Gesamtsystem gebracht hat, als Ersatzoperator in der „stabilisierten“ Topologie eingesetzt wurde. Dies führt im schlechtesten Fall zu einem Hin- und Herspringen zwischen verschiedenen neuen Ersatzoperatoren, die immer wieder durch andere ersetzt werden, ohne dass ein wirklicher Fortschritt stattfindet. Das System wäre nicht lebendig in einem globalen Kontext, da keine Ereignisströme aus dem Umfeld der sich immer wieder neu startenden Operatoren fließen. Es lässt sich nicht vermeiden, dass ein lebendiger Operator irgendwann einmal doch entfernt wird, aber durch die Forderung der Wiederherstellung **und** des Fortschritts kann das nur geschehen, wenn ein anderer wirklich lebendiger Operator im Sinne eines Gesamtfortschrittes ihn ersetzt. Da mindestens einer der F Operatorenklone lebendig ist und einen Fortschritt macht, muss der Fehlerdetektor nur abwarten, wo der Fortschritt geschehen ist, um einen wirklich lebendigen Operator ausfindig zu machen.

Um diese Fortschrittsanforderung umzusetzen, muss ein Operator, wenn er einem wiederhergestellten Operatoren eine Bestätigungsnachricht schickt, durch eine „Flag“ anzeigen, ob die bestätigte Sequenznummer in dieser Nachricht bereits an einen anderen Operatoren mit der selben Korrelationsfunktion gegangen ist, oder ob es sich um einen Fortschritt handelt. Eine solche Flag wird als bool'sche Variable mit dem Namen Progress der Bestätigungsnachricht angehängt.

6.5.3.4 Beispiele

Falsch-positive Ausfallerkennung mit Entfernung des Ersatzoperators In Abb. 6.7 ist eine Topologie mit 5 Operatoren (ω_i bis ω_m) abgebildet, die von einem Fehlerdetektor (FD) auf ihre Lebendigkeit überprüft werden. Das Schaubild zeigt die (falsche) Verdächtigung von ω_k, die Initialisierung eines Ersatzoperators und schließlich die Verwerfung des Ersatzoperators. Schritt für Schritt läuft die Wiederherstellung folgendermaßen ab:

1. Der FD erkennt, dass eine Heartbeat-Nachricht von ω_k sich über die Zeitschranke T hinaus verzögert hat und verdächtigt ω_k infolgedessen, ausgefallen zu sein. Ein Ersatzoperator ω_k' wird initialisiert.
2. Der Ersatzoperator ω_k' wird gestartet und schickt an seine Vorgänger ω_i und ω_j einen RecoveryRequest (vgl. Abschnitt 6.5.2.1).

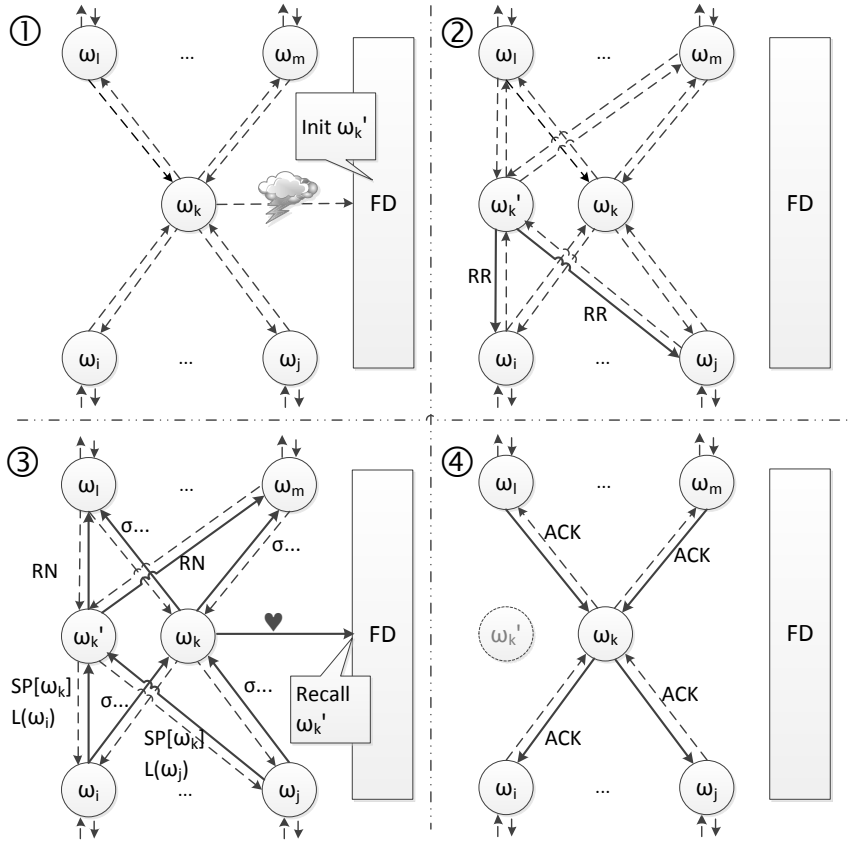

Abbildung 6.7: Der Fehlerdetektor (FD) schätzt den Operator ω_k fälschlicherweise als ausgefallen ein und korrigiert den Fehler später.

3. ω_i und ω_j antworten auf den RecoveryRequest, indem sie alle relevanten Wiederherstellungsinformationen an ihren neuen Nachfolger ω_k' senden (siehe Abschnitt 6.5.2.1). Diese Informationen beinhalten auch den jeweils aktuellen Sicherungsbaum der Operatoren und den jeweiligen relevanten Teil der Ausgangslogs. Die Verbindung zu ω_k wird dabei nicht abgebrochen: Die ausgehenden Ereignisse werden von ω_i und ω_j weiterhin ebenfalls an ω_k gesendet, die beiden Operatoren verhalten sich so, als hätten sie 2 Nachfolger. ω_k, der von der Initialisierung seines Ersatzoperators nichts mitbekommen hat, arbeitet in der Zwischenzeit ganz normal weiter und sendet seine erzeugten Ereignisse an die beiden Nachfolger ω_l und ω_m. Der Ersatzoperator ω_k' kann seine Ereignisverarbeitung mit Hilfe der von seinen Vorgängern empfangenen Information wiederherstellen und sendet dann eine RecoveryNotification an seine Nachfolger ω_l und ω_m.

   Während dieses Vorganges kommt irgendwann doch noch die Heartbeat-Nachricht des als ausgefallen eingeschätzten Operators ω_k beim Fehlerdetektor an, d.h. aus Sicht des FD muss ω_k lebendig sein. Da der FD noch keinen Bescheid vom Ersatzoperator

ω_k' über dessen *vollständige Wiederherstellung* und den *Fortschritt im Gesamtsystem* bekommen hat, entscheidet er sich dafür, ihn zurückzuziehen und stattdessen weiter ω_k im CEP-System zu nutzen.

4. Das System läuft wieder im Normalbetrieb ohne redundante Operatoren.

Anmerkung: Wenn der Fehlerdetektor den Originaloperator ω_k fälschlicherweise als ausgefallen eingeschätzt hatte und diese Einschätzung korrigiert, kann er die Zeitschranke T für den Jitter der Heartbeat-Nachrichten nach oben korrigieren, um künftige Fehleinschätzungen zu vermeiden.

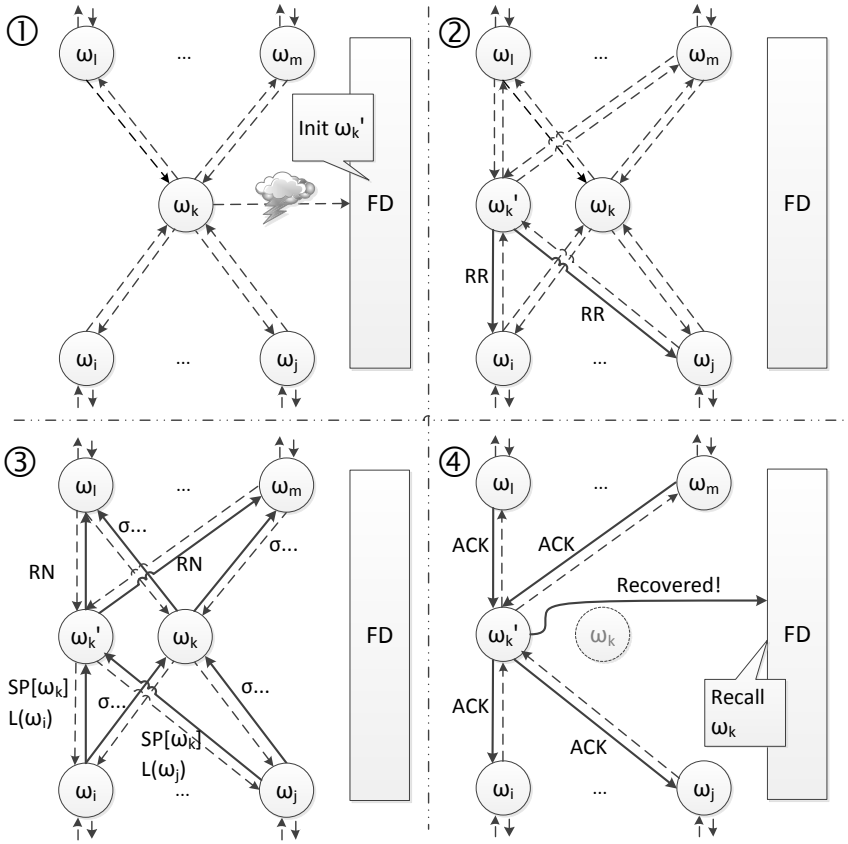

Abbildung 6.8: Der Fehlerdetektor (FD) schätzt den Operator ω_k fälschlicherweise als ausgefallen ein und übernimmt den Ersatzoperator ω_k' in die Topologie.

Falsch-positive oder korrekte Ausfallerkennung mit Entfernung des Originaloperators In Abb. 6.8 ist eine Topologie mit 5 Operatoren (ω_i bis ω_m) abgebildet, die von einem Fehlerdetektor (FD) auf ihre Lebendigkeit überprüft werden. Das Schaubild zeigt die (falsche oder korrekte) Verdächtigung von ω_k, die Initialisierung eines Ersatzoperators und schließlich die Verwerfung des Originaloperators. Ob ω_k tatsächlich ausgefallen war, spielt in dem Fall, in

dem ein Ersatzoperator in der stabilisierten Topologie eingesetzt wird, letztendlich keine Rolle mehr und es ist für den Fehlerdetektor auch nicht ersichtlich, ob er sich bezüglich ω_k geirrt hat. Schritt für Schritt läuft die Wiederherstellung folgendermaßen ab:

1. Der FD erkennt, dass eine Heartbeat-Nachricht von ω_k sich über die Zeitschranke T hinaus verzögert hat und verdächtigt ω_k infolgedessen, ausgefallen zu sein. Ein Ersatzoperator ω_k' wird initialisiert.

2. Der Ersatzoperator ω_k' wird gestartet und schickt an seine Vorgänger ω_i und ω_j einen RecoveryRequest (vgl. Abschnitt 6.5.2.1).

3. ω_i und ω_j antworten auf den RecoveryRequest, indem sie alle relevanten Wiederherstellungsinformationen an ihren neuen Nachfolger ω_k' senden (siehe Abschnitt 6.5.2.1). Diese Informationen beinhalten auch den jeweils aktuellen Sicherungsbaum der Operatoren und den jeweiligen relevanten Teil der Ausgangslogs. Die Verbindung zu ω_k wird dabei nicht abgebrochen: Die ausgehenden Ereignisse werden von ω_i und ω_j weiterhin ebenfalls an ω_k gesendet, die beiden Operatoren verhalten sich so, als hätten sie 2 Nachfolger. ω_k, der von der Initialisierung seines Ersatzoperators nichts mitbekommen hat, arbeitet in der Zwischenzeit ganz normal weiter und sendet seine erzeugten Ereignisse an die beiden Nachfolger ω_l und ω_m. Der Ersatzoperator ω_k' kann seine Ereignisverarbeitung mit Hilfe der von seinen Vorgängern empfangenen Information wiederherstellen und sendet dann eine RecoveryNotification an seine Nachfolger ω_l und ω_m.

   Im Gegensatz zum vorherigen Beispiel in Abb. 6.7 kommt von ω_k auch weiterhin keine Heartbeat-Nachricht beim Fehlerdetektor an. Die Ereignisverarbeitung über den Ersatzoperator ω_k' schreitet weiter voran, bis ihm schließlich einer der Nachfolger eine Bestätigungsnachricht sendet, die ein Ereignis mit höherer Sequenznummer bestätigt als die von diesem Nachfolger zuvor gesendeten Bestätigungen.

4. Wenn ω_k' eine Bestätigungsnachricht von einem der Nachfolger erhalten hat, die einen *Fortschritt* in der Ereignisverarbeitung für das Gesamtsystem anzeigt, benachrichtigt er den Fehlerdetektor darüber („Recovered!“). Der Fehlerdetektor, der in der Zwischenzeit immer noch keine Nachrichten von ω_k erhalten hat, entscheidet sich daraufhin, den (nachgewiesenermaßen lebendigen) Ersatzoperator ω_k' zu dem Operator zu machen, der in der stabilisierten Topologie den Platz des Originaloperators einnimmt, und nimmt ω_k aus dem System.

Anpassung der Zeitschranke T Die Zeitschranke T für die Übertragungszeit und den Jitter von Heartbeat-Nachrichten wird zu Beginn auf einen sinnvollen Wert initialisiert. Dieser Wert hängt von der Netzwerktopologie ab, die der Operatortopologie zugrunde liegt und muss vom Systemarchitekten bestimmt werden. Ein sinnvoller Wert ist ein Wert, der üblicherweiser nicht überschritten wird, der aber andererseits die Durchschnittszeit nicht zu sehr übersteigt, damit das System auf Ausfälle relativ zügig reagieren kann. Der Initialwert für T kann in einer bestimmten Situation angepasst werden: Wenn der Fehlerdetektor einen Operator fälschlicherweise verdächtigt, ausgefallen zu sein, und diese Entscheidung rückgängig macht,

d.h. den dafür initialisierten Ersatzoperator wieder aus dem System entfernt, kann T für die Kommunikation mit diesem Operator höher gesetzt werden, um diesen Fehler in Zukunft zu vermeiden. Für die Anpassung von Zeitschranken für Fehlerdetektoren gibt es in der Literatur schon verschiedene Verfahren der dynamischen Adaption, z.B für *Eventually Perfect Failure Detectors* [CT96].

6.5.3.5 Verteilung des zentralen Fehlerdetektors auf die Operatoren

Die Abstraktion des zentralen Fehlerdetektors vereinfacht die Sicht auf das Wiederherstellungsverfahren und ist daher für die theoretische Betrachtung und Analyse zweckmäßig. Wenn das CEP-System implementiert wird, kann man auch andenken, den Fehlerdetektor auf verschiedene Operatoren zu verteilen. Beispielsweise könnte jeweils einer der Vorgängeroperatoren den Heartbeat seiner Nachfolger überprüfen und dann entsprechend handeln. Allerdings kommen bei einem solchen verteilten Fehlerdetektor neue Fragestellungen auf, die vor allem mit der Asynchronität zusammenhängen. Beispielsweise müssen sich verschiedene Operatoren, die für einen Einsatz als Fehlerdetektor für einen bestimmten anderen Operatoren in Frage kommen, darauf einigen, welcher Operator diese Rolle innehat. Es muss sichergestellt sein, dass der Operator, welcher Fehlerdetektor ist, auch tatsächlich lebendig ist. Eine ungeeignete Anordnung von Operatoren, die sich gegenseitig überprüfen, kann zu Deadlock-Situationen führen. Die genaue Untersuchung der Möglichkeiten für einen verteilten Fehlerdetektor auf Fail-Silent-Operatoren ist ein spannendes Forschungsgebiet, das im Rahmen dieser Arbeit allerdings nicht vertieft werden kann.

6.5.4 Beweis der Korrektheit

In diesem Abschnitt folgt nun der Beweis, dass die Invarianten 6.3.1 und 6.3.2 aus Abschnitt 6.3.2 eingehalten werden und somit das Verfahren korrekt arbeitet.

Invariante 6.3.1, die Wiederherstellbarkeit von Operatoren, ist durch das Konzept der Sicherungspunkte und Sicherungsbäume sichergestellt. Ein Operator sendet stets einen aktuellen Sicherungspunkt an alle seine Vorgänger, diese bauen ihn in einen Sicherungsbaum ein und senden diesen an ihre Vorgänger, und so weiter, sodass der Sicherungspunkt des Operators in allen Vorgänger bis zu F Ebenen weit gespeichert ist. Selbst wenn also der Operator und F-1 Vorgänger in einer Sequenz gleichzeitig ausfallen, ist der Sicherungspunkt des Operators nach der Wiederherstellung seiner Vorgänger noch verfügbar. Somit ist der Operator auch beim gleichzeitigen Ausfall von F Operatoren zu einem Sicherungspunkt wiederherstellbar.

Invariante 6.3.2, die definiert, dass die Sicherungspunkte so gewählt sein müssen, dass jederzeit alle von den Konsumenten noch nicht bestätigten $\sigma_{consumer}$-Ereignisse wieder erzeugt werden können, ist ebenfalls jederzeit erfüllt. Dies folgt daraus, dass ja gerade die Konsumenten durch ihre Bestätigungsnachrichten (ACKs) überhaupt erst eine Aktualisierung der Sicherungspunkte anstoßen, die sich „wellenförmig“ stromabwärts über die

Operatoren fortsetzt. Durch die Berechnung der Umkehrfunktion f^{-1}, die jeweils das Startereignis des Korrelationsfensters eines bestätigten Ereignisses zurückgibt, ist sichergestellt, dass nur Ereignisse, die *älter* als die an der Erzeugung von $\sigma_{consumer}$ beteiligten Ereignisse sind, gelöscht werden.

Nach Abschnitt 6.3.2.1 folgt daraus die Korrektheit des Wiederherstellungsverfahrens. Dass die Lebendigkeit des Gesamtsystems ebenfalls sichergestellt ist (d.h. es macht nicht nur „nichts Falsches“, sondern es macht auch „das Richtige“), wird in Abschnitt 6.5.3.3 ausführlich dargelegt.

6.6 Kritik des Ansatzes

Das in diesem Kapitel eingeführte Rollback-Recovery-Verfahren ohne persistente Checkpoints bringt einige Vorteile gegenüber anderen Wiederherstellungsverfahren mit sich. Die Operatoren müssen über keinen persistenten Speicher zur Sicherung von Zustandsinformationen oder Ereignislogs verfügen und eine Sicherung dieser Informationen im volatilen Speicher verspricht schnellere Zugriffszeiten. Es müssen nicht dauerhaft redundante Operatoren vorgehalten werden, damit das System eine Ausfallsituation überstehen kann. Dies sind die substantiellen Vorteile gegenüber den gängigen Wiederherstellungsverfahren, wie sie auch in Kapitel 5 untersucht werden.

Doch es gibt auch Nachteile bei dem Verfahren, insbesondere wenn die Zahl der Operatoren und der Parameter F für die maximal verkraftbare Anzahl der gleichzeitigen Ausfälle sehr hoch sind. Es müssen mitunter sehr viele Zustandsinformationen in einem Operator gespeichert werden, denn je nach Topologie und Tiefe kann der Sicherungsbaum schnell anwachsen, im schlechtesten Fall muss in manchen Operatoren von beinahe jedem anderen Operatoren ein Sicherungspunkt gespeichert werden. Ebenso problematisch kann der Ausfall einer Sequenz von Operatoren sein: Dadurch, dass in einem solchen Fall keine parallele Wiederherstellung aller ausgefallenen Operatoren möglich ist, sondern die Operatoren sequentiell von unten nach oben wiederhergestellt werden müssen, kann es relativ lange dauern, bis alle Operatoren wieder korrekt laufen.

Im folgenden Kapitel werden diese Probleme analysiert und es wird ein Lösungsvorschlag erarbeitet.

7 Erweiterung des Ansatzes

In Abschnitt 6.6 wurden einige mögliche Schwachstellen des vorgestellten Modells angesprochen, die insbesondere die Skalierbarkeit des Systems und die Latenzzeit bei Operatorausfällen beeinträchtigen können. In diesem Kapitel soll genauer untersucht werden, in welchen Fällen sich durch die Topologie des CEP-Systems der Aufwand für Kommunikation und Berechnungen und der Speicherbedarf für die einzelnen Operatoren besonders verstärken kann. Es wird gezeigt, dass manche Topologieformen die Anzahl der übertragenen Bestätigungsnachrichten, die Größe des Ausgangslogs und das Ausmaß des Sicherungsbaumes für bestimmte Operatoren signifikant verstärken können. Durch die Nutzung *persistenter Speicherschichten* können bestimmte Operatoren besonders entlastet werden, indem die Menge an zu sichernder Zustandsinformation verringert wird. Zudem kann die Latenzzeit bis zur Wiederherstellung einer Sequenz von ausgefallenen Operatoren ebenfalls verringert werden. Es wird gezeigt, wie persistente Speicherschichten das CEP-System in verschiedene Teilsysteme *unterteilen* können, was den einzelnen Operatoren in einem solchen Teilsystem entlastet.

7.1 Problematische Topologien

7.1.1 Operatoren mit vielen Vorgängern

In Abb. 7.1 ist eine Topologie dargestellt, in der ein Operator ω_i n direkte Vorgänger ω_1 bis ω_n hat. Ein solches Muster in der Topologie hat spezielle Folgen für die Zustandsinformationen, die in den Vorgängern gespeichert werden. Die Bestätigungsnachrichten von ω_i werden n Mal repliziert und haben damit auch einen n-fachen Effekt auf die Aktualisierung von Sicherungspunkten und die Kürzung von Ausgangslogs und gespeicherten Konsumoperationen. Ein Operator mit vielen Vorgängern ist also ein *Bestätigungs-Verstärker*. Zudem wird der Sicherungsbaum von ω_i bei einer neuen Bestätigungsnachricht n mal versendet und n mal gespeichert.

Es wäre vorteilhaft, wenn ω_i einen möglichst kleinen Sicherungsbaum besitzt, um die übertragene Datenmenge in den Bestätigungsnachrichten zu minimieren und damit auch die Größe der Sicherungsbäume der Vorgänger und deren Vorgänger bis zu einer Tiefe von n-1 Verarbeitungsebenen. Zudem wäre es vorteilhaft, wenn ω_i empfangene Ereignisse möglichst schnell bestätigen kann, damit die Vorgänger (und deren Vorgänger bis zu den Ereignisquellen) ihre Ausgangslogs und gespeicherten Konsumoperationen möglichst schnell nach der Erzeugung wieder kürzen können. Je tiefer der Operatorbaum sich bis zu den

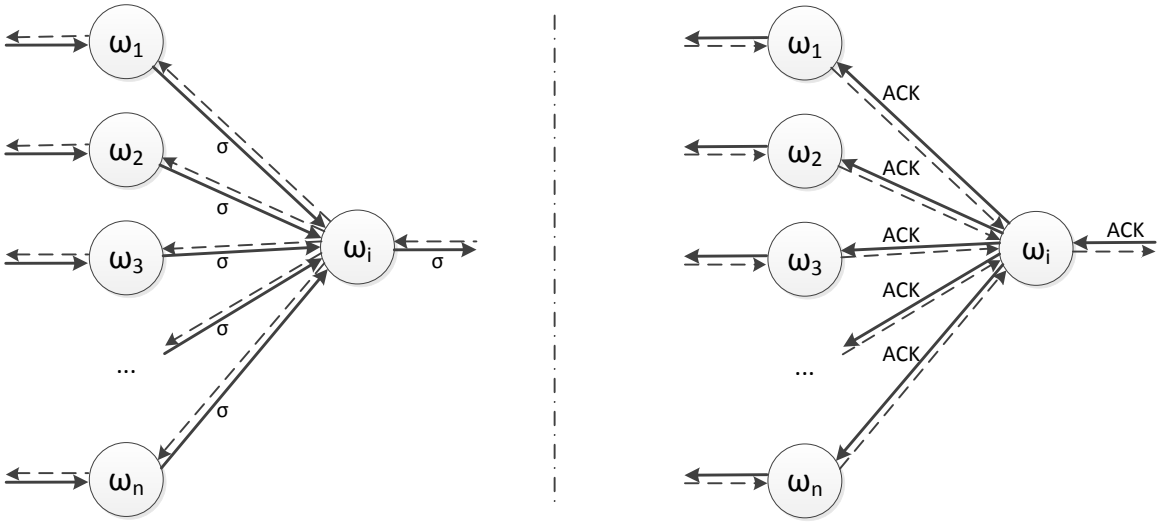

Abbildung 7.1: Eine Topologie, in der ein Operator viele Vorgänger hat.

Konsumenten zieht, desto größer werden im Allgemeinen die Mengen an für die Wiederherstellung vorgehaltenen Ereignissen in allen Zwischenstufen bis zu einem fertiggestellten und bestätigten Ereignis, das an einen Konsumenten ausgeliefert wurde. Die Tatsache, dass ein Operator viele Vorgänger hat, vervielfacht diesen Effekt.

7.1.2 Operatoren mit vielen Nachfolgern

Ein anderes Extrem in Topologien ist ein Operator, der viele direkte Nachfolger hat. In Abb. 7.2 ist eine solche Topologie dargestellt: Der Operator ω_i hat mit den Operatoren ω_1 bis ω_n n direkte Nachfolger. Ein solches Muster hat für den Operatoren ω_i und dessen Vorgänger wiederum spezielle Folgen. So setzt sich der Sicherungsbaum von ω_i aus Teilen der Sicherungsbäume aller seiner n Nachfolger zusammen, bei einem großen n kann der Sicherungsbaum also entsprechend groß werden. Dieser potentiell große Sicherungsbaum wird gekürzt auch an die Vorgänger von ω_i weitergegeben und dort gespeichert. Die Vergrößerung des Sicherungsbaumes kann sich somit bis zu F-1 Ebenen stromabwärts von ω_i aus fortsetzen, wobei sich der Effekt durch die Kürzung der Sicherungsbäume auf eine Tiefe von F nach und nach abschwächt. Eine Bestätigungsnachricht eines Nachfolgers führt nicht immer direkt zu einer Aktualisierung des Sicherungspunkts von ω_i, sondern nur wenn sich die niedrigste bestätigte Sequenznummer aller Vorgänger ändert, ein Operator mit vielen Nachfolgern kann in diesem Sinne also als ein *Bestätigungs-Aggregator* angesehen werden. Der Sicherungsbaum wird allerdings bei jeder empfangenen Bestätigungsnachricht aktualisiert.

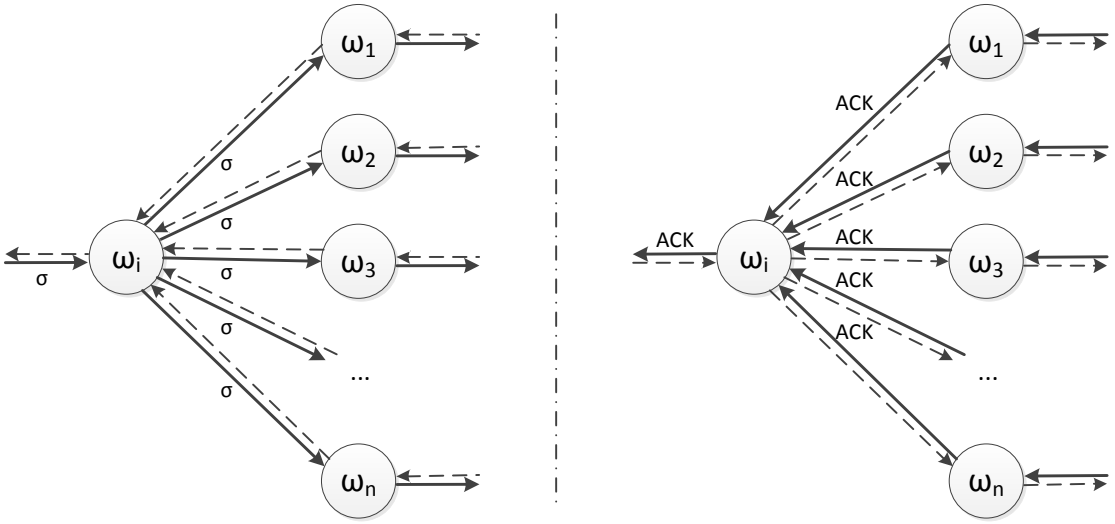

Abbildung 7.2: Eine Topologie, in der ein Operator viele Nachfolger hat.

Bei sehr vielen Nachfolgern kann es für den Operatoren ω_i schwierig werden, die Sicherungsbäume zu speichern. Wenn er dazu viele Vorgänger hat, muss er seinen eigenen großen Sicherungsbaum auch noch regelmäßig an diese versenden (vgl. Abschnitt 7.1.1). Es wäre also vorteilhaft, wenn der Sicherungsbaum von ω_i verkleinert werden könnte.

7.1.3 Viele Operatoren in einer Sequenz

Bei vielen Operatoren in einer Sequenz zwischen Ereignisquellen und -konsumenten kann sich der Speicherbedarf für das Ausgangslog eines Operators erhöhen, wenn zwischen dem Operator und den Konsumenten viele weitere Operatoren und damit viele weitere Zwischenschritte in der Ereignisverarbeitung positioniert sind. Dadurch, dass in den einzelnen Verarbeitungsschritten im Allgemeinen die Anzahl der Ereignisse eher abnimmt, da „einfache“ Ereignisse zu „komplexen“ Ereignissen verdichtet werden, sind jeweils viele Ereignisse aus niedrigen Verarbeitungsebenen an der Erzeugung eines Ereignisses aus einer hohen Verarbeitungsebene beteiligt. Das heißt auch, dass sich die Größe des Ausgangslogs im Sinne der Anzahl der gespeicherten Ereignisse im Allgemeinen stromabwärts erhöht. Mit jedem weiteren sequentiellen Operatoren vergrößert sich das Ausgangslog aller seiner Vorgänger in der transitiven Hülle *pred** sogar um den Faktor der durchschnittlichen Anzahl *ws* an neuen Ereignissen, die der Operator von jedem Vorgänger für eine Korrelation benötigt. Dies hängt damit zusammen, dass für die Produktion eines Ereignisses, das an einen Konsumenten ausgeliefert wird, dann eine *ws*-fache Menge an Ereignissen der niedrigeren Verarbeitungsebenen benötigt wird. Die Ausgangslogs können somit mit der Anzahl der sequentiellen Operatoren exponentiell anwachsen. In der Evaluation in Kapitel 8.5.1 wird

diese Überlegung durch Messungen bestätigt.

Ein weiterer Nachteil einer tiefen Topologie mit vielen Verarbeitungsebenen tritt bei der Wiederherstellung einer großen Menge von Operatoren, die in einer Sequenz liegen, auf. Um den letzten Operator in der Sequenz wiederherzustellen, müssen erst alle seine Vorgänger wiederhergestellt werden. Dies setzt sich fort bis zur Wiederherstellung der ausgefallenen Operatoren der niedrigsten Verarbeitungsebenen. Im schlechtesten Fall muss beim Ausfall von F Operatoren der Operator der höchsten Verarbeitungsebene bis zu seiner eigenen Wiederherstellung zunächst auf die Wiederherstellung von F-1 anderen Operatoren warten. Eine Parallelisierung der Wiederherstellung ist im Falle eines Ausfalls sequentiell angeordneter Operatoren nicht möglich.

7.2 Persistente Speicherschichten

Bisher sind die Operatoren und die Algorithmen für ihre Wiederherstellung gezielt so entworfen worden, dass sie ohne einen persistenten Speicher funktionieren. Dies hat mehrere Vorteile: Zum einen sind persistente Speicheroperationen vergleichsweise teuer, da sie mehr Zeit benötigen als eine Speicherung in einem volatilen Speicher. Zum anderen kann ein solcher Algorithmus auch auf Computersystemen laufen, die gar keinen beschreibbaren persistenten Speicher für die Anwendung bereitstellen.

Es kann jedoch sinnvoll sein, in bestimmten Situationen auf einen persistenten Speicher zurückzugreifen, um die Operatoren und die Kommunikationsverbindungen zu entlasten. Die Idee dahinter ist, dass eine Sicherung bestimmter Zustandsdaten von Operatoren in einem persistenten Speicher trotz der höheren einmaligen Speicherungskosten immer noch günstiger sein kann als die Replikation dieser Daten in mehreren volatilen Speichern, die auf verschiedene Operatoren verteilt sind. Wenn eine persistente Speicherung günstiger ist und ein persistenter Speicher mit ausreichender Datenverfügbarkeit vorhanden ist, sollte ein solcher Speicher verwendet werden. Außerdem kann die Skalierbarkeit erhöht werden, indem mit der Speicherung von Zustandsdaten überlastete Operatoren durch einen persistenten Speicher entlastet werden können. Was ist dann aber der Unterschied zu einem „klassischen“ Rollback-Recovery-Verfahren, in dem jeder Knoten von Zeit zu Zeit einen persistenten Checkpoint anlegt? Es ist die Flexibilität: Eine persistente Speicherschicht *kann* eingesetzt werden, sie muss es nicht. Vielmehr kann in jeder Topologie an jeder Stelle die Abwägung getroffen werden, ob ein persistenter Speicher oder die Verteilung der Zustandsinformationen auf volatile Speicher die günstigere Alternative ist. Eine Kostenrechnung führt zur Entscheidung für oder gegen einen persistenten Speicher an bestimmten Stellen in der Topologie des Systems. Schließlich könnte diese dann in einem dynamischen Algorithmus angewendet werden, der die automatisierte Verwaltung von persistenten Speicherschichten ermöglicht.

In diesem Abschnitt wird das Konzept des Einsatzes eines persistenten Speichers im Rahmen des Systemmodells erläutert.

7.2.1 Konzept und Funktionsweise

Der persistente Speicher arbeitet wie ein weiterer Operator, der zwischen zwei oder mehr miteinander verbundene Operatoren geschaltet wird. Die entsprechenden Operatoren kommunizieren dann nicht mehr direkt miteinander, sondern mit dem persistenten Speicher. Dieser speichert die übertragenen Daten, d.h. Ereignisse von seinen Vorgängern und Sicherungsbäume von seinen Nachfolgern. Die Ereignisse werden nach der Speicherung an die Nachfolger zur Sequenzierung in deren Eingangsströmen weitergeleitet, die Sicherungsbäume hingegen werden persistent in ihrer jeweils aktuellsten Version gespeichert, aber *nicht* vollständig an die Vorgänger zur volatilen Speicherung weitergeleitet. Statt der Übertragung der kompletten Sicherungsbäume der Tiefe F werden in den Bestätigungsnachrichten an die Vorgänger lediglich die empfangenen und persistent gespeicherten Ereignisse bestätigt. Damit arbeitet ein persistenter Speicher wie eine *Kombination aus Ereigniskonsument und*

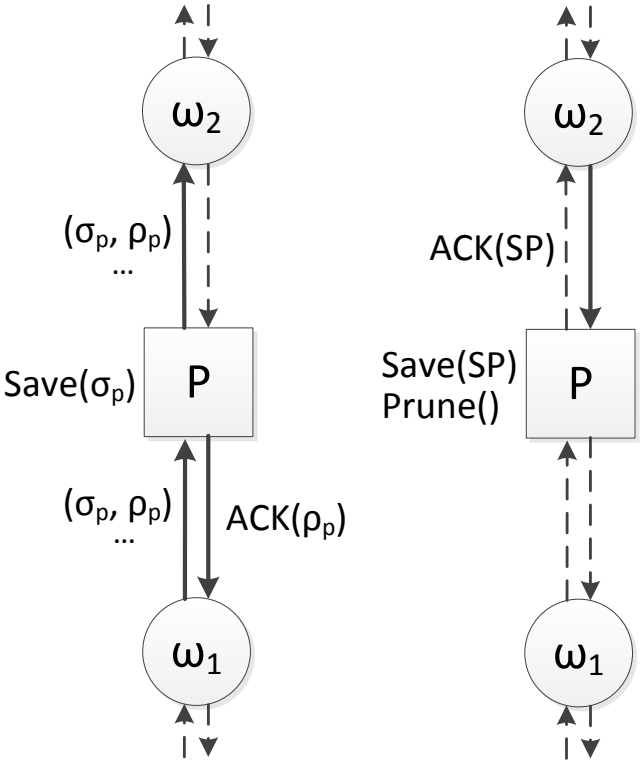

Abbildung 7.3: Funktionsweise eines persistenten Speichers in der Kommunikation mit 2 Operatoren.

-quelle. Gegenüber den Vorgängern des persistenten Speichers gibt er sich als Konsument aus, der empfangene und persistent gespeicherte Ereignisse mit einfachen Nachrichten bestätigt. Gegenüber seinen Nachfolgern liefert er diese Ereignisse wie eine Ereignisquelle aus,

speichert und verwaltet die in den Bestätigungsnachrichten empfangenen Sicherungsbäume und bereinigt die Ausgangslogs.

Abb. 7.3 zeigt einen persistenten Speicher P, der zwischen zwei ursprünglich miteinander verbundene Operatoren ω_1 und ω_2 geschaltet wurde. Ereignisse, die von ω_1 an P gesendet werden, werden von P persistent gespeichert und dann mit einer einfachen Bestätigungsnachricht (Bestätigung der Sequenznummer) beim Produzenten bestätigt. Insofern verhält sich P gegenüber ω_1 wie ein Ereigniskonsument. Die von P persistent gespeicherten Ereignisse werden an den Empfänger ω_2 gesendet. Dieser wird die Ereignisse weiterverarbeiten, bis sie schließlich als $\sigma_{consumer}$-Ereignisse an einen Konsumenten ausgeliefert und von diesem bestätigt werden. Die Bestätigungen werden stromabwärts weiterverarbeitet und führen zum Aufbau eines Sicherungsbaums der maximalen Tiefe F. Schließlich sendet ω_2 eine Bestätigungsnachricht an P, die einen Sicherungsbaum enthält. P speichert bzw. aktualisiert den Sicherungsbaum, um ω_2 und dessen Nachfolger wiederherstellen zu können, und kann nun anhand des Sicherungspunktes von ω_2 sein eigenes persistentes Log von Ereignissen kürzen, indem alle Ereignisse mit einer niedrigeren Sequenznummer als das zuletzt von ω_2 bestätigte Ereignis aus dem persistenten Speicher entfernt werden.

8 Evaluation

Basierend auf dem eingeführten Systemmodell (siehe Kapitel 3) wurde das entwickelte Wiederherstellungsverfahren (siehe Kapitel 6) im ereignisbasierten Simulations-Framework OMNeT++ [Omn] implementiert. Anhand dieser Implementierung wird eine Evaluation des Systemverhaltens unter verschiedenen Simulationsparametern durchgeführt. Ziel der Evaluation ist es vor allem, einige der in Kapitel 7 analysierten Fragen und Annahmen über das Systemverhalten durch eine Untersuchung der gewonnen Messdaten zu hinterfragen und somit auch zu Schlussfolgerungen über den Einsatz von persistenten Speicherschichten zu kommen. Zudem wird durch die praktische Implementierung der Algorithmen und der Prüfung der Ergebnisse das Wiederherstellungsverfahren validiert.

Ein grober Überblick über die Implementierungsdetails wird in Abschnitt 8.1 gegeben. In Abschnitt 8.2 werden die Topologien definiert, auf denen die Simulationsläufe ausgeführt werden und es wird erklärt, wie die Simulation im Einzelnen abläuft. Schließlich werden in Abschnitt 8.3 die Szenarien bzw. Parameter definiert, unter denen die Simulation abläuft, bevor in Abschnitt 8.4 die entsprechenden Ergebnisse in grafischen Schaubildern dargestellt werden. Das Kapitel schließt in Abschnitt 8.5 mit Schlussfolgerungen darüber, was die erzielten Ergebnisse für den Einsatz von persistenten Speicherschichten im Kontext des Wiederherstellungsverfahrens bedeuten.

8.1 Details zur Implementierung

In Abb. 8.1 wird die Architektur der Implementierung vereinfacht dargestellt. Ein zentraler Aspekt ist hierbei der sog. `Broker`, der einen Host implementiert, auf dem jeweils **ein** Operator platziert werden kann. Diese Broker laufen im Gegensatz zur Definition in Kapitel 3.3.2 zur Vereinfachung des Verfahrens in einem Crash-Recovery-Fehlermodell, d.h. bei einem Absturz verliert der Broker alle Zustandsinformationen, fährt allerdings automatisch wieder hoch und steht zur Wiederherstellung des platzierten Operators zur Verfügung. Somit sind Rekonfigurationen der Operatortopologie unnötig, die Evaluation zielt ausschließlich auf die Wiederherstellung der Ereignisströme ab (vergleiche dazu auch den Aufbau in Kapitel 6.5).

Quellen und Konsumenten sind über Kommunikationsverbindungen jeweils mit den Brokern verbunden, die in OMNeT++ über `gates` und `connections` realisiert werden. Die Ströme von Ereignissen und Bestätigungsnachrichten laufen wie in Kapitel 6.4 erläutert ab.

Der `Coordinator` überwacht die Heartbeat-Nachrichten der Broker und veranlasst und

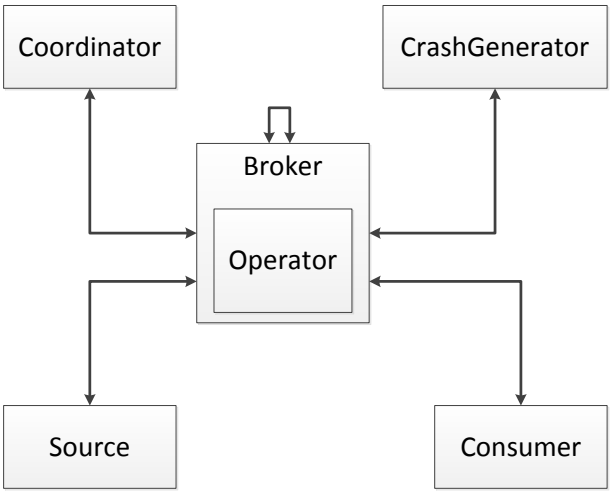

Abbildung 8.1: Architektur der Implementierung. Die Rechtecke stellen Komponenten dar, die Pfeile Kommunikationsverbindungen.

koordiniert die Wiederherstellung von abgestürzten Brokern. Ob und wann ein Broker abstürzt, wird von der Komponente `CrashGenerator` gesteuert, der periodisch an verschiedene Broker `CrashRequests` verschickt. Über diese Komponente lassen sich die Frequenz und die Wahrscheinlichkeit von Abstürzen steuern.

8.2 Topologien und Simulationsablauf

Abbildung 8.2: Topologien in der Evaluation.

Bei der Evaluation in dieser Arbeit wird keine „praxistypische“ Topologie untersucht, sondern ein sog. *kritischer Pfad*, anhand dessen die Verhaltensweisen des Systems in bestimmten Situation beobachtet werden können. Das heißt, dass die untersuchten Topologien immer nach dem folgenden Muster aufgebaut sind (Abb. 8.2): Eine Ereignisquelle ist mit einer Sequenz von 1 bis N Operatoren verbunden und der letzte Operator in dieser Reihe ist schließlich mit einem Konsumenten verbunden. Eine solche einfache Topologie lässt direkte Schlüsse auf Verhältnismäßigkeiten zwischen Parametereinstellungen und gemessenen Daten zu.

In der Simulation werden verschiedene Konstellationen von Parametern getestet. Dabei werden stets alle Parameter bis auf einen festgehalten, und die Änderungen im Systemverhalten bei verschiedenen Werten für den veränderbaren Parameter untersucht. Bei der Entscheidung des CrashGenerator über Abstürze von Brokern wird auf einen Zufallszahlengenerator zurückgegriffen, was ausschließlich Szenario 3 (siehe Abschnitt 8.3.3.3) betrifft. Hierbei wird eine Gleichverteilung des *Mersenne Twister Random Number Generator* [MN98] eingesetzt, der im OMNet++-Framework implementiert ist. Für jede Parameterkonfiguration wird eine Reihe von 5 Simulationsdurchläufen durchgeführt, bei denen jeweils verschiedene Seeds für den RNG eingesetzt werden. Die Ergebnisse der Simulationsläufe werden dann in einem Konfidenzintervall dargestellt. In Szenario 1 und 2 (siehe Abschnitte 8.3.3.1 und 8.3.3.2) wird nicht auf Zufallswerte zugegriffen, sodass dort lediglich ein Simulationslauf durchgeführt wird. Für alle Szenarien gilt, dass in jedem Simulationslauf 1.000.000 Ereignisse in der Ereignisquelle produziert und durch die Operatortopologie verarbeitet werden.

Die Operatoren sind so eingestellt, dass sie immer eine bestimmte Anzahl von eingehenden Ereignissen in einem Korrelationsfenster sammeln. Sobald die festgelegte Zahl an Ereignissen eingegangen ist, wird das Korrelationsfenster geschlossen, die Korrelation wird durchgeführt und es wird ein ausgehendes Ereignis produziert. Das nächste Korrelationsfenster wird dann mit dem ersten eingehenden Ereignis nach dem Endereignis des vorherigen Fensters geöffnet. Damit haben alle Korrelationsfenster in den Operatoren die selbe Größe und überlappen nicht.

8.3 Getestete Szenarien und Parameter

8.3.1 Parameter

Auf Basis der beiden vorhergehenden Abschnitte, in denen die Architektur der Implementierung und der Aufbau der Operatortopologie in der Simulation beschrieben werden, lassen sich verschiedene Parameter feststellen, anhand deren die Simulation in ihren wesentlichen Aspekten gesteuert wird. Einige der Parameter werden in allen Parameterkonfigurationen auf einen festen Wert eingestellt sein, andere wiederum sind veränderlich, um bestimmte Verhaltensweisen des Systems aufzuzeigen. In Tabelle 8.1 folgt eine Übersicht über die wesentlichen Parameter; die veränderbaren Parameter bekommen dabei einen speziellen Bezeichner, der in den Auswertungen zur Referenzierung benutzt wird.

8.3.2 Was wird gemessen?

Gemessen wird die Lebensdauer von Ereignissen in den Quellen, das heißt der Zeitraum zwischen der Erzeugung eines Ereignisses (und damit der Speicherung im Ausgangslog der Quelle) und dem Empfang der Bestätigungsnachricht, die zur Entfernung des Ereignisses aus dem Ausgangslog führt. Über die gemessenen Zeiträume lassen sich direkte Rückschlüsse

Bezeichner	Name	Wertebereich	Beschreibung
	Verzögerung	fest: 20 ms	Verzögerung der Kommunikationsverbindungen. Sämtliche Kommunikationsverbindungen haben diesen Verzögerungswert fest eingestellt.
	Anzahl Quellen	fest: 1	Anzahl der Ereignisquellen.
	Anzahl Konsumenten	fest: 1	Anzahl der Ereigniskonsumenten.
	Frequenz der Quellen	fest: 1 Event / ms	Frequenz, in der Ereignisquellen neue Ereignisse produzieren.
ws	Korrelationsfenstergröße	variabel: 5 bis 40	Anzahl von Ereignissen in einem Korrelationsfenster.
pl	Pfadlänge	variabel: 1 bis 5	Anzahl der in Sequenz geschalteten Operatoren zwischen Quelle und Konsumenten.
	Heartbeat-Frequenz	fest: 1 / 2000 ms	Frequenz der Heartbeat-Nachrichten (bestimmt auch Verzögerung der Ausfallerkennung).
P(crash)	Absturzwahrscheinlichkeit	variabel: 0,00 bis 0,40	Wahrscheinlichkeit für einen Operator, während einer Entscheidung des CrashGenerator abzustürzen.
	Frequenz des CrashGenerator	fest: 1 / 1000 ms	Frequenz, in welcher der CrashGenerator über Abstürze von Operatoren (jeweils mit P(crash)) entscheidet.
	Maximale Anzahl der Abstürze	fest: Pfadlänge	Es können in jeder Entscheidung des CrashGenerator sämtliche Operatoren abstürzen (falls P(crash) > 0 ist).
	Simulationslaufzeit	fest: 1000 s	Die Simulation wird nach Ablauf der Simulationslaufzeit abgeschlossen und die Ergebnisse werden ermittelt.
	Verarbeitungsdauer	fest: 0 ms	Korrelationen werden sofort durchgeführt, eine Verzögerung durch benötigte Berechnungszeit wird nicht einberechnet.

Tabelle 8.1: Simulationsparameter

auf die Größe des Ausgangslogs in den Quellen ziehen. Es wird stets die minimale und die maximale Lebensdauer über alle Ereignisse in der Ereignisquelle gemessen sowie ein aus allen Messungen des Simulationslaufes gebildeter Mittelwert berechnet. Die Werte werden dann über alle Simulationsläufe gemittelt. Die maximale Lebenszeit aller Ereignisse bestimmt die Größe, die das Ausgangslog einer Ereignisquelle annehmen kann. Durch die Produktionsfrequenz der Quellen von genau 1 Event / ms sind die Lebensdauer eines Ereignisses in ms und Anzahl der gespeicherten Ereignisse im Ausgangslog identisch.

8.3.3 Parametereinstellungen in den verschiedenen Szenarien

8.3.3.1 Szenario 1: Variable Pfadlänge

Das erste Szenario untersucht den Einfluss der Pfadlänge *pl* auf die Lebensdauer der Ereignisse in den Ausgangslogs der Ereignisquelle. Die variablen Parameter *ws* und *P(crash)* sind auf folgende Werte eingestellt:

ws = 10
P(crash) = 0,00

Der Einfluss der Pfadlänge auf die Lebensdauer der Ereignisse in den Ausgangslogs der Ereignisquelle wird durch theoretische Betrachtungen als signifikant eingeschätzt. Jeder zusätzliche Operator und damit eingehender Korrelationsschritt vergrößert die Zahl der Ereignisse aus den Ereignisquellen, die zu einem an den Konsumenten ausgelieferten Ereignis korreliert werden. Damit ist auch ein Einfluss auf die Lebensdauer der Ereignisse in der Ereignisquelle zu erwarten.

8.3.3.2 Szenario 2: Variable Größe der Korrelationsfenster

Das zweite Szenario untersucht den Einfluss der Größe der Korrelationsfenster *ws* auf die Lebensdauer der Ereignisse in den Ausgangslogs der Ereignisquelle. Die variablen Parameter *pl* und *P(crash)* sind auf folgende Werte eingestellt:

pl = 3
P(crash) = 0,00

Die Größe der Korrelationsfenster verändert die Menge der Ereignisse aus der Ereignisquelle, die für ein korreliertes und an den Konsumenten ausgelieferten Ereignis benötigt werden. Damit ist auch ein Einfluss auf die Lebensdauer der Ereignisse in der Ereignisquelle zu erwarten.

8.3.3.3 Szenario 3: Variable Absturzwahrscheinlichkeit

Das dritte Szenario untersucht den Einfluss der Absturzwahrscheinlichkeit *P(crash)* auf die Lebensdauer der Ereignisse in den Ausgangslogs der Ereignisquelle. Die variablen Parameter *ws* und *pl* sind auf folgende Werte eingestellt:

ws = 10
pl = 3

Abstürze und Wiederherstellungen von Operatoren haben zwar keinen Einfluss auf die Menge der Ereignisse aus der Ereignisquelle, die zu einem an den Konsumenten ausgelieferten Ereignis korreliert werden, doch durch die Dauer des Wiederherstellungsverfahrens wird zwischen der Erzeugung und der Bestätigung von Ereignissen eine zusätzliche Verzögerung verursacht. Diese beinflusst die Lebensdauer der Ereignisse in den Ereignisquellen.

8.4 Ergebnisse

Die folgenden Abbildungen stellen die Simulationsergebnisse der verschiedenen Szenarien dar. Unter den Schaubildern ist jeweils eine Tabelle mit den genauen Messergebnissen angegeben. Die Ergebnisse der Simulation werden in Abschnitt 8.5 analysiert.

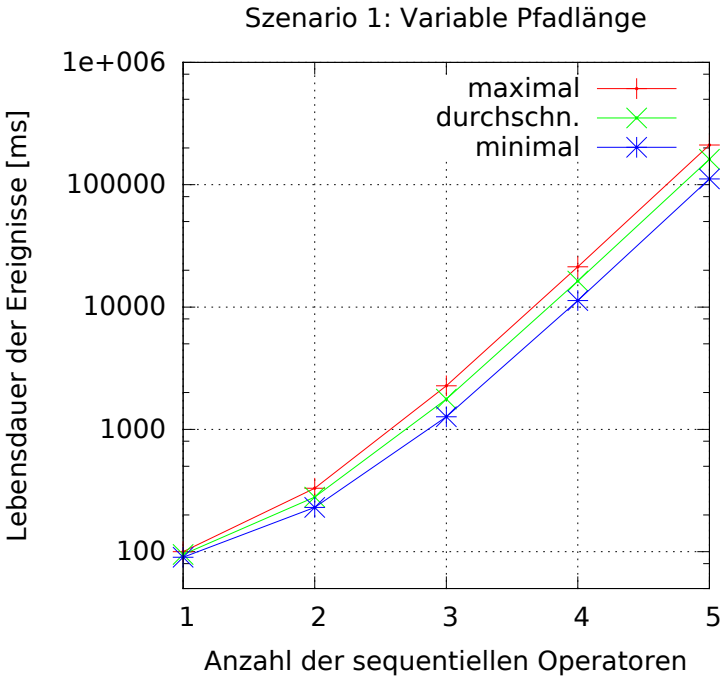

Abbildung 8.3: Szenario 1: Variable Pfadlänge mit ws = 10 und P(crash) = 0,00.

Pfadlänge	min. Lebensdauer [ms]	max. Lebensdauer [ms]	durchsch. Lebensdauer [ms]
1	90	100	95
2	230	330	280
3	1270	2270	1770
4	11310	21310	16310
5	111350	211350	161350

Tabelle 8.2: Szenario 1: Messergebnisse

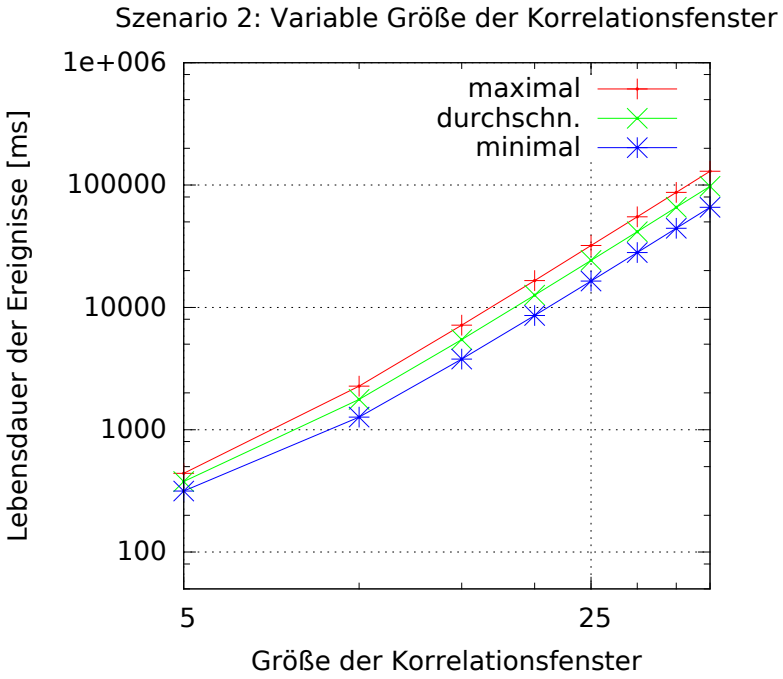

Abbildung 8.4: Szenario 2: Variable Größe der Korrelationsfenster mit pl = 3 und P(crash) = 0,00.

Größe d. Korrelations-fensters	min. Lebensdauer [ms]	max. Lebensdauer [ms]	durchsch. Lebens-dauer [ms]
5	315	440	377.5
10	1270	2270	1770
15	3775	7150	5462.5
20	8580	16580	12580
25	16435	32060	24247.5
30	28090	55090	41590
35	44295	87170	65732.5
40	65800	129800	97800

Tabelle 8.3: Szenario 2: Messergebnisse

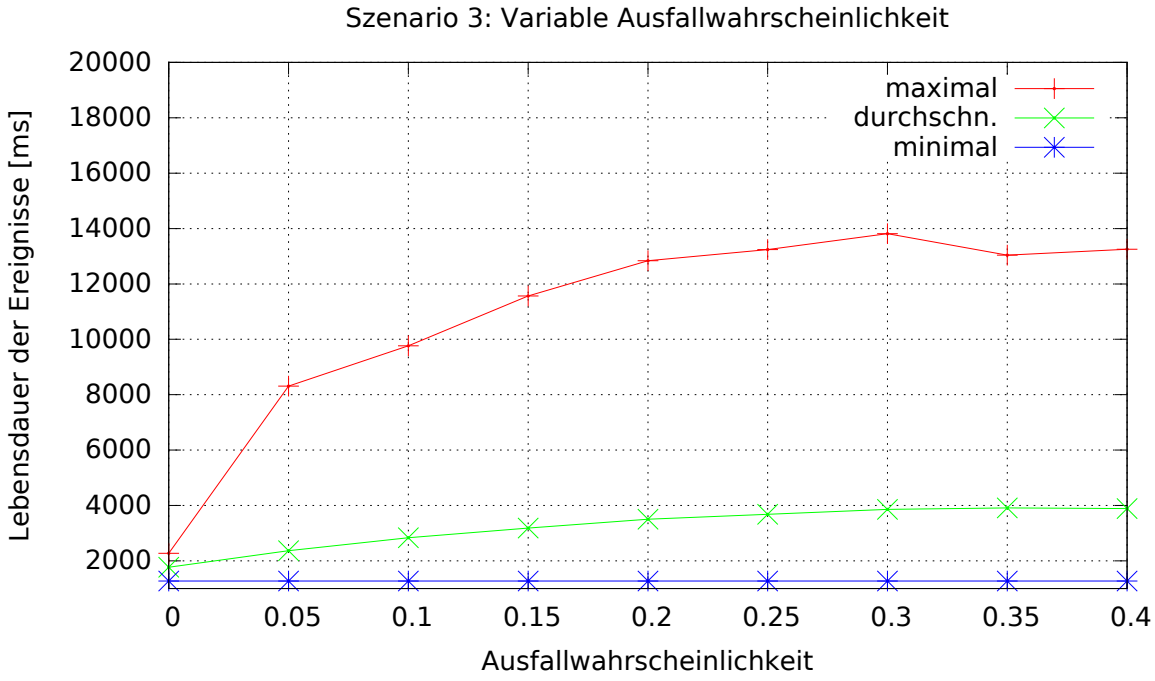

Abbildung 8.5: Szenario 3: Variable Ausfallwahrscheinlichkeit mit ws = 10 und pl = 3.

P(crash)	min. Lebensdauer [ms]	max. Lebensdauer [ms]	durchsch. Lebensdauer [ms]
0.00	1270	2270	1770
0.05	1270	8310	2365
0.10	1270	9766	2837
0.15	1270	11566	3182
0.20	1270	12838	3502
0.25	1270	13238	3679
0.30	1270	13814	3861
0.35	1270	13038	3915
0.40	1270	13250	3887

Tabelle 8.4: Szenario 3: Messergebnisse

8.5 Analyse und Schlussfolgerungen

8.5.1 Einfluss der Pfadlänge

Wie in Abb. 8.3 zu erkennen ist, steigt die Lebensdauer der Ereignisse mit der Anzahl der Operatoren exponentiell an. Für jeden Operatoren, der im sequentiellen Verarbeitungspfad zwischen Quelle und Konsumenten hinzugefügt wird, steigen sowohl die minimale, als auch die durchschnittliche und die maximale Lebensdauer der Ereignisse annähernd um den Faktor 10. Dieser Faktor entspricht genau der Größe der Korrelationsfenster. Dies entspricht exakt den Erwartungen, die in Kapitel 7.1.3 an das Systemverhalten gestellt wurden. Ein Operator mit der Korrelationsfenstergröße von *ws* verlängert also die Lebensdauer der Ereignisse annähernd um den Faktor *ws*, und zwar gleichermaßen die minimale, die maximale und die durchschnittliche Lebensdauer. Der Faktor nähert sich dabei *ws* lediglich an, da sich die Gesamtverzögerung neben den Korrelationen und der damit verbundenen Wartezeit auch noch aus den Verzögerungen der Kommunikationsverbindungen zusammensetzt. Diese Parameter multiplizieren sich nicht, sondern addieren sich lediglich, sodass der Faktor *ws* lediglich auf die Wartezeit für Korrelationen angerechnet werden kann.

Eine weitere Beobachtung ist, dass sich die Unterschiede zwischen der minimalen und der maximalen Lebensdauer mit jedem weiteren Operatoren ebenso verzehnfachen und auf das Verhältnis 1 zu 2 zulaufen. Dies erklärt sich dadurch, dass im optimalen Fall ein produziertes Ereignis im nächsten Operatoren sofort zu einer Korrelation und damit der Erzeugung eines neuen Ereignisses führt, dieses Ereignis wiederum beim nächsten Operatoren zur Korrelation führt, und so weiter, d.h. das Ereignis führt direkt ohne Wartezeiten auf weitere Ereignisse zur Erzeugung eines Ereignisses, das vom Konsumenten bestätigt wird. Die Bestätigungen laufen nun stromabwärts vom Konsumenten bis zur Quelle. Jedoch wird in jedem Operator jeweils das Startereignis des Korrelationsfenster bestätigt, das zur Erzeugung eines bestätigten Ereignisses geführt hat, da das Startereignis des *nächsten* Korrelationsfensters noch unbekannt ist (vergleiche dazu Kapitel 6.4.1.3). Dadurch verzögert sich die Bestätigung des Ereignisses, das ja eigentlich direkt zur Korrelation geführt hat, in jedem Operator um den Faktor *ws*. Im schlechtesten Fall wurde das Ereignis in der Quelle direkt nach dem Ereignis im gerade beschriebenen optimalen Fall erzeugt. Dann muss zunächst ein komplettes weiteres komplexes Ereignis korreliert werden, um das Ereignis in der Quelle freigeben zu können, was asymptotisch doppelt so lange dauert. Das Durchschnittsereignis liegt genau in der Mitte zwischen dem optimalen und dem schlechtesten Fall, und hat daher genau eine mittlere Lebensdauer zwischen beiden Extremen.

Die Pfadlänge führt also zu einem exponentiellen Wachstum der Größe der Ausgangslogs in Abhängigkeit zur Korrelationsfenstergröße. Daher sind persistente Schichten wie in Kapitel 7.2 besprochen bei langen kritischen Pfaden insbesondere sinnvoll, um die Größe der Ausgangslogs zu vermindern. Ab einer bestimmten Pfadlänge wird durch das exponentielle Wachstum die Belastung für die unteren Verarbeitungsschichten so hoch, dass an einer Partitionierung der Topologie durch persistente Schichten kein Weg mehr vorbei führt.

8.5.2 Einfluss der Größe der Korrelationsfenster

In Abb. 8.4 ist der Einfluss der Korrelationsfenstergröße auf die Lebensdauer der Ereignisse dargestellt. Beide Achsen sind exponentiell skaliert. Es ist zu beobachten, dass eine Verdoppelung der Fenstergröße asymptotisch zur Verachtfachung der Lebensdauer führt. Da im Experiment eine Pfadlänge von 3 festgelegt wurde, deckt sich dies mit der Beobachtung aus dem ersten Szenario (vgl. Abschnitt 8.5.1), dass jeder Operator im kritischen Pfad zu einer Multiplikation der Lebensdauer der Ereignisse um den Faktor der jeweiligen Fenstergröße führt. Wenn wl also um den Faktor x verändert wird und die Pfadlänge pl konstant bleibt, ist die neue Lebensdauer näherungsweise $(wl * x)^{pl}$, analog wie in Szenario 1 gleichermaßen für die minimale, die durchschnittliche und die maximale Lebensdauer. Die Beobachtungen aus dem zweiten Szenario stützen also die Ergebnisse der Analyse des ersten Szenarios.

8.5.3 Einfluss von Ausfällen

Im dritten Szenario wurde die Ausfallwahrscheinlichkeit der Broker verändert, was zu einer Entwicklung der Lebensdauer wie in Abb. 8.5 dargestellt geführt hat. Die minimale Lebensdauer liegt konstant bei einem Wert von 1270 ms, da auch bei häufigen Ausfällen letztendlich ein optimaler Durchlauf erreicht werden kann, in dem keine Ausfälle die Verarbeitung und Bestätigung des Ereignisses stören. Interessanter ist die maximale Lebensdauer, die schon bei einer Ausfallwahrscheinlichkeit von 0,05 signifikant auf einen knapp vierfachen Wert ansteigt. Der Maximalwert steigt immer weiter, bis er sich bei einer Ausfallwahrscheinlich um 0,30 stabilisiert bzw. sogar wieder leicht abfällt. Dies hängt mit der Art und Weise zusammen, wie in dem Szenario die Ausfälle erzeugt werden. Einmal alle 1000 ms wird von der `CrashGenerator`-Komponente der Ausfall von Brokern ausgelöst. Das heißt auch, dass es immer wieder Zeiträume gibt, in denen auf keinen Fall ein Ausfall geschieht, sodass alleine dadurch schon die maximale Lebenszeit der Ereignisse begrenzt ist. Die größte Verzögerung tritt dann auf, wenn die Broker nicht gleichzeitig, sondern zeitversetzt ausfallen: Während ein Broker wiederhergestellt wird, fällt ein anderer aus, sodass der wiederhergestellte Broker wiederum warten muss, und so weiter. Wenn die Broker gleichzeitig ausfallen, werden sie sequentiell wiederhergestellt und sind danach für eine gewisse Zeit wieder alle lebendig, bis die nächste Ausfallrunde beginnt. Wenn die Ausfallwahrscheinlichkeit sehr hoch ist, kommt dieser Fall häufiger vor und es wird unwahrscheinlicher, dass die Broker zeitversetzt ausfallen.

Abgesehen von dieser Einschränkung ist zu beobachten, dass durch Ausfälle und Wiederherstellungen von Operatoren die maximale Lebensdauer von Ereignissen bis annähernd einem Faktor von 6 erhöht werden konnte. Dies zeigt, dass zwar auch Wiederherstellungen von Operatoren eine signifikante Vergrößerung der Ausgangslogs zur Folge haben können, diese Vergrößerung im Vergleich zu anderen Faktoren aber nicht so stark ins Gewicht fällt. Dass ein Operator bzw. sein Host in jeder Sekunde mit einer Wahrscheinlichkeit von bis zu 40 Prozent ausfällt, ist auch schon sehr hoch gegriffen und wird in der Realität natürlich so nicht auftreten. Die eigentlichen Faktoren, die die Ausgangslogs exponentiell

anwachsen lassen, sind nicht die Ausfälle von Operatoren, sondern die Pfadlänge und die Korrelationsfenstergröße. Dies ist die Beobachtung aus dem dritten Szenario.

8.5.4 Einfluss von Sicherungsbäumen und Kommunikationsaufwand

Messungen während der Simulationen haben gezeigt, dass den Sicherungsbäumen nur ein kleiner Anteil an der Gesamtmenge der gespeicherten Zustandsinformationen zugeschrieben werden kann. Wenn bisweilen mehr als 200000 Ereignisse im Ausgangslog einer Quelle persistent gespeichert werden müssen, kommt es nicht mehr unbedingt auf die zusätzlichen Sicherungsbäume der Nachfolgeroperatoren an, es sei denn, die Topologie bestünde aus zehntausenden von Knoten. Auch in der Versendung der Bestätigungsnachrichten spielt die Größe der angehängten Sicherungsbäume keine besonders herausragende Rolle, da im Vergleich zur Zahl der versendeten Ereignisse nur sehr wenige Bestätigungsnachrichten erzeugt werden. Bei den vorliegenden Zahlen und insbesondere in Anbetracht des exponentiellen Wachstums sind die Ausgangslogs der entscheidende Faktor für den Entwurf der Operatortopologie und die Entscheidung über die Nutzung persistenter Schichten nach Kapitel 7.

9 Zusammenfassung und Ausblick

Aufgrund der Anforderungen, die aus verschiedenen Anwendungsgebieten an CEP-Systeme gestellt werden, ist vor allem in Hinblick auf die Skalierbarkeit eine verteilte Architektur meist der sinnvollste Ansatz. Es wird für mancherlei Anwendungsfälle ein hochskalierbares und ausfallsicheres System benötigt, das sich zudem durch eine hohe Verarbeitungsgeschwindigkeit auszeichnet, das fehlertolerant ist und dessen Ergebnisse unter allen einplanbaren Umständen, d.h. auch in definierten Fehlersituationen, korrekt sind. Auf Basis dieser Anforderungen wurde in dieser Diplomarbeit ein Verfahren zur korrekten Wiederherstellung von Ereignisströmen entwickelt und evaluiert.

Auf dem Weg dorthin wurden zunächst die in der Forschung bisher vorhandenen Lösungsansätze zur Gewährleistung der Ausfallsicherheit verteilter Systeme untersucht. Es wurde gezeigt, welche Stärken und Schwächen diese Ansätze in ihrer Anwendung für verteilte CEP-Systeme besitzen und es wurde der Bedarf nach einem neuen Verfahren begründet, das ohne eine persistente Zustandsspeicherung in jedem Berechnungsknoten bzw. Operatoren und ohne redundante Berechnungen auskommt. Ein solches Verfahren wurde als *Rollback-Recovery ohne persistente Zustandsspeicher* bezeichnet.

Um eine hinreichend redundante volatile Sicherung der Zustandsinformationen eines Operatoren zu ermöglichen, wurde untersucht, aus welchen Teilen ein solcher Zustand besteht. Durch ein geeignetes Verarbeitungsmodell, das auf sogenannten Korrelationsfenstern aufbaut, konnte erreicht werden, dass der Zustand der Korrelationsoperationen zu bestimmten Zeitpunkten leer ist, was die Menge der wiederherzustellenden Daten signifikant reduzieren konnte. Die Zustandsinformationen zu solchen Zeitpunkten wurden als *Sicherungspunkte* definiert und es wurde anhand verschiedener Operatortopologien Schritt für Schritt ein Verfahren entwickelt, um diese Sicherungspunkte redundant in den volatilen Speichern der Operatoren zu verteilen. Zudem wurde ein Algorithmus entwickelt, nach dem ein Operator aus den entsprechend verteilten Zustandsinformationen zu einem bestimmten Sicherungspunkt wiederhergestellt werden kann. Das entwickelte Wiederherstellungsverfahren führt auch nach dem Ausfall und der Wiederherstellung beliebig vieler Operatoren zu korrekten Ergebnissen, es arbeitet mit einer effizienten Verarbeitung und Verteilung der Zustandsinformationen der Operatoren, es ist skalierbar, hat eine hohe Ausdrucksstärke bezüglich der möglichen Korrelationen und benötigt keinen persistenten Speicher in den Operatoren.

Schließlich wurde untersucht, welchen Einfluss verschiedene Operatortopologien auf die Menge der übertragenen und gespeicherten Zustandsinformationen haben und es wurden bestimmte Verstärkungseffekte ausgemacht, die in solchen Topologien in Bezug auf

die übertragene und gespeicherte Menge an Zustandsinformationen auftreten. Um diese Datenmenge reduzieren zu können, wurde durch das Konzept der persistenten Speicherschichten ein Werkzeug entwickelt, mit dem man eine Topologie partitionieren kann, sodass die verstärkenden Effekte abgeschwächt werden. Durch dieses Konzept ermöglicht das Wiederherstellungsverfahren einen dynamisch implementierbaren Mittelweg im Umgang mit Zustandsinformationen zwischen persistenter Speicherung und redundanter Verteilung. Eine solche dynamisch regulierbare Zustandsverwaltung ist schließlich der Beitrag dieser Diplomarbeit zum Forschungsfeld der Ausfallsicherheit und Wiederherstellung verteilter Systeme, die alle typischen Anforderungen an verteiltes Complex Event Processing erfüllt. Schließlich wurden noch einige Systemeigenschaften in einer Simulation untersucht und die theoretischen Überlegungen durch praktische Daten untermauert.

Ausblick

Zu dem in dieser Diplomarbeit vorgestellten Verfahren können noch verschiedene Fragen untersucht werden. So wäre eine Evaluierung über den Einfluss von Berechnungszeiten für die Korrelationen in den Operatoren interessant. Außerdem kann das Verfahren mit weiteren aktuellen Forschungsgebieten im verteilten Complex Event Processing in Verbindung gebracht werden, wie zum Beispiel Verfahren zur Platzierung von Operatoren [SKR11] und für mobile CEP-Szenarien [KORR12]. Für die Platzierung von persistenten Schichten kann ein Algorithmus zur dynamischen Einteilung entwickelt werden, der aus verschiedenen Parametern eine optimale Partitionierung der Operatortopologie berechnet.

Das Verfahren kann auch noch in verschiedener Hinsicht verfeinert werden, um in speziellen Fällen wie dem Auftreten besonders großer Korrelationsfenster oder der Platzierung mehrerer Operatoren auf einem Host die Verarbeitungseffizienz zu steigern. Von Interesse ist auch die Frage, wie man das Verfahren mit neuen Speichertechnologien, die ähnliche Zugriffseigenschaften wie Arbeitsspeicher bieten, verbinden kann (*Storage Class Memory* [Lam10]).

Literaturverzeichnis

[ABW06] A. Arasu, S. Babu, J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15:121–142, 2006. 10.1007/s00778-004-0147-z. (Zitiert auf Seite 4)

[AIM10] L. Atzori, A. Iera, G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010. doi:10.1016/j.comnet.2010.05.010. (Zitiert auf Seite 1)

[BK09] A. Buchmann, B. Koldehofe. Complex Event Processing. *it - Information Technology*, 51(5):241 – 242, 2009. (Zitiert auf den Seiten 1 und 5)

[BKR11] A. Benzing, B. Koldehofe, K. Rothermel. Efficient support for multi-resolution queries in global sensor networks. In *Proceedings of the 5th International Conference on Communication System Software and Middleware*, COMSWARE '11, S. 11:1–11:12. ACM, New York, NY, USA, 2011. doi:10.1145/2016551.2016562. (Zitiert auf Seite 1)

[CDMRV10] L. Coppolino, D. De Mari, L. Romano, V. Vianello. SLA compliance monitoring through semantic processing. In *11th IEEE/ACM International Conference on Grid Computing (GRID), 2010*, S. 252 –258. 2010. doi:10.1109/GRID.2010.5697975. (Zitiert auf den Seiten 19 und 20)

[CM94] S. Chakravarthy, D. Mishra. Snoop: an expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(1):1–26, 1994. doi:10.1016/0169-023X(94)90006-X. (Zitiert auf den Seiten 8, 5, 14, 15, 30, 31 und 33)

[CM10] G. Cugola, A. Margara. TESLA: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, S. 50–61. ACM, New York, NY, USA, 2010. doi:10.1145/1827418.1827427. (Zitiert auf Seite 5)

[CT96] T. D. Chandra, S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996. doi:10.1145/226643.226647. (Zitiert auf den Seiten 57 und 65)

[EAWJ02] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002. doi:10.1145/568522.568525. (Zitiert auf den Seiten 1, 23 und 25)

[EB09] M. Eckert, F. Bry. Informatiklexikon: Complex Event Processing (CEP), 2009. URL http://www.gi.de/nc/service/informatiklexikon/detailansicht/article/complex-event-processing-cep.html. (Zitiert auf den Seiten 1, 3 und 4)

[FGK11] F. C. Freiling, R. Guerraoui, P. Kuznetsov. The failure detector abstraction. *ACM Computing Surveys*, 43(2):9:1–9:40, 2011. doi:10.1145/1883612.1883616. (Zitiert auf Seite 57)

[GO96] R. Guerraoui, R. Oliveira. Stubborn Communication Channels. Technischer Bericht, LSE, Departement d'Informatique, Ecole Polytechnique Federale de Lausanne, 1996. (Zitiert auf Seite 16)

[GR06] R. Guerraoui, L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Zitiert auf den Seiten 15 und 57)

[KBG08] Y. Kwon, M. Balazinska, A. Greenberg. Fault-tolerant stream processing using a distributed, replicated file system. *Proceedings of the VLDB Endowment*, 1(1):574–585, 2008. doi:10.1145/1453856.1453920. (Zitiert auf Seite 25)

[KKR10] G. G. Koch, B. Koldehofe, K. Rothermel. Cordies: expressive event correlation in distributed systems. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, S. 26–37. ACM, New York, NY, USA, 2010. doi:10.1145/1827418.1827424. (Zitiert auf den Seiten 1 und 5)

[KORR12] B. Koldehofe, B. Ottenwälder, K. Rothermel, U. Ramachandran. Moving range queries in distributed complex event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, S. 201–212. ACM, New York, NY, USA, 2012. doi:10.1145/2335484.2335507. (Zitiert auf den Seiten 1, 5 und 86)

[Lam77] L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, SE-3(2):125 – 143, 1977. doi:10.1109/TSE.1977.229904. (Zitiert auf Seite 58)

[Lam10] C. Lam. Storage Class Memory. In *10th IEEE International Conference onSolid-State and Integrated Circuit Technology (ICSICT), 2010*, S. 1080 –1083. 2010. doi:10.1109/ICSICT.2010.5667551. (Zitiert auf Seite 86)

[Luc01] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. (Zitiert auf den Seiten 1, 3 und 16)

[MN98] M. Matsumoto, T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998. doi:10.1145/272991.272995. (Zitiert auf Seite 75)

[OB10] D. O'Keeffe, J. Bacon. Reliable complex event detection for pervasive computing. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, S. 73–84. ACM, New York, NY, USA, 2010. doi: 10.1145/1827418.1827429. (Zitiert auf Seite 20)

[Omn] URL `http://www.omnetpp.org/`. (Zitiert auf Seite 73)

[RDR10] S. Rizou, F. Dürr, K. Rothermel. Solving the Multi-Operator Placement Problem in Large-Scale Operator Networks. In *Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), 2010*, S. 1 –6. 2010. doi:10.1109/ICCCN.2010.5560127. (Zitiert auf Seite 7)

[Rey05] M. Reynal. A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News*, 36(1):53–70, 2005. doi:10.1145/1052796.1052806. (Zitiert auf Seite 57)

[RH85] C. Raghavendra, S. Hariri. Reliability Optimization in the Design of Distributed Systems. *IEEE Transactions on Software Engineering*, SE-11(10):1184 – 1193, 1985. doi:10.1109/TSE.1985.231866. (Zitiert auf Seite 23)

[RHI+12] U. Ramachandran, K. Hong, L. Iftode, R. Jain, R. Kumar, K. Rothermel, J. Shin, R. Sivakumar. Large-Scale Situation Awareness With Camera Networks and Multimodal Sensing. *Proceedings of the IEEE*, 100(4):878 –892, 2012. doi:10.1109/JPROC.2011.2182093. (Zitiert auf Seite 1)

[SKPR10] B. Schilling, B. Koldehofe, U. Pletat, K. Rothermel. Distributed heterogeneous event processing: enhancing scalability and interoperability of CEP in an industrial context. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, S. 150–159. ACM, New York, NY, USA, 2010. doi:10.1145/1827418.1827453. (Zitiert auf Seite 5)

[SKR11] B. Schilling, B. Koldehofe, K. Rothermel. Efficient and Distributed Rule Placement in Heavy Constraint-Driven Event Systems. In *IEEE 13th International Conference on High Performance Computing and Communications (HPCC), 2011*, S. 355 –364. 2011. doi:10.1109/HPCC.2011.53. (Zitiert auf den Seiten 7 und 86)

[SM11] Z. Sebepou, K. Magoutis. CEC: Continuous eventual checkpointing for data stream processing operators. In *IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN), 2011*, S. 145 –156. 2011. doi: 10.1109/DSN.2011.5958214. (Zitiert auf Seite 23)

[SS83] R. D. Schlichting, F. B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, 1983. doi:10.1145/357369.357371. (Zitiert auf Seite 12)

[SSMW10] C. Seel, J. Schimmelpfennig, D. Mayer, P. Walter. Conceptual modeling of complex events of the Internet of Things. In *eChallenges, 2010*, S. 1 –8. 2010. (Zitiert auf Seite 19)

[VKR11] M. Volz, B. Koldehofe, K. Rothermel. Supporting Strong Reliability for Distributed Complex Event Processing Systems. In *IEEE 13th International Conference on High Performance Computing and Communications (HPCC), 2011*, S. 477 –486. 2011. doi:10.1109/HPCC.2011.69. (Zitiert auf den Seiten 7, 1, 19, 20, 23 und 24)

[WDR06] E. Wu, Y. Diao, S. Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, S. 407–418. ACM, New York, NY, USA, 2006. doi:10.1145/1142473.1142520. (Zitiert auf Seite 5)

[WKL+08] E. Welbourne, N. Khoussainova, J. Letchner, Y. Li, M. Balazinska, G. Borriello, D. Suciu. Cascadia: A System for Specifying, Detecting, and Managing RFID Events. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, S. 281–294. ACM, New York, NY, USA, 2008. doi:10.1145/1378600.1378631. (Zitiert auf Seite 5)

Alle URLs wurden zuletzt am 23.09.2012 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Ruben Mayer)