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

Bachelorarbeit Nr. 7

Ereigniskorrelation auf energiebeschränkten mobilen Endgeräten

Marcus Vetter

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dipl.-Inform. Beate Ottenwälder
begonnen am:	03. Mai 2012
beendet am:	08. Oktober 2012
CR-Klassifikation:	C.1.4, C.2.1, C.2.4

Kurzfassung

Der Einsatz von Systemen mit komplexer Ereignisverarbeitung ist heutzutage in vielen Gebieten der Informationstechnologie anzutreffen. Beispielsweise können mithilfe eines mobilen Stau-Warn-Systems Autofahrer vor einem Stauende gewarnt und folglich Auffahrunfälle vermieden werden. Ein solches CEP-System kann hinsichtlich der guten Prozessorleistung eines Smartphones lokal auf einem mobilen Endgerät ausgeführt werden, wobei der Akku des Smartphones dabei stark belastet wird. Im Gegensatz dazu besteht die Möglichkeit, die berechnungsintensive Ausführung des CEP-Operators in einer Infrastruktur durchzuführen. In diesem Fall entsteht jedoch eine hohe Netzwerk-Kommunikation, die ebenfalls den Akku des Smartphones stark beansprucht.

In dieser Arbeit wird daher der Energieverbrauch eines CEP-Systems mit lokaler, auf einem Smartphone durchgeführter Ausführung eines CEP-Operators mit der Ausführung in einer Infrastruktur verglichen. Dabei werden CEP-Operatoren anhand spezifischer Merkmale, wie der Anzahl der eingehenden Ereignisströme, der Berechnungskomplexität sowie der Frequenz der Ereigniskorrelation, klassifiziert. Aufgrund der Konfiguration eines CEP-Operators kann folglich entschieden werden, ob sich dessen Ausführung in einer entfernten Infrastruktur oder lokal auf dem Smartphone energieeffizienter darstellt.

Die durchgeführte Evaluation hat gezeigt, dass die Frequenz der Ereigniskorrelation maßgeblich für den Energieverbrauch des CEP-Systems verantwortlich ist. Die Berechnungskomplexität des CEP-Operators bei der Ereigniskorrelation hat hingegen einen geringen Einfluss auf den Energiebedarf des Systems.

Inhaltsverzeichnis

1 Einleitung

Mobile Endgeräte, beispielsweise *Smartphones*, sind heutzutage weit verbreitet und erlauben Nutzern den weltweiten Zugriff auf Daten. Zudem verfügen Smartphones über diverse Sensoren wie zum Beispiel einem Mikrofon, Helligkeitssensoren oder GPS, die zum Erfassen und Speichern von Informationen der Umgebung verwendet werden können. Mithilfe dieser Informationen können Applikationen mobiler Endgeräte ihren Nutzern ein großes Funktionsspektrum in unterschiedlichen Anwendungsbereichen bieten. Aufgrund der häufig vorhandenen Daten-Verbindung über Schnittstellen wie beispielsweise WLAN, kann eine Applikation insbesondere auch von anderen mobilen Endgeräten erfasste Informationen beziehen und zur weiteren Verarbeitung verwenden.

Das Paradigma *Komplexe Ereignisverarbeitung*, kurz CEP, beschreibt eine schnelle, zuverlässige und verteilte Verarbeitung von erfassten Informationen. Auftretende Korrelationen der Sensorereignisse können verwendet werden, um beispielsweise Gefahrensituationen im Verkehr zu erkennen. Das Paradigma erlaubt hierbei eine von den Geräten unabhängige, flexible Ausführung. So kann zum Beispiel die Korrelation von Ereignissen des Systems in einer dedizierten Infrastruktur berechnet werden, während lokal auf dem entsprechenden Smartphone Sensoren zum Erfassen von Informationen zum Einsatz kommen und schlussendlich das Ergebnis präsentiert wird.

Wird die Berechnung der Ereigniskorrelation in einer Infrastruktur durchgeführt, entsteht aufgrund der Kommunikation zwischen Sensoren und Operatoren über eine Daten-Verbindung sowohl ein hoher Energieverbrauch als auch eine beträchtliche Netzwerklast. Alternativ kann die Berechnung des CEP-Operators auf dem Smartphone selbst ausgeführt werden. Aufgrund der gegebenenfalls hohen Belastung der CPU entsteht auch hier ein hoher Energiebedarf. Abhängig von der Semantik des CEP-Operators und der Komplexität der Ereigniskorrelation differiert der Energiebedarf des CEP-Systems.

In dieser Bachelor-Arbeit wird untersucht, welche CEP-Operatoren lokal auf dem Smartphone und welche CEP-Operatoren in einer Infrastruktur ausgeführt werden müssen, um die Belastung des Akkus eines Smartphones möglichst gering zu halten. Um CEP-Operatoren näher spezifizieren zu können, wird zunächst eine Architektur für ein mobil ausführbares CEP-System entwickelt und anschließend dessen Implementierung durchgeführt. Schlussendlich werden die Merkmale der CEP-Operatoren herausgearbeitet und eine Klassifikation vorgestellt.

Gliederung

Die Arbeit ist wie folgt gegliedert: Im Kapitel 2 wird auf die Grundlagen der Bachelor-Arbeit eingegangen. Dabei wird der Fokus auf das Paradigma *Komplexe Ereignisverarbeitung*, das Betriebssystem *Android* sowie die Energiebetrachtung bei Smartphones gerichtet. Anschließend wird im Kapitel 3 das in dieser Bachelor-Arbeit untersuchte Problem bei der Ausführung eines CEP-Systems auf mobilen Endgeräten beschrieben. Aufgrund der unterschiedlichen Anwendungsgebiete der komplexen Ereignisverarbeitung werden im Kapitel 4 einige verwandte Arbeiten vorgestellt und daraus resultierend eine Abgrenzung zu dieser Bachelor-Arbeit formuliert.

Im Kapitel 5 werden das der Implementierung zugrunde gelegte Systemmodell sowie die getroffenen Systemannahmen erläutert. Im anschließenden Kapitel 6 wird die Architektur des CEP-Systems anhand einiger Architekturdiagramme verdeutlicht. Die Ausführung eines CEP-Operators und insbesondere das dahinterstehende Ausführungsmodell wird im Kapitel 7 detailliert beschrieben. Das zentrale Kapitel 8 befasst sich mit dem im Rahmen dieser Bachelor-Arbeit implementierten, mobil ausführbaren CEP-System.

Eine Klassifikation der in dieser Bachelor-Arbeit spezifizierten CEP-Operatoren befindet sich im Kapitel 9. Nachfolgend werden im Kapitel 10 einige Experimente durchgeführt und final das Ergebnis präsentiert. Abschließend werden im Kapitel 11 die resultierenden Erkenntnisse dieser Bachelor-Arbeit zusammengefasst.

2 Grundlagen

Im Kapitel 2.1 werden die Grundlagen des Paradigmas *Komplexe Ereignisverarbeitung* detailliert erläutert sowie auf dessen Komponenten und wesentlichen Merkmale eingegangen. Weiterhin werden elementare und komplexe Ereignisse vorgestellt, die mithilfe von Ereignisströmen von einer zur anderen CEP-Komponente übertragen werden.

Das in dieser Bachelor-Arbeit präsentierte CEP-System basiert auf dem im Abschnitt 2.2 vorgestellten Betriebssystem *Android*. Im Abschnitt 2.3 wird der Fokus auf die Energiebetrachtung bei mobilen Endgeräten gerichtet. Die im Kapitel 6 abgebildeten Architekturdiagramme wurden mithilfe der im Abschnitt 2.4 beschriebenen Modellierungssprache *Fundamental Modeling Concepts* erstellt.

2.1 Komplexe Ereignisverarbeitung

Das Paradigma *Komplexe Ereignisverarbeitung*, kurz CEP, wird in vielen Echtzeit-Systemen angewandt, die die Verarbeitung von komplexen, zusammengesetzten Ereignissen und deren Korrelation realisieren [EB09, LRD+11]. Dabei spielt neben der Geschwindigkeit der Ereignisverarbeitung auch die Lokation der verwendeten CEP-Komponenten eine wesentliche Rolle. CEP-Systeme werden unter anderem bei der Service Oriented Architecture (SOA), Event-Driven Architecture (EDA) oder vergleichbaren Ansätzen eingesetzt.

Daher wird ein CEP-System meist in einer verteilten Umgebung ausgeführt. Die Hauptbestandteile eines CEP-Systems sind verschiedene CEP-Komponenten, elementare und komplexe Ereignisse sowie Ereignisströme. In den folgenden Unterkapiteln wird eine Einführung in diese Bestandteile gegeben. Dabei wird im Abschnitt 2.1.2 der Zusammenhang zwischen Ereignissen und Ereignisströmen verdeutlicht und im anschließenden Abschnitt 2.1.3 werden komplexe Ereignisse, die als Resultat einer Korrelation auftreten, erläutert.

2.1.1 Komponenten

Grundsätzlich besteht ein CEP-System aus den Komponenten *Sensoren*, *Operatoren* und *Konsumenten*. Ein Sensor erfasst Informationen aus der Umgebung, die zur weiteren Verarbeitung durch einen Operator verwendet werden. Eine von einem Sensor erfasste Information kann beispielsweise die gemessene Temperatur oder die GPS-Position des mobilen Endgeräts sein. Entsprechend der Semantik des Operators können mithilfe der eingehenden Informationen neue, komplexe Ereignisse berechnet werden. Der Konsument eines CEP-Systems stellt die final von einem Operator berechneten Ereignisse dar. Eine Applikation eines Smartphones kann zum Beispiel als Konsument eines CEP-Systems die von einem Operator berechneten

Positionen anderer mobiler Endgeräte auf einer Landkarte graphisch aufbereitet darstellen.

Dabei werden Sensoren zur Kategorie der Ereignisproduzenten zugeordnet, da sie für das Erzeugen von Ereignissen verwendet werden. Im Gegensatz zu den Sensoren gehören Konsumenten der Kategorie der Ereigniskonsumenten an, da sie für das Konsumieren von Ereignissen benötigt werden. Operatoren können sowohl Ereignisse konsumieren, als auch die Ereignisproduktion durchführen, wodurch sie beiden Kategorien zugeordnet werden. Die hier beschriebene Relation zwischen den CEP-Komponenten Sensoren, Operatoren und Konsumenten und den Kategorien Ereignisproduzent und -konsument wird in Tabelle 2.1.1 veranschaulicht.

CEP-Komponente	Ereignisproduzent	Ereigniskonsument
Sensoren	X	
Operatoren	X	X
Konsumenten		X

Tabelle 2.1: CEP-Komponenten

2.1.2 Ereignisse und Ereignisströme

Ein *Ereignis* in einem CEP-System ist ein abgeschlossenes Datenpaket, das im Allgemeinen über ein Netzwerk übertragen werden kann. Das Ziel eines Ereignisses ist es, von einer CEP-Komponente zu einer anderen gesendet zu werden. Die Übertragung von Ereignissen kann entweder lokal auf einem Gerät oder über ein Netzwerk realisiert werden. Sowohl serialisierbare Objekte, im Sinne der Objektorientierung bei Programmiersprachen, als auch einfache Zeichenketten werden als Ereignisse betrachtet.

Um Komponenten eines CEP-Systems zu verknüpfen, werden sogenannte *Ereignisströme* verwendet. Mithilfe eines Ereignisstroms können Ereignisse zwischen CEP-Komponenten transportiert werden. Ein Ereignisstrom hat eine definierte Richtung und transportiert Ereignisse von einem Ereignisproduzent zu einem Ereigniskonsument.

2.1.3 Komplexe Ereignisse als Resultat einer Korrelation

Wie im Kapitel 2.1.1 beschrieben, erzeugen sowohl Sensoren als auch Operatoren Ereignisse, da beide der Kategorie der Ereignisproduzenten zugeordnet sind. Dabei ist allerdings die Art der produzierten Ereignisse zu unterscheiden. Sensoren erzeugen *elementare Ereignisse*, die bislang nicht weiterverarbeitet oder mit anderen einfachen Ereignissen korreliert wurden. Im Gegensatz hierzu werden *komplexe Ereignisse* ausschließlich von Operatoren erzeugt. Operatoren lesen elementare oder komplexe Ereignisse von ihren eingehenden Ereignisströmen. Tritt eine Korrelation der eingehenden Ereignisse gemäß der Spezifikation des Operators auf, findet die Verarbeitung der Ereignisse im Operator statt. Dabei führt der

Operator eine Selektion der eingehenden Ereignisse durch und wendet anschließend auf die ausgewählten Ereignisse eine definierte Funktion an. Ein vom Operator erzeugtes, aus der Verarbeitung resultierendes komplexes Ereignis wird schlussendlich über alle ausgehenden Ereignisströme des Operators versendet, wobei optional ein Filter zum Einsatz kommen kann. Das Ausführungsmodell, das den in dieser Bachelor-Arbeit beschriebenen Operatoren zugrunde liegt, wird im Kapitel 7 detailliert vorgestellt.

Um beispielsweise Ereignisse von eingehenden Strömen in Data Stream Management Systemen kontinuierlich auslesen zu können, wurde die deklarative Anfragesprache CQL entwickelt, die sich stark an die Datenbanksprache SQL anlehnt [ABW03a, ABW03b]. Die in dieser Bachelor-Arbeit verwendete Anfragemethodik für Datenströme wird im Kapitel 8 detailliert erläutert.

Der in diesem Abschnitt beschriebene Sachverhalt zwischen einfachen und komplexen Ereignissen wird in Abbildung 2.1 veranschaulicht. Im abgebildeten Beispiel werden CEP-Komponenten mit Kreisen, Ereignisströme anhand gerichteter Pfeile und Ereignisse mithilfe von Rechtecken dargestellt. Schwarz eingefärbte Rechtecke symbolisieren komplexe Ereignisse, wobei weiß eingefärbte Rechtecke elementare Ereignisse abbilden.

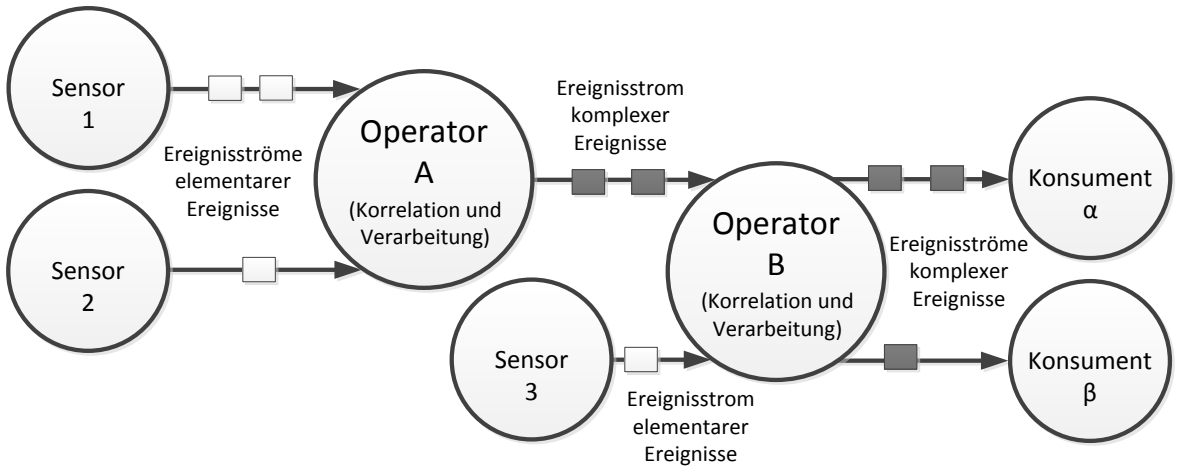


Abbildung 2.1: Elementare und komplexe Ereignisverarbeitung eines CEP-Operators

Die im Beispiel 2.1 dargestellten Sensoren '1' und '2' versenden elementare Ereignisse an Operator 'A'. Nachdem Operator 'A' gemäß seiner Spezifikation eine Korrelation eingehender Ereignisse und folglich die Verarbeitung anhand einer definierten Funktion durchgeführt hat, wird ein ausgehendes, komplexes Ereignis erzeugt. Das komplexe Ereignis wird mithilfe eines Ereignisstroms an Operator 'B' transportiert. Sowohl dieser Ereignisstrom, als auch der von Sensor '3' erzeugte Ereignisstrom werden von Operator 'B' als eingehende Ereignisströme verwendet. Nachdem Operator 'B' analog zu Operator 'A' eine Korrelation und Ereignisverarbeitung durchgeführt hat, werden die erzeugten, komplexen Ereignisse an die Konsumenten α und β versendet.

Ein Operator kann beliebig viele eingehende Ereignisströme besitzen, die sowohl elementare als auch komplexe Ereignisse transportieren. Im Gegensatz dazu versendet ein Operator mithilfe der ausgehenden Ereignisströme ausschließlich komplexe Ereignisse .

2.2 Android

Das für diese Bachelor-Arbeit entwickelte CEP-System muss sowohl in einer Infrastruktur als auch auf mobilen Endgeräten ausgeführt werden können. Aufgesetzt wird das CEP-System dabei auf einem Betriebssystem, das wichtige Schnittstellen für das CEP-System bereit stellt. Das für die Ausführung auf Smartphones verwendete CEP-System wurde in dieser Bachelor-Arbeit auf Basis des für mobile Endgeräte entwickelte, quelloffene Betriebssystem *Android* implementiert. Android wird seit 2007 von der Open Handset Alliance entwickelt. Die Firma Google ist das Hauptmitglied der Open Handset Alliance. Das Betriebssystem Android basiert auf einem Linux Kernel, wobei die von Google entwickelte, auf dem Linux Kernel aufgesetzte, virtuelle Maschine 'Dalvik VM' das Ausführen von Java-Programmcode erlaubt. Sowohl das Betriebssystem als auch Anwendungen (englisch Applications, kurz Apps), werden in Java implementiert, wobei das Layout der Anwendungen mithilfe von XML-Dateien definiert wird.

Die Open Handset Alliance stellt das Software Development Kit, kurz SDK, für Android quelloffen zur Verfügung. Das Android SDK erweitert das Java SDK, schränkt es aber nicht ein. Somit können alle aus Java bekannten Funktionen verwendet werden.

Die Grundelemente einer Anwendung für das Android-Betriebssystem sind '*Activities*', '*Content-Provider*', '*Broadcast-Receiver*' und '*Services*'. Jede auf einem Smartphone graphisch dargestellte Bildschirmmaske basiert auf einer Activity. Eine Activity erlaubt die Interaktion mit dem Benutzer und stellt die graphischen Elemente der Bildschirmmaske bereit. Das Layout einer Activity kann mithilfe von XML-Dateien definiert werden. Im Gegensatz zu Activities dienen Content-Provider, Broadcast-Receiver und Services nicht zur graphischen Darstellung von Elementen, sondern zur Datenverarbeitung und Dienstbereitstellung. Content-Provider stellen anderen Anwendungen Daten bereit, während publizierte Ereignisse anderer Anwendungen oder des Betriebssystems unter Verwendung von Broadcast-Receivern empfangen werden können. Der vierte Grundbestandteil einer Android-Anwendung sind Services. Sie werden direkt an eine Anwendung geknüpft, werden mit dieser gestartet, führen jedoch ihre angebotenen Dienste im Hintergrund aus. Wird eine Anwendung vom Nutzer geschlossen, läuft der Service weiter.

2.3 Energie bei Smartphones

Neuartige mobile Endgeräte bieten Nutzern viele unterschiedliche Funktionen, deren Ausführung stets Energie benötigt. Dabei müssen sowohl der Bildschirm, die CPU, das WLAN-Modul als auch andere Hardware-Komponenten eines Smartphones energetisch versorgt werden. Damit dem Gerät ohne angeschlossenes Netzkabel Energie zugeführt werden kann, besitzen mobile Endgeräte einen Akkumulator. Ein solcher Akkumulator, kurz Akku, besteht bei Smartphones meist aus Lithium-Ionen. Er versorgt das Smartphone mit einer Spannung von üblicherweise 3,7 Volt bis 4,2 Volt und besitzt eine Nennkapazität von 1000 mAh bis 2500 mAh. Die Nennkapazität, abgekürzt mit den Buchstaben NK, wird in Amperestunden, kurz Ah, angegeben und durch das in Formel 2.1 angegebene Produkt errechnet [NEN].

$$NK[Ah] = I[A] \cdot t[h] \tag{2.1}$$

Die hier beschriebene Nennkapazität darf unter keinen Umständen mit der physikalischen Größe der 'Elektrischen Kapazität', kurz C, in Verbindung gebracht werden. Die elektrische Kapazität wird mit der Division von Ladung in Coulomb und der Spannung in Volt definiert.

Die Nennkapazität eines Smartphones bezeichnet das sogenannte Strom-Fassungsvermögen [NEN]. Beispielsweise kann der Akku eines Smartphones mit einer Nennkapazität von 1000 mAh das Gerät konstant zwei Stunden mit 500 mA Strom versorgen, oder vier Stunden mit einem Strom von 250 mA.

Da im Allgemeinen ein Nutzer eines Smartphones eine lange Akkulaufzeit präferiert, sollte bei der Entwicklung von Applikationen auf deren Energieverbrauch geachtet werden. Je weniger Energie eine Applikation benötigt, desto länger kann der Nutzer das Smartphone und dessen Funktionalität verwenden.

2.3.1 Energieverbrauch bei Android

Das Betriebssystem Android bietet für die Entwickler von Applikationen eine Schnittstelle an, die zum Auslesen einiger Informationen über den Energieverbrauch des Smartphones verwendet werden kann. Der sogenannte *Batterymanager*, der seit der ersten Version des Betriebssystems den Entwicklern zur Verfügung steht, stellt wichtige Informationen über den Zustand des Akkus bereit. So kann eine Applikation mithilfe eines Broadcast Receivers vom Betriebssystem informiert werden, sobald sich der Akku um einen Prozentpunkt entlädt oder aufgeladen wird. Teil des Informationspakets des Betriebssystems ist die aktuelle Spannung des Akkus, die in Millivolt angegeben wird.

Die aus dem Akku gelieferte Leistung wird in der Einheit *Wattstunde* angegeben. Sie definiert, wie viel Watt der Akku in einer Stunde abgibt und errechnet sich wie in Formel 2.2 angegeben. Dabei wird die im Abschnitt 2.3 beschriebene Nennkapazität mit der Spannung des Akkus multipliziert.

$$1Wh = 1Ah \cdot 1V \tag{2.2}$$

Folglich kann der Entwickler anhand der Information des Betriebssystems und den Kenngrößen des Akkus den Energieverbrauch seiner Anwendung annähernd bestimmen. Dazu kann der Energieverbrauch des Betriebssystems ohne das Starten der Anwendung, subtrahiert vom Energieverbrauch des Betriebssystems mit aktiver Anwendung, gemessen werden.

2.4 FMC - Fundamental Modeling Concepts

Um die Architektur eines Softwaresystems modellieren zu können, entwickelte die Firma Intervista AG Deutschland die Sprache *Fundamental Modeling Concepts*, kurz FMC [FMCa]. Die in diesem Abschnitt vorgestellte Modellierungssprache wird im Kapitel 6 verwendet, um die Architektur des in dieser Bachelor-Arbeit implementierten CEP-Systems zu beschreiben.

FMC bietet für die strukturelle Modellierung einer Software-Architektur das sogenannte *FMC Blockdiagramm* an, das in diesem Abschnitt beschrieben wird. Die Modellierung des Verhaltens eines Systems wird in der Modellierungssprache anhand von Petri-Netzen dargestellt, die Struktur von Daten und Wertübertragungen anhand von Entity-Relationship-Diagrammen.

Ein FMC Blockdiagramm besteht aus 'Agents', 'Storages' und 'Channels'. Dabei wird ein Agent mithilfe eines Rechtecks, ein Storage anhand eines Ovals und ein Channel als Kreis dargestellt. Ein Agent wird zur aktiven Datenverarbeitung verwendet, wobei ein Storage eine passive Datenspeicherung darstellt. Unter der Verwendung von Channels können Agents mit Agents kommunizieren.

Ein Beispiel für eine in FMC modellierte Software-Architektur ist in Abbildung 2.2 veranschaulicht. Hierbei wurden die abgebildeten Agenten mit 'A.1', 'A.2' und 'A.3', die Storages mit 'S.1', 'S.2', 'S.3.1', 'S.3.2', 'S.4.1' und 'S.4.2' gekennzeichnet.

Pfeile werden in der FMC-Modellierung für den Speicherzugriff verwendet. Zeigt ein Pfeil von einer Storage-Komponente auf einen Agenten, kann der Agent lesend auf die Daten des Storages zugreifen. Verläuft der Pfeil vom Agent zum Storage, signalisiert dies einen schreibenden Zugriff. Werden Pfeile in beide Richtungen verwendet, spricht man vom 'Modifying Access'. Im abgebildeten Beispiel 2.2 greift der Agent 'A.2' folglich schreibend auf 'S.4.2' zu, während 'S.1' von 'A.1' sowohl gelesen als auch geschrieben wird.

Die in FMC verwendeten Kanäle zur Kommunikation zwischen zwei Agents können gerichtet oder nicht gerichtet sein, wobei die Richtung des Pfeils der Richtung des Datenflusses entspricht. Wird ein Kanal für das Paradigma 'Request-Response' verwendet, kann dies anhand des Buchstabens 'R' am Kanal und einem Pfeil in Richtung der 'Request'-Nachricht modelliert werden.

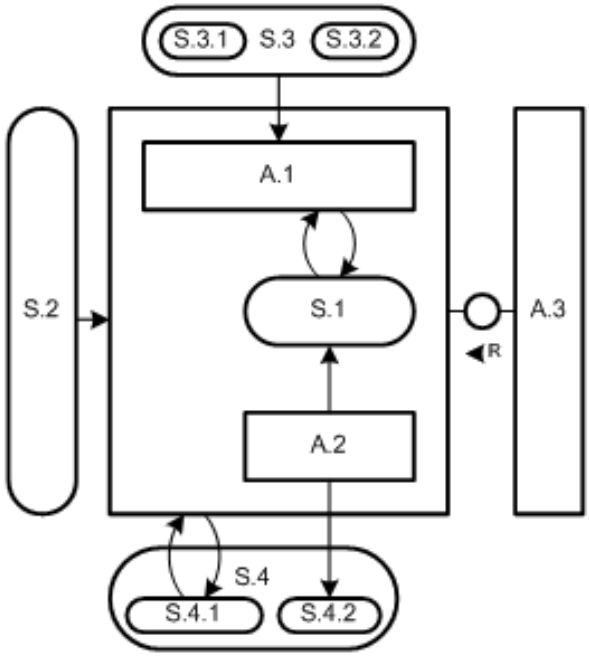


Abbildung 2.2: Beispiel einer in FMC modellierten Architektur [FMCb]

Die in diesem Abschnitt vorgestellten Komponenten der FMC-Modellierung lassen sich gruppieren, indem Komponenten verschachtelt werden. Ein Agent kann dabei jede Art von Komponenten enthalten, während ein Storage ausschließlich andere Storage-Komponenten enthält. Im Beispiel 2.2 besteht die Gruppe 'S.4' aus den Storage-Komponenten 'S.4.1' und 'S.4.2'. Mithilfe der Notation von '...' wird verdeutlicht, dass mehrere gleiche Komponenten in einer Komponente enthalten sein können.

Die vollständige Tabelle aller Elemente der FMC-Modellierungssprache befindet sich unter der hier angegebenen Internetseite [FMCb].

3 Problembeschreibung

In diesem Kapitel wird anhand des konkreten Beispiels eines mobilen Stau-Warn-Systems der Einsatz eines mobilen CEP-Systems beschrieben. Weiterhin wird die energetische Belastung des mobilen Endgeräts durch das CEP-System und eine mögliche Lösungsstrategie erläutert.

3.1 Mobiles Stau-Warn-System

Nach Angaben des Statistischen Bundesamts ereigneten sich im Jahr 2011 auf deutschen Autobahnen 18.290 Unfälle mit Personenschäden [DES]. Trotz der seit 2001 erfreulicherweise rückläufigen Anzahl der Autounfälle investieren Autobauer viel Geld in Forschung und Entwicklung neuartiger Technologien, um Autounfälle zu vermeiden.

Eine solche Software, die potentielle Autounfälle frühzeitig erkennen kann und dem Autofahrer einen entsprechenden Warnhinweis ausgibt, kann beispielsweise mit dem Paradigma der komplexen Ereignisverarbeitung realisiert werden.

Ein konkretes Beispiel eines Autounfalls auf einer Autobahn ist das Auffahren auf ein Stauende. Sowohl bei zu wenig Abstand zum Vordermann als auch beim Entstehen eines Stauendes im Kurvenbereich ist der Auffahrunfall eine häufige Unfallursache.

Um einen Autofahrer vor einem vorausliegenden Stauende zu warnen, kann ein mobiles CEP-Sytem zum Einsatz kommen. Das Smartphone des Autofahrers wirkt dabei als Konsument des CEP-Systems, der von einem 'CEP-Stau-Operator' informiert wird, sobald sich ein Stauende auf seiner Strecke befindet. Als Sensoren des CEP-Systems dienen die Smartphones aller Verkehrsteilnehmer, die das CEP-System auf ihren Smartphones installiert haben. Da neuartige Smartphones mit einem Mikrofon, Lage-, GPS-, Geschwindigkeits-, Helligkeitssensor und vielen weiteren Sensoren ausgestattet sind, dienen sie als optimale Datenquelle.

Senden nun alle Verkehrsteilnehmer ihre Daten, beispielsweise Position und Geschwindigkeit des Kraftfahrzeugs, zum 'CEP-Stau-Operator', kann dieser ein Stauende berechnen und die nachfolgenden Autofahrer frühzeitig warnen.

In Abbildung 3.1 wird dieses mobile Stau-Warn-System veranschaulicht. Dabei werden die Smartphones der Autofahrer mit grün eingefärbten Kraftfahrzeugen als Sensoren verwendet. Diese senden ihre Position und Geschwindigkeit in einem Ereignis an den 'Stau-Operator'. Die in der Abbildung grün eingefärbten Rechtecke symbolisieren die Ereignisse, während Pfeile den Ereignisstrom und dessen Richtung darstellen.

Nachdem der 'Stau-Operator' ein Stauende errechnet hat, werden die Konsumenten des CEP-Systems über die Gefahrenquelle informiert. In Graphik 3.1 sind Kraftfahrzeuge der Konsumenten orange eingefärbt. Analog zum Ereignisstrom vom Sensor zum Operator werden Ereignisse auch vom Operator zum Konsumenten gesendet, wobei das Ereignis mit einem orange eingefärbten Dreieck abgebildet ist.

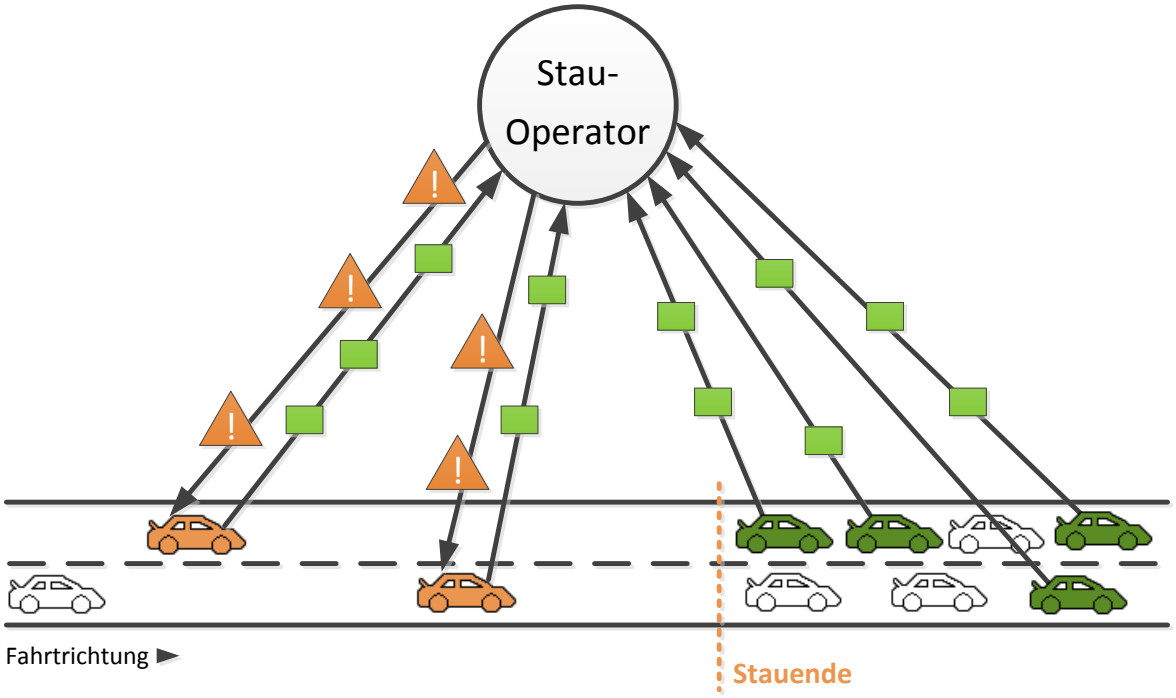


Abbildung 3.1: Beispiel eines mobilen Stau-Warn-Systems

Funktionsweise des CEP-Operators

Der beschriebene Operator zur Erkennung eines Stauendes erhält die aktuellen Geschwindigkeiten und Positionen der Verkehrsteilnehmer. Im dargestellten Beispiel haben die grün eingefärbten Autos eine sehr geringe Geschwindigkeit und ihre Positionen unterscheiden sich nur geringfügig. Im Gegensatz dazu weisen die orange dargestellten Autos sowohl eine höhere Geschwindigkeit als auch eine größere Entfernung zu den grün eingefärbten Autos auf. Der Operator ermittelt anhand der Autos mit geringer Geschwindigkeit und derer Positionen das Stauende. Alle Autos mit höherer Geschwindigkeit werden mithilfe eines erzeugten Ereignisses gewarnt. Das Ereignis enthält die Position des berechneten Stauendes, sodass sich der Konsument anhand seiner Position die aktuelle Entfernung zum Stauende berechnen kann.

3.2 Problem der Ausführung von CEP-Operatoren in mobilen Szenarien

Um einen CEP-Operator auszuführen, kann entweder eine Infrastruktur verwendet werden, oder aber die Ausführung des CEP-Operators auf dem Smartphone des Konsumenten in Betracht kommen. Bei der Ausführung des Operators auf einer Infrastruktur entsteht dem Konsumenten des CEP-Systems keine Rechenlast, jedoch eine nicht zu vernachlässigende Netzwerk-Kommunikation.

Im Gegensatz hierzu kann die Ausführung des Operators adäquat auf einem Smartphone realisiert werden. Die Netzwerklast wird hierdurch reduziert, wobei aufgrund der lokalen Ausführung des Operators die CPU des Smartphones beansprucht wird.

Der im Abschnitt 3.1 beschriebene Operator für ein mobiles Stau-Warn-System berechnet entsprechend der Position des Konsumenten die aktuellen Stauenden und dadurch entstehende Gefahren. Dazu erhält der Operator als eingehende Ereignisströme die Ereignisse der anderen Verkehrsteilnehmer, deren Positionen und Geschwindigkeiten. Zu beachten ist hierbei, dass nur die Ereignisströme der Verkehrsteilnehmer verwendet werden, die sich auf der selben Autobahn und in Fahrtrichtung vor dem Konsumenten befinden. Trotzdem können dabei große Datenströme entstehen, die vom Operator verarbeitet werden müssen.

Zusammenfassend lässt sich feststellen, dass zwei mögliche Szenarien zur Ausführung des CEP-Operators zur Wahl stehen. Wird der Operator auf dem mobilen Endgerät ausgeführt, entsteht weniger Netzwerklast, jedoch wird die CPU des Smartphones stark beansprucht. Die zweite Möglichkeit ist die Ausführung des Operators in einer entfernten Infrastruktur, wodurch die CPU des Smartphones für die Berechnung des Operators nicht verwendet werden muss. Aufgrund der entfernten Berechnung des CEP-Operators entsteht jedoch eine erhöhte Netzwerklast, die, analog zur CPU, den Akku des Smartphones stark beansprucht.

3.3 Ziel der Bachelor-Arbeit

In dieser Bachelor-Arbeit wird untersucht, in wie weit sich die Ausführung eines CEP-Operators auf den Energieverbrauch eines Smartphones auswirkt. Dabei wird ein im Kapitel 7 erläutertes Modell herangezogen, um die Architektur eines CEP-Systems erstellen und schlussendlich das mobile System implementieren zu können. Anschließend wird eine im Kapitel 9 beschriebene Klassifikation der CEP-Operatoren vorgestellt. Schlussendlich wird für jede Klasse eines Operators bestimmt, ob es für das Smartphone energetisch günstiger ist, den CEP-Operator in einer entfernten Infrastruktur oder auf dem lokalen Smartphone selbst auszuführen.

4 Related work

In diesem Kapitel werden Arbeiten beschrieben, die sich mit Themen des Gebiets der komplexen Ereignisverarbeitung befassen. Sie diskutieren dabei ähnliche Fragestellungen wie diese Bachelor-Arbeit. Das in Abschnitt 4.1 beschriebene Paper beschäftigt sich ausführlich mit der Ausführung eines mobilen CEP-Systems, das die theoretische Grundlage für diese Bachelor-Arbeit, insbesondere die im Kapitel 8 beschriebene Implementierung, beinhaltet.

Außerdem werden in den Abschnitten 4.2, 4.3 und 4.4 weitere Arbeiten beschrieben, die ähnliche und weiterführende Ansätze bei CEP-Systemen diskutieren. Im Abschnitt 4.5 wird verdeutlicht, wie sich diese Bachelor-Arbeit von den beschriebenen, existierenden Arbeiten abgrenzt.

4.1 Moving Range Queries in Distributed Complex Event Processing

Die Arbeit 'Moving Range Queries in Distributed Complex Event Processing' beschreibt die Ausführung eines CEP-Systems in einer mobilen Umgebung [KORR12, DEB]. Dabei gehen die Autoren detailliert auf die Erstellung eines Graphen für die logische Anordnung der CEP-Komponenten und auf ein Modell für die Ereignisverarbeitung ein. Sie definieren einen *Operatorbaum* als gerichteten Graphen $G = (\Omega \cup S \cup C, L)$, wobei Ω die Menge der Operatoren, S die Menge der Sensoren und C die Menge der Konsumenten bezeichnet. L definiert die im Graph verwendeten Kanten, wobei $L \subseteq (\Omega \cup S \times \Omega \cup C)$ die Kanten zwischen Operatoren und Sensoren sowie zwischen Operatoren und Konsumenten als Tupel darstellt. Anschließend wird auf die Operatoren eingegangen, die laut dieser Arbeit auf sogenannten Brokern des CEP-Netzes ausführbar sind.

In den weiteren Abschnitten des Papers wird auf das in einer mobilen Umgebung ausführbare CEP-System eingegangen. Dabei wird unter anderem das Konzept des 'Dynamic Interest Queries', kurz DIQ, beschrieben. Es handelt sich hierbei um eine Anfrage des Konsumenten an das CEP-System, mit dem Befehl die Ausführung zu starten und ist definiert als $DIQ = (T, mo, R, \delta)$. Hinter der Variablen T verbirgt sich der Operatorbaum, der für die Ausführung des CEP-Systems notwendig ist und alle beteiligten Komponenten enthält. mo bezeichnet das beobachtete, zentrale Objekt, englisch 'Monitored Focal Object' (MFO), das in den meisten Beispielen der Konsument selbst ist. Das MFO sendet in kontinuierlichen Zeitintervallen die aktuelle Position an das CEP-System. Die Variable R bezeichnet das Gebiet, das für den Konsumenten von Interesse ist, während δ einen Zeitraum für Ereignisse definiert, die in der Vergangenheit liegen, trotzdem für eine eventuelle Korrelation zur Verfügung stehen sollen.

Das Konzept des DIQs, das die Definition eines Operatorbaums einschließt, ist für diese Bachelor-Arbeit grundlegend. Die Einschränkung, dass Operatoren nur in einer Infrastruktur ausführbar sind, wird jedoch aufgehoben. Operatoren müssen für die Theses der Bachelor-Arbeit, die im Abschnitt 9.3 des Kapitels 9 formuliert ist, sowohl auf einer Infrastruktur als auch auf dem lokalen, mobilen Endgerät ausgeführt werden können.

Im vierten Kapitel der Arbeit von Koldehofe et al. wird ein spezielles Ausführungsmodell beschrieben. Hierbei wird der Fokus auf die Ausführung eines Operators, der Korrelation von Ereignissen sowie die Selektion und der Konsum vor bzw. nach einer Korrelation gerichtet. Außerdem wird ein Algorithmus angegeben, der eine verteilte Neu-Konfigurierung der Komponenten des CEP-Systems durchführt, sobald sich die Position des 'monitored focal objects' ändert. Schlussendlich werden einige Optimierungen des CEP-Systems diskutiert sowie eine Evaluation durchgeführt.

Zusammenfassend kann festgehalten werden, dass die Arbeit 'Moving Range Queries in Distributed Complex Event Processing' viele theoretische Grundlagen für diese Bachelor-Arbeit enthält, von denen hauptsächlich im Kapitel 8 Gebrauch gemacht wird. Auf eine Betrachtung der benötigten Energie eines Smartphones zur Ausführung des beschriebenen CEP-Systems wird allerdings verzichtet. Weiterhin werden Operatoren nicht genauer analysiert und folglich nicht klassifiziert. Auf Grundlage des Papers kann in dieser Bachelor-Arbeit sowohl eine Klassifikation der Operatoren als auch die Energiebetrachtung der mobilen Endgeräte durchgeführt werden.

4.2 CQL - Continuous Query Language

Neben der Arbeit von Koldehofe et al. geht es im Paper 'CQL: A Language for Continuous Queries over Streams and Relations' von Arvind Arasu et al. um ein Konzept der Anfrage von Ereignissen in Ereignisströmen [ABW03a]. Die in dieser Arbeit beschriebene Sprache 'Continuous Query Language', kurz CQL, übernimmt viele Konzepte von der in den 1970er Jahren entwickelten Datenbanksprache 'Structured Query Language', kurz SQL. Im Gegensatz zu SQL realisiert CQL eine Technik, Ereignisse auf Ereignisströmen anzufragen, während SQL auf relationalen Datenbanken eingesetzt wird. Das vorhandene Konzept der 'Relationen' von SQL wird auf das Konzept der Ereignisströme in CQL transformiert.

Die Arbeit stellt die Sprache CQL vor und geht dabei hauptsächlich auf das Konzept der Anfrage von Ereignissen in Ereignisströmen ein. Dabei wird der Fokus jedoch nicht auf die mobile Ausführung des CEP-Systems und die Energiebelastung eines Smartphones gerichtet. Weiterhin findet keine Klassifikation der CEP-Operatoren statt.

4.3 TESLA - Trio-based Event Specification Language

Um eine formale Definition für die Verarbeitung von Ereignissen in einem CEP-System erstellen zu können, wurde das von Gianpaolo Cugola und Alessandro Margara verfasste Paper 'TESLA: A Formally Defined Event Specification Language' veröffentlicht [CM]. Die Autoren beschreiben die entwickelte Sprache TESLA, mit deren Hilfe Regeln zur Verarbeitung von Ereignissen definiert werden können. Dabei besteht TESLA aus einer einfachen Syntax und formalen Semantiken. Weiterhin ermöglicht TESLA eine starke Ausdrucksfähigkeit und Flexibilität mithilfe eines strikten Rahmenwerks, das inhaltliche und zeitliche Filter, Timer, Aggregate sowie frei definierbare Regeln für die Ereignisselektion und den Ereigniskonsum beinhaltet.

Auch die Arbeit von Gianpaolo Cugola und Alessandro Margara geht dabei nicht auf die mobile Ausführung eines CEP-Systems und folglich nicht auf den Energiekonsum eines Smartphones ein.

4.4 SASE Event Language - RFID reading

In der Arbeit 'High-Performance Complex Event Processing over Streams' von Eugene Wu, Yanlei Diao und Shariq Rizvi beschreiben die Autoren ein System, das eine CEP-Ausführung mit RFID-Werten in Echtzeit durchführt [WDR06]. Dabei werden zu Ereignissen kodierte, gelesene RFID-Werte als eingehende Ereignisströme verwendet. Um ein effizientes RFID-Überwachungssystem zu realisieren, werden vorhandene CEP-Ereignissprachen erweitert. Die dadurch resultierende Sprache *SASE* ist eine deklarative Anfragesprache, die Filterung, Korrelation und Transformation von Ereignissen unterstützt. Bei der Anfrage von Ereignissen in einem Ereignisstrom werden, ähnlich zur Arbeit 'Moving Range Queries in Distributed Complex Event Processing' eine Selektion der Ereignisse in einem Fenster vorgenommen. Hierbei formulieren die Autoren das Anwendungsgebiet abstrakt, wodurch kein Bezug zu einer Ausführung eines RFID-Systems mithilfe von mobilen Endgeräten hergestellt wird.

Eugene Wu et al. stellen in ihrer Arbeit eine Anfragesprache für ein CEP-System vor und beschreiben einige Anwendungsgebiete. Sie gehen dabei aber nicht konkret auf eine mobile Ausführung und weiterführend auf den Energieverbrauch ein.

4.5 Abgrenzung zur Bachelor-Arbeit

Die in den Abschnitten 4.1, 4.2, 4.3 und 4.4 beschriebenen Arbeiten beschäftigen sich alle mit diversen Konzepten auf dem Gebiet des Complex Event Processings. Sie gehen dabei auf verschiedene Anfragetechniken der Ereignisströme und Verarbeitungsszenarien ein, wobei sich einzig die Arbeit 'Moving Range Queries in Distributed Complex Event Processing' mit der Ausführung eines CEP-Systems auf einer mobilen Umgebung beschäftigt. Die verschiedenen Ansätze zur Anfrage von Ereignissen in Ereignisströmen werden als Grundlage dieser Bachelor-Arbeit herangezogen. Weiterhin werden die Konzepte eines Operatorbaums und der Ausführung eines Operators der Implementierung zugrunde gelegt.

Zusammenfassend lässt sich feststellen, dass keine der hier vorgestellten Arbeiten eine Klassifikation der CEP-Operatoren bei einer mobilen Ausführung fokussiert. Außerdem wird der Energieverbrauch eines Smartphones bei der Ausführung eines CEP-Operators nicht in Betracht gezogen. Diese Bachelor-Arbeit baut auf den beschriebenen Arbeiten auf und stellt eine Klassifikation der CEP-Operatoren bei einer mobilen CEP-Ausführung vor. Weiterhin wird im Kapitel 10 der Fokus auf den Energieverbrauch von Smartphones gelegt.

5 Systemmodell

In diesem Kapitel wird das der Implementierung zugrunde gelegte Systemmodell erläutert. Weiterhin werden im letzten Abschnitt dieses Kapitels einige Annahmen über das System getroffen.

Das im folgenden beschriebene Systemmodell wird in Abbildung 5.1 veranschaulicht. In der Mitte der Darstellung befindet sich die Infrastruktur des CEP-Systems, die aus einem oder mehreren Servern aufgebaut sein kann. Die im Kreis um die Infrastruktur angeordneten CEP-Komponenten, wie beispielsweise Smartphones oder Server, können eine Verbindung zur Infrastruktur aufbauen. Eine Netzwerkverbindung ist in der abgebildeten Graphik mit einer einfachen, schwarzen Linie dargestellt. Jedes im CEP-System existierende Gerät kann Sensor, Operator und Konsument sein.

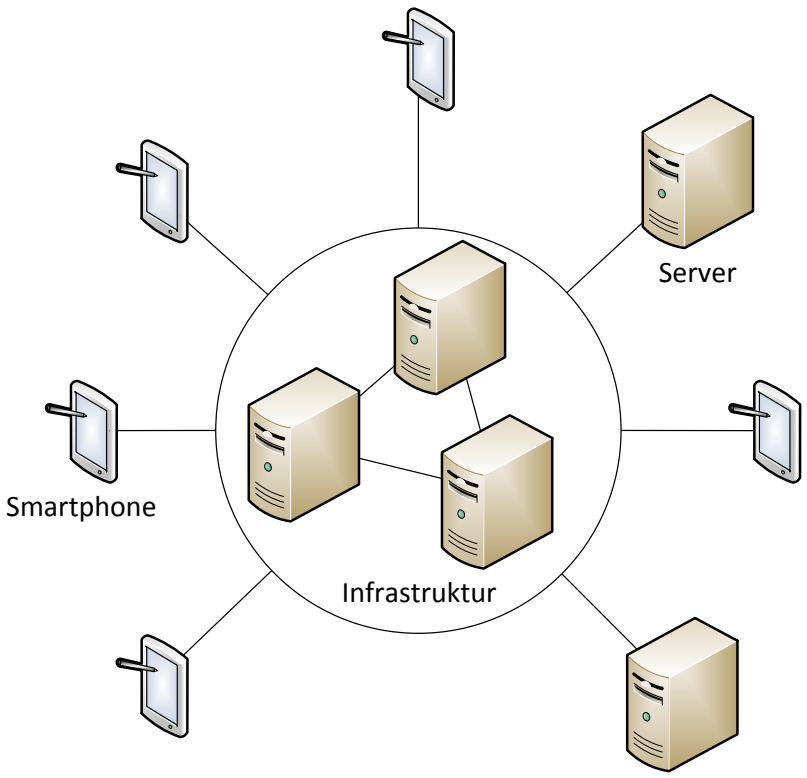


Abbildung 5.1: Das Systemmodell

Jedes Gerät des CEP-Systems stellt im Initialisierungsprozess eine Verbindung zur Infrastruktur her, deren Adresse öffentlich bekannt ist. Sowohl Smartphones als auch Server oder Computer können eine Verbindung, beispielsweise über ein WLAN- oder mobiles Datennetz, zur Infrastruktur aufbauen. Dabei kann jedes verbundene Gerät von anderen Geräten identifiziert und verwendet werden. So kann zum Beispiel ein Smartphone als Konsument im CEP-System ein anderes Smartphone als Sensor und einen auf einem Server befindlichen Operator nutzen, um einen Operatorbaum zu erstellen.

In dieser Bachelor-Arbeit wird der Fokus auf die Verwendung von Smartphones gerichtet. Ein Smartphone kann sowohl über ein WLAN-Netz als auch über ein mobiles Datennetz eine Verbindung zur Infrastruktur herstellen. Sollte die Verbindung kurzzeitig unterbrochen werden, können Ereignisse, die über die Infrastruktur zum Smartphone gesendet werden, in einem Puffer zwischengespeichert werden. Kann die Verbindung neu aufgebaut werden, sendet die Infrastruktur zwischengespeicherte Ereignisse zum Smartphone.

Die Infrastruktur übernimmt dabei das Routing. Sie kennt jedes im CEP-System aktive Gerät und leitet Ereignisse, die von einem Gerät an ein anderes bestimmt sind, weiter. Außerdem informiert sie jedes im CEP-System existierende Gerät, falls ein Gerät hinzukommt oder ausscheidet.

Systemannahmen

In diesem Abschnitt werden Annahmen getroffen, die während der gesamten Ausführung gelten.

Im Hinblick auf die Konnektivität wird davon ausgegangen, dass jedes Smartphone dauerhaft und zuverlässig über ein WLAN- oder Datennetz mit der Infrastruktur verbunden ist. Dabei tritt niemals ein Verlust von Datenpaketen aufgrund von Netzwerkpartitionierungen auf. Diese Annahme erleichtert die im Kapitel 8 durchgeführte Implementierung des mobilen CEP-Systems. Eine Möglichkeit, die Zuverlässigkeit von mobilen Netzwerk-Verbindungen zu verbessern, wird in der Arbeit 'Improving TCP/IP Performance over Wireless Networks' von Hari Balakrishnan et al. detailliert beschrieben [BSAK95].

Außerdem wird angenommen, dass jedes CEP-Gerät, mindestens jedoch das beobachtete, zentrale Objekt eines CEP-Systems, dauerhaft die aktuelle Position per GPS oder anderen Positionssystemen abfragen und an andere CEP-Komponenten im Operatorbaum versenden kann.

6 Architektur

Die Architektur des CEP-Systems, der das im Kapitel 5 beschriebene Systemmodell zugrunde liegt, veranschaulicht verschiedene Konfigurationsmöglichkeiten der CEP-Komponenten. Sie beschreibt außerdem, wie die Kommunikation zwischen einzelnen Komponenten ablaufen kann und zeigt dabei verschiedene Szenarien auf.

Zunächst wird im Abschnitt 6.1 eine Architektur für die lokale Ausführung eines CEP-Systems vorgestellt. Da die hier vorgestellte Architektur sehr generisch und modular aufgebaut ist, bedarf es keiner Änderung am System, um von einer lokalen Ausführung auf eine entfernte Ausführung umzustellen. Die entfernte Ausführung eines CEP-Systems kann mithilfe von mobilen Endgeräten oder unter der Verwendung von Servern durchgeführt werden. Eine detaillierte Beschreibung der entfernten CEP-Ausführung findet sich im Abschnitt 6.2.

6.1 Lokale Ausführung

Die Ausführung eines CEP-Systems wird *lokal* genannt, wenn alle Komponenten auf einem Gerät ausgeführt werden. In der hier beschriebenen Architektur wird dabei von einem mobilen Endgerät ausgegangen. Folglich ist die Kommunikation mit anderen Endgeräten oder entfernten Servern nicht notwendig.

Das in Abbildung 6.1 dargestellte mobile Endgerät beinhaltet die Komponenten *Sensoren*, *Operatoren* und *Konsumenten*. Es können parallel mehrere Sensoren, Operatoren und Konsumenten auf einem Gerät existieren und mithilfe eines Operatorbaums verwendet werden. Jeder der auf dem Gerät aktiven Konsumenten wird mit einem mit FMC modellierten Kanal zu einem oder mehreren Operatoren verbunden. Es handelt sich hierbei um einen gerichteten Kanal, sodass Ereignisse ausschließlich vom Operator zum Konsumenten gesendet werden können. Analog besteht zwischen jedem Sensor und Operator des CEP-Systems ein gerichteter Kanal, der den Transfer von Ereignissen vom Sensor zum Operator ermöglicht. Im Unterschied zu den Kanälen zwischen Sensor und Operator sowie Operator und Konsument, besteht zwischen Operatoren ein ungerichteter Kanal. Ein Operator kann folglich Ereignisse an einen anderen Operator senden und parallel Ereignisse von diesem Operator empfangen.

Die während der Ausführung vorhandene Konfiguration der Komponenten spezifiziert der Operatorbaum. Jeder Konsument kann mithilfe dieses Baumes dem CEP-System mitteilen, wie der Ereignisfluss stattfinden soll und welche Komponenten einbezogen werden sollen.

Bei dem in diesem Abschnitt beschriebenen Szenario werden keine CEP-Komponenten einbezogen, die sich außerhalb des lokalen, mobilen Endgeräts befinden. Als resultierender

Vorteil ergibt sich, dass das CEP-System unabhängig von einem WLAN- oder Datennetz ausgeführt werden kann. Als Folge kann jedoch keine Berechnung auf anderen mobilen Endgeräten oder Servern durchgeführt werden. Außerdem können keine Sensordaten, die von anderen Smartphones oder fest installierten Sensoren empfangen werden, zur Korrelation verwendet werden. Da jede Berechnung einer Korrelation von Ereignissen auf dem Gerät selbst ausgeführt werden muss, kann dessen CPU unter Umständen stark belastet werden.

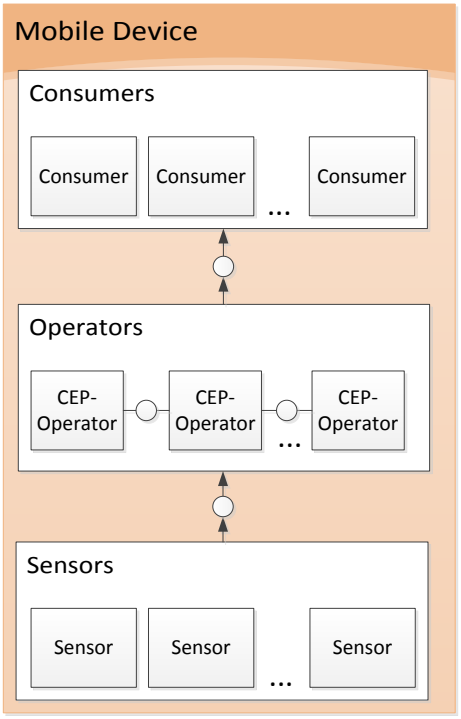


Abbildung 6.1: Architektur für eine lokale Ausführung eines CEP-Systems auf einem mobilen Endgerät

6.2 Entfernte Ausführung

Im Unterschied zur im Abschnitt 6.1 beschriebenen lokalen Ausführung eines CEP-Systems, kann das mobile Endgerät zusätzlich entfernte CEP-Komponenten verwenden. Es liegt eine *entfernte Ausführung* eines CEP-Systems vor, falls das System Komponenten verwendet, die nicht lokal auf dem Gerät ausgeführt werden können. Die Kommunikation mit diesen Komponenten findet über ein WLAN- oder Datennetz statt.

Die im Systemmodell beschriebene Infrastruktur, die Kommunikationskanäle zwischen den entfernten CEP-Komponenten aufbaut und für den Transfer der Ereignisse Sorge trägt, wird in den folgenden Architekturdiagrammen der Übersichtlichkeit dienend nicht darge-

stellt. Statt der hier direkt dargestellten Verbindung zweier entfernter CEP-Komponenten, werden Ereignisse über die Infrastruktur von einer zur anderen Komponente übertragen.

Ein mobiles Endgerät kann zur Ausführung eines CEP-Systems lokale Komponenten als auch Komponenten eines Servers oder anderer mobiler Endgeräte verwenden. In den nachfolgenden Abschnitten werden die drei Szenarien, Komponenten entfernt zu verwenden, unterteilt erläutert. Im Abschnitt 6.2.1 wird zunächst der CEP-Operator nicht lokal, sondern auf einem Server ausgeführt. Anschließend wird im Abschnitt 6.2.2 die Architektur für das Verwenden von Komponenten eines anderen Smartphones dargestellt. Im darauffolgenden Abschnitt 6.2.3 wird die Kommunikation mit einem Server und dessen CEP-Komponenten beschrieben.

6.2.1 Mobiles Gerät mit serverseitiger Ausführung des Operators

Im Gegensatz zur im Abschnitt 6.1 beschriebenen Architektur einer lokalen CEP-Ausführung kann beispielsweise der CEP-Operator nicht lokal auf dem Smartphone, sondern auf einem entfernten Server ausgeführt werden. Die dazugehörige Architektur ist in Abbildung 6.2 dargestellt.

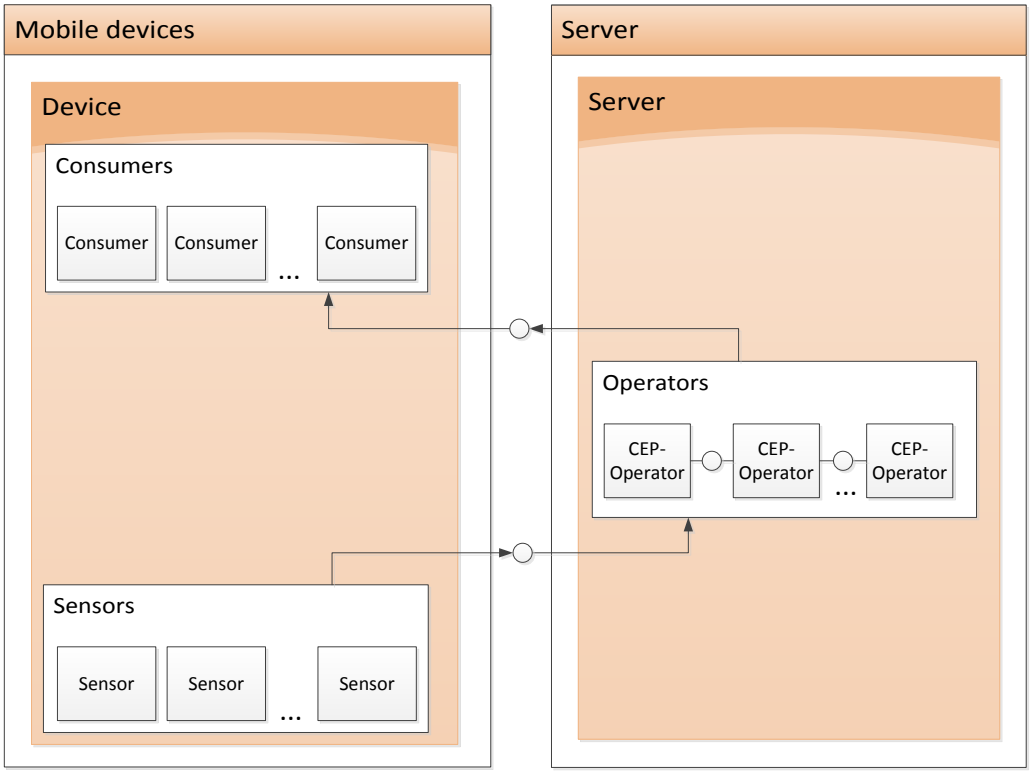


Abbildung 6.2: Architektur für eine entfernte Ausführung eines CEP-Operators auf einem Server

Der serverseitig ausgeführte Operator erhält Ereignisse, die von auf dem Smartphone befindlichen Sensoren erzeugt werden. Nachdem der Operator die Ereigniskorrelation und Verarbeitung durchgeführt hat, sendet er die erzeugten, komplexen Ereignisse zurück an das Smartphone.

6.2.2 Mobile Geräte

Besteht der Operatorbaum einer Ausführung eines CEP-Systems aus Sensoren oder Operatoren, die sich auf anderen mobilen Endgeräten befinden, kommt die in diesem Abschnitt beschriebene Architektur zum Einsatz.

Ein Smartphone kann beispielsweise sowohl einen lokalen Operator als auch einen Operator eines entfernten mobilen Endgeräts verwenden. Hierbei können Operatoren von lokalen oder entfernten Sensoren sowie von lokalen oder entfernten Operatoren Ereignisse beziehen.

In Abbildung 6.3 sind die beschriebenen Konfigurationsmöglichkeiten und verwendeten Kommunikationskanäle dargestellt. Der Ereignisfluss wird mit den in der Sprache FMC modellierten gerichteten und ungerichteten Kanälen veranschaulicht.

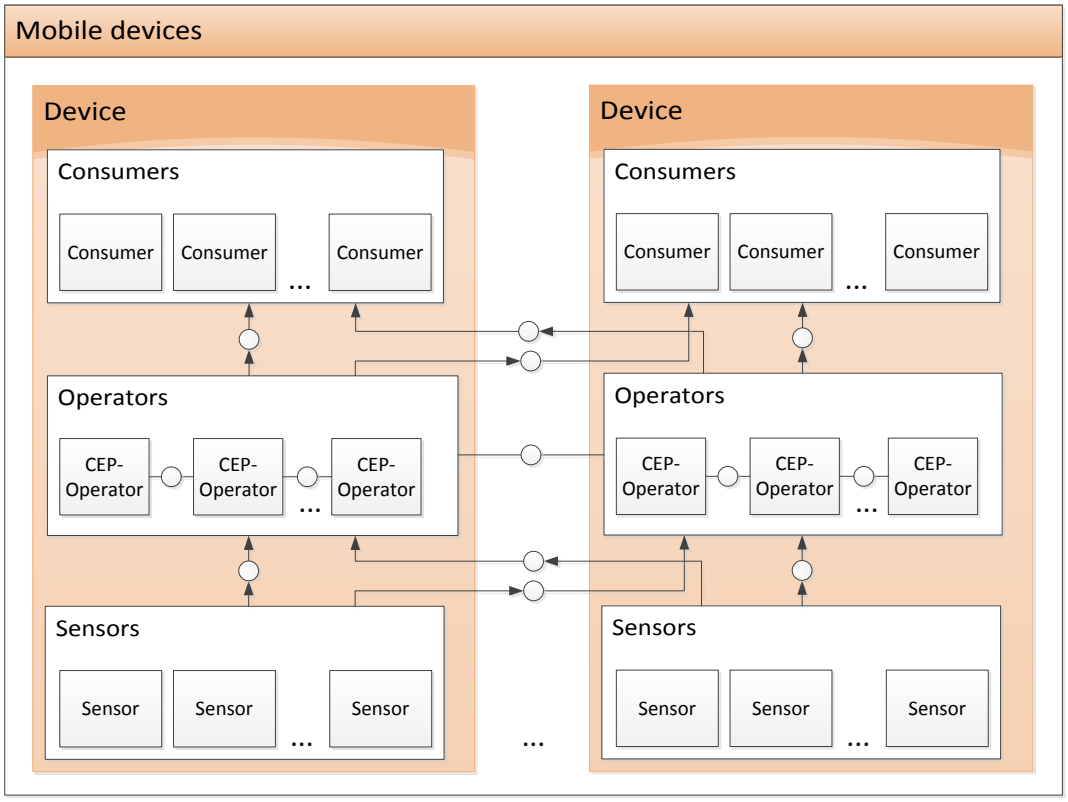


Abbildung 6.3: Architektur für die verteilte Ausführung eines CEP-Systems auf mehreren mobilen Endgeräten

Bei der Initialisierung des CEP-Systems spezifiziert der Konsument mithilfe eines Operatorbaums, welche Komponenten er für die Ausführung verwenden möchte. Falls es sich dabei um entfernte Komponenten handelt, kann definiert werden, ob das entfernte Gerät determiniert werden soll, oder ob das CEP-System ein passendes Gerät zur Ausführung wählen kann. Im Kapitel 8 wird der Initialisierungsprozess und die Wahl der im Operatorbaum enthaltenen Komponenten ausführlich erläutert.

6.2.3 Mobile Geräte und Server

Analog zur im Abschnitt 6.2.2 beschriebenen Ausführung entfernter CEP-Komponenten mithilfe anderer mobiler Endgeräte können ein oder mehrere Server verwendet werden. Die Kommunikation findet ebenfalls über in FMC modellierte Kanäle zwischen den CEP-Komponenten statt.

In Abbildung 6.4 ist die Kommunikation eines mobilen Endgeräts mit einem Server dargestellt. Anhand der drei Punkte rechts unterhalb der Komponenten 'Device' und 'Server' wird verdeutlicht, dass mehrere Geräte bzw. Server verfügbar sein können. Ihr Aufbau ist dabei analog zu den abgebildeten Komponenten.

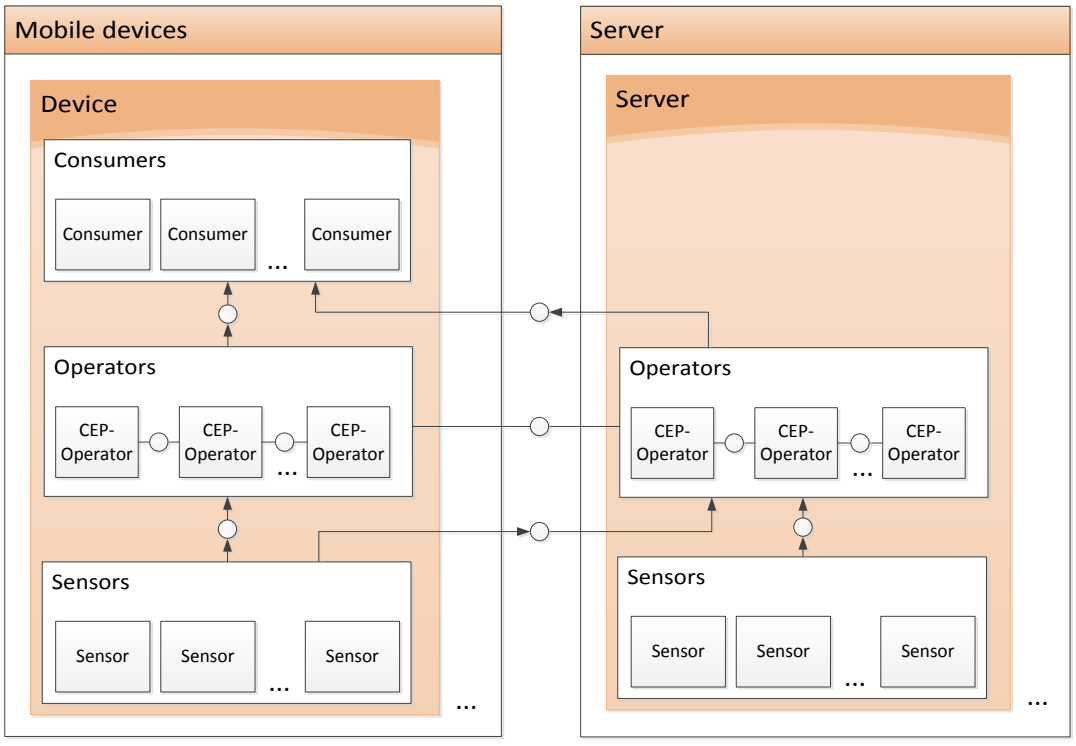


Abbildung 6.4: Architektur für die verteilte Ausführung eines CEP-Systems auf mehreren mobilen Endgeräten sowie auf mehreren Servern

Im Unterschied zur Kommunikation zwischen zwei mobilen Endgeräten kann ein Server keine Konsumenten beinhalten. Er kann aber Sensoren und Operatoren, ebenso wie ein Smartphone, dem CEP-System zur Ausführung bereit stellen. Ein auf einem Server ausgeführter Operator kann eingehende Ereignisse sowohl von eigenen, ebenfalls auf dem Server existierenden Sensoren, als auch von anderen Servern oder mobilen Endgeräten zur Verfügung gestellten Sensoren für die Ereigniskorrelation beziehen.

Zusammenfassend kann verdeutlicht werden, dass die hier vorgestellten Architekturen aus den Abschnitten 6.2.2 und 6.2.3 kombiniert werden können. Folglich kann ein Konsument eines mobilen Endgeräts mithilfe des Operatorbaums CEP-Komponenten adressieren, die sowohl lokal, auf anderen mobilen Endgeräten als auch auf einem oder mehreren Servern ausgeführt werden. Die verwendeten Komponenten können hierbei Ereignisströme anderer lokaler oder entfernter Komponenten des CEP-Systems verwenden.

7 Ausführungsmodell

In diesem Kapitel wird das Modell, das der Ausführung eines CEP-Operators zugrunde gelegt wird, detailliert beschrieben. Dabei werden unabhängig von der Implementierung Konzepte zur Selektion, Korrelation, Verarbeitung und dem Konsum der Ereignisse von Ereignisströmen erläutert. Das Ausführungsmodell basiert auf der Arbeit 'Moving Range Queries in Distributed Complex Event Processing' von Koldehofe et al., weshalb die Begrifflichkeiten übernommen wurden [KORR12].

Im Abschnitt 7.1 wird zunächst ein Überblick in die in diesem Kapitel verwendeten Konzepte gegeben. Darauffolgend wird im Abschnitt 7.2 die Selektion von Ereignissen auf einem Puffer des Operators dargestellt, während die Korrelation und anschließende Verarbeitung der Ereignisse im Abschnitt 7.3 ausgeführt werden. Schlussendlich wird im Abschnitt 7.4 der Fokus auf den Konsum der Ereignisse nach erfolgreicher Korrelation gerichtet.

7.1 Überblick der Konzepte

In diesem Abschnitt werden die verschiedenen Konzepte des Ausführungsmodells erläutert.

Bevor der Operator eine Korrelation und Ereignisverarbeitung durchführen kann, müssen vom Puffer eingehende Ereignisse gelesen werden. Um Ereignisse auszuwählen, wird das im Abschnitt 7.2 beschriebene Konzept der *Selektion von Ereignissen* verwendet.

Der zweite Schritt des Operators ist die Durchführung der *Ereigniskorrelation und Verarbeitung*. Dabei wird das im Abschnitt 7.3 erläuterte Konzept herangezogen.

Nachdem der Operator entsprechend seiner Semantik die Ereignisverarbeitung durchgeführt hat, werden die zur Korrelation verwendeten Ereignisse auf dem Puffer bearbeitet. Das Konzept des *Ereigniszustands nach der Korrelation* findet sich im Kapitel 7.4.

7.2 Selektion von Ereignissen

Ein CEP-Operator besteht sowohl aus einer Berechnungseinheit, die die Verarbeitung der Ereignisse einer Korrelation durchführt, als auch aus Puffern für eingehende Ereignisse. Für jeden eingehenden Ereignistyp, insbesondere für jeden eingehenden Ereignisstrom, erstellt der Operator einen Puffer. In Abbildung 7.1 wird ein möglicher Zustand der eingehenden Ereignispuffer eines CEP-Operators mit drei Puffern dargestellt. Ereignisse werden in dieser Darstellung mit einem grünen Rechteck symbolisiert. Hervorzuheben ist, dass die Abbildung eine Momentaufnahme des Zustands der eingehenden Ereignispuffer darstellt.

Um Ereignisse eines Puffers für eine Korrelation zu selektieren, werden sogenannte *Selection Windows* verwendet. Diese Fenster für die Auswahl von Ereignissen sind in der Abbildung 7.1 orange dargestellt. Ein Selection Window hat einen definierten Startzeitpunkt und eine definierte Länge, wodurch der Endzeitpunkt des Fensters implizit gegeben ist. Das Fenster enthält zusätzlich ein ganzzahliges Attribut 'Slots', womit ausgedrückt werden kann, ob sich ein Selection Window im Zustand 'bereit zur Korrelation' oder 'nicht bereit zur Korrelation' befindet. Das Attribut definiert, wie viele Ereignisse im Fenster enthalten sein müssen, damit das Fenster den Zustand 'bereit zur Korrelation' erhält.

Ist das Attribut 'Slots' beispielsweise auf '5' gesetzt und es befinden sich aktuell nur vier Ereignisse im Selection Window, so verbleibt das Fenster im Zustand 'nicht bereit zur Korrelation'. Empfängt der Puffer des Operators ein weiteres Ereignis, das in den Zeitrahmen des Selection Windows fällt, befinden sich nun fünf Ereignisse im Fenster. Sobald die Zahl der Ereignisse in einem Fenster größer gleich der im Attribut 'Slots' definierten Anzahl ist, wird der Zustand des Selection Windows auf 'bereit zur Korrelation' gesetzt.

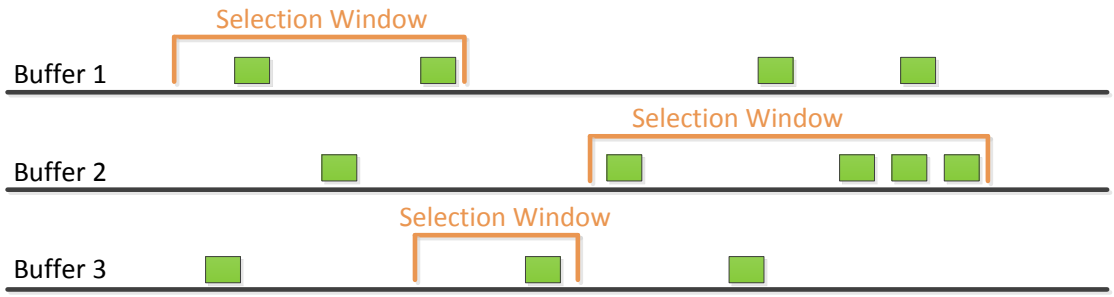


Abbildung 7.1: Möglicher Zustand der Puffer eingehender Ereignisströme eines CEP-Operators

7.3 Ereigniskorrelation und Verarbeitung

Eine Ereigniskorrelation wird vom Operator genau dann ausgeführt, wenn alle im Abschnitt 7.2 beschriebenen Selection Windows den Zustand 'bereit zur Korrelation' erreicht haben. Der Operator wird alle Ereignisse der Puffer mit den zugehörigen Selection Windows zur Korrelation heranziehen und die individuelle Berechnung vornehmen. Dabei definiert jeder Operator ω eine Funktion $f_\omega(S)$, die die vom Selection Window ausgewählten Ereignisse (S) der eingehenden Ströme auf die ausgehenden, vom Operator erstellten Ereignisse abbildet.

Nachdem der Operator die Korrelation und anschließende Berechnung durchgeführt hat, werden die Selection Windows auf der Zeitachse der Puffer verschoben. Dabei definiert jeder Operator eine sogenannte *Selection Policy*. Hier kann definiert werden, ob der Operator ein Selection Window auf dem Puffer nach rechts oder links bewegt. Zusätzlich kann festgelegt werden, ob das Selection Window 'time-dependend', unter Angabe einer Zeit

in Millisekunden, oder 'event-dependend', unter der Angabe der Anzahl von Ereignissen, verschoben wird.

Wird in der Selection Policy definiert, dass das Selection Window nach einer Korrelation beispielsweise 'time-dependend' um '+5000 ms' verschoben werden soll, erhöht das System den Startzeitpunkt des Fensters um fünf Sekunden. Wird jedoch 'event-dependend' mit der Konstanten '-2' gewählt, wird das Selection Window so weit nach links in die Vergangenheit geschoben, bis zwei Ereignisse einen höheren Zeitstempel als der Startzeitpunkt des Selection Windows aufweisen.

7.4 Ereigniszustand nach der Korrelation

Nach der Korrelation, Verarbeitung der korrelierten Ereignisse sowie dem Verschieben der Selection Windows aufgrund ihrer Selection Policy können korrelierte Ereignisse des Puffers bearbeitet werden. Dabei kann mithilfe einer *Consumption Policy* definiert werden, welchen Zustand korrelierten Ereignissen zugewiesen werden soll. Der Entwickler des CEP-Operators kann hierbei zwischen den folgenden drei Varianten wählen:

- Delete
- Ignore
- Mark as correlated

Wird als Consumption Policy 'Delete' gewählt, so werden alle korrelierten Ereignisse aller Puffer vom System gelöscht. Sie stehen folglich nicht für weitere Korrelation zur Verfügung. Im Gegensatz dazu werden Ereignisse vom System nicht weiter bearbeitet, stehen aber für weitere Korrelationen zur Verfügung, wenn der Entwickler 'Ignore' wählt. Besagt die Consumption Policy 'Mark as correlated', wird das System die korrelierten Ereignisse markieren. Sie stehen damit vorläufig nicht zur weiteren Korrelation zur Verfügung, verbleiben jedoch auf dem Puffer. Ereignisse, die als korreliert markiert wurden, können zu einem späteren Zeitpunkt frei geschaltet werden, sodass sie zukünftigen Korrelationen zur Verfügung stehen. Diese Möglichkeit wird benötigt, um eine optionale Optimierung der Ereignisverarbeitung, die in der Arbeit [KORR12] von Koldehofe et al. beschrieben wird, zu gewährleisten.

8 Implementierung

Die Implementierung des mobilen CEP-Systems gliedert sich in den folgenden Abschnitten wie folgt: Das Framework des CEP-Systems wird im Abschnitt 8.1 beschrieben. Detailliert wird dabei auf die Komponenten des CEP-Frameworks, Ausführungsumgebungen und die Transportschicht eingegangen. Das Framework ist modular und sehr generisch aufgebaut, sodass sich die Transportschicht auswechseln lässt, ohne eine Beeinträchtigung oder Anpassung der anderen im CEP-System enthaltene Komponenten oder Ausführungsumgebungen hervorzurufen.

In den angeschlossenen Abschnitten 8.2, 8.3 und 8.4 wird auf die Implementierung der im CEP-System verwendeten Komponenten eingegangen, insbesondere auf die im Ausführungsmodell beschriebenen CEP-Operatoren.

Eine Besonderheit der Implementierung des CEP-Systems ist die Variabilität und Generizität. Das CEP-System wurde mithilfe der Programmiersprache Java entwickelt und ist daher sowohl auf Computern und Servern mit einer Ausführungsplattform für Java-Programmcode als auch auf mobilen Endgeräten mit dem Betriebssystem Android ausführbar.
Außerdem ist die Verwendung der Transportschicht variabel. Das Framework bietet die Möglichkeit, die Transportschicht auszutauschen und somit neben der Verbindung zu einer Infrastruktur oder dem Versenden von Ereignissen mithilfe eines Broadcasts auch andere Transport-Paradigmen verwenden zu können.

8.1 CEP-Framework

Da das Framework für die mobile Ausführung eines CEP-Systems auf diversen Geräten lauffähig sein soll, erfordert es einen hohen Grad an Modularität und Generizität. Die Transportschicht des CEP-Systems ist mithilfe von Interfaces gekapselt, sodass diese leicht ersetzbar ist. Es besteht somit sowohl die Möglichkeit der Verwendung einer Transportschicht, die Paradigmen des Broadcastings verwendet, als auch das Einsetzen einer zentralen, entfernten Infrastruktur, die für die Kommunikation zwischen den CEP-Komponenten verwendet wird.

In den folgenden Unterkapiteln werden zunächst die in der Architektur beschriebenen Komponenten, die Ausführungsumgebungen und schlussendlich die Transportschicht beschrieben.

8.1.1 Komponenten

Das CEP-Framework bietet Interfaces für die aus dem CEP-Paradigma bekannten Komponenten an. Die Wahl der Schnittstellen basiert dabei auf den im Abschnitt 2.1.1 des Grundlagenkapitels 2 beschriebenen Komponenten eines CEP-Systems. Die Hierarchie der Komponenten ist in Abbildung 8.1 beschrieben. Da Operatoren sowohl eingehende als auch ausgehende Ereignisströme besitzen, erbt die Komponente 'IOperator' von beiden Interfaces. Eine weitere Verfeinerung des 'IOperators' ist der 'ILocalOperator'. Beim 'ILocalOperator' handelt es sich um einen 'IOperator', der lokal auf dem Ausführungsgerät vorhanden ist und von der lokalen Ausführungsumgebung gesteuert wird.

Die Komponente 'ISensor' mit ihrer abstrakten Implementierung 'AbstractSensor' erbt vom Interface 'ICEPStreamProducer', da sie ausschließlich ausgehende Ereignisströme produziert. Eine weitere Implementierung des 'ISensor'-Interfaces, der 'DefaultSensor', bietet die grundlegende Implementierung der vom Interface definierten Methoden an. Der Entwickler muss bei der Verwendung des 'DefaultSensor' nur das spezifische 'Sensoring' implementieren.

Das Interface 'IConsumer' beschreibt Methoden des Konsumenten eines Ereignisstroms und erbt daher vom Interface 'ICEPStreamConsumer'.

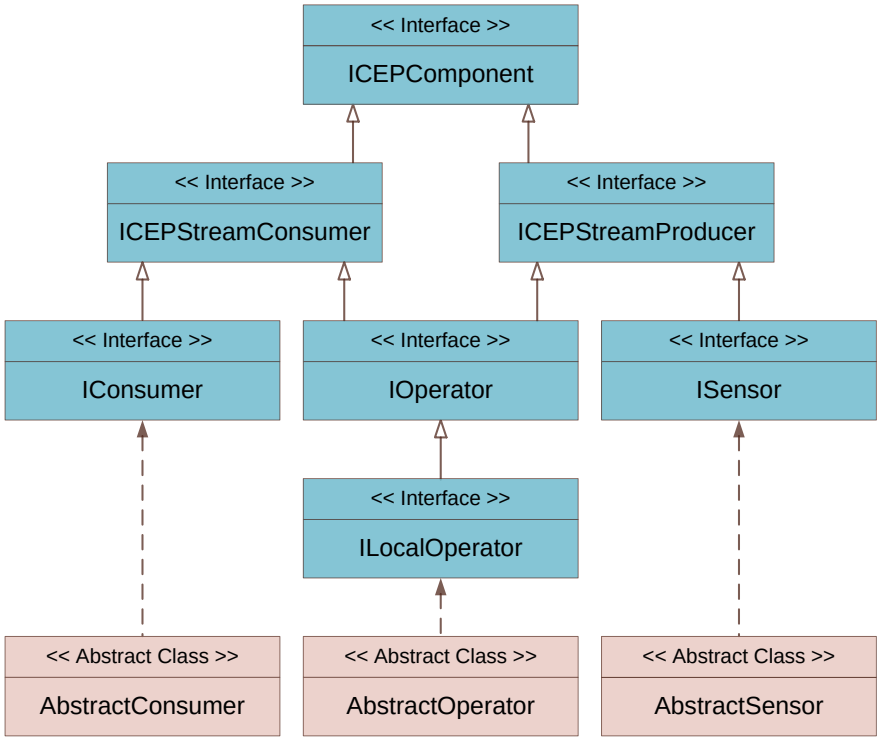


Abbildung 8.1: Hierarchie der Interfaces und abstrakten Implementierungen der CEP-Komponenten

Ereignistypen

Um Ereignisse in einem CEP-System unterscheiden zu können, wird jedem Ereignis ein bestimmter Typ zugeordnet. Der Typ eines Ereignisses wird in Form eines Strings definiert.

Weiterhin wird in den im Abschnitt 8.1.1 beschriebenen Interfaces 'ICEPStreamProducer' und 'ICEPStreamConsumer' der Typ der entsprechend konsumierten bzw. produzierten Ereignisse definiert. Dabei bietet das Interface 'ICEPStreamProducer' die Methode 'getOutputEvents()' an, die die entsprechend produzierten Ereignistypen zurückgibt. Analog kann die Methode 'getInputEvents()' im Interface 'ICEPStreamConsumer' verwendet werden, um die konsumierten Ereignistypen zu erhalten. Im Listing 8.1 ist eine beispielhafte Implementierung der beiden Methoden abgebildet. Der Operator, der sowohl die Interfaces 'ICEPStreamProducer ' als auch 'ICEPStreamConsumer' realisiert, muss beide angegebenen Methoden implementieren. In Zeile 3 bis 5 wird eine Liste erstellt, der zwei konstante Zeichenketten angehängt werden. Die beiden Konstanten definieren die Ereignistypen, die vom Operator als eingehende Ereignisse verarbeitet werden können. Analog werden die ausgehenden Ereignistypen in der Methode 'getOutputEventTypes()' in den Zeilen 11 bis 18 definiert.

Listing 8.1 Beispiel für Input- und Output-Eventtypen

```
1  public List<String> getInputEventTypes() {
2
3          List<String> inputEventTypes = new ArrayList<String>();
4          inputEventTypes.add(SensorAndroidGPSType.EVENT_TYPE_PRODUCED_LOCATION);
5          inputEventTypes.add(SensorDatabaseGPSType.EVENT_TYPE_PRODUCED_POSITION);
6
7          return inputEventTypes;
8
9  }
10
11 public List<String> getOutputEventTypes() {
12
13         List<String> outputEventTypes = new ArrayList<String>();
14         outputEventTypes.add(OperatorType.EVENT_TYPE_PRODUCED);
15
16         return outputEventTypes;
17
18 }
```

Initialisieren des CEP-Frameworks

Jedes im CEP-System vorhandene Gerät muss das CEP-Framework initialisieren, bevor die Kommunikation mit anderen CEP-Komponenten sowie die Ausführung der lokalen Operatoren möglich ist. Die beispielhafte Implementierung des Initialisierungsprozesses ist im Listing 8.2 zu finden.

In den Zeilen 4, 5 und 6 werden die Klassen des Transport Layers, der Execution Engine und der Transport Engine gesetzt. Sollte keine eigene Implementierung der Execution beziehungsweise Transport Engine vorhanden sein, kann die Standard-Implementierung

'DefaultExecutionEngine' bzw. 'DefaultTransportEngine' aus dem CEP-Framework verwendet werden. Sollten eigene Implementierungen angegeben werden, müssen die vom Framework definierten Interfaces implementiert werden. Nachdem die 'initialize()'-Methode des CEP-Frameworks in Zeile 9 aufgerufen wurde, können lokal auf dem Gerät ausführbare CEP-Komponenten dem CEP-Framework bekannt gemacht werden. Hierbei wird eine Instanz einer CEP-Komponente mithilfe der Methode 'registerLocalComponent()' dem Framework und schlussendlich dem kompletten CEP-System bekannt gemacht. So wird in Zeile 12 eine Instanz des Konsumenten erstellt und weiter in Zeile 13 die Registrierung der Instanz per CEP-Framework durchgeführt. Analog wird der lokale Operator 'LocalOperator' dem CEP-Framework in Zeile 16 und 17 bekannt gemacht. Um alle lokalen Sensoren zu registrieren, wird die in Zeile 20 bis 23 abgebildete Schleife verwendet. Entsprechend des Parameters 'anzahlSensoren' wird die Schleife mehrmals durchlaufen. Bei jedem Schleifendurchlauf wird eine neue Instanz des Sensors 'LocalSensor' erstellt und beim CEP-Framework registriert. Sind alle lokalen CEP-Komponenten registriert, kann das CEP-Framework anhand des Methoden-Aufrufs in Zeile 26 gestartet werden.

Listing 8.2 Initialisieren des CEP-Frameworks

```
1  private void initializeCEPFramework(int anzahlSensoren) {
2
3          // Setzen der Klassen des Transport Layers, der Execution und Transport Engine
4          CEPFramework.get().setTransportLayer(TransportLayerInfrastructure.class);
5          CEPFramework.get().setExecutionEngineClass(DefaultExecutionEngine.class);
6          CEPFramework.get().setTransportEngineClass(DefaultTransportEngine.class);
7
8          // Initialisierungsprozess des CEP-Frameworks starten
9          CEPFramework.get().initialize();
10
11         // Instantiierung und Registrierung des Consumers
12         IConsumer consumer = new LocalConsumer();
13         CEPFramework.get().registerLocalComponent(consumer);
14
15         // Instantiierung und Registrierung des Operators
16         IOperator operator = new LocalOperator();
17         CEPFramework.get().registerLocalComponent(operator);
18
19         // Instantiierung und Registrieren der Sensor-Komponenten
20         for (int itr = 0; itr < anzahlSensoren; itr++) {
21                ISensor sensor = new LocalSensor();
22                CEPFramework.get().registerLocalComponent(sensor);
23         }
24
25         // Starten des CEP-Frameworks
26         CEPFramework.get().start();
27
28 }
```

8.1.2 Transport Layer

Bei der im Abschnitt 8.1.1 beschriebenen Initialisierung des CEP-Frameworks wird unter anderem die Klasse des zu verwendenden Transport Layers angegeben. In diesem Abschnitt werden die Aufgaben, die Funktionsweise und die Modularität des Transport Layers vorgestellt. Der für diese Bachelor-Arbeit speziell entwickelte Transport Layer für die Einbindung und Verwendung einer Infrastruktur wird im Abschnitt 8.1.2 ausführlich erläutert.

Zu den Aufgaben eines Transport Layers zählt sowohl der Kommunikationsauf- und abbau als auch das Versenden und Empfangen von Datenpaketen. Das CEP-Framework definiert das Interface 'ITransportLayer', das vom verwendeten Transport Layer implementiert werden muss. Die Schnittstellenspezifikation ist im Listing 8.3 angeführt.

Listing 8.3 Das Interface 'ITransportLayer'

```
1
2  public interface ITransportLayer {
3
4      // Adresse des Geraets in Form einer URI
5      public URI getLocalAddress();
6
7      // Veroeffentlichen und Abmelden eines Geraets
8      public void publishDevice(IDevice device);
9      public void unpublishDevice(IDevice device);
10
11     // Veroeffentlichen und Abmelden einer CEP-Komponente
12     public void publishComponent(ICEPComponent component);
13     public void unpublishComponent(ICEPComponent component);
14
15     // Registrieren und Deregistrieren eines Empfaengers fuer sich aendernde Geraete
16     public void registerDeviceChangedListener(DeviceChangedListener listener);
17     public void unregisterDeviceChangedListener(DeviceChangedListener listener);
18
19     // Registrieren und Deregistrieren eines Empfaengers fuer sich aendernde
           CEP-Komponenten
20     public void registerComponentChangedListener(ComponentChangedListener listener);
21     public void unregisterComponentChangedListener(ComponentChangedListener listener);
22
23     // Beginnen und Beenden der Uebertragung von Ereignissen ueber einen Kanal
24     public void startEventTransmission(IChannel channel);
25     public void stopEventTransmission(IChannel channel);
26
27     // Senden von Ereignissen ueber einen Kanal
28     public void sendEvent(IChannel channel, IEvent event);
29
30 }
```

Das CEP-Framework veröffentlicht während des Initialisierungsprozesses das Gerät im CEP-Netzwerk und ruft dazu die Methode 'publishDevice(IDevice device)' im Transport Layer auf. Analog wird die Methode 'unpublishDevice(IDevice device)' aufgerufen, falls ein Gerät das CEP-Framework beendet.

Damit das auf dem Gerät ausgeführte CEP-Framework über Änderungen anderer Geräte oder derer Komponenten informiert wird, werden über die Methoden 'registerDeviceChangeListener(DeviceChangeListener listener)' und 'registerComponentChangedListener(ComponentChangedListener listener)' Empfänger am Transport Layer registriert. Das Entfernen der Empfänger kann analog mit den 'unregister'-Methoden durchgeführt werden.

Anhand der Methoden in den Zeilen 24 und 25 kann die Übertragung von Ereignissen gestartet bzw. beendet werden. Der als Parameter gesetzte Kanal definiert den Ereignisproduzenten und -konsumenten. Die Methode 'sendEvent(IChannel channel, IEvent event)' erlaubt das Versenden eines Ereignisses über einen spezifizierten Kanal.

Die in diesem Abschnitt beschriebene Schnittstelle für eine Transportschicht kann von beliebigen Klassen implementiert werden. Als Folge kann der Entwickler einen eigenen Transport Layer implementieren und ohne Änderung des Frameworks die eigene Transport-Implementierung verwenden.

Infrastruktur

Der für diese Bachelor-Arbeit implementierte Transport Layer zur Einbindung einer entfernten Infrastruktur wird in diesem Abschnitt vorgestellt.

Die Infrastruktur besteht in dieser Bachelor-Arbeit aus einem Server, der eine in Java implementierte Server-Software ausführt. Jedes im CEP-System enthaltene Gerät kann mithilfe einer öffentlich bekannten Adresse eine TCP-Verbindung zur Infrastruktur herstellen. Dabei registriert sich ein Gerät zunächst mit dem Befehl 'PUBLISH_DEVICE', bevor es die lokal vorhandenen CEP-Komponenten anhand des Befehls 'PUBLISH_COMPONENT' anmeldet. Die Infrastruktur versendet bei der An- oder Abmeldung von Geräten oder Komponenten Nachrichten an alle anderen im System befindlichen Geräte.

Zur Erstellung von Transportkanälen wird das Kommando 'CREATE_CHANNEL' verwendet. Die Infrakstruktur stellt folglich einen Kanal zwischen den spezifizierten Komponenten her. Dabei wird der Produzent und der Konsument des Kanals mittels Nachrichten informiert. Sendet der Produzent ein Ereignis über den entsprechenden Kanal, wird das Ereignis an den Konsumenten des Kanals weitergeleitet. Scheidet der Konsument kurzzeitig aus dem CEP-System aus, werden die Ereignisse in der Infrastruktur zwischengespeichert und bei erneut erfolgreichem Verbindungsaufbau an den Konsumenten gesendet.

Für die Kommunikation und Einbindung der Infrastruktur wurde das Interface 'ITransportLayer' entsprechend implementiert und bei der in diesem Kapitel beschriebenen Ausführung verwendet.

8.1.3 Dynamic Interest Query

In diesem Abschnitt wird zunächst das Konzept des *Dynamic Interest Query*, kurz DIQ, beschrieben. Anschließend werden die Bestandteile eines DIQ detailliert erläutert sowie eine Beispiel-Implementierung vorgestellt.

Ein *Dynamic Interest Query* ist die Anfrage eines Konsumenten an das CEP-System, eine CEP-Ausführung zu starten. Dabei muss der Konsument die Parameter *Monitored Focal Object*, *Operator Tree*, *Range* und *Delta* spezifizieren.

Das Monitored Focal Object, kurz MFO, beschreibt das Objekt von Interesse. Wie auch in dieser Bachelor-Arbeit ist das MFO in den meisten Fällen der Konsument der CEP-Ausführung. Der Konsument muss dabei in regelmäßigen Abständen die aktuelle Position an das CEP-System übermitteln. In der hier vorgestellten Implementierung kann als Parameter eine Instanz der Schnittstelle 'IConsumer' verwendet werden, da diese vom Interface 'IMonitoredFocalObject' ableitet. Der Entwickler muss in dieser Implementierung dafür sorgen, dass die aktuelle GPS-Position dem CEP-System zur Verfügung steht.

Der Parameter *Range* spezifiziert den geographischen Bereich, in dem vom CEP-System berechnete Ereignisse von Interesse sind. Der Mittelpunkt der Range ist gegeben durch die aktuelle Position des Monitored Focal Objects. Berechnet das CEP-System Ereignisse für den Konsumenten, werden diese nur dann versendet, wenn das Ereignis dem geographisch definierten Bereich zugeordnet ist. Das Framework bietet die zwei Implementierungen 'CircleRange' und 'RectangleRange' der Schnittstelle 'IRange' an. Mithilfe der Klasse 'CircleRange' kann ein kreisförmiger Bereich mit Radius definiert werden. 'RectangleRange' erlaubt das Erstellen eines rechteckigen Bereichs, der vom Entwickler anhand der Parameter 'Länge' und 'Breite' definiert wird.

Verändert das 'Monitored Focal Object' seine Position, wird zunächst die spezifizierte 'Range' angepasst. Aufgrund dieser Bereichsänderung würde das CEP-System Ereignisse, die im Bereich vor der Bereichsänderung liegen, nicht an den Konsumenten weiterleiten. Damit über eine definierte Zeitspanne hinweg Ereignisse aus Bereichen der Vergangenheit an den Konsumenten gesendet werden können, beinhaltet ein DIQ den Parameter *Delta*. Der Entwickler kann mithilfe dieses Parameters die Zeitspanne definieren, in der Ereignisse aus der Vergangenheit trotz einer geographisch außerhalb des berechneten Gebiets liegenden Position an den Konsumenten versendet werden.

Der vierte und wichtigste Parameter eines DIQs definiert die zu verwendenden CEP-Komponenten und deren Ereignisfluss. Der Parameter wird als 'Operator Tree' betitelt und wird im nachfolgenden Abschnitt 8.1.3 beschrieben.

Nachdem ein Entwickler alle Parameter eines DIQs spezifiziert und somit einen DIQ vollständig erstellt hat, kann er den DIQ am CEP-Framework anmelden. Das CEP-Framework startet nach erfolgreicher Prüfung des DIQs auf dessen Korrektheit die CEP-Ausführung. Im Listing 8.4 sind die entsprechenden Methoden abgebildet. Anhand des in Zeile 4 dargestellten Methodenaufrufs kann ein erstellter DIQ registriert werden. Analog erfolgt die Abmeldung eines DIQs in Zeile 11. Die Prüfung und Ausführung des DIQs wird automatisch vom CEP-Framework durchgeführt und muss daher nicht explizit vom Entwickler veranlasst werden.

Listing 8.4 Registrieren bzw. De-Registrieren eines 'Dynamic Interest Querys'

```
1  private void registerDIQ() {
2
3          // Registriere Dynamic Interest Query beim CEP-Framework
4          CEPFramework.get().getExecutionEngine().registerDIQ(Consumer.this.diq);
5
6  }
7
8  private void unregisterDIQ() {
9
10         // De-Registriere Dynamic Interest Query beim CEP-Framework
11         CEPFramework.get().getExecutionEngine().unregisterDIQ(Consumer.this.diq);
12
13 }
```

Operator Tree

Ein Parameter des im Abschnitt 8.1.3 beschriebenen DIQs ist der hier detailliert erläuterte *Operator Tree*. Der Operatorbaum spezifiziert alle von der CEP-Ausführung verwendeten Komponenten und deren Ereignisfluss.

In der Abbildung 8.2 ist eine beispielhafte Struktur eines Operatorbaums dargestellt.

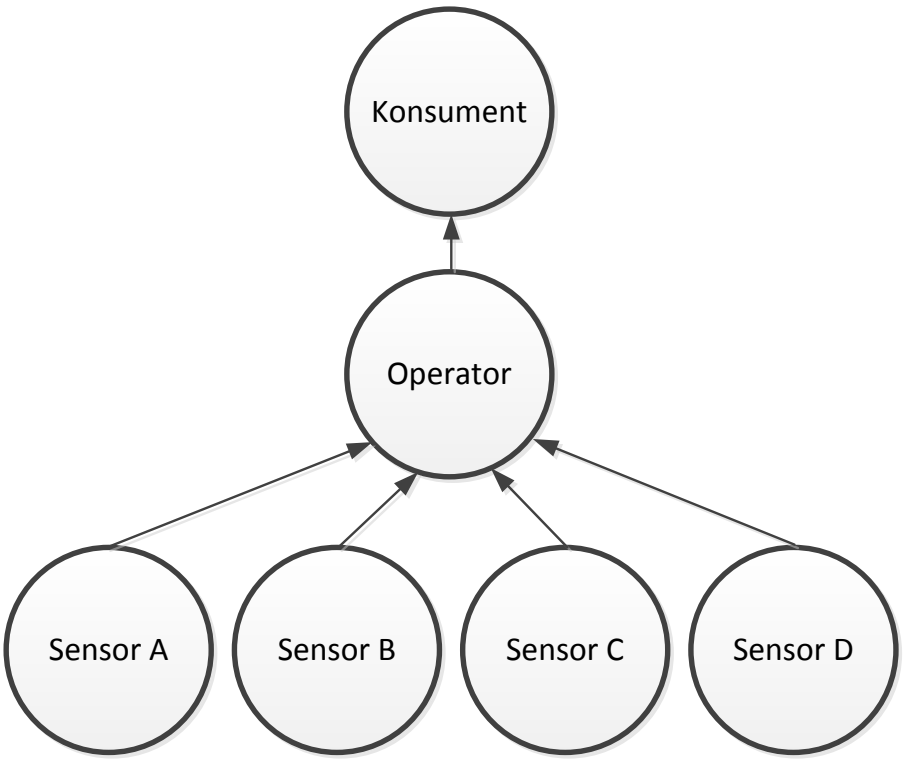


Abbildung 8.2: Beispiel eines Operatorbaums

Die Kreise der Abbildung 8.2 symbolisieren CEP-Komponenten, wobei das zur Ausführung der Komponente erforderliche Gerät nicht spezifiziert werden muss. Pfeile zwischen den Kreisen verdeutlichen den gerichteten Ereignisfluss.

Der Ersteller eines Operatorbaums kann exakt spezifizieren, ob eine CEP-Komponente auf einem angegebenen Gerät ausgeführt werden muss oder ob das System nach einem passenden Gerät suchen kann. Im Listing 8.5 ist eine beispielhafte Implementierung der Erstellung eines Operatorbaums abgebildet.

Nachdem der Bezeichner des Geräts in Zeile 5 und 6 ausgelesen wurde, kann in Zeile 9, 10 und 11 ein Knoten des Operatorbaums erstellt werden. Der Konstruktor der Klasse 'Node' besitzt als Parameter ein Objekt der Klasse 'IComponentFilter'. Anhand des hier angegebenen 'DefaultComponentFilter' kann eine CEP-Komponente exakt spezifiziert werden. Zunächst wird der Typ der Komponente angegeben, der den Wert 'Sensor', 'Operator' oder 'Consumer' haben kann. Danach folgt ein als Zeichenkette spezifizierter Subtyp, der den Namen der Komponente enthält. Die letzten beiden Parameter des 'DefaultComponentFilter'-Konstruktors können optional angegeben werden. Der Entwickler kann sowohl den Bezeichner des Geräts der Komponente, als auch den Bezeichner der Komponente selbst angeben. Werden beide Parameter mit 'null' belegt, sucht das System im CEP-Netzwerk nach geeigneten Komponenten.

Nachdem ein Objekt der Klasse 'OperatorTree' in Zeile 14 angelegt wurde, wird in Zeile 17, 18 und 19 der Knoten des Operators angelegt. Dies erfolgt analog zum Erstellen des Knotens für den Konsumenten.

Um den Ereignisfluss beschreiben zu können, werden sogenannte Kanten, englisch Edges, verwendet. Eine Kante wird anhand einer Quell- und einer Ziel-Komponente des Operatorbaums spezifiziert. Die Komponenten 'Consumer' und 'Operator' werden im abgebildeten Beispiel 8.2 anhand einer instanziierten Kante 'edgeConsumerOperator' in Zeile 22 miteinander verknüpft. Anschließend wird in Zeile 23 dem Operatorbaum die erstellte Kante hinzugefügt.

In den Zeilen 22 bis 36 wird analog für jeden Sensor des CEP-Systems ein Knoten erstellt. Zuletzt wird mithilfe einer Kante der Ereignisfluss von jedem Sensor zum Operator definiert.

8.1.4 Execution Engine

Das CEP-Framework stellt jedem Gerät eine sogenannte *Execution Engine* zur Verfügung. Die Execution Engine besteht aus zwei Modulen, die von verschiedenen CEP-Komponenten genutzt werden können.

In den nachfolgenden Abschnitten werden die beiden Module und deren Implementierung kurz vorgestellt.

Consumer Execution Engine

Die *Consumer Execution Engine* wird von der CEP-Komponente 'Consumer' verwendet und startet eine CEP-Ausführung anhand eines im Abschnitt 8.1.3 beschriebenen DIQs.

Listing 8.5 Erstellen eines Operatorbaums

```
1  private void createOperatorTree(String operatorIdentifier,
2                                   List<String> sensorIdentifierList) {
3
4          // Der Bezeichner des Geraets
5          String deviceIdentifier = CEPFramework.get().getLocalDevice()
6                          .getIdentifier();
7
8          // Erstelle Knoten fuer den Consumer
9          Node consumerNode = new Node(new DefaultComponentFilter(
10                         ComponentType.CONSUMER, Consumer.this.getSubType(),
11                         deviceIdentifier, Consumer.this.getIdentifier()));
12
13         // Erstelle den Operatorbaum. Als Parameter wird der Knoten des Consumers
               mitgegeben.
14         Consumer.this.operatorTree = new OperatorTree(consumerNode);
15
16         // Erstelle Knoten fuer den Operator
17         Node operatorNode = new Node(new DefaultComponentFilter(
18                         ComponentType.OPERATOR, LocalOperatorType.COMPONENT_TYPE,
19                         null, null));
20
21         // Erstelle die Kante zwischen Operator und Consumer und fuege sie dem
               Operatorbaum hinzu.
22         Edge edgeConsumerOperator = new Edge(operatorNode, consumerNode);
23         Consumer.this.operatorTree.addEdge(edgeConsumerOperator);
24
25         // Erstelle die Knoten der Sensoren
26         int i = 0;
27         for (String sensorID : sensorIdentifierList) {
28                Node sensorNode = new Node(new DefaultComponentFilter(
29                               ComponentType.SENSOR, LocalSensorType.COMPONENT_TYPE + i,
30                               deviceIdentifier, sensorID));
31
32                // Erstelle Kante vom Sensor zum Operator und fuege sie dem Operatorbaum
                      hinzu.
33                Edge edgeOperatorSensor = new Edge(sensorNode, operatorNode);
34                Consumer.this.operatorTree.addEdge(edgeOperatorSensor);
35                i++;
36         }
37
38 }
```

Der Entwickler muss hierfür die vom Framework zur Verfügung gestellte und im Listing 8.6 abgebildete Methode 'registerDIQ(IDynamicInterestQuery diq)' aufrufen.

Nachdem die Ausführung eines DIQs veranlasst wurde, traversiert die Consumer Engine den im DIQ enthaltenen Operatorbaum und sucht nach den angeforderten Komponenten. Sollte eine Komponente nicht verfügbar sein, wird der Aufrufer mithilfe einer entsprechenden Nachricht informiert. Findet das CEP-Framework alle im Operator definierten Komponenten, werden mithilfe des Transport Layers alle Kanäle zwischen den Komponen-

ten initialisiert. Die Transportkanäle entsprechen dabei den im Operatorbaum definierten Kanten.

Listing 8.6 Ausführung eines 'Dynamic Interest Querys' mithilfe des CEP-Frameworks

```
1
2  private void startDIQExecution() {
3
4          // Starte die Ausfuehrung des DIQs mithilfe des CEP-Frameworks
5          CEPFramework.get().getExecutionEngine().registerDIQ(this.diq);
6
7  }
```

Nach der von der Consumer Engine ausgeführten Vernetzung aller im Operatorbaum spezifizierten Komponenten beginnt die Ausführung des CEP-Systems. Die verwendeten Sensoren wurden aufgrund der Erstellung der Kanäle informiert und senden nun Ereignisse an einen Operator. Schlussendlich erreichen die vom Operator berechneten, komplexen Ereignisse den Konsumenten des Systems.

Operator Execution Engine

Das zweite Modul der Execution Engine eines CEP-Frameworks ist die Ausführung eines Operators mit der *Operator Execution Engine*. Die Ausführung eines Operators wird mithilfe der im Listing 8.7 beschriebenen Methode gestartet.

Als Parameter der in Zeile 5 dargestellten Methode 'startOperatorExecution(IOperator operator)' muss der Operator, der das Interface 'IOperator' implementiert, gesetzt werden. Das Framework erstellt eine Operator Engine für den spezifizierten Operator. Die erstellte Instanz der Operator Engine wird bei der aufgerufenen Methode zurückgegeben. Anschließend muss jedem Puffer für eingehende Ereignisströme die Instanz der Operator Engine gesetzt werden.

Listing 8.7 Ausführung eines Operators mithilfe des CEP-Frameworks

```
1
2  private void startOperatorExecution() {
3
4          // Starte die Ausfuehrung des Operators mithilfe des CEP-Frameworks und speichere
               die zurueck gegebene Instanz der Operator-Engine.
5          OperatorEngine engine =
               CEPFramework.get().getExecutionEngine().startOperatorExecution(Operator.this);
6
7          // Setzte jedem Puffer der Eingangsstroeme die Instanz der Operator-Engine.
8          for (IInstreamBuffer buffer : Operator.this.buffers) {
9                  buffer.setOperatorEngine(engine);
10         }
11
12 }
```

Die Operator Engine führt nun selbstständig die Ausführung des Operators durch. Dabei werden eingehende Ereignisse detektiert und Fenster für die Auswahl von Ereignissen verwendet. Die Ausführung der Korrelation, die Bewegung der Auswahlfenster und der Konsum der korrelierten Ereignisse wird gemäß des im Kapitel 7 beschriebenen Ausführungsmodells eines CEP-Operators durchgeführt.

Nachdem eine Korrelation einer oder mehrerer Ereignisse von der Operator Engine erkannt wurde, wird die Methode 'correlate()' des entsprechenden Operators aufgerufen. Folglich wird der vom Entwickler implementierte Java-Code zur Korrelation von Ereignissen ausgeführt und die Verarbeitung der korrelierten Ereignisse gestartet.

8.2 Sensor

Ein Sensor eines CEP-System wird für die Erzeugung von Ereignissen verwendet. Der Entwickler eines Sensors muss dabei für die Ausführung im CEP-System eine Klasse implementieren, die die Schnittstelle 'ISensor' bedient.

Da die Implementierung einiger Methoden für die vielen Sensoren verwendet werden kann, ist die abstrakte Klasse 'AbstractSensor' im CEP-Framework enthalten. Sie implementiert das Interface 'ISensor' und stellt für einige Methoden eine Basis-Implementierung bereit. Leitet der Entwickler von der Klasse 'AbstractSensor' ab, wird von ihm einzig die Implementierung der Methoden 'startSensoring()' und 'stopSensoring()' sowie die Angabe des Subtyps des Sensors als Zeichenkette gefordert.

Initialisiert der Entwickler das CEP-Framework wie in Abschnitt 8.1.1 beschrieben und registriert den implementierten Sensor, übernimmt das CEP-Framework die Ausführung und das Versenden der vom Sensor erzeugten Ereignisse.

8.3 Operator

Für die Implementierung eines Operators muss die Schnittstelle 'IOperator' verwendet werden. Soll der Operator lokal auf einem Gerät ausgeführt werden, muss der Entwickler die vom Interface 'IOperator' abgeleitete Schnittstelle 'ILocalOperator' implementieren. Analog zur Entwicklung eines Sensors steht eine abstrakte Implementierung eines lokalen CEP-Operators im Framework zur Verfügung.

Der Entwickler eines Operators muss, neben der Angabe des Subtyps als Zeichenkette, die Instantiierung von Puffern mithilfe der Klasse 'InstreamBuffer' und von Auswahlfenstern mit der Klasse 'SelectionWindow' durchführen.

Nachdem der Initialisierungsprozess abgeschlossen wurde, kann die im Abschnitt 8.1.4 beschriebene Ausführung eines Operators mithilfe des CEP-Frameworks durchgeführt werden. Das Framework ruft beim Auftreten einer Korrelation die im Interface 'ILocalOperator' definierte Methode 'correlate()' auf. Dabei werden dem Operator die korrelierten Ereignisse übergeben. Nachdem die Verarbeitung der Ereignisse stattgefunden hat, kann ein komplexes, ausgehendes Ereignis generiert und dem Framework überliefert werden. Das Framework des CEP-Systems, insbesondere der Transport Layer, führt die Übertragung des Ereignisses an den entsprechenden Konsumenten durch.

8.4 Konsument

Der in diesem Abschnitt beschriebene Konsument des CEP-Systems muss nach der Initialisierungsphase einen im Abschnitt 8.1.3 beschriebenen DIQ zur Ausführung eines CEP-Systems am CEP-Framework anmelden. Der Entwickler eines Konsumenten muss die Schnittstelle 'IConsumer' verwenden und dabei die Methode 'receivedEvents()' implementieren. Die beschriebene Methode wird vom CEP-Framework aufgerufen, sobald ein eingehendes Ereignis an den Konsumenten gesendet wurde.

Der Entwickler kann schlussendlich die Verarbeitung und Visualisierung des komplexen Ereignisses durchführen.

9 Klassifikation der CEP-Operatoren

Um verschiedene CEP-Operatoren klassifizieren zu können, werden im Abschnitt 9.1 zunächst die spezifischen Merkmale der CEP-Operatoren diskutiert. Anhand der beschriebenen Merkmale wird im anschließenden Abschnitt 9.2 eine Klassifikation vorgestellt. Dieses Kapitel beinhaltet als letzten Abschnitt die These dieser Bachelor-Arbeit. Sie ist im Abschnitt 9.3 formuliert und definiert, in welcher Klasse ein Operator energiesparender in einer entfernten Infrastruktur bzw. lokal auf einem mobilen Endgerät ausgeführt werden kann.

9.1 Merkmale der CEP-Operatoren

CEP-Operatoren können im Allgemeinen sehr unterschiedlich aufgebaut sein und diverse Funktionen bieten. Trotzdem ist innerhalb eines CEP-Systems definiert, dass CEP-Operatoren mindestens einen eingehenden Ereignisstrom verarbeiten können. Weiterhin wird durch Ereigniskorrelation mindestens ein ausgehender Ereignisstrom produziert. Ein Merkmal eines CEP-Operators ist somit die Anzahl der von ihm verarbeiteten, eingehenden Ereignisströme sowie die Anzahl der ausgehenden Ereignisströme.

Aufgrund der unterschiedlichen Funktionalität der CEP-Operatoren kann dessen Korrelation, insbesondere der Selektion von Ereignissen, unterschiedlich komplex sein. Außerdem kann die Berechnung, die der Operator nach der Korrelation mehrerer Ereignisse durchführt, sowohl trivial als auch hoch komplex sein. Die Berechnungskomplexität und damit die CPU-Last kann stark zwischen zwei verschiedenen Operatoren differieren.

Ein weiteres Merkmal ist die Frequenz der eingehenden Ereignisse. Abhängig von der Spezifikation des Operators kann eine Relation zwischen der Frequenz der eingehenden Ereignisse und der Ereigniskorrelation, und damit auch den ausgehenden Ereignissen bestehen. In diesem Fall wäre auch die Häufigkeit der Berechnung eines ausgehenden Ereignisses nach einer Ereigniskorrelation abhängig von der Frequenz der eingehenden Ereignisse.

Zusammenfassend kann festgehalten werden, dass folgende drei Merkmale einen CEP-Operator charakterisieren:

- Anzahl der eingehenden Ereignisströme
- Berechnungskomplexität nach der Ereigniskorrelation
- Frequenz der eingehenden Ereignisse

Bei der Charakterisierung der CEP-Operatoren lassen sich weitere Merkmale nennen, wobei die hier genannten aufgrund der praktischen Erfahrung mit CEP-Systemen durch Variation die größte Wirkung hervorrufen. Zusätzliche Merkmale sind beispielsweise der

Speicherverbrauch, insbesondere der Arbeitsspeicherverbrauch, als auch die Möglichkeit der parallelen Ausführung der Berechnung des CEP-Operators mithilfe mehrerer Server bzw. mobiler Endgeräte.

9.2 Klassifikation

Anhand der im Abschnitt 9.1 beschriebenen Merkmale eines CEP-Operators und der Erfahrung, die durch die praktische Arbeit mit dem mobilen CEP-System erlangt werden konnte, kann nun eine Klassifikation der Operatoren formuliert werden.

Ein Operator kann aufgrund der im Abschnitt 9.1 beschriebenen Merkmale charakterisiert werden.

Werden über die eingehenden Ereignisströme hochfrequent Ereignisse an den Operator gesendet, muss im Fall einer serverseitigen Ausführung des Operators eine große Anzahl an Datenpaketen versendet werden. Im Gegensatz dazu können die häufig durchzuführenden Berechnungen direkt mithilfe der CPU des mobilen Endgeräts berechnet werden, sofern die Ausführung des Operators lokal auf dem Gerät durchgeführt wird.

Wird der Fokus auf die Anzahl der eingehenden Ereignisströme und die Berechnungskomplexität nach der Ereigniskorrelation gerichtet, so muss die Rechenleistung der CPU und der dadurch entstehende Energiekonsum näher betrachtet werden. Führt ein entfernter Server die Berechnung nach der Ereigniskorrelation durch, wird die CPU des Smartphones nicht verwendet. Steigt im Fall der serverseitigen Berechnung die Berechnungskomplexität an, ändert sich der Energieverbrauch des CEP-Systems nicht.

Im Vergleich hierzu steigt der Energiekonsum des CEP-Systems aufgrund der Belastung der CPU signifikant an, falls eine große Berechnung auf dem Smartphone durchgeführt wird.

Aufgrund der formulierten Zusammenhänge zwischen der Frequenz der Ereignisverarbeitung, der Berechnungskomplexität und der Anzahl der eingehenden Ereignisströme lässt sich folgende Klassifikation vornehmen:

Operator der Klasse I

Ein CEP-Operator wird Klasse I zugeordnet, falls er eine geringe Anzahl an Ereignisströmen, eine sehr geringe Berechnungskomplexität, jedoch eine hochfrequente Ereignisverarbeitung aufweist. Der Operator kann beispielsweise drei eingehende Ereignisströme und eine Berechnungskomplexität von 1.000 Gleitkommaoperationen je Ereigniskorrelation besitzen als auch mit einer Frequenz von 90 Ereignisverarbeitungen pro Minute arbeiten.

Operator der Klasse II

Steigt die Zahl der eingehenden Ereignisströme sowie die Berechnungskomplexität bei niedriger Frequenz stark an, wird ein CEP-Operator der Klasse II zugeordnet. Ein Operator kann dabei beispielsweise 50 eingehende Ereignisströme besitzen. Die Berechnung pro

Korrelation kann zum Beispiel 1.000.000 Gleitkommaoperationen beinhalten, wobei eine Korrelation fünf mal je Minute durchgeführt wird.

Operator der Klasse III

Besitzt ein Operator sowohl eine geringe bis mittlere Anzahl an Ereignisströmen, eine geringe bis durchschnittlich hohe Berechnungskomplexität als auch eine mittel- bis hochfrequente Ereignisverarbeitung, so wird er Klasse III zugeordnet. Beispielsweise besitzt ein Operator der Klasse III zehn eingehende Ereignisströme, die durchzuführende Berechnung nach der Ereigniskorrelation beinhaltet 100.000 Gleitkommaoperationen und wird mit einer Frequenz von 12 Berechnungen pro Minute ausgeführt. Außerdem wird ein Operator dieser Klasse zugeordnet, falls alle genannten Merkmale sehr geringe, bzw. sehr hohe Werte aufweisen.

9.3 These

In diesem Abschnitt wird die These der Bachelor-Arbeit formuliert, wobei für die Klassen der Operatoren definiert wird, ob die Berechnung einer Ereigniskorrelation in einer Infrastruktur oder bei der lokalen Ausführung energieeffizienter ist. Die These wird im Kapitel 10 anhand von Experimenten überprüft. Schlussendlich wird im Abschnitt 10.3 das Ergebnis präsentiert.

These *Ein Operator eines mobilen CEP-Systems, der nach der Klassifikation aus Abschnitt 9.2 der Klasse I zugeordnet wird, kann energiesparender auf dem lokalen Gerät als in einer entfernten Infrastruktur ausgeführt werden. Wird dem Operator die Klasse II zugeordnet, ist es für das Smartphone im Sinne der Energiebetrachtung besser, den Operator auf einer entfernten Infrastruktur auszuführen. Sollte ein Operator der Klasse III angehören, kann keine allgemein gültige Aussage getroffen werden, da bei minimaler Veränderung der Konfiguration der Operator entweder Klasse I oder Klasse II zugeordnet werden muss und entsprechend energieeffizienter lokal auf einem Smartphone bzw. in einer entfernten Infrastruktur ausgeführt werden kann.*

Erläuterung Die Berechnung der Ereigniskorrelation und Verarbeitung eines CEP-Operators auf einem mobilen Endgerät muss auf einer meist schwächeren CPU als der eines Servers durchgeführt werden. Dabei steigt der Energiekonsum stark an, wenn die Berechnungskomplexität zunimmt. Folglich wird der Akku eines Smartphones stark belastet, falls intensive Berechnungen nötig sind und die CPU ausgelastet wird. Der Vorteil einer lokalen Ausführung besteht aus kurzen Transportwegen. Ereignisse müssen nicht über ein WLAN- oder Datennetz an einen entfernten Server versendet werden, sondern können direkt auf der CPU des Smartphones berechnet werden. Erfordert die Ausführung eines CEP-Operators eine hochfrequente Berechnung, benötigt das Übertragen der Ereignisse viel Energie.

Aufgrund dieser Zusammenhänge benötigt eine hochfrequente CEP-Ausführung mit niedriger Berechnungskomplexität weniger Energie, wenn sie auf dem lokalen Smartphone ausgeführt wird. Besitzt die Ausführung des CEP-Systems hingegen eine hohe Berechnungskomplexität, wird jedoch selten ausgeführt, benötigt die entfernte Berechnung weniger Energie.

10 Evaluation

Um die in Kapitel 9 definierte Klassifikation der CEP-Operatoren verifizieren zu können, wurden einige Experimente durchgeführt.

Dabei werden im Abschnitt 10.1 zunächst der Versuchsaufbau sowie die Eigenschaften der verwendeten Smartphones erläutert. Im Abschnitt 10.2 wird sowohl die Durchführung der Experimente als auch deren Auswertung beschrieben. Schlussendlich wird im Abschnitt 10.3 das Ergebnis präsentiert.

10.1 Versuchsaufbau

In diesem Abschnitt werden die Vorbereitungen und der Versuchsaufbau beschrieben. Dabei werden im Abschnitt 10.1.1 die zur Versuchsdurchführung verwendeten Smartphones näher beschrieben sowie deren Systemzustand erläutert. Weiterhin wird im Abschnitt 10.1.2 die Konfiguration des mobilen CEP-Systems formuliert. Hierbei wird sowohl auf den Operatorbaum und die versendeten Ereignisse als auch auf die Ausführung des Operators eingegangen.

10.1.1 Verwendung mobiler Endgeräte

Zur Ausführung eines mobilen CEP-Systems wurden zwei 'Samsung Galaxy Nexus I9250' Smartphones verwendet. Dabei wurde das zur Zeit der Ausführung aktuellste Android-Betriebssystem der Version 4.1.1 mit dem Namen 'Jelly Bean' verwendet. Beide Smartphones wurden auf den Werkszustand gesetzt und Anwendungen wie E-Mail-Clients, Kalender-Programme oder ähnliche deaktiviert. Außerdem waren die Smartphones während der Ausführung nicht mit einem Mobilfunknetzwerk verbunden, sondern wurden im Flugmodus betrieben. Alle Anwendungen, die nicht zur Versuchsdurchführung dienten, wurden beendet. Somit konnte sichergestellt werden, dass die Einflüsse durch andere Anwendungen auf den Akku des Smartphones so gering wie möglich waren.

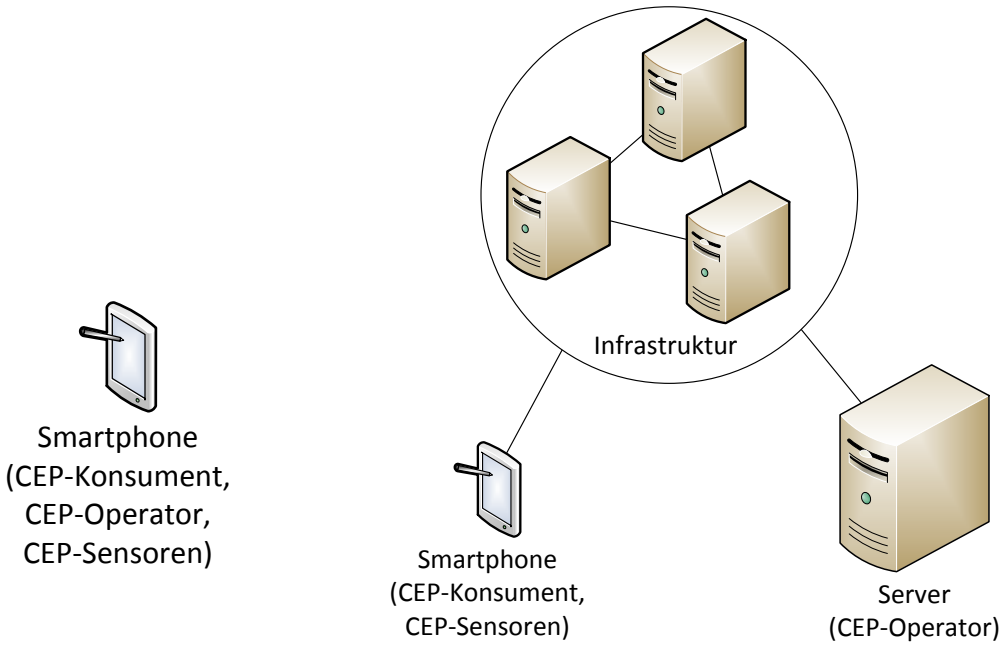
Vor jedem Experiment wurde das Smartphone vollständig aufgeladen. Die Dauer der CEP-Ausführung entspricht der Zeit, die vom vollständig geladenen bis zum vollständig entladenen Akku vergeht und wird in den folgenden Kapiteln als 'Laufzeit' tituliert. Während der Versuchsdurchführung wurde das Display auf voller Helligkeit mit weißem Hintergrund betrieben.

10.1.2 Konfiguration des mobilen CEP-Systems

In diesem Abschnitt wird die Konfiguration des mobilen CEP-Systems detailliert erläutert. Zunächst wird die Anordnung der verwendeten Geräte und die Eigenschaften des CEP-Systems beschrieben. Weiterführend wird der verwendete Operatorbaum sowie die Funktionsweise des CEP-Operators vorgestellt.

Anordnung der Geräte

Um den Energieverbrauch eines CEP-Systems mit lokaler bzw. entfernter Operator-Ausführung vergleichen zu können, wurden zwei Szenarien zur Anordnung der Geräte verwendet. Abbildung 10.1 stellt beide Szenarien dar.



(a) Szenario A (b) Szenario B

Abbildung 10.1: Konfigurationen des mobilen CEP-Systems

Szenario A, dargestellt in Abbildung 10.1(a), zeigt die lokale Ausführung eines CEP-Konsumenten, eines CEP-Operators sowie mehrerer CEP-Sensoren. Dabei werden alle CEP-Komponenten auf dem Smartphone ausgeführt, sodass keine Netzwerk-Verbindung zu einer Infrastruktur oder anderen Servern notwendig ist.

Das in Abbildung 10.1(b) beschriebene Szenario B nutzt die Möglichkeit zur entfernten Ausführung des CEP-Operators. Der CEP-Konsument und alle CEP-Sensoren werden weiterhin auf dem lokalen Smartphone ausgeführt, wobei die Berechnung des CEP-Operators auf einem entfernten Server durchgeführt wird. CEP-Sensoren senden die erzeugten Ereignisse zu einer Infrastruktur, die die Ereignisse zum Server weiterleitet. Nach der Ereigniskorrelation und Verarbeitung durch den Server werden die erzeugten, komplexen Ereignisse

erneut mithilfe der Infrastruktur an das Smartphone versendet. Schlussendlich kann der CEP-Konsument des Smartphones die Ergebnisse anzeigen.

Funktion des CEP-Systems

Das für die Evaluation verwendete und in diesem Abschnitt beschriebene CEP-System führt die mathematische Operation der Matrixmultiplikation durch. Dabei erzeugt ein Sensor des CEP-Systems eine Matrix bestehend aus Integer-Zahlen und sendet diese an den CEP-Operator. Ein Sensor wird mit dem Parameter n gestartet, wodurch die Dimension $n \times n$ der erzeugten Matrix definiert wird.

Zunächst wird im folgenden Abschnitt die logische Anordnung der CEP-Komponenten mithilfe eines Operatorbaums erläutert. Darauf folgend wird die Funktionsweise des CEP-Operators verdeutlicht.

Operatorbaum

Für die im Abschnitt 10.2 beschriebene Versuchsdurchführung und Auswertung wird ein Konsument, ein Operator sowie eine mithilfe des Parameters m definierte Anzahl an Sensoren verwendet. Jeder Sensor versendet die erzeugten Ereignisse anhand eines Ereignisstroms an den CEP-Operator. Nachdem der CEP-Operator die Ereigniskorrelation und Verarbeitung durchgeführt hat, versendet er das komplexe Ereignis an den Konsumenten.

Der hier formulierte Ereignisfluss sowie die logische Anordnung der CEP-Komponenten veranschaulicht der in Abbildung 10.2 dargestellte Operatorbaum.

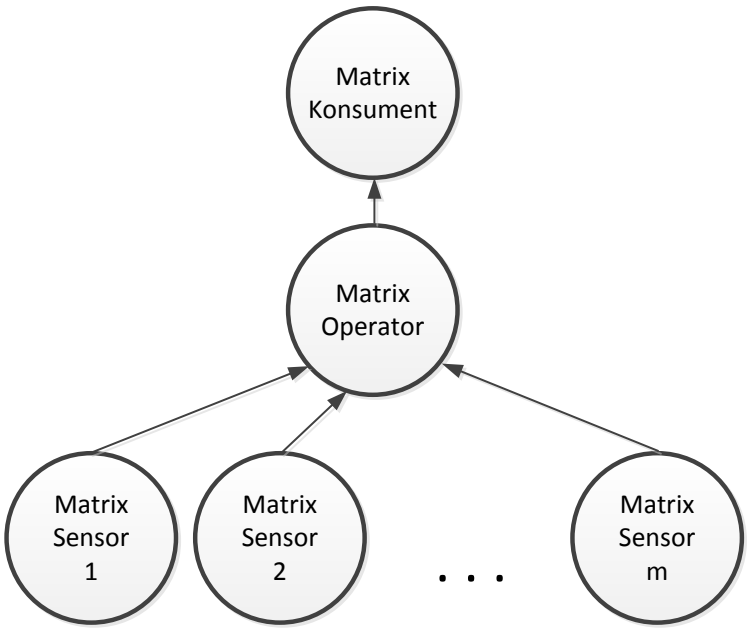


Abbildung 10.2: Operatorbaum für die Versuchsdurchführung

Die Aufgabe des CEP-Operators besteht aus der Multiplikation mehrerer Matrizen. Der CEP-Operator wird mithilfe des Parameters m gestartet, der die Anzahl der eingehenden Ereignisströme festlegt. Wird der Operator beispielsweise mit $m = 3$ gestartet, werden drei Puffer erzeugt, die eingehende Ereignisse empfangen können. Ist mindestens ein Ereignis in jedem Puffer vorhanden, beginnt der Operator mit der Ereigniskorrelation. Hierbei wird das jeweils zeitlich älteste Ereignis eines Puffers verwendet, die im Ereignis enthaltene Matrix extrahiert und mit den aus den anderen Puffern entnommenen Matrizen multipliziert. Nachdem die Matrixmultiplikation abgeschlossen wurde, löscht der Operator die verwendeten Ereignisse auf den Puffern und erzeugt ein neues Ereignis, setzt die Werte der errechneten Matrix und versendet das Ereignis an den CEP-Konsumenten.

Zur Übertragung der Matrizen werden Zeichenketten verwendet. Zunächst wird die Dimension der Matrix abgebildet, gefolgt vom Sonderzeichen $ und der Matrix, deren Werte durch Kommata getrennt sind.

10.2 Durchführung und Auswertung

Zunächst wird im Abschnitt 10.2.1 die Durchführung der Versuche beschrieben. Im darauf folgenden Abschnitt 10.2.2 werden die Resultate der durchgeführten Versuche abgebildet und diskutiert.

10.2.1 Durchführung

In diesem Abschnitt werden die Versuchsdurchführung sowie die Merkmale der einzelnen Versuche erläutert.

Um der großen Varianz eines CEP-Operators sowohl im Hinblick auf die eingehenden Ereignisströme, die Berechnungskomplexität während der Ereignisverarbeitung als auch der Frequenz der Ereignisverarbeitung gerecht zu werden, wurden im Kapitel 9 die Merkmale eines CEP-Operators beschrieben und nachfolgend eine Klassifikation der Operatoren in drei Klassen vorgenommen.

In den nachfolgend beschriebenen Versuchen wurden die drei identifizierten Merkmale eines Operators variiert. Dabei werden die Merkmale wie folgt abgekürzt:

EES	*bezeichnet*	Anzahl der eingehenden Ereignisströme
GKOP	*bezeichnet*	Berechnungskomplexität der Ereignisverarbeitung in Gleitkommaoperationen
FREQ	*bezeichnet*	Frequenz der eingehenden Ereignisse pro Minute

Die Berechnungskomplexität der Ereignisverarbeitung wird in Gleitkommaoperationen angegeben und berechnet sich bei der Versuchsdurchführung wie in Formel 10.1 beschrieben. Dabei bezeichnet die Variable n die Dimension der Matrix, während m die Anzahl der eingehenden Ereignisströme abbildet.

$$GKOP(n, m) = (m - 1) \cdot (2n^3 - n^2) \tag{10.1}$$

Es wurden insgesamt 24 Versuche durchgeführt, wobei 12 Versuche mit lokaler und 12 Versuche mit entfernter Ausführung des CEP-Operators konfiguriert wurden. Dabei wurden die eingehenden Ereignisströme von 2 bis 100, die Gleitkommaoperationen der Ereignisverarbeitung von 12 bis 4,7 Millionen und die Frequenz von 2 bis 180 Ausführungen pro Minute variiert.

Eine Übersicht der Konfigurationen der Versuchsdurchführungen findet sich in Tabelle 10.2.1. Dabei wurden viele unterschiedliche Konfigurationen gewählt, sodass ein großes Spektrum der Konfigurationsmöglichkeiten abgedeckt werden konnte.

Versuch	Ausführung des Operators		EES	GKOP	FREQ
	lokal	entfernt			
#01	X		2	12	120
#02		X	2	12	120
#03	X		2	12	180
#04		X	2	12	180
#05	X		5	900	60
#06		X	5	900	60
#07	X		50	764400	15
#08		X	50	764400	15
#09	X		30	1539900	12
#10		X	30	1539900	12
#11	X		30	1539900	30
#12		X	30	1539900	30
#13	X		30	1539900	45
#14		X	30	1539900	45
#15	X		30	1539900	60
#16		X	30	1539900	60
#17	X		30	1539900	120
#18		X	30	1539900	120
#19	X		100	1544400	30
#20		X	100	1544400	30
#21	X		60	3132900	2
#22		X	60	3132900	2
#23	X		20	4702500	15
#24		X	20	4702500	15

Tabelle 10.1: Konfigurationen der Versuchsdurchführungen

Während der Versuchsdurchführung wurde der Zustand des Akkus des Smartphones gemessen und in einer Datei festgehalten. Eine für diese Messung implementierte Anwendung wurde vom Betriebssystem informiert, sobald sich der Zustand des Akkus um einen Prozentpunkt änderte. Der Zeitpunkt der Änderung sowie der aktuelle Prozentsatz der Ladung des Akkus wurden aufgezeichnet.

Gemessene Laufzeit ohne Ausführung des CEP-Systems

In Tabelle 10.2 sind dabei die Laufzeiten der beiden Smartphones sowie der Durchschnitt dargestellt, wobei das CEP-System nicht ausgeführt wurde. In der ersten Zeile findet sich die Messreihe mit deaktivierter Verbindung zum WLAN-Netz, in der zweiten Zeile mit aktivierter Verbindung.

	Laufzeit in Minuten		
WLAN-Verbindung	Gerät A	Gerät B	Durchschnitt
deaktiviert	327,84	326,54	327,19
aktiviert	323,68	320,16	321,92

Tabelle 10.2: Gemessene Laufzeit der Geräte ohne Ausführung des CEP-Systems

Deutlich zu erkennen ist der geringe Unterschied der Laufzeiten der Geräte zwischen aktivierter und deaktivierter WLAN-Verbindung. Der Energieverbrauch der aktivierten WLAN-Verbindung ist sehr gering, da keine Daten übertragen wurden.

10.2.2 Auswertung

In diesem Abschnitt werden die erzielten Messergebnisse der Versuchsdurchführungen präsentiert.

Tabelle 10.3 zeigt die Versuchsdurchführungen mit den gemessenen Laufzeiten der beiden Smartphones sowie deren Durchschnitt. Jeder Versuch wurde dabei einmal durchgeführt, da in der realisierten Evaluation der Fokus nicht auf eine häufige Wiederholung eines Versuchs, sondern auf das Erreichen eines möglichst großen Konfigurationsspektrums gerichtet wurde.

Auf Grundlage der gemessenen Laufzeiten werden in den folgenden Abschnitten signifikante Messreihen präsentiert. Zunächst wird im folgenden Abschnitt die Anzahl der eingehenden Ereignisströme variiert. Im darauffolgenden Abschnitt wird die Frequenz der Ereigniskorrelation verändert, wobei die durchgeführten Gleitkommaoperationen und die Anzahl der eingehenden Ereignisströme konstant bleibt.

Versuch	Laufzeit in Minuten		
	Gerät A	Gerät B	Durchschnitt
#01	165,55	184,00	174,77
#02	160,76	169,24	165,00
#03	166,32	164,39	165,36
#04	166,40	164,80	165,60
#05	172,20	168,66	170,43
#06	184,54	169,90	177,22
#07	228,11	231,48	229,79
#08	251,26	246,12	248,69
#09	255,38	263,50	259,44
#10	282,97	287,95	285,46
#11	177,64	176,10	176,87
#12	225,09	202,82	213,96
#13	161,70	167,23	164,47
#14	185,90	180,42	183,16
#15	168,84	168,01	168,42
#16	175,30	169,32	172,31
#17	152,73	160,97	156,85
#18	164,31	156,56	160,43
#19	168,55	175,21	171,88
#20	214,95	204,07	209,51
#21	319,42	319,82	319,62
#22	315,52	311,04	313,28
#23	221,19	219,78	220,48
#24	266,90	275,06	270,98

Tabelle 10.3: Messergebnisse der Versuchsdurchführungen

Variation der Anzahl der eingehenden Ereignisströme

In diesem Abschnitt werden Messergebnisse einiger Versuche veranschaulicht, bei denen die Anzahl der Gleitkommaoperationen sowie die Frequenz der Ausführung konstant gehalten, die Anzahl der eingehenden Ereignisströme variiert wurde.

Bei den Versuchen #11, #12 und #19, #20 wurden rund 1,54 Millionen Gleitkommaoperationen mit einer Frequenz von 30 mal pro Minute durchgeführt. Die Anzahl der eingehenden Ereignisströme wurde dabei variiert. Bei Versuch #11 und #12 wurden 30 eingehende Ereignisströme verwendet, bei Versuch #19 und #20 hingegen 100.

Abbildung 10.3 veranschaulicht den Zustand des Akkus über die Zeit in Minuten. Die kräftigen Farben kennzeichnen die lokalen Berechnungen des Operators, die blassen Farben hingegen die Ausführung des Operators auf einem entfernten Server.

Sowohl der Unterschied der Laufzeit bei den Versuchen #11 und #19 mit lokaler Ausführung des Operators als auch bei den Versuchen #12 und #20 mit entfernter Ausführung ist gering. Folglich hat die Anzahl der eingehenden Ereignisströme nur einen geringen Einfluss auf die Laufzeit des Akkus eines Smartphones.

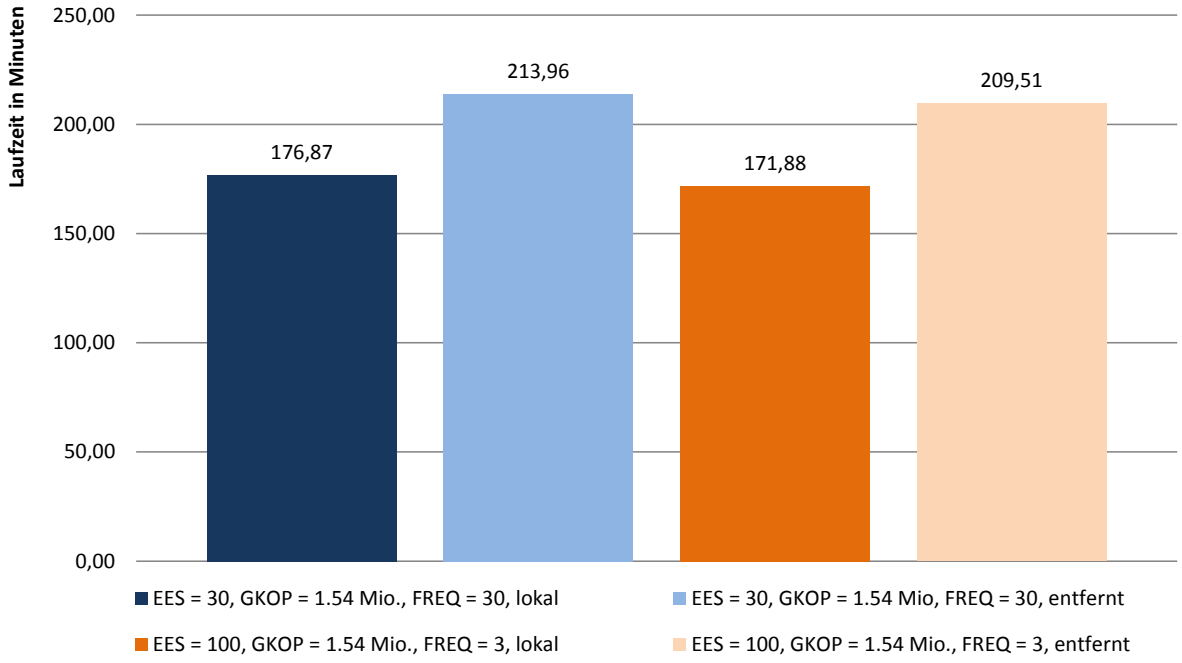


Abbildung 10.3: Laufzeit lokaler und entfernter Ausführungen bei variierender Anzahl eingehender Ereignisströme

Variation der Frequenz der Ereigniskorrelation

Bei den in diesem Abschnitt erläuterten Versuchen blieb die Anzahl der durchgeführten Gleitkommaoperationen sowie die Anzahl der eingehenden Ereignisströme konstant. Die Versuche #9 bis #18 wurden mit 30 eingehenden Ereignisströmen und 1,54 Millionen Gleitkommaoperationen durchgeführt. Dabei wurde die Frequenz der Ereigniskorrelation wie folgt variiert: Versuch #9 und #10 mit 12 Ausführungen pro Minute, #11 und #12 mit 30, #13 und #14 mit 45 ,#15 und #16 mit 60 und #17 und #18 mit 120 Ausführungen pro Minute. Die gemessenen Laufzeiten sind in Abbildung 10.4 graphisch dargestellt.

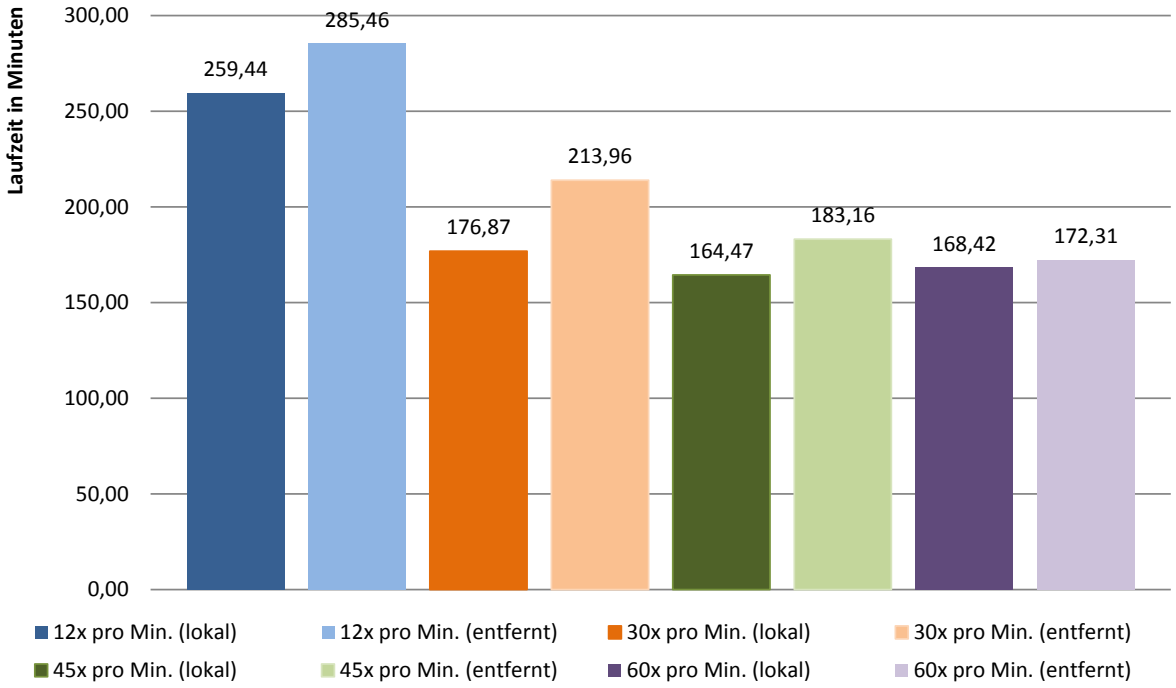


Abbildung 10.4: Laufzeit lokaler und entfernter Ausführungen bei variierender Frequenz

Die Laufzeit der entfernten Ausführung des Operators war bei allen hier dargestellten Messreihen größer als die der lokalen Ausführung. Allerdings ist der Unterschied der lokalen Ausführung zur entfernten Ausführung bei einer Frequenz von 30 Ausführungen pro Minute deutlich höher als der Unterschied bei einer Frequenz von 60 Ausführungen pro Minute.

In Abbildung 10.5 sind die Laufzeitunterschiede zwischen der entfernten und lokalen Ausführung dargestellt. Dabei wurde die gemessene Laufzeit der lokalen Ausführung von der Laufzeit der entfernten Ausführung subtrahiert.

Hiermit konnte gezeigt werden, dass bei zunehmender Frequenz der Ereigniskorrelation ab 30 Ausführungen pro Minute der Unterschied des Energiebedarfs zwischen der lokalen und entfernten Ausführung abnimmt. Die Laufzeit des Akkus war mit der entfernten Ausführung bei einer Frequenz von 30 Ausführungen pro Minute im Gegensatz zur lokalen Ausführung circa 37 Minuten länger. Bei einer Frequenz von 60 Ausführungen pro Minute sank der Vorteil der entfernten Ausführung hingegen auf rund 4 Minuten.

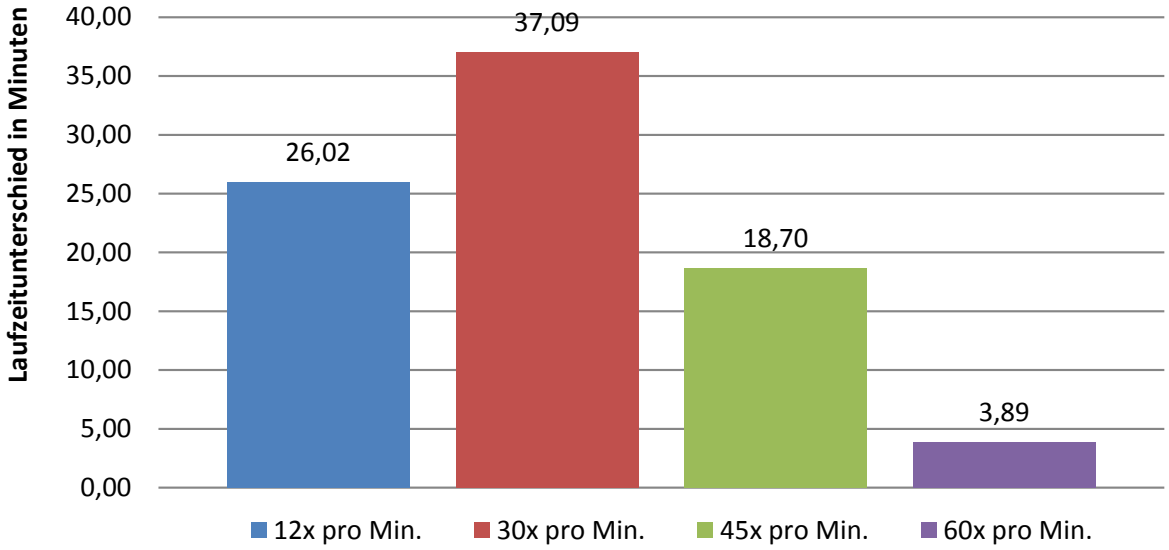


Abbildung 10.5: Laufzeitunterschied zwischen lokaler und entfernter Ausführungen bei variierender Frequenz

Unterschied der Laufzeit bei lokaler und entfernter Ausführung

In diesem Abschnitt werden sechs Messreihen präsentiert, die die im Kapitel 9 beschriebene Klassifikation der Operatoren verdeutlicht.

Die Ereigniskorrelation der Versuche #1 und #2 wurden mit einer hohen Frequenz von 120 Ausführungen pro Minute, jedoch mit einer geringen Anzahl Gleitkommaoperationen und zwei eingehenden Ereignisströmen durchgeführt. Die Messergebnisse der Versuche #15 und #16 wurden mithilfe einer CEP-Ausführung erzielt, die mit 1,54 Millionen Gleitkommaoperationen, 30 eingehenden Ereignisströmen und einer Frequenz von 60 Ausführungen pro Minute konfiguriert wurde. Mit einer niedrigen Frequenz von 15 Ausführungen pro Minute, 20 eingehenden Ereignisströmen und 4,70 Millionen Gleitkommaoperationen je Ereigniskorrelation wurde das CEP-System der Versuche #23 und #24 ausgeführt.

Die Laufzeiten der in diesem Abschnitt beschriebenen Versuche sind in Abbildung 10.6 visualisiert.

Bei der Konfiguration EES = 2, GKOP = 12 und FREQ = 120 der Versuche #1 und #2 ist zu erkennen, dass die Laufzeit der lokalen Ausführung die der entfernten Ausführung übersteigt. Der Operator der Versuche wird dabei der Klasse I zugeordnet. Bei den Versuchen #15 und #16 mit der Konfiguration EES = 30 , GKOP = 1.54 Mio. und FREQ = 60 sowie bei den Versuchen #23 und #24 mit der Konfiguration EES = 20, GKOP = 4.70 Mio. und FREQ = 15 war hingegen die entfernte Ausführung des CEP-Operators energieeffizienter. Der bei den Versuchen #15 und #16 verwendete Operator wird Klasse III, der bei den Versuchen #23 und #24 ausgeführte Operator Klasse II zugeordnet.

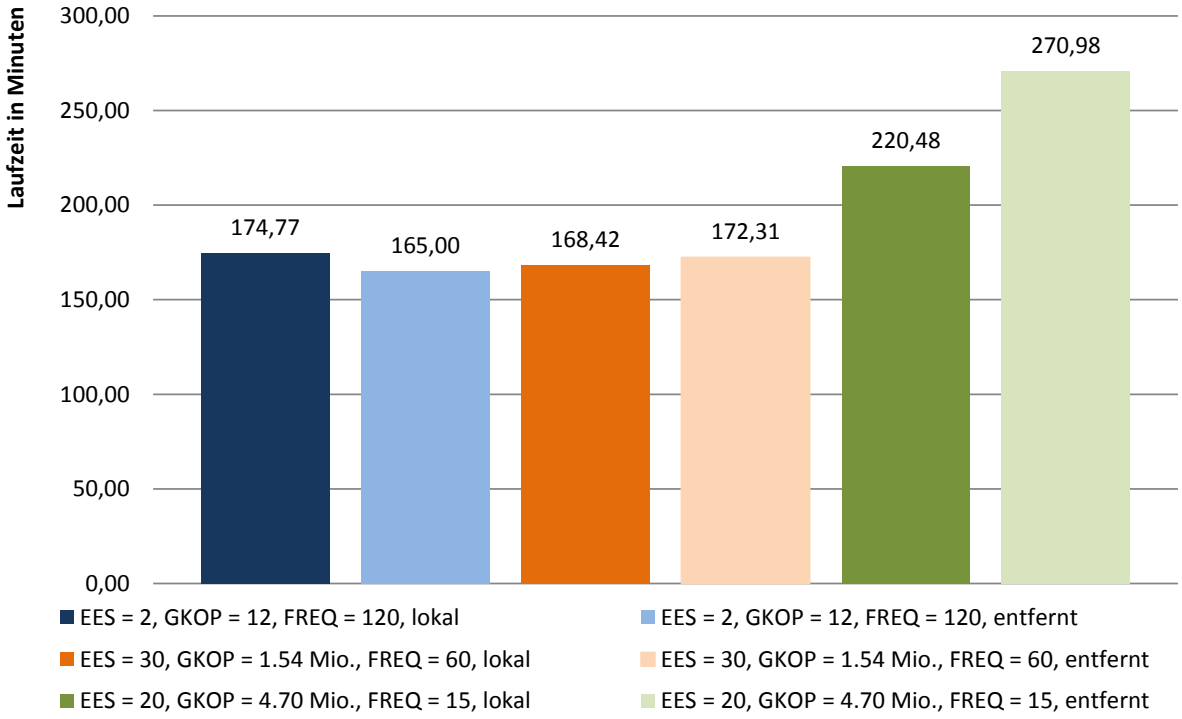


Abbildung 10.6: Laufzeit lokaler und entfernter Ausführungen der Versuche #1, #2, #15, #16, #23, #24

10.3 Ergebnis

Zusammenfassend lässt sich feststellen, dass Ereigniskorrelationen eines CEP-Operators mit geringer Berechnungskomplexität, einer geringen Anzahl eingehender Ereignisströme, jedoch einer hohen Ausführungsfrequenz auf einem Smartphone energieeffizienter als in einer entfernten Infrastruktur ausgeführt werden können. Steigt die Berechnungskomplexität und die Anzahl der eingehenden Ereignisströme bei abnehmender Frequenz, ist der Energieverbrauch des CEP-Systems bei einer entfernten, auf einer Infrastruktur berechneten Ausführung geringer als bei der lokalen Ausführung. Die im Kapitel 9 vorgestellte Klassifikation kann daher mit der hier vorgestellten Versuchsdurchführung verifiziert werden.

Weiterhin kann bemerkt werden, dass eine hohe Frequenz der Ereigniskorrelation maßgeblich zum Energiebedarf des CEP-Systems beiträgt. Wird eine Ereigniskorrelation häufiger als 30 mal pro Minute ausgeführt, steigt der Energiebedarf signifikant an. Im Gegensatz hierzu ist der Unterschied des Energiebedarfs zwischen einer hohen und niedrigen Berechnungskomplexität einer Ereigniskorrelation sehr gering.

11 Zusammenfassung und Ausblick

In dieser Arbeit wurde der Fokus auf den Energieverbrauch von CEP-Operatoren bei der Ausführung eines CEP-Systems mithilfe von mobilen Endgeräten gerichtet. Dabei wurden CEP-Operatoren anhand der Anzahl der eingehenden Ereignisströme, der Berechnungskomplexität sowie der Frequenz der Ereigniskorrelation klassifiziert. Daraufhin wurden mithilfe einiger Versuchsdurchführungen sowohl die definierte Klassifikation verifiziert als auch gezeigt, dass die Frequenz der Ereigniskorrelation den Energieverbrauch des CEP-Systems maßgeblich beeinflusst.

Zunächst wurde in dieser Bachelor-Arbeit eine Architektur für mobile sowie in einer Infrastruktur ausführbare CEP-Systeme vorgestellt. Auf dieser Grundlage konnte ein Framework implementiert werden, mit dessen Hilfe die komplexe Ereignisverarbeitung automatisch durchgeführt wird und dem Anwendungsprogrammierer ausgewählte Schnittstellen zur Verfügung stellt. Die Besonderheit der Implementierung ist die Modularität und Generizität der Transportschicht.

Ein CEP-Operator kann im Allgemeinen sowohl in einer Infrastruktur als auch lokal auf einem Smartphone berechnet werden. Bei der lokalen Berechnung entsteht aufgrund der hohen CPU-Last während der Ausführung ein hoher Energiebedarf, wohingegen bei der entfernten, mithilfe einer Infrastruktur durchgeführten Berechnung das Netzwerk beansprucht und folglich ebenfalls der Akku stark belastet wird.
Anhand der herausgearbeiteten Merkmale eines CEP-Operators, wie die Anzahl der eingehenden Ereignisströme, die Berechnungskomplexität der Ereigniskorrelation sowie dessen Frequenz, konnte eine Klassifizierung vorgenommen werden. Besitzt ein Operator demnach eine große Anzahl eingehender Ereignisströme, eine hohe Berechnungskomplexität, jedoch eine geringe Frequenz der Ereigniskorrelation, kann der Operator energieeffizienter in einer entfernten Infrastruktur als auf einem Smartphone ausgeführt werden. Im Gegensatz dazu wird die Ausführung auf einem lokalen Endgerät empfohlen, falls die Berechnungskomplexität des CEP-Operators sowie die Anzahl der eingehenden Ereignisströme gering, die Frequenz der Ereigniskorrelation hingegen sehr hoch ist.

Mithilfe einiger Messreihen wurden verschiedene Konfigurationen der CEP-Operatoren untersucht und die Laufzeit des Smartphones gemessen. Als Folge konnte die vermutete Klassifikation verifiziert sowie die Relevanz der Frequenz der Ereigniskorrelation verdeutlicht werden. Wird ein Operator häufiger als 30 mal pro Minute ausgeführt, steigt der Energiebedarf des CEP-Systems signifikant an, da sich das Gerät dauerhaft im aktiven Zustand befindet und die CPU kontinuierlich verwendet wird.

Mithilfe der durchgeführten Bachelor-Arbeit konnten spezifische Merkmale der CEP-Operatoren identifiziert sowie eine Klassifikation definiert und evaluiert werden. Dabei wurden die Versuche mit 'Samsung Galaxy Nexus I9250' Smartphones durchgeführt.

Um die vorgestellte Klassifikation weiter zu verfeinern, könnten ergänzend zu dieser Arbeit weitere Versuche durchgeführt werden, wobei Smartphones anderer Hersteller und Modelle verwendet werden, die sich im Hinblick auf die CPU-Leistung, Kapazität des Akkus sowie des Hauptspeichers von den verwendeten Smartphones unterscheiden.

Ein weiterer Aspekt der mobilen Ereigniskorrelation ist die verteilte Berechnung. Wird beispielsweise ein CEP-Operator sowohl lokal auf einem mobilen Endgerät als auch in einer Infrastruktur ausgeführt, besteht die Möglichkeit zur Nutzung beider Operatoren. Hierbei könnte die CPU-Last verteilt werden, indem nur jede zweite Ereigniskorrelation mithilfe einer Infrastruktur berechnet wird. Folglich wird die CPU-Last beim mobilen Endgerät im Vergleich zur rein lokalen Ausführung halbiert, während das Netzwerk nur teilweise beansprucht wird.

Dabei muss das in dieser Bachelor-Arbeit entwickelte, mobil ausführbare CEP-System erweitert werden, sodass eingehende Ereignisströme der Operatoren geteilt und nach der Ereigniskorrelation zusammengeführt werden können.

Schlussendlich könnte mithilfe der in dieser Arbeit vorgestellten und gegebenenfalls weiter verfeinerten Klassifikation der CEP-Operatoren das bestehende Framework erweitert werden, sodass der CEP-Operator anhand seiner Konfiguration selbst entscheiden kann, ob sich dessen Ausführung in einer entfernten Infrastruktur oder lokal auf dem mobilen Endgerät energieeffizienter darstellt. Hierfür müsste der CEP-Operator in einer entsprechenden Tabelle einen Look-up durchführen, als Schlüssel die Konfiguration angeben und schließlich als Ergebnis des Look-ups die Entscheidung erhalten.

Abkürzungsverzeichnis

Ah	Amperestunde
App	Application
CEP	Complex Event Processing
CPU	Central Processing Unit
CQL	Continuous Query Language
DIQ	Dynamic Interest Query
EDA	Event-Driven Architecture
FMC	Fundamental Modeling Concepts
GPS	Global Positioning System
mA	Milliampere
mAh	Milliamperestunde
MFO	Monitored Focal Object
ms	Millisekunde
NK	Nennkapazität
RFID	Radio-Frequency Identification
SDK	Software Development Kit
SOA	Service Oriented Architecture
SQL	Structured Query Language
TCP	Transmission Control Protocol
TESLA	Trio-based Event Specification Language
V	Volt
VM	Virtual Machine
Wh	Wattstunde
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Literaturverzeichnis

[ABW03a] A. Arasu, S. Babu, J. Widom. CQL: A Language for Continuous Queries over Streams and Relations. In G. Lausen, D. Suciu, Herausgeber, *DBPL*, Band 2921 von *Lecture Notes in Computer Science*, S. 1–19. Springer, 2003. URL http://dblp.uni-trier.de/db/conf/dbpl/dbpl2003.html. (Zitiert auf den Seiten 9 und 18)

[ABW03b] A. Arasu, S. Babu, J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical Report 2003-67, Stanford InfoLab, 2003. URL http://ilpubs.stanford.edu:8090/758/. An earlier version this technical report, titled 'An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations', appears on this publications server as technical report number 2002-57. A short version of technical report 2002-57 also appears in the proceedings of the 9th International Conference on Data Base Programming Languages (DBPL 2003). (Zitiert auf Seite 9)

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, Y. H. Katz. Improving TCP/IP Performance over Wireless Networks. S. 2–11. 1995. (Zitiert auf Seite 22)

[CM] G. Cugola, A. Margara. TESLA: a formally defined event specification language. S. 50–61. doi:http://doi.acm.org/10.1145/1827418.1827427. URL http://portal.acm.org/ft_gateway.cfm?id=1827427&type=pdf&coll=ACM&dl=ACM&CFID=96927647&CFTOKEN=63976875. (Zitiert auf Seite 19)

[DEB] The 6th ACM International Conference on Distributed Event-Based Systems. URL http://www.csw.inf.fu-berlin.de/debs2012/. (Zitiert auf Seite 17)

[DES] Statistisches Bundesamt. URL https://www.destatis.de/DE/ZahlenFakten/Wirtschaftsbereiche/TransportVerkehr/Verkehrsunfaelle/Tabellen/PolizeilichErfassteUnfaelle.html. (Zitiert auf Seite 14)

[EB09] M. Eckert, F. Bry. Aktuelles Schlagwort: Complex Event Processing (CEP). *Informatik Spektrum*, 32(2):163–167, 2009. URL http://www.pms.ifi.lmu.de/publikationen/. (Zitiert auf Seite 7)

[FMCa] Fundamental Modeling Concepts. URL http://www.fmc-modeling.org. (Zitiert auf Seite 12)

[FMCb] Fundamental Modeling Concepts - Notation Reference. (Zitiert auf den Seiten 13 und 62)

[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. URL http://doi.acm.org/10.1145/2335484.2335507. (Zitiert auf den Seiten 17, 29 und 31)

[LRD+11] M. Liu, E. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, I. Ari, A. Mehta. High-performance nested CEP query processing over event streams. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, S. 123 –134. 2011. doi:10.1109/ICDE.2011.5767839. (Zitiert auf Seite 7)

[NEN] Definition Nennkapazität. URL http://www.itwissen.info/definition/lexikon/Nennkapazitaet-rating.html. (Zitiert auf Seite 11)

[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. URL http://doi.acm.org/10.1145/1142473.1142520. (Zitiert auf Seite 19)

Alle URLs wurden zuletzt am 30.09.2012 geprüft.

Erklärung

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

(Marcus Vetter)