Efficient and Secure Event Correlation in Heterogeneous Environments

Von der Fakultät Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Björn Schilling

aus Sindelfingen

Hauptberichter:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Mitberichter:	Prof. Dr.-Ing. Dr. h.c. Andreas Reuter
Tag der mündlichen Prüfung:	19. November 2015

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart
2015

Acknowledgements

First of all, I would like to thank my advisor, Kurt Rothermel, for giving me the opportunity to work, learn, and do research in his group. He always gave helpful feedback and was on point in asking the right questions whenever necessary. A special thanks also goes to Boris Koldehofe for being a great co-advisor. His love in doing research and dedication in pursuing new ideas heavily inspired the approaches presented in this dissertation.

A big thank you goes out to my colleagues at the VS Research Group, especially to Andreas Grau, Harald Weinschrott, Adnan Tariq, Gerald Koch, Beate Ottenwälder, Frank Dürr, Marco Völz, Stefan Föll, Christian Hiesinger, Stamatia Rizou and Andreas Benzing for creating an amazing atmosphere within the institute which was both inspiring and fun to be part of. Also, I would like to thank Udo Pletat for his support in the DHEP research project.

Finally, I want to thank my family and friends for being supportive throughout the years of my studies and the work on this dissertation. The last thank you goes out to Marina, who was always there when I needed it. You never lost the patience with me and stayed positive through all the years. Thank you!

Contents

List of Figures

Abstract

The importance of managing events has increased steadily over the last years and has reached a great magnitude in science and industry. The reasons for that are twofold. On the one hand, sensor devices are cheap and provide event information which is of great interest for a large variety of applications. In fact, sensors are ubiquitous in modern life. Nowadays, RFID tags are attached to goods and parcels to allow an easy tracking. Numerous weather stations are providing up-to-date information about temperature, pressure, and humidity to allow for precise weather forecasts worldwide. Lately, mobile phones are equipped with various sensor devices like Global Positioning System sensors or acceleration sensors to increase the applicability of the phone. On the other hand, reacting on events has become an increasingly important factor especially for business applications. The occurrence of a system failure, a sudden drop in the stock exchange, or a missing parcel can cause huge costs for the company if their appearance is not handled properly. As a consequence, detecting and reacting on events quickly is of great value and has lead to a change in the design of modern software systems, where event-driven architectures and service-oriented architectures have become more and more important.

With the emerging establishment of event-driven solutions, *complex event processing* (CEP) has become increasingly important in the context of a wide range of business applications such as supply chain management, manufacturing, or ensuring safety and security. CEP allows applications to asynchronously react to the changing conditions of possibly many business contexts by describing relevant business situations as correlations over many events. Each event corresponds either to a change of a business context or the occurrence of a relevant business situation.

This thesis adresses the need to cope with *heterogeneity* in distributed

event correlation systems in order to i) reuse expressive correlation and efficient technology optimized for processing speed, ii) increase scalability by distributing correlation tasks over various correlation engines, iii) allow migration of correlation tasks between heterogeneous engines and security domains, and iv) provide security guarantees among domains in order to increase interoperability, availability and privacy of correlation results. In particular, a framework called DHEP is presented that copes with such requirements.

Deutsche Zusammenfassung

Umsetzung von effizienter und sicherer Ereigniskorrelation in verteilten multi-domänen Infrastrukturen

1 Einleitung

Ereignisverarbeitung wird in verschiedensten Applikationen eingesetzt, um aus vielen Ereignissen komplexe Situationen zu erkennen, auf die dann reagiert werden kann. Grundlage für solche ereignisbasierten Systeme ist eine Vielzahl unterschiedlicher Basisereignisse, wie beispielsweise Messwerte von RFID Sensoren, Bewegungs- oder Temperatursensoren, oder wahrgenommene Veränderungen an Anwendungen oder Datenbanken. Ziel ist es dabei bestimmte charakteristische Muster von Ereignissen, sogenannte Situationen, zu erkennen und weiterzuverarbeiten. Diese Situationserkennung geschieht mit Hilfe von Regeln, welche diese Muster beschreiben. Da oft viele einfache Ereignisse zu dieser Situationserkennung beitragen spricht man auch von Ereigniskorrelation. Ein wichtiges Merkmal für Ereigniskorrelation ist die Aneinanderreihung von Regeln. Das heißt, erkannte Situationen werden selbst als Ereignisse angesehen, welche dann wiederum Grundlage für nachfolgende Situationserkennungen sein können. Da durch diese fortlaufende Ereigniskorrelation zwischen Datenquelle und Datenziel immer aussagekräftigere Ereignisse entstehen, ihre Komplexität also stetig wächst, spricht man von komplexer Ereignisverarbeitung. Zu beachten ist, dass in diesem Zusammenhang der Begriff *Ereignis* semantisch doppelt belegt ist. Er steht neben dem tatsächlichen in der Realität vorkommenden Ereignis auch für die Nachricht, die dieses

Ereignis beschreibt und in das System gesendet wird. In dieser Dissertation ist, wenn nicht explizit anders angegeben, ein Ereignis stets eine Nachricht im Ereignissystem.

Da ereignisbasierte Systeme die Eigenschaft besitzen, automatisiert Situationen zu erkennen und darauf zu reagieren, haben sie eine hohe Bedeutung für Anwendungen im Sicherheitsbereich und Überwachungssysteme in verschiedensten Varianten. Beispielhaft für die Verwendung von Ereignisverarbeitung stehen Anwendungen zur Logistikverwaltung, zum Management großer Anlagensysteme, zum Energiemanagement oder für die allgemeine Überwachung von Geschäftsprozessen. Auch Börsenanwendungen unterliegen typischerweise einer Ereignisverarbeitung. Wichtige Aspekte in solchen Systemen sind die schnelle Reaktionszeit auf auftretende (Problem-) Situationen, eine hohe Skalierbarkeit sowie der Schutz der oft vertraulichen Daten vor Zugriffen von Dritten.

Ihren Ursprung haben ereignisbasierte Systeme in aktiven Datenbanksystemen der späten 80er Jahre, wie HiPAC oder Postgres [DBB+88, SK91]. Es wurde erkannt, dass die automatisierte Reaktion auf Änderungen in Datenbanktabellen einen essentiellen Mehrwert für Anwendungen hat. So war es unter anderem möglich, Fehlerfälle schnell zu erkennen und zu beheben, was in vielen Fällen einen Sicherheits- und nicht zuletzt Kostenvorteil mit sich bringt.

Durch das große Potential dass sich durch die reaktive Funktionsweise ergibt, wurde das Interesse sowohl von Forschungseinrichtungen als auch der Industrie geweckt. Es entstanden im Verlauf der 90er Jahre eine Vielzahl von Systemen zur Ereigniskorrelation sowie dazugehörige Regelbeschreibungssprachen, stellvertretend seien hier Snoop, SAMOS und ODE genannt [CM94, GD92, GJS92]. Die neuen Korrelationssysteme wurden immer mächtiger, und es konnten mehr und mehr Ereignisse analysiert und korreliert werden. Gleichzeitig wurden auch die Beschreibungssprachen immer ausdrucksstärker. Neue Operatoren wurden hinzugefügt die die Beschreibung von komplexeren Situationen erlauben, und erweiterte Verarbeitungsrichtlinien ermöglichten eine weitere Verfeinerung der Situationserkennung [CM94].

Durch die immer größer und wichtiger werdenden Anwendungen, gekennzeichnet durch eine wachsende Anzahl an Ereignissen und Datenquellen,

rückte eine verteilte Ereignisverarbeitung in den Fokus [Cou02]. Anstatt die Ereigniskorrelation an einer zentralen Stelle zu verwalten und durchzuführen, wurden mehrere Korrelationssysteme hintereinander geschaltet, um die Last zu verteilen. Bei weit entfernten Datenquellen war es möglich bereits nah an den Quellen erste Verarbeitungsschritte durchzuführen, und somit die Reaktionszeit sowie die Netzauslastung des Gesamtsystems spürbar zu verringern. Die Vorteile der verteilten Ereignissysteme fanden vor allem in der akademischen Anwendung schnell Anklang, was zu einer stetigen Weiterentwicklung und Verbesserung führte. Um auf Veränderungen von Datenquellen, Ereignislast oder Regeln besser reagieren zu können, wurden Regeln in mobilen *Operatoren* gekapselt [Pet04]. Dadurch war es möglich, zur Laufzeit die Ereignisverarbeitung anzupassen, indem einzelne Komponenten an optimale Stellen des verfügbaren Korrelationsnetzes verschoben werden konnten.

Allerdings hat die Anwendung von verteilter Ereigniskorrelation bisher wenig Anwendung in industriell eingesetzten Systemen gefunden. Die Gründe hierfür sind vielfältig, haben aber zu großen Teilen ihre Ursache in der Heterogenität heutiger verteilter industrieller Applikationen. Durch die Globalisierung der Märkte und der damit einhergehenden Kooperation und Interaktion verschiedenster Marktteilnehmer, sind die resultierenden kollaborierenden Systeme unterschiedlichen Problemen und Schwierigkeiten unterlaufen, deren Lösung Grundlage für einen erfolgreichen Einsatz in industriellen Anwendungen ist:

- **Einheitliche Schnittstellen**: Durch das Fehlen von Standards ist eine Vielzahl verschiedenster Korrelationssysteme mit unterschiedlichen Schnittstellen auf dem Markt erhältlich. Probleme bei der Interaktion verschiedener eingesetzter Technologien/Systeme sind die Folge.
- **Valide und effiziente Platzierung** der einzelnen Korrelationskomponenten (Operatoren): Da die Situationserkennung in vielen Fällen eine vertrauliche und unternehmenskritische Geschäftslogik beinhaltet, ist es nötig, dies bei der Verteilung der Komponenten im Netz zu berücksichtigen.
- **Handhabung vertraulicher Daten** in heterogenen System mit

vielen Teilnehmern: Ähnlich wie die Regeln zur Situationserkennung, sind auch die erzeugten Daten in vielen Fällen sensibel. Vor allem in großen Systemen mit vielen Teilnehmern, muss die Vertraulichkeit der Ereignisse beachtet werden.

In dieser Dissertation werden die Probleme heutiger verteilter Korrelationssysteme analysiert und Lösungsmöglichkeiten erörtert. In den folgenden drei Unterkapiteln wird eine Kurzübersicht gegeben.

2 Verteilte Ereignisverarbeitung in heterogenen Systemen

Obwohl sich in den letzten Jahren mehrere Forschungsprojekte mit verteilten Ereignissystemen beschäftigt haben, und durch die Verteilung der Korrelationsprozesse große Performance-Fortschritte erzielt werden können, zögern industrielle Anwender mit dem Einsatz von verteilten Lösungen. Ein Grund dafür ist die Heterogenität verfügbarer Ereignisverarbeitungssysteme. Durch das Fehlen einheitlicher Standards haben sie keinerlei einheitliche Schnittstelle und verwenden unterschiedliche Beschreibungssprachen für die Spezifikation von Ereignissen und Regeln. Dies macht eine Kooperation von Geschäftspartnern mittels ereignisbasierter Systeme nicht möglich, wenn diese verschiedene Technologien einsetzen.

Dieses Problem macht einen großen Vorteil der verteilten Anwendung von komplexen Ereignisverarbeitung zunichte: Das effiziente Ausnutzen der Standorte von Datenquellen und -empfängern durch vorteilhafte Platzierung der Korrelationskomponenten (Operatoren) im verteilten System.

Im Rahmen dieser Dissertation ist das Ereignisverarbeitungssystem DHEP entstanden. Ziel von DHEP ist es, verteilte Ereigniskorrelation zwischen verschiedenen Einheiten, über Domänengrenzen und interagierende Nutzer hinweg zu ermöglichen, um somit die Akzeptanz von verteilter Ereignisverarbeitung zu erhöhen. Insbesondere ermöglicht DHEP eine Kooperation von heterogenen Ereignissystemen, selbst wenn diese verschiedene Technologien und Beschreibungssprachen einsetzen.

Grundlage dafür ist ein Framework, an die sich die verschiedenen Ereignis-Korrelationssysteme über Adapter andocken (siehe Abbildung 1). Das Framework verhält sich daher ähnlich zu einer Middleware. Es abstrahiert die Netzwerkkommunikation und übernimmt sämtliche Aufgaben zum Empfangen und Versenden der Ereignisse im System (*Event Bus*). Der Einsatz von DHEP ermöglicht somit auch eine verteilte Ausführung von ursprünglich rein zentralisiert laufenden Ereignissystemen.

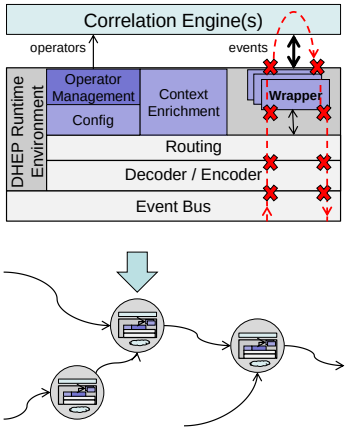


Abbildung 1: Kurzübersicht und Ereignisfluss des DHEP Systems

Um den korrekten Austausch von Ereignissen in heterogenen Systemen zu gewährleisten, verwendet DHEP eine Metasprache. Sämtliche Ereignisse werden in der Metasprache codiert, und bei der Übergabe an Korrelationssysteme in deren eigene Beschreibungssprache übersetzt. Dies geschieht in einer Adapterkomponente (*Wrapper*), die auch sämtliche Kommunikation zwischen Framework und Korrelationssystem übernimmt. Durch die Verwendung einer Metasprache wird eine Interaktion verschiedener Systeme möglich, selbst wenn diese verschiedene Beschreibungssprachen für Ereignisse nutzen.

Zusätzlich bietet das DHEP System Komponenten zur Verwaltung und Verteilung von Operatoren (*Operator Management*) sowie zur Anreicherung von Metainformationen (*Context Enrichment*). Während die Regelverteilung, gekapselt in Operatoren, dazu verwendet wird, die Performance im verteilten System zu optimieren, bietet die Kontext-Anreicherung eine Möglichkeit, die Ereignisse im System mit Metainformationen zu erweitern. Dies ist insbesondere in Business Anwendungen notwendig, da hier oftmals Informationen eines Ereignisses in direkter Relation zu Informationen in Datenbank-Tabellen stehen (wie beispielsweise Kundeninformationen).

Das DHEP Framework bietet die Grundlage für die Lösungen weiterer Probleme, die für den erfolgreichen Einsatz von verteilter Ereigniskorrelation in heterogenen Systemen notwendig sind. Von besonderer Bedeutung sind dabei die effiziente Verteilung von Operatoren und das garantierte Einhalten von Sicherheitsrichtlinien, die in den nachfolgenden Unterkapiteln zusammengefasst werden.

3 Effiziente Verteilung von Operatoren

Eine der Kernaufgaben für skalierbare verteilte Ereignisverarbeitung ist es, die einzelnen Operatoren im Netzwerk so zu platzieren, dass die Verarbeitung möglichst effizient ist. Die Platzierung hat einen großen Einfluss auf die Kommunikationswege im gesamten Netz und somit auf die für die Verbarbeitung relevanten Faktoren wie Reaktionszeit, verursachte Netzwerklast oder die maximale Ereignislast. Obwohl in den letzten Jahren in diesem Gebiet geforscht wurde, und effiziente Algorithmen zur optimalen Platzierung veröffentlicht worden sind [RDR10, Pet04], sind diese für den Einsatz in einem System wie DHEP nicht geeignet. Diese bestehen in großen, unternehmensübergreifenden Netzwerken aus heterogenen Hosts, die sich in Verarbeitungsgeschwindigkeit, Domänenzugehörigkeit, Netzanbindung und nicht zuletzt verwendeter Korrelationstechnologie unterscheiden. Diese Unterschiede müssen beachtet werden, wenn die Operatoren im System platziert werden. Einige typische Fall-Beispiele sind:

- einzelne Teilnehmer im System fordern, dass ihre Operatoren nur in bestimmten Domänen eingesetzt werden
- manche Operatoren erfordern eine bestimmte Korrelationtechnologie bzw. Korrelationsengine
- es bestehen Mindestanforderungen für die Netzanbindung

Die Präsenz vieler strikter, binärer Bedingungen und Einschränkungen verändert das Optimierungsproblem ausschlaggebend. Anstatt eine möglichst optimale Positionierung jedes Operators im Gesamtsystem zu erreichen, kann jeder Operator nur in einer sehr eingeschränkten Menge an Positionen im Netzwerk positioniert werden: Die Position ist an Bedingungen geknüpft. Da aus dem ursprünglichen Problem ein Bedingungserfüllungsproblem (englisch: *Constraint-Satisfaction-Problem*) wird, können bisherige Algorithmen zur Operator-Platzierung nicht mehr verwendet werden.

Im Rahmen dieser Dissertation wurde die verteilte Platzierung, Migration und Optimierung von Operatoren in heterogenen Netzstrukturen untersucht. Dabei ist ein Platzierungsalgorithmus entstanden, der die Bedingungen jedes Operators bei der Platzierung beachtet, und gleichzeitig nach einer möglichst optimalen Platzierung sucht. Das Ziel der Optimierungsstrategie ist es, die Auslastung des Netzes zu minimieren. Dies führt zu im Mittel kurzen Antwortzeiten sowie geringem Bandbreitenverbrauch. Der entstandene Optimierungsalgorithmus arbeitet dezentral und adaptiv. Alle zum System gehörenden Hosts werden bei der Suche nach einer optimalen Platzierung eingebunden. Bei Veränderungen im System (z.B. Wegfallen eines Knotens oder eine veränderte Ereignislast), wird eigenständig nach neuen Lösungen gesucht.

4 Durchsetzung von Sicherheitsrichtlinien

Eine wichtige Fragestellung beim Einsatz von verteilten Systemen ist die Einhaltung von Sicherheitsgarantien. Da in verteilten Netzen die Daten- und Verwaltungshoheit nicht an einer zentralen Stelle liegt, ist die Garantie von Richtlinien deutlich komplexer als in einem zentralen System. So

besteht die Gefahr, dass durch Abhören von Netzwerk-Leitungen Daten von Unberechtigten abgehört werden, oder dass sich Dritte als Teil des Systems ausgeben und damit versuchen Einfluss auf die Ereigniskorrelation zu nehmen. Diese Gefahren haben auch in der Forschung zu verteilten Ereignissystemen zu einem verstärkten Interesse im Umgang mit Sicherheitsgarantien geführt.

Während in Zuge dessen Konzepte entwickelt wurden, wie Ereignisse durch Verschlüsselngsmechanismen sicher durch ein Ereignissystem versendet werden können [TKAR10, OP01], sind die Mechanismen zur Verwaltung und Einhaltung dieser Verschlüsselungen bisher nicht ausgereift und haben Schwächen. Insbesondere das Fehlen von Mechanismen, die die Vertraulichkeit der Ereignisdaten sicherstellen, stellt eine hohe Sicherheitslücke dar.

Im Rahmen dieser Arbeit wurden verschiedene Sicherheitsansätze für verteilte Ereignissysteme vor dem Hintergrund eines multi-domänen Netzes analysiert, bewertet und verbessert. Dabei ist unter anderem ein Algorithmus entworfen worden, der die Vertraulichkeit von Ereignisdaten im Verlauf von mehreren Ereigniskorrelationsschritten gewährleistet. Dies stellt einen wichtigen Baustein dar, um die Akzeptanz von verteilter Ereigniskorrelation entscheidend zu verbessern.

1 Introduction

1.1 Motivation

In today's business world, event-based communication is of increasing importance. The ongoing globalization of markets forces companies to be flexible and adaptable in order to stay on par with their competitors and not have a competitive disadvantage. This forces the market players to continuously monitor and analyze both their business processes and the global market in order to reveal how well the processes perform and to react and adapt on various events [LKS+10, IBM12, CD97]. Since it may become very costly if a problem is not recognized or reacted upon, the topic gained interest of both the industrial and academic world. For example, manufacturers have to quickly identify a shortage of materials or faulty supplies, a crash of important server systems, or a stock market collapse [CA08, HSB09].

Against this background, many modern (cooperative) application platforms are designed based on the paradigm of reacting on events, called event-driven architecture (EDA, cf. [Mic06]). In contrast to classical communication patterns like RPC or RMI, which are active in terms of communication, event-based system are typically more passive, waiting on a certain event input before they start any action. As a result, the EDA-based systems have no control about *when* data is processed. Instead, they are triggered by the input received from event producers, like sensor devices transmitting their events via event channels. Sensors may not only measure the physical environment (e.g., for tracking, or intrusion detection), but also the state of an observed computer system or application (e.g., business processes), making EDA systems flexible and holistic, independent of their application area.

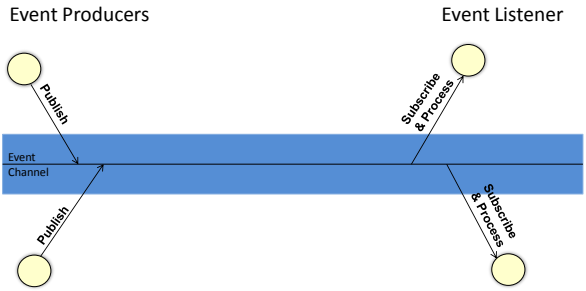


Figure 1.1: EDA Communication based on the reaction on Events

With the emerging establishment of event-driven solutions, and the contemporary rapid increase of event sources, *complex event processing* (CEP) has become increasingly important in the context of a wide range of business applications such as supply chain management, manufacturing, or ensuring safety and security [Luc11]. CEP allows applications to asynchronously react to the changing conditions of possibly many business items by describing relevant business situations as *correlations* over many simple and complex events. Each event corresponds either to a change of a business item or the occurrence of a relevant business situation.

CEP provides high flexibility in writing and reconfiguring business applications, e.g., by decoupling low level information related to technical content and high level information related to business content. Consider an organization that wants to monitor access to a restricted area. In this context, sensor readings that provide position information about movements within the area represent technical data, while the business application works on process-relevant events like the entry of a person into a dangerous area. Though substituting the sensor technology might require new correlations to detect such a situation, applications that work on this situation will not require any change at all.

CEP systems have seen a change of perspective recently. While, originally, powerful CEP systems were used in a central way to efficiently correlate events and detect situations, the emerging increase of event sources

and event consumers has lead towards a decentralized handling of events [Pie04, LJ05, KKR10]. In addition, the increasingly collaborative nature of today's economy [Hag02] results in large-scale networks, where different users, companies, or groups exchange data (cf. Figure 1.2). As a result, event processing networks are heterogeneous in terms of processing capabilities and technologies, consist of differing participants, and are spread across multiple security domains.

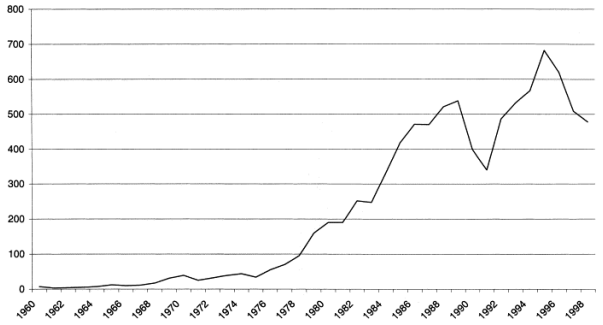


Figure 1.2: R&D Partnerships of Companies since 1960 [Hag02]

The increasing need for interoperability of applications leads to additional challenges for CEP systems deployed in industrial environments. CEP applications have to run on distributed networks, whose participants are highly heterogeneous. While cooperating companies act as hosts for a business network on the one hand, they do, on the other hand, not necessarily use the same event processing technologies. That said, modern large-scale CEP networks may differ in terms of hardware and software, but also the methods of communication may be different. Still, seamless event processing among different cooperating companies is necessary to guarantee a successful application of distributed CEP in industrial environments.

This thesis focuses on the challenge of establishing an efficient and secure

CEP in a distributed multi-domain infrastructure. As a foundation, a platform is introduced that enables complex event processing over heterogeneous networks, consisting of different CEP processing technology. Furthermore, methods and algorithms are discussed which allow for an efficient processing of events in such environments. Finally, we investigate security mechanisms for cross-domain event processing systems. Specifically, methods for maintaining the privacy of event information are presented.

To reflect the diversity of applications utilizing CEP we will use a couple of application examples illustrating our concepts. In particular, we focus on the deployment of a CEP system in energy & utility applications, which manage modern large-scale energy grids where multiple and diverse energy providers are opposed to a multitude of heterogeneous event consumers. Furthermore we utilize a logistics example to illustrate application scenarios where event processing is done over a long chain of different entities, because various heterogeneous logistic partners are involved in today's logistic chains.

1.2 Technological Trend towards Complex Event Processing

1.2.1 Sensing and Detecting

In recent years, the importance of complex event processing has increased dramatically. Technological progress and high demand for sensing applications has had a major influence of event processing [CA08]. Nowadays, sensor devices are cheap, and the ongoing use of sensors in everyday life has lead to a massive increase of event data. Today, a considerably large amount of the more than 3 billion mobile phones are smart phones [Rid07] which are equipped with multiple sensors measuring the environment, like GPS [HWLC93] or acceleration sensors. Production systems are periodically reporting their current state, and RFID-Tags [Fin10] are attached to wares and parcels to allow easy tracking of items [HSB09]. Today,

several billion RFID sensors and GPS receivers are deployed worldwide [CRHPP09].

In parallel to the availability of sensed information, the interest in events has been increased steadily. The continuous increase of complexity of business processes [VVL07] demands for a more and more precise analysis of current system and business states and a consequent reaction on it. Furthermore, many modern applications are context-aware and rely on measurements of its users and their context [DHN+04]. This need has lead to a massive increase in sensing applications (e.g., [WHFG92, MLF+08, ASSC02, LM03]). The basic motivation behind these applications is to gain business benefits from the additional information retrieved by sensors. Examples are optimization of business processes (e.g., logistic throughput of warehouses), security applications (e.g., intrusion detection), or public sensing applications (e.g., traffic news on smart phones based on current locations).

To be able to make efficient use of the events produced by sensor devices, additional logic is needed [Luc08]. Therefore, event processing engines have been developed, which create valuable, higher level information based on received basic events [CM94, AE04]. This is done by correlating the received events, which typically includes comparing, relating, aggregating, and/or filtering them. The result of the correlation process is a complex event, generated by the engine as soon as the conditions, specified in the correlation description, are fulfilled. The complex event can be thought of as an indicator that the situation of interest has been detected. Based on this complex event, a user or a system can then react on the findings.

To increase the efficiency of event processing engines, the generated complex events can not only be send towards a user, but also be again an input of the engine. By doing this, the process of correlating event information can be decomposed in multiple consecutive steps. In every step, the generated event output has a higher level of semantic information. At the same time, the event throughput gets reduced in every step. This is often pictured in an event pyramid, where basic events make up the foundation of the pyramid, and get correlated on their way to the top, where a high level complex event represents a detected situation (cf. Figure 1.3).

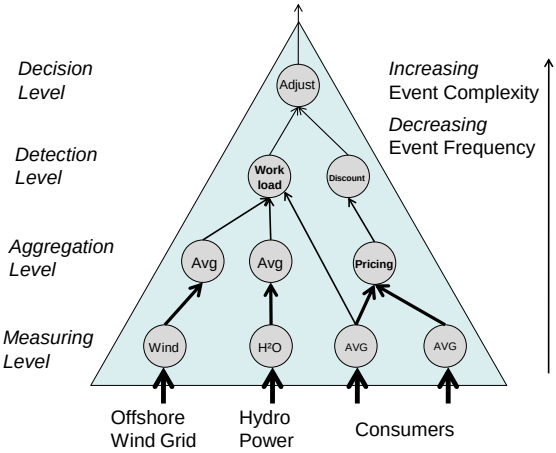


Figure 1.3: Event Pyramid for a Smart Energy Processing

1.2.2 Communication and Computation

The original design of correlation systems has been inherently centralized, thus limiting scalability as the number of event sources increases in the system [AE04, CM94, GD92, GJS92]. However, the dramatic increase in the complexity of applications, business situations and the increase of sensors that contribute to them shows that more and more event sources need to be accounted for in the future. This growth is amplified by the industrial need of interoperability.

To be able to distribute the correlation steps in the network, the abstraction of operators has been introduced (cf. [LJ05, Pet04]). Operators encapsulate correlation functionality, like aggregating or filtering an event stream, and can be deployed on the existing network nodes, called hosts.

The introduction of operators demands for concepts of communication and placement of operators, as well as composition and decomposition of correlation mechanisms. The first two concepts are inherently connected to the distributed network, the available hosts, and their respective characteristics. It has to be assured, that the operators are correctly deployed on a host, so that they can send and receive events to and from other operators. The latter two concepts are needed to be able to split the whole situation detection into multiple correlation steps, so that each step can be encapsulated in an operator. This is closely related to the splitting of an event pyramid, as described in Section 1.2.1.

Currently, the distributed networks using correlation systems are homogeneous, allowing for constraint-less implementation of CEP [LJ05, MSS99, PSB03, LFA+06, KKR10, LLS08, RDR10, SMW05, ZOTW06]. However, multi-domain networks, as they are present in today's business world, are far from being homogeneous in terms of technology and hardware. The deployment of CEP systems is subject to a lot of constraints that have to be met. Consequently, the necessary step in the development of distributed CEP systems is the efficient interconnection of hosts in heterogeneous networks, to allow for multi-domain event processing.

1.3 Focus and Contributions

In today's business world, CEP is not yet fully exploited and is still establishing itself [Luc11]. Distributed CEP is even one step behind. The most decisive reason is, that for industrial event processing, not only performance is important. For example, in many business applications also orthogonal attributes like functionality, expressiveness, reusability of present knowledge, user friendly interfaces, and flexibility are essential [SKPR09]. These attributes are not present in current distributed approaches, which focus mostly on an efficient distributed detection of situations.

Therefore, many industrial users still rely on existing, mature complex event technology [AE04] which is capable of providing the required functionality, at the expense of distribution (cf. [TIB, jBo]). Scalability is typically achieved by providing more powerful servers hosting the CEP System (cf. [BER08]). This, however, is a tremendous disadvantage in scenarios where the application requirements vary or are even completely unknown before. For example, when the event rates in a system increase over time due to a rise in users and/or sensor devices, the deployed system's capabilities will eventually reach its limits. Also, many scenarios, like measuring current power consumption within a nationwide or even continental power grid, are inherently dispersed. A centralized CEP system handling all the sensor data would not only cause a huge processing cost, but in addition cause a major communication overhead.

In the future, integration of universal, heterogeneous CEP technology will be key to allow flexible CEP systems that are capable of adapting to user needs. The reason for this is that many users rely on different products (because they specialize in some certain criteria) that are not capable to interact. The main reason for that is the missing existance of a generally accepted definition language for complex event processing, though first steps in this direction have been made (cf. [LS11]). This is a major drawback for many applications, since interoperability and communication is mandatory in today's business world. Business partners are forced to interact, and with the emerge of event-driven architectures this also affects CEP systems. Hence, commonly used and accepted communication and event descriptions have to be developed to enable interoperability.

This work focuses on bringing CEP to the next level. Therefore, we elaborated which problems have to be solved to allow the establishment of efficient and secure event processing in a distributed multi-domain infrastructure [SKPR09]. This leads to three main contributions discussed in this thesis.

First, the interconnection of event processing systems and event communication among heterogeneous domains is elaborated. Here, the DHEP framework is introduced, which focusses on these aspects, such that business processes can benefit from a distributed processing of events. The framework supports the integration of various established centralized CEP systems in a distributed environment. The system has been developed and implemented in collaboration with the IBM Research and Development Laboratory Böblingen. It will be shown that DHEP is a scalable complex event processing system, which enables interaction with business processes and context information. It therefore enables an efficient approach for distributed CEP in business contexts [SKRR10].

Second, means are provided to optimize the efficiency of the DHEP system. Both the placement and migration of operators in a distributed heterogeneous environment is discussed. Strategies are presented, which ensure a correct functionality of the CEP on one hand, and provide an efficient usage of the network resources, such as latency and bandwith consumption, on the other hand. Furthermore, the placement and migration strategies guarantee, that all constraints set towards the placement of operators are met [SKR11]. This may include for example security policies, which do not allow operators to be placed in certain domains.

Third, this thesis describes methods and mechanisms which provide security in multi-domain environments. While basic security mechanisms like authentication and confidential event delivery build the foundation, the focus in this work lies on the privacy of event information. Especially if multiple participants are interconnected in a multi-domain network, the privacy of events plays a major role in the acceptance of the CEP system. Therefore, methods are presented which protect the event information, even if the event is processed and correlated over multiple steps [SKRR13].

1.4 Structure of the Thesis

This thesis is structured as follows. First, a detailed overview about the fundamentals of complex event processing is given in Chapter 2. Both conventional event-based systems, as well as distributed CEP systems are discussed in-depth. Afterwards, the DHEP framework is introduced in Chapter 3. DHEP is our middleware platform allowing a distributed event processing over heterogeneous environments. Chapter 4 focuses on the optimization of heterogeneous distributed CEP systems. We give insight in current placement algorithms for complex event processing and propose two algorithms that focus on an optimized placement in heterogeneous networks. In Chapter 5 we elaborate how a sophisticated access policy management is required to close the security gap that emerges in multi-domain CEP networks. We propose a way to efficiently solve this problem by calculating obfuscation values of event information. Finally, in Chapter 6 we summarize the contributions of this work and give an outlook to promising future research topics.

2 Fundamentals

In this chapter, we will give an overview about the current state of event-based systems and distributed complex event processing. More important, we will give definitions for the components of the system model we use throughout this thesis. This is necessary to avoid misconception, as there is no standardized terminology available by the time this thesis is written.

2.1 Event Based Systems

2.1.1 The Concept of an Event

The definition of an event has brought up a lot of discussion and also misconception within the event processing community. The reason is that when event processing became popular in the 1990's, researchers and companies with different scientific background put research effort in it. As a result, multiple event definitions have been introduced over the years by different researchers and the lack of standardization attempts has lead to a state, where almost every event processing system made use of its own definition of an event. One of the biggest disputes about the definition of an event originated in the ambiguity of the term *event*. Both the event occurrence, as well as the notification about it (i.e. the transmitted message) were defined as an event.

To overcome the existence of multiple definitions, a group of researchers and organizations formed the event processing technical society (EPTS) in 2008 [LS08]. The EPTS has, among others, the goal of fostering a standardization of event processing. As an important first step towards

this, researchers and organizations agreed on a common glossary, specifying the most important event processing terms, including the notion of an event [LS11]. However, the glossary still contains two definitions of an event, one for the actual event occurrence (e.g. a door is opened), one for the event message reporting about this event occurrence (e.g. a sensor notification about the door being opened). To avoid the double definition of the same term, this thesis only uses the EPTS definition of an event message:

Definition 1 (Event (event object, event message, event tuple)) *An object that represents, encodes, or records an event, generally for the purpose of computer processing.*

Examples are an email confirmation about an online purchase, a message reporting about a stock exchange trade, or incoming goods in a warehouse.

While today's agreement in the event processing community is that both the event message as well as the actual event occurrence can be referred to as an *event*, it is ambiguous throughout a written document. Since this thesis addresses the technical and structural aspects of heterogeneous event processing networks the definition of an event as a message sent in the network is sufficient.

Every event contains a timestamp, discussed in the following, as well as one or more attributes (e.g., the *name*). Event attributes have discrete values.

2.1.2 Timestamps of Events

The timing of events is crucial in event processing systems. However, different syntax and semantics are existent in related work. The reason is, that events can be either *primitive* or *complex* whose semantics differ substantially. Primitive events are the initial events entering the system. Typically they are produced by sensor devices or monitoring systems. Complex events are self-created events of the event processing system.

The difficulty with the timing of events stems from the fact that complex events are often the result of multiple correlated events which have differing timestamps. While *primitive* events can be usually captured at a certain discrete point in time, this is not necessarily the case with complex events. Consequently, the occurrence of complex events may also be described as a timespan instead of discrete timestamps, e.g., by using the earliest and latest timestamp of the correlated events [YB05, LCB99, AC06]. However, the handling of events with non-discrete timestamps is rather complex, especially if the timespans of different complex events overlap. Finally, it has to be stated that, if uncertainty and reliability of the event sources are taken into account, the definition of timestamps gets even more ambiguous [BR04].

To simplify the handling of complex event processing, in this work the event timestamp is defined as the time when the event is created at one of the source nodes or a CEP node due to correlation. This is in line with most existing distributed CEP systems (e.g., [Fid06, Pie04, KKR10]), which is important for our system, since it aims towards enabling event correlations among heterogeneous environments. This means, each event, whether complex or not, has a definite timestamp. The timestamp is assigned by event detectors (e.g., sensor devices) which create primitive events or by the host's correlation engines which create complex events.

2.1.3 From Active Databases towards Complex Event Processing

Event processing was first introduced by database monitoring and management concepts like HiPAC [DBB+88] or Postgres [SK91] in the late 80's, respectively early 90's. At this time, the usage of databases was more and more common for modern systems and applications, and their popularity came hand in hand with a continuous extension and improvement of database systems [CA08]. One of these extensions was the ability of a prompt reaction on changes to the database, causing the terminology of *active databases*. This was typically done by a database management component which monitored the database activity. This feature was a big

benefit for many business applications which focus on reacting quickly to certain changes, like security or banking applications.

To specify the reaction on specific database events, database programmers made use of the ECA (Event-Condition-Action) paradigm, first introduced by HiPAC [DBB+88]. The paradigm contains all necessary information how to react on a certain event occurrence. The ECA paradigm soon gained attention, and while initially only one event was specified to trigger the action, the event part was soon capable of handling several correlated events, e.g., an event sequence.

Definition 2 (Rule (CEP/ECA rule, operator description)) [1]
An expression formalizing the detection of a complex event. Including (but not necessarily limited to) the three pieces of information:

1. *It contains the event(s) that should be reacted on.*
2. *It specifies the conditions that have to be met. For example, this could be some specific value of the event's attribute.*
3. *It specifies the action that should be executed when the conditions of the event were fulfilled. For example, the definition of the complex event.*

In many CEP languages, the parts 1. and 2. are referred to as the *event pattern*. It is the most prominent characteristic of each rule. And, although the syntax of it may vary heavily between different expression languages, their semantic expressiveness is often similar, which allows for mappings from one language to another.

While an event pattern always refers to a single rule, a situation can comprise a potentially large set of rules, which may contribute to its outcome. An example is the event pyramid depicted in Section 1.2.1, where multiple rules contribute to a decision to adjust the workload. An event pattern

[1] With the increasing popularity of distributed CEP systems, where the denotation of *operators* is widely accepted as the implementation of a rule (cf. 3), the term *operator description* has also been used to name a rule. As in most contemporary work, it will be used in throughout the contributions of this thesis.

can, however, be interpreted as the smallest situation detectable in a CEP system.

ECA rules have been a key concept on many subsequent pioneer CEP systems developed by academic projects. Prominent examples are Snoop [CM94], A-RDL [SPC96], ARIEL [Han96], COMPOSE [GJS93], ODE [LGA96], REACH [BBKZ94] or SAMOS [GD92]. Furthermore, ECA was also present in the field of embedded and reactive systems, where the reaction on an event was not restricted to a database object but also on system states [ACV97].

With the conceptual advances made, the ECA paradigm was further extended in the action part being able to create a new, higher level event which itself could be the input of another ECA rule. Therefore distinction has been made between primitive events, which are triggered by the database, and composite (complex) events, which represent a subset of a primitive event history [GJS92]. All primitive events that are matched by the event expression are mapped towards the complex event subset history. Primitive events typically originate from a transaction, a changing object state, or the execution of a method. Also, time events can be considered [GJS92, Cha97]. Furthermore, an event was able to have attributes associated with it [CM94], i.e. it contains multiple pieces of information which can be used in the specification of the complex event detection (cf. the condition part of a rule (Definition 2)).

With the number of primitive event sources increasing, also the event expressions got more and more sophisticated. They were originally formed by primitive events, potentially related by logical operators like conjunction ($\wedge$, and) or disjunction (!, not). Over time, expressions also were capable of handling interval detection [GJS92, CM94]. While operators for conjunction and disjunction are ignoring the detection time of events, others like interval detection consider the sequence of incoming events. For example, in ODE $relative(A, B)[h]$ expresses the occurrences of event (or complex event) B, after the occurence of event (or complex event) A, in an event history h.

To give the user more influence on the correlation behavior, the application for parameter contexts became apparent, first introduced by Snoop [CM94]. Parameter context enables the user to specify the computational

behavior of Snoop during the creation of complex events. In particular, the user is able to specify which events should be used in order to calculate a certain complex event. This is important when multiple events (with potentially differing attribute values) match a certain rule, since using different events may lead to different results. Hence, Snoop allows the user to influence the application semantic, which made the language very popular.

Hence, with the progress made in description languages, it was soon possible to create high level event information based on the occurence of multiple, possibly already correlated events. Basically, a new research field was named Complex Event Processing (CEP). ECA rules are considered as the foundation of CEP description languages and although CEP expressions from modern approaches may have differing structures, their content is very similar to those of traditional ECA rules. Some example descriptions are presented in Section 2.1.3.

On the downside, most of the Active Database products lack scalability due to the fact that they work on a single instance [SKPR09]. This is not only problematic when we consider a large-scale use of the correlation approach, but additionally the single node turns out to be a single point of failure. This holds true in particular when event processing is used in an enterprise-wide context. For example, it is used for central control of multiple vehicle manufacturing plants with cross-plant event-based communication or for monitoring assets or people in multi-building premises of some enterprise. The key idea to deal with these problems is to design a distributed correlation system placed on an interconnected set of hosts. By spreading correlation tasks across the network the system becomes more scalable. Additionally, fault-tolerance can be increased by enhancing the availability of the components (i.e., replicating correlation nodes, hosts, within the network).

Examples of Language Syntax

To depict the differences in language syntax, Figure 2.1 shows examples of different expressions from the languages ODE, Snoop and AMiT.

All complex events in ODE can be expressed as regular expressions. This is advantageous for the implementation, as finite state automata can be used to detect complex events easily and efficiently. However, ODE also has multiple shortcomings. For example, it does neither enable a fine-grained selection of events, nor is it able to reuse an event (for multiple event expressions). Also, it is unable to create events expressing the semantic information of a detected complex event, as detected complex events are always expressed as a (subset) history of occured primitive events. Finally, ODE lacks some operators which later turned out to be useful in many CEP applications, like counting operators.

Snoop, which has been developed by event processing pioneer Sharma Chakravarthy in the early 90's at the University of Florida, played a major role in the beginnings of CEP [CM94]. In contrast to ODE, Snoop is not a database, but rather an expressive language to specify detection rules for complex events. It is designed and implemented for the Sentinel object-oriented database [Cha97].

With the increasing popularity of complex event processing, the capabilities of active databases increased. One modern example of a dedicated event processing engine is AMiT. AMiT has been developed by IBM in 2004, featuring a wide range of possibilities to detect complex events. Expressions are very modular and the user is able to specify fine-grained parameter context to every object of the events and the rules. To have more control about the semantics of over-time-correlations, AMiT introduces the notion of a *lifespan*, which can be seen as a time window wherein situations can be detected. However, due to the fine-grained control the user has over the execution of its event processing, the AMiT language is much more complex. Hence, operators that can be easily expressed in older language concepts require multiple lines of AMiT operator description.

2.2 Distributed CEP

While, originally, powerful engines with expressive correlation languages were used in a central way to efficiently correlate events and detect sit-

Operator \ Language	ODE	Snoop	Amit
Conjunction	E andsign F	E ∧ F	Operator = conjunction First operand event: “E” Second operand event: “F”
Disjunction	E orsign F	E ∨ F	Operator = nth 1 First operand = event: “E” Second operand = event: “F”
Sequence	relative (E, F)	E ; F	Lifespan initiator = event: “F” correlation: “ignore” Operator = nth 1 Detection mode = immediate First operand = event: “E”

Figure 2.1: Examples for Different Syntax of Expression Languages

uations, the increasing popularity of event-based communication, for example in Publish/Subscribe networks, promoted the investigation in the distributed detection of complex events. The main motivation for distribution comes from the emerging increase in event sources and consumers, as well as their often inherent dispersal. Instead of transmitting every event to a central correlation engine, communication can be reduced if the correlation functionality is distributed over multiple entities across the network.

Consequently, multiple distributed CEP approaches have been designed over the last decade. Examples for systems supporting distributed CEP are HERMES [Pie04], PADRES [JCL+10], and CORDIES [KKR10]. Furthermore, a lot of research has been conducted on how to efficiently distribute the correlation of events (e.g., [Jak03, LJ05, LGL08, PSB03]). The general conception is that the logic to create complex events is split into multiple parts. These parts can in turn be deployed on a set of available hosts in the network. By splitting the event processing task, the load for processing events can be distributed over multiple hosts. However, the complexity of handling the event processing is increased. Yet this opens space for various optimization. For example, one can reduce the used network bandwidth by placing an aggregation task closer to the event sources, or influence the detection delay by favoring powerful processing hosts for computation-critical tasks.

Especially with the increasing popularity of Publish/Subscribe systems (cf. [TKKR10, BKR09]), event processing has become a highly distributed technology [JCL+10, KKR10]. In the Publish/Subscribe interaction scheme, participants of the network can express their interest in certain messages (i.e., *events*) as well as publish their own messages to anyone interested. Over the years, different variants of Publish/Subscribe systems have been proposed, a good overview can be found in Eugster et al. [EFGK03].

Complex event processing can benefit from the the method of subscribing to specific events of interest, which can be extended to support complex events. A CEP engine attached to the Publish/Subscribe system takes care of the detection of complex events. Furthermore, the possibility to detect complex events on any host in the system opens a lot of optimization possibility, such as moving functionality closer to the source in order to reduce the overall network usage.

In the following subsections, a description of the major challenges of distributed CEP is given. First, Section 2.2.1 discusses system and network models of distributed CEP systems, specifically introducing the system model used in this work. Second, Section 2.2.2 discusses how placement of functionality influences the behavior and efficiency of these systems. Finally, Section 2.2.3 discusses the basic security challenges that occur.

2.2.1 System and Network Model

While the advantages of event-based communication have been widely recognized, the implementation and deployment of such systems requires a more sophisticated model. More components interacting in networked environments lead to a more complex and error-prone communication. As a result, pioneering work in distributed complex event processing had a strong focus on modeling distributed event-based systems in the presence of communication specific characteristics, like delay and loss of event messages. Therefore, we first briefly discuss a couple of pioneer DCEP models, before detailing the model used throughout this thesis.

As one of the first works in CEP modeling, Schwiderski describes an architecture for distributed event processing in 1996 [Sch96]. The work

focuses specifically on the timing of complex events. To handle the timing complexity in distributed systems, the architecture is based on the 2g precedence model, which defines a global time approximation. When using the 2g precedence model, events can reliably be ordered if they are at least two clock ticks apart (hence *2g*). Since the clock modeling affects the efficiency of the event processing, the handling of clocks is reflected in the architecture. At every network node (called *site*), both a local and a global event detector is deployed. The introduced model is affecting the behavior for the global detectors, receiving events from the whole network, while the local event detector is not affected by the global clock timing and acts only on local events, similar to the centralized, database centric CEP systems available at that time. The language used by Schwiderski is similar to Snoop (cf. Section 2.1.3). The proposed architecture is a first attempt to deploy a CEP system in a distributed architecture. However, it does not tackle any efficiency related challenges, like placement of detectors in the network.

In 2003, Pietzuch et al. presented a *Framework for Event Composition in Distributed Systems* [PSB03]. It is basically a middleware, which can be deployed on top of a Publish/Subscribe infrastrucure for event dissemination. The middleware introduces the abstraction of a *mobile CE detector*. A CE detector encapsulates a finite state automaton detecting a complex event expression, similar to ECA rules. Each of the detectors uses the Publish/Subscribe infrastructure to subscribe to events needed for the detection as well as to publish the detected complex events. The detectors have an agent-like behavior, allowing them to move from one node to another. Furthermore, the framework supports sub-expressions, resulting in the capability to reuse some existing CE detectors for newly added expressions.

These pioneer works form the basis for many of today's approaches, and many of the introduced concepts can be found in them. As our work is not aiming towards creating a new detection mechanism or expression language, but in enabling interoperability among existing systems, our focus lies on using a simple, basic system and network model which can work well with most available CEP systems. Consequently, our system model used throughout this work is influenced by these available distributed CEP models [PSB03, Sch96].

Similar to [PSB03], we introduce the notion of an operator, which acts as a mobile complex event detector.

Definition 3 (Operator) *An implementation of a* rule *for a distributed environment. Operators can be deployed on any CEP host participating in the CEP network. They are mobile, i.e. they can be migrated to another host during runtime.*

Operators receive/access events from the the CEP system and detect/output complex events based on the rule they implement, which will be formally defined in the upcoming subsection.

Operators are deployed on the hosts in the CEP system and can be moved from one host to another. Throughout this work, we assume the presence of an overlay network capable of migrating components. Distributed CEP systems are typically built on top of an *event broker* network which we will discuss in the following (cf. [PSB03, LJ05]).

CEP Overlay Network

We assume a distributed correlation network $N = \{n_1, n_2, ...\}$, where dedicated hosts (i.e., event brokers) are interconnected. These hosts are capable of deploying operators, which are executed to collaboratively detect situations and form the distributed CEP system. The cooperative interaction of the operators is modeled by a *directed operator graph*.

The operator graph $G = (\Omega, S)$ consists of *operators* $\omega \in \Omega$ and *event streams* $(\omega_i, \omega_j) \in S \subseteq (\Omega \times \Omega)$ directed from ω_i to ω_j.

In every event stream (ω_i, ω_j), we call ω_i the event *producer* and ω_j the *consumer* of these events.

Operators with an incoming stream degree of 0 are called *sources*, and operators with an outgoing stream degree of 0 are called *event consumers*. By treating event sources (e.g., sensor devices) and event consumers (e.g., user applications) as operators, we do not need to care about interfaces between them, which considerably simplifies our model.

We assume that the overlay network will deliver every message. The event streams are FIFO, guaranteeing the ordered delivery of events. Operators are not static, but may leave and reenter the system due to network and node failures.

Every operator ω implements a function $f_\omega : I_\omega \rightarrow O_\omega$ that maps incoming event streams I_ω to outgoing event streams O_ω. In particular, f_ω identifies which events of its incoming streams are selected, how event patterns are identified (correlated) between those events, and finally how events for its outgoing streams are produced.

We illustrate the definition of a CEP system by means of a simple logistic chain example. In the scenario, a manufacturer, a shipping company, and a customer each constitute a domain and each is providing an operator in its domain. The operators establish event streams as depicted in Figure 2.2. The manufacturer wants to send an item to one of its customers. Therefore, it sends events to a shipping company providing information about the item's destination, its production place as well as the time when the product is ready. The shipping company receives these event attributes and uses them in a correlation function f_{sc}. The correlation creates a tuple of organizational information: the warehouse the item is going to be shipped to for further delivery, and the estimated day of delivery.

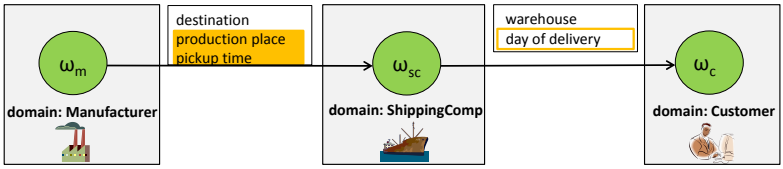


Figure 2.2: Operator Graph of a simple Shipping Scenario

Large-scale, publish/subscribe messaging systems, as we make use of them within DHEP, are subject to failures of nodes and links. Furthermore,

hosts participating in the CEP overlay network may join and leave dynamically. Consequently, a CEP service like DHEP has to be robust in the sense that it can handle these conditions hence can be dynamically reconfigurable.

2.2.2 Operator Placement

One of the core challenge towards scalable complex event processing is to efficiently distribute the event correlating operators within the available hosts of the event processing network. As already motivated in Section 1.2.1, the requirement to place operators is an inherent and crucial consequence of the distribution of CEP. The placement can heavily influence the performance of the detection process as well as the load of the network.

Depending on where the different operators receive, process, and emit events, the behavior of the whole CEP system will change in terms of resource consumption, responsiveness, and reliability. The challenge is to find an operator placement, such that CEP is efficient with respect to the applications optimization goals, such as network usage.

Consequently, the placement problem has found consideration by many researchers (cf. [Fid06, LJ05, PSB03]). Depending on the optimization goal, the placement approaches differ. For example, some systems try to detect situations as soon as possible, others seek towards balancing the load. In large-scale distributed applications, like industrial systems working on global scale, an efficient utilization of the network resources is very important, as they often can be a bottleneck. An optimized network usage reduces the induced network load and network congestion and therefore leads to a robust application that is able to process as much events as possible (cf. [RDR10]). Network utilization is computed as the product of bandwidth and delay.

To reach the optimization goal, the CEP functionality is encapsulated and then migrated within the network. Up till now, distributed CEP is characterized by relying on a homogeneous system, where all hosts have the same capabilities and no constraints are imposed on the placement. Hence, handling the placement problem in heterogeneous systems, where

it is often not possible to place operators on certain hosts due to constraints, is not considered in existing work. As a result, today's placement solutions are not applicable in scenarios with many constraints which is ubiquitous in industrial settings.

Consequently, we investigate how operator placement has to adapt in order to fit highly heterogeneous, industrial networks in Chapter 4.

2.2.3 Security in Event Processing

With the establishment of distributed CEP systems, CEP became not only of interest for individuals or single companies, but has found application in collaborative networks where different users, companies, or groups exchange events [HSB09]. Hence, event processing networks consist of different participants which may be spread across multiple security domains, resulting in the necessity to consider the security of the network participants and the data they provide [WCEW02].

Consequently, security of event data has been tackled by research in the context of Publish/Subscribe systems [TKAR10, PEB07, BESP08]. While, in a distributed Publish/Subscribe system, the publishers have no knowledge of the recipients of their events, and receivers might have no knowledge about the source of their events, some of the transmitted event data might be sensitive. And as a result, the system has to take care of the availability and visibility of this data.

In this context, a couple of distributed CEP systems have proposed mechanisms to secure the event dissemination, like Hermes [Pie04] or EventGuard [SL05]. Also, PADRES has the possibility to perform actions after processing events, which can be used for access restrictions [WJ07]. While these examples focus on broker-based Publish/Subscribe systems, there have also been proposals to ensure security in broker-less Publish/Subscribe systems, for example [TKAR10].

Security mechanisms typically use key servers, which manage authentication of network participants and their credentials. Their goal is to enforce confidentiality towards the events transmitted while guaranteeing scalability with the size of the network [TKAR10].

However, this is not sufficient for industrial, heterogeneous CEP systems, where information is processed over a long chain of different hosts. In todays security solutions for CEP, the provider of the original event information masks its information to have control about who can read the events. In other words, he restricts the access to any host which does not know how to decipher the information. However, the provider does not have any influence on what the recipients do, and how they process their legitimately received information. One can say the provider loses its influence on access control, once the information is received by its recipients. The access control is moved to the recipients, and they continue to mask the information they provide. In an extreme example, the original provider may restrict access to its confidential event information, which only its closes business partners are allowed to access and use. However, these partners are allowed to provide its own information freely to everyone, even if it is heavily dependent on the original data, maybe even sharing same attributes. This constitutes a security problem, since it enables recipients of the partners events to infer the original, confidential information.

This work investigates, how security management needs to be improved in order to fit highly heterogeneous, industrial networks in Chapter 5.

3 Enhancing Interoperability of Complex Event Processing

Although a significant amount of research has investigated the benefits of distributed CEP in terms of scalability and extensibility, there is an ongoing reluctance in deploying distributed CEP in an industrial context. We developed the DHEP (Distributed Heterogeneous Event Processing) system in cooperation with the IBM® laboratory in Böblingen, Germany. It is designed to address one of the key problems in increasing the acceptance of distributed CEP: supporting interoperability between heterogeneous event processing systems. In this chapter, the concepts behind the DHEP system are presented and it is shown how those concepts help to achieve scalable and extensible event processing in an environment where heterogeneous CEP technology co-exists.

Our event processing system DHEP focusses on providing interoperability and communication among heterogeneous, multi-domain CEP systems. In particular, we contribute the following:

- A *framework* that supports the integration of various established centralized CEP systems in a distributed environment. It takes care of all features necessary to build up a distributed processing system, like managing the communication between the hosts or distributing operators in the network.

- The concept of a *meta language* to be able to interact without some form of event translation. Therefore, DHEP comes along with a powerful modeling language that allows the design of distributed applications. The language enables the use of ontology to design and manage events, operators (i.e. operator descriptions) and context information within the distributed system. Hence, we are able

to adapt to a wide range of possible applications and can use this semantic knowledge to enable efficient complex event processing.

- A configuration tool for the application design. On the one hand, it is a graphical editor, where the user is modeling and managing the application. On the other hand, it can be used for managing the placement and deployment of operators.

Based on the DHEP framework, the algorithms for placement and security propagation are developed and evaluated. These algorithms are discussed in the subsequent chapters.

3.1 Motivation

In today's industrial applications distributed CEP is often not used to its full potential. Although many applications are inherently distributed, industrial event processing systems often rely on existing, centralized, mature complex event technology [AE04] instead of choosing a distributed approach. While the benefits of CEP are well understood and used in modern business processes, there are certain factors which impede an interconnection and distribution of the CEP applications. First, the deployed, centralized CEP technology is well-understood by system users and accepted as a reliable and efficient tool. Switching the technology as a whole is often not acceptable for industrial users. Second, interconnecting centralized CEP technology is not supported as for now. While business processes of heterogeneous entities are often strongly tied in today's industrial world, the non-homogeneous, non-standardized design of different traditional CEP technologies does not allow to seamlessly interconnect them. Finally, rather than targeting orthogonal attributes like expressiveness and user-friendly design which are essential for industrial use, distributed approaches tend to focus on efficiency in detecting situations, e.g. by reducing the detection latency to its maximum (cf. [BER08, PSB03, Jak03, TOK08, LGL08, LJ05]). Industrial event processing systems rely on centralized complex event technology [AE04] which is capable of providing the required functionality, at the expense of

distribution. Scalability is typically achieved by providing more powerful servers hosting the CEP System (cf. [BER08]).

This, however, is a disadvantage in scenarios where the applications requirements vary or are even completely unknown upfront. When the requirements, for example the event rates, increase over time, the deployed system's capabilities will eventually reach its limits. Also, many scenarios are inherently dispersed. We will exemplify this in the following section with a scenario in which we measure the current power consumption within a continental power grid (cf. smart grid). A centralized CEP system handling all the sensor data would not only cause a huge processing cost, but cause a major communication overhead.

In the future, it will be crucial to integrate universal, heterogeneous CEP technology which can adapt to user needs (cf. [SKPR09]). While many users currently rely on different CEP products (which specialize in some certain criteria of their need), these products are not capable to interact. The main problem is that they lack a fundamental basis to cooperate on, most importantly CEP is missing a standardization including a generally accepted event specification and a common language for the description of CEP operations. This is a major drawback for many applications, since interoperability and communication is an integral part in today's business world. A commonly used communication platform and standardized event descriptions would be a significant improvement to enable interoperability.

DHEP focusses on closing the gaps that prevent an efficient interoperation, such that business processes can benefit from a distributed processing of events. The framework supports the integration of various established centralized CEP systems in a distributed environment. It further makes use of a meta-language to describe events and operators. The meta-language serves as an abstraction layer of the integrated CEP systems' event desciptions, and allows them to communicate. This chapter will show that DHEP is a scalable complex event processing system, which enables interaction with business processes and context information.

The rest of this chapter is structured as follows. In Section 3.2 we present the challenges for a distributed heterogeneous CEP system on the basis of our driving efficient energy (smart grid) scenario. After presenting the

DHEP framework in Section 3.3 we will show some evaluation results in Section 3.3.2. Finally, we discuss related work in Section 3.6 before we conclude the chapter.

3.2 Challenges for Distributed Heterogeneous CEP

When analyzing today's usage of event processing in large business applications, it can be observed that almost none of the distributed CEP approaches proposed in the scientific literature has made it into industrial applications so far. Instead, a wide range of dedicated, centralized CEP systems are used in various application domains to perform event correlation tasks. Also, it is interesting that current development in industrial CEP technologies tend to favor investigating on highly efficient clustering mechanisms (cf. [BER08]) instead of distributing the process and moving the CEP functionality to dispersed locations.

There are a couple of reasons for this tendency. First, these 'distributed but centralized' approaches are customized to perform well when massive event loads can be processed in centralized data centers. Second, especially in business applications the CEP technology is strongly connected to business processes of the company. Thus, in order to perform the processing of the events, access to context information related to business processes is needed and such context information often resides in centralized databases. Therefore, the DHEP approach is driven by demands and requirements from distributed business processes where event processing can play an essential role when implementing the required business functionality.

We will describe our DHEP system throughout this chapter by means of a smart grid scenario established in a large area. In smart grid scenarios, as they are emerging nowadays, the efficiency of large power grids is enhanced by smart meters placed in households from where they send power consumption events to an energy agency. A simplified scenario is depicted in Figure 3.1. Here, smart meters in the consumers' households send the energy requests to substations owned by energy (transmission) providers. The substations keep track of the currently requested power

within their subnet. If the requested power exceeds the provided power within the subnet, the substation requests additional energy from the providers' mains. The provider itself may request additional power either from other providers or from energy producers. On the return path, the provider sends notifications to the smart meters about the available energy.

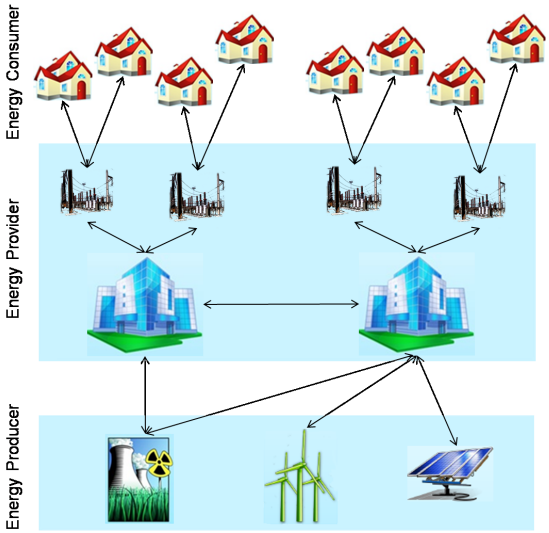


Figure 3.1: Simple E-Energy Scenario

The need for extensibility, the inherent distribution of energy consumers, along with the hierarchical structure of today's power management favors a distributed CEP solution which imposes the following specific challenges: (cf. [CIM09, McM07]).

- *Communication*: The power grid of almost every country contains

a large number of substations that need to interact according to the (simplified) power consumption forecast scenario described previously. Therefore, intelligent event emission avoiding unnecessary event traffic is a must when power consumption shall be forecast reliably in shorter time intervals like once per 15 minutes.
As can be seen in the example, different energy transmission providers, energy consumers, and energy producers have to interact in a network. Hence, communication between heterogeneous CEP environments has to be supported. [1] Events and operators have to be exchangeable between the different CEP systems running in the network.

- *Heterogeneity*: Heterogeneity has various aspects. For one, the substations within the power grid can have different complexity and tasks. While low-footprint event processing is sufficient in some places, major substations may require more powerful processing capabilities. Therefore, having different kinds of event processing operators, provided by different kinds of event processing technology, within the power grid can be beneficial. Second, todays power grids are shared among several providers which are closely connected, trading energy even among country borders. As a result, CEP networks are likely to exist across multiple domains, each using seperate description languages and CEP concepts.

- *Purposeful deployment*: Furthermore, the placement and deployment of operators is essential for a distributed CEP system and has special challenges in this kind of setting. For example, domain restrictions and engine constraints may preclude the placement of operators on some hosts. [2]

- *Context Modeling*: Context information like the structure of the power grid or the power consumption per household is important. Hence, a powerful language for modeling context information is required. In a distributed system, necessary access to context informa-

[1] The relationship between multiple business partners inevitably also rises the question of security. However, the security related challenges are excluded in this chapter, as they are elaborated extensively in Chapter 5

[2] The placement of operators will be elaborated extensively in Chapter 4

tion may be expensive from some places in the network. Therefore, it must be possible that events transport context information and that operators are able to deal with context information.

By providing solutions to these key aspects, our system is a first step to deliver the benefits of distributed event processing systems to industrial event processing solutions.

3.3 Approach

The main idea behind the DHEP system architecture is to enable distributed event processing without enforcing users to use a new correlation technology. Hence, our goal is to interconnect existing, typically centralized, engines within a network of correlation nodes (hosts) that is likely to be widely dispersed. As a consequence, we embed these centralized engines within a framework which takes care of all features necessary to build up a distributed processing system, like managing the communication between the hosts or distributing operators. Moreover, since heterogeneous engines are unable to interact without some form of event translation, we propose the concept of a meta language (cf. [SKPR09]). Therefore, DHEP comes along with a powerful modeling language that allows the design of distributed applications. The language enables the use of ontology to design and manage events, operators, and context information within the distributed system. Hence, we are able to adapt to a very wide range of possible applications and can use this semantic knowledge to enable efficient complex event processing. Finally, we provide a configuration tool for the application design. On the one hand, the configuration tool is a graphical editor, where the user can model and manage the application. On the other hand, it is responsible for managing the placement and deployment of operators. Figure 3.2 gives an overview of our system. We will discuss the meta language, the configuration tool and the runtime environment, in more detail in the following.

The DHEP system is very flexible. By integrating more hosts into the event processing network, the load of each host can be decreased, making the whole system more robust to a higher event input. By reducing the

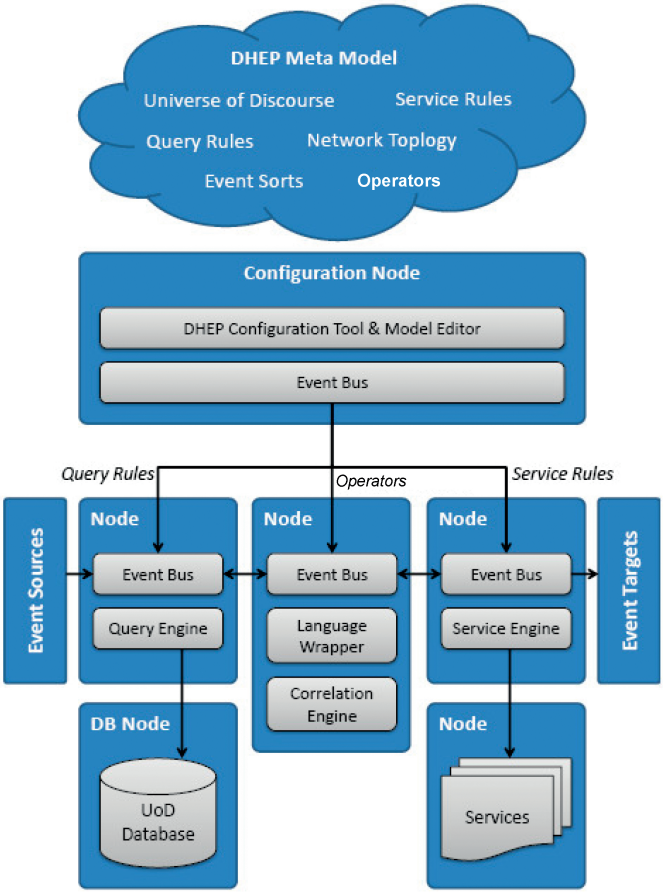


Figure 3.2: DHEP System Overview

number of hosts, network resources can be saved. Therefore, the CEP network configuration can be adapted depending on the desired overall system behavior. Additionally, latency can be improved by several means. First, the system can benefit from placing latency-relevant operators on hosts with low-latency network connection. Second, semantic knowledge about operators sets can be exploited: If operators are unrelated, they may be placed on different hosts within the network, thus reducing the overall detection time by using parallel computation.

3.3.1 The DHEP Meta Language

The DHEP system is based on a meta language for defining the elements of distributed event-based applications that can be implemented. The meta language is a necessity based on the requirement to enable interoperation of existing, mostly centralized, CEP technologies. It works as an abstraction layer towards the original description languages of the deployed engines and also contains elements relevant for the distribution of operators, like the expression of placement restrictions. The goal of the meta-language is not to be an all-embracing language which is able to cover every functionality of all existing description languages. Instead, the DHEP meta language is a modular, easily extendable language framework, such that the most common language concepts can be expressed. In particular this means that the meta-language will match a Snoop-expressiveness (cf. [AC06]).

The meta language is based upon the following elements:

- an object-oriented context-model defining the information model underlying an event processing application;
- event type definitions allowing to include complex objects in events being transmitted within the system;
- operators for standard event correlation, access to context model information, and invocation of web services;
- event processing nodes (hosts) within a network; each host can be equipped with different event processing engines;

- deployment descriptions that allow to express which operator shall be executed on which engine on which host in the network.

In order to illustrate the language concepts, the smart grid example from Section 3.2 is modeled in the following. This object model allows to express that power consumption can be measured at smart meters installed at a customer's house (cf. Listing 3.1). Customers have a contract with an energy agency which sell and bill consumed energy on behalf of energy providers owning the power plants.

The context modeling part of the DHEP language features typical object-oriented capabilities which are powerful enough to represent complex information models like the IEC 61970 standard for modeling power grids as published by the International Electrotechnical Commission (cf. [CIM09, McM07]). Standards like this gain more and more importance for the implementation of state-of-the-art industry applications.

object sort *NamedObject*
    attribute *name : String;*
    attribute *description : String;*

object sort *PowerMeter* $<$ *NamedObject*
    attribute *currentConsumption : Float;*
    attribute *accumulatedConsumption : Float;*
    attribute *parent : SubStation;*

object sort *SubStation* $<$ *NamedObject*

object sort *Customer* $<$ *NamedObject*
    attribute *myMeter : PowerMeter;*

object sort *EnergyAgency* $<$ *NamedObject*
    relation *myCustomers : Customer;*

object sort *EnergyProvider* $<$ *NamedObject*
    attribute *sellThrough : EnergyAgency;*

Listing 3.1: Example of Object Definitions

Some typical events that need to be handled include power consumption at a certain meter or power production information provided by a power plant operator (cf. Listing 3.2). Event Attributes can be both standard

attribute types like Integer or String and complex attribute types like defined DHEP object sorts, e.g., a *PowerMeter* defined in Listing 3.1.

event sort *PowerConsumption*
    field *meter : PowerMeter;*
    field *amount : Float;*

event sort *PowerForecast*
    field *meter : PowerMeter;*
    field *amount : Float;*

event sort *AggregatedPowerForecast*
    field *station : SubStation;*
    field *amount : Float;*

event sort *PowerProduction*
    field *provider : EnergyProvider;*
    field *amount : Float;*

Listing 3.2: Example of Event Definitions

Operators are the foundation of every situation detection. Their behavior is specified in operator descriptions. DHEP operator descriptions are identified by their name and consist of four parts:

1. The **event pattern** (*WHEN*) specifies the events used in the operator as well as their relation.
2. The correlation **condition** (*IF*) is used to access and verify certain attribute values of the incoming events.
3. The **restrictions** that have to be met when placing the operator (*RESTRICT*).
4. The **action** (*EMIT*), which describes the event that will be sent to the network once the situation is detected. Also, the user specifies how the values of the result event are set.

Not all of the DHEP language parts are relevant for the operator itself. For example, the restrictions only influence the placement of the operator, but not its functionality. Listing 3.3 shows the operator description of an aggregation of two smart meter values. In the example, we attach

the added *amount* values of the incoming *PowerForecast* events to the resulting *AggregatedPowerForecast* event.

Our operator language supports typical correlation operators like **SEQ**, **ALL**, **OR**, and **NOT** for detecting event patterns. Together with being able to define time constraints for the detection of event patterns, we are well positioned with respect to expressiveness of our operator language[3]. This holds true in particular as we also support sophisticated context model access and service invocations via dedicated operators (cf. Section 3.3.2).

OpDesc *AggregateConsumptionForecasts*
WHEN *SEQ(pcf1 : PowerForecast,*
pcf2 : PowerForecast)
IF *pcf1.meter* $\neq$ *pcf2.meter*
RESTRICT *engine.type = amit*
EMIT *AggregatedPowerForecast*
(station = pcf1.meter.parent
amount = pcf1.amount + pcf2.amount)

Listing 3.3: Example of a Simple Aggregation Operator

As can be seen in the example, deployment restrictions can be formulated (cf. Listing 3.3). These restrictions influence where an operator can be placed. We currently support two such restriction types. First, we support binary restrictions. These are conditions which only have two possible values (true, false), and are common in heterogeneous systems like tackled in our work. A typical example is the definition of a operator type classification expressing on what type of engine the operator can be processed. Another example is the definition of a domain restriction expressing the requirement of an operator to be processed on a host within a certain domain. Upon deployment this will be used to assure that the operator is only deployed to an engine and host capable of processing operators of the respective type. Second, the operator can be annotated with restrictions on the resources of the host it should be deployed on. For example, attributes like required CPU speed, memory size, or network

[3] Adding more sophisticated correlation operators is no big effort and not so essential to our approach.

connection bandwidth. More detailed information about the operator restricions can be found later in this thesis while discussing the placement (cf. Chapter 4).

Finally, the event processing network can be defined by a set of hosts (see Listing 3.4), each of which can be equipped with multiple event processing engines. Furthermore, the user has the possibility to influence the placement by manually deploying operators on specific hosts. However, this is optional as we also provide an automatic deployment mechanism (cf. Section 3.3.3).

host *a correlation host*
    correlation engine *AMiT;*
host *a context model server*
    query engine *queryEngine;*
      deployed operator *R1;*
    service invocation engine *serviceEngine;*

Listing 3.4: Example of a Node Model

3.3.2 The DHEP Runtime Environment

Every host in our network is running the DHEP runtime environment (RE) which is basically a middleware that encapsulates the event processing technology. It abstracts the network functionality and performs all tasks that are necessary to enable a distributed heterogeneous event processing system. Figure 3.3 shows the most important components of the RE. The dashed path marks the typical event flow we have to handle in the RE. In the following subsections, we describe the components as they are used in the event flow. Also, we discuss the functionality of the Operator Management and Context Enrichment components.

Event Bus

The event bus constitutes the foundation of our system, as it contains all low-level elements which are needed to distribute the data (events) in our system. Since heterogeneous systems do also include heterogeneous

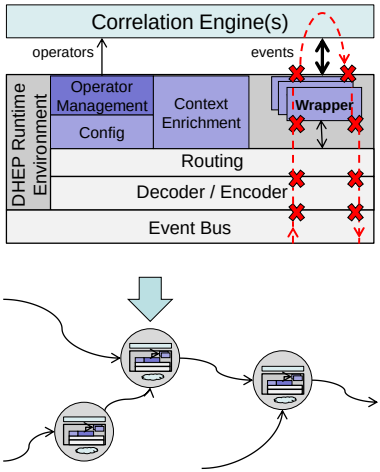


Figure 3.3: Eventflow in the Runtime Environment

communication methods, the communication interface of the runtime environment has to be designed in a modular fashion. Thus, all communication modules implement the same common interface, which provides the *getMessage* method to receive events.

Modules share the same interfaces to make them exchangeable. This enables the integration of multiple heterogeneous message systems and communication principles, as depicted in Figure 3.4. Typically, DHEP makes use of a publish/subscribe system, which is connected to the event bus machanism. For testing purposes in the remainder of this chapter, we will use a socket-based communication module.

The socket module is based on the reactor design pattern (also known as dispatcher/notifier, cf. [Sch94]) and uses non-blocking sockets. The idea behind the pattern is to demultiplex and dispatch each incoming request to its corresponding service before invoking the service. A *Selector* acts as a demultiplexer to determine which connections are ready to have operations invoked on them, without blocking the application process.

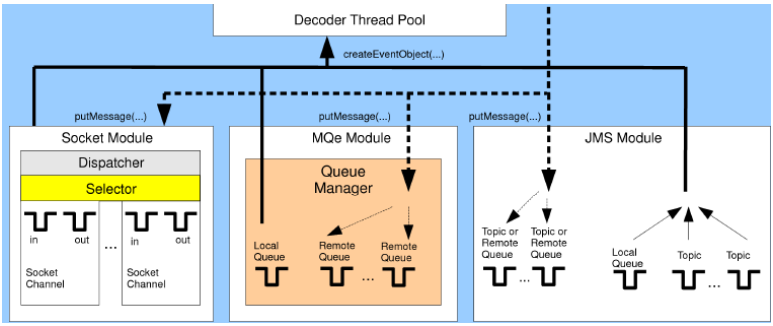


Figure 3.4: Event Bus & Decoder

Decoding and internal Routing

As typical for a model and ontology based system, DHEP has to deserialize every incoming event and match it against our description model, in order to encapsulate the event within an event object. This happens within the *Decoder* (see Figure 3.4). To simplify this process and reduce programming overhead in this step, we make use of Dynamic EMF (Eclipse Modelling Framework [SBPM08]). We store our events in a Dynamic EMF format and use deserialization methods provided by EMF. Accordingly, the encoder acts vice versa as a serializer which creates event messages from DHEP objects. We use a thread pool within the Decoder component to be able to decode events in parallel.

A routing component receives incoming event objects from the decoder and forwards them internally (cf. Figure 3.3). A routing table contains the information about the local engines which are meant to process each event at the host. Hence, the event bus distributes every incoming event accordingly, sending it towards one (or more) of the local engines or other hosts in the network. The values of this table are given by the RESTRICT section in operator descriptions for ingoing events and influenced by the placement algorithm (by means of configuration events, cf. *Event Wrapper*).

Event Wrapper

The framework architecture has the ability to integrate different existing CEP engines even within the same network. The wrapper component is responsible for the integration of the different engines. It acts primarily as the adapter between the routing component and an event processing unit, i.e. correlation engine, on the host. A wrapper is required for every different engine, because its main task is to translate event messages from the meta language into the language of the target engine. Due to a modular design, a wrapper is in fact an adapter not only to engines but to any unit that can process and act with events. A new wrapper just has to implement the provided wrapper interface.

We use this concept not only for the integration of various correlation engines but also for context enrichment as well as the configuration of our system. For context enrichment, we attach a query engine, which reacts on user-defined query rules to enrich event data with related information. For configuration, we implemented a configuration wrapper, that receives all system relevant information via configuration events which are distributed via the event bus. The configuration events contain for example routing information and operator placements. The configuration wrapper takes care of the proper settings of the routing table and moves operators to their correct target engines (according to the placement algorithm).

Besides context enrichment and configuration, we use the wrapper architecture to access a service engine which can invoke web services.

Operator Management

The operator management is responsible for the distribution, migration, and deployment of operators in our system. It is strongly coupled with the configuration events, since it uses these events to manage the deployment of operators. The component offers interfaces to move operators to other hosts as well as to deploy them on a local event processing engine which is connected via one of the wrappers. The deployment process consists of sophisticated re-organization and optimization mechanisms performed in a distributed manner. It also tackles the constraints and optimization criteria of our heterogeneous environments. The operator placement for heterogeneous systems will be elaborated in detail in Chapter 4.

Similar to the event translation done by the wrappers, the deployment of operators requires a translation process, because operator descriptions defined with the DHEP meta language are not understood by the various CEP engines. As a result, translators are attached to the *Operator Management*. Their input is the DHEP operator description and their output is the equivalent operator defined in the description language of the respective target engine. While we provide the interface specifications, the implementation of the translators has to be done by system administrators, i.e. DHEP users. It is worth mentioning that not all parts of DHEP operator descriptions are relevant to the operators itself and need to be

translated. Especially the specified *restrictions* are used by the placement algorithm and not the operator.

Event Context Enrichment

In distributed scenarios, access to context information is not always available or might be very expensive. However, it is often needed during the event processing. Especially in business applications, context information is important since many business decisions can not be made solely by information contained in the events, but require additional meta information applying to the events. For example, events from a smart meter do contain little information, typically an identifier of the customer as well as some value representing the requested energy amount. However, this information is most likely not sufficient for the energy provider which requires additional data about the users contract. This information is typically located at the data center, where all business processes are running. It needs to be added to either events or the event processing system in order to use it within the complex event processing, hence allowing to make correct decisions.

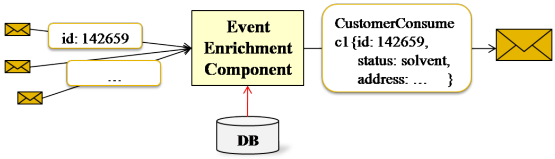


Figure 3.5: Enrichment of Smart Meter Events

In contrast to centralized approaches, distributed CEP systems did not address the access to external information so far. To be able to use external sources for event information, context enrichment and transport

becomes necessary. The DHEP system is providing this feature. For accessing context information in our system, we make use of the modular, heterogeneous architecture by providing a *query engine*, which is capable of enriching events with context retrieved from external sources, such as a database. To configure the query engines we are able to define *query rules* which are also covered by the DHEP language. The transport of context is ensured since all specified objects can be serialized and deserialized within the encoder/decoder.

To allow a configurable, independent context enrichment with the query engine, a three tier architecture is chosen (see Figure 3.6). The architecture consists of the query engine, an information access layer and the database containing the actual context data.

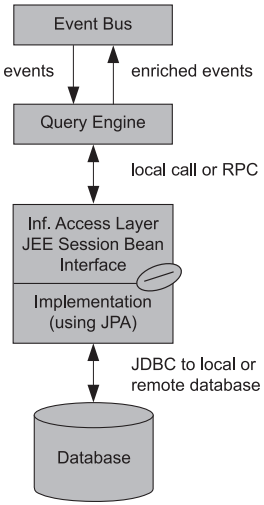


Figure 3.6: Three-Tier Architecture to access Context Information

The reason behind this partitioning is to keep the actual data access independent from the query engine, where the events are processed according to the deployed query rules. Therefore, the information access layer is designed as a separate component with a standardized interface which is called by the query engine. This enables the exchange of the data access implementation easily without touching the query engine code. Furthermore, this makes it easy to also use other information systems than relational databases. For example, object-oriented databases, XML files, and web services can be used for the retrieval of context information.

Finally, by not accessing the database directly, context information can be cached in the access layer. This can reduce the expected cost produced by the database access by a large margin.

3.3.3 Configuration Tool

The configuration tool is the meeting point of the DHEP meta language and the actual CEP network. With the configuration tool the user models, deploys and interacts with the application. From the users perspective, the tool is a graphical modeling editor (see Figure 3.7). Here, users can define operator descriptions or create wrappers for event translations. Furthermore, the tool initiates the deployment mechanisms. During this process, the configuration tool communicates with the operator management component of the framework nodes, using configuration events (see Section 3.3.2).

3.4 Benchmarking

With the Configuration Tool, the DHEP framework is, architecture-wise, capable of integrating a variety of centralized placement and optimization algorithms. Then, the placement is computed in the Configuration Tool and the deployment is handled with configuration events. In scenarios where no central configuration tool is available, the operator management component described in Section 3.3.2 provides interfaces to integrate distributed placement algorithms. In the upcoming Chapter 4, we discuss

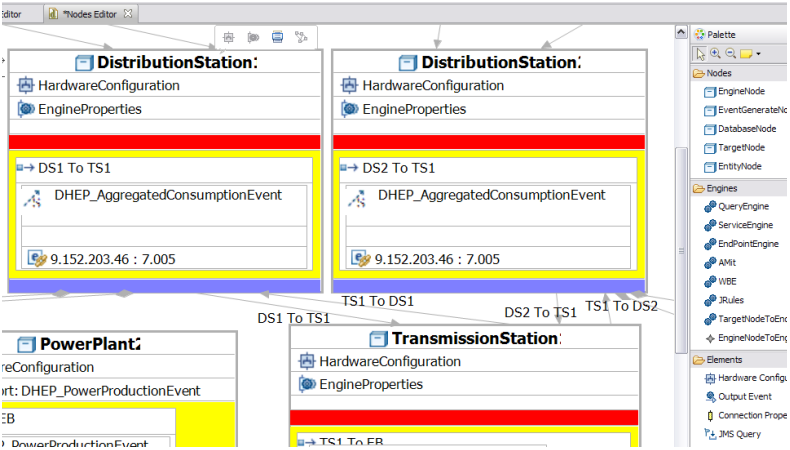


Figure 3.7: Excerpt of the Graphical Editor

and present a distributed placement algorithm designed specifically for heterogeneous environments, which can be efficiently used and deployed within the DHEP framework. Before we can elaborate about efficient placement algorithm in the next chapter, we will benchmark our framework itself.

While DHEP has no influence on the usage of network resources, compared to a homogeneous CEP system, it does have an influence on the processing costs, as the processing within the framework causes CPU load. Furthermore, the CPU load caused by operators in heterogeneous systems can vary dependent on the hosts. This is caused by different engines, which can cause different processing costs while deploying the same operator. Therefore, we measure the CPU load caused by the framework in order to integrate this knowledge into the operator placement algorithm, discussed in Chapter 4.

In our system, every deployed operator produces a CPU load which is based on the following parameters: the operator itself (when processing an event), the used CEP engine, the event rate, and the composition of

the event stream. The latter is important, as depending on the incoming events, the operator might trigger a situation, i.e., a result event, or not. Figure 3.3 shows a typical event flow within our system. As can be seen, 7 components are passed, after an event has been received, each contributing to the resource usage: i) Decoder, ii) Routing, iii) Translation, iv) CEP, v) (Back-)Translation, vi) Routing, vii) Encoder. Only component iv) is engine-dependent.

Hence we can calculate the total resource usage of every event for operator e_ω on host n as the sum of the resource consumption of every used component c_i. We use the required processing time of each component as the *cost* in our function. This also covers the host configuration like CPU speed, as it is directly related to the processing time.

$$C(e_\omega, n) = \sum cost(c_i) \tag{3.1}$$

Equation 3.1 is in fact a pessimistic calculation, as the components v) to vii) are only used if the CEP engine produces a result, and therefore require no resource consumption in many cases where no event is produced. However, with this calculation we can give a pessimistic forecast of how many events can be handled by a host.

$$C_{max}(\omega, n) = \lambda * C(e_\omega, n) \tag{3.2}$$

Since Equation 3.1 determines the required CPU time of one event, we can calculate an operator's maximum resource consumption C_{max} (Equation 3.2) for a given event rate λ and give a lower bound for the maximum event throughput λ_{max} (Equation 3.3):

$$\lambda_{max} = \frac{1}{C(e_\omega, n)} \tag{3.3}$$

3.4.1 Correlation Cost Analysis

To be able to fulfill placement conditions and make useful cost approximations for operator placements, knowledge about the resource requirements of operators is mandatory. In DHEP, two concept elements produce

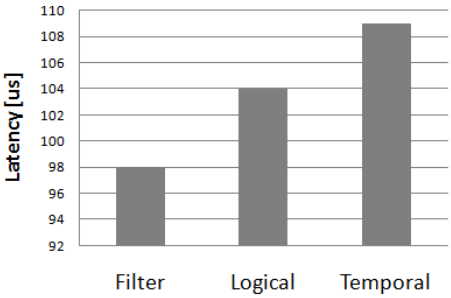


Figure 3.8: Cost vs. Operator Types

(CPU) cost: the framework (i.e., the runtime environment), and the correlation engines. The framework cost is not dependent on the operator itself, but on the event rate coming along with the operator, we will tackle this in the evaluation of our framework (Section 3.5). However, this is not true for the correlation cost. Here, operator types and operator characteristics have a major influence on the cost.

To understand the engines behavior and resource consumption, benchmarks on the processing cost are necessary which are collected exemplary for the IBM AMiT^{TM}engine (see Figure 3.8 and following). For the benchmarks, the processing time (latency) between the input of an event to the engine's interface and the eventual output of a result event was measured.

Figure 3.8 shows the latency of different basic operator types: *Filter operators*, *logical operators*, and *temporal operators* (like a sequence). The results show that the cost increases with the operator's complexity. As can be seen in figure 3.8, the cost increases about 20% from 100 microseconds for the most simple filter operator to about 120 microseconds for the most complex operator, the sequence detection.

The second benchmark shows the impact of specific event streams on AMiT's processing time of operators (see Figure 3.9). This is affecting sequential operators, as they are producing new instances. For example,

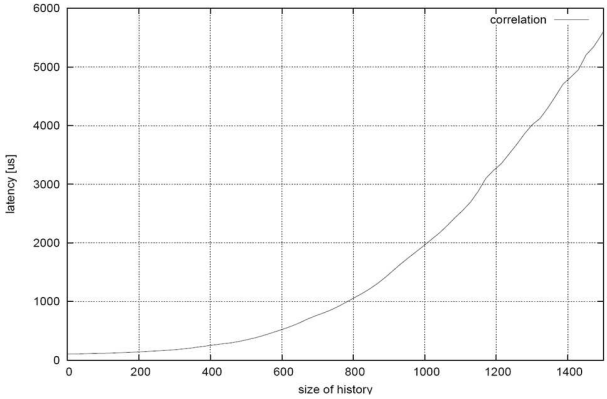


Figure 3.9: Cost vs. Size of History

an event stream A^nB produces an instance for every incoming event A, while waiting for the matching event B that triggers the rule pattern $A;B \rightarrow C$. The benchmark shows that the increase is approximately quadratic. The reason is that every event has to the go through all n previously created instances until a new instance is created. This effect causes quadratic runtime for the event stream A^nB.

Figure 3.10 shows the impact of event attributes. As can be seen, the operator's latency increases logarithmic with the number of accessed event attributes. While the increase of processing cost is apparent at first, about 20% from zero to four attributes, the increase of required processing time is pretty minimal after an event is checked for 4 attributes.

Finally, Figure 3.11 shows the impact of multiple operators that are affected by an event. As can be seen, the latency increases linearly, which shows that the AMiT engine processes every operator after the other. However, the cost increase is not an accumulation of the individual operator costs, but the engines scheduling mechanism creates an overhead for every additional operator.

As a result from the evaluations, we can approximate the correlation cost

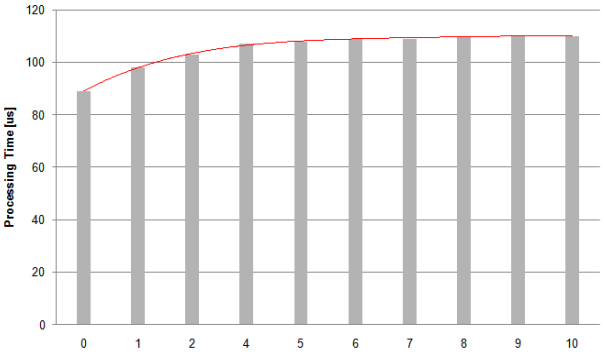


Figure 3.10: Cost vs. number of Attributes

of different operators for different engines. This information will be gathered for all available engines and will further be used for more sophisticated placement algorithms, as will be discussed in the remainder of this thesis.

3.5 Evaluation

The evaluation of the DHEP framework was driven by two guiding questions:

1. What is the framework performance characteristic?

2. When is the framework applicable?

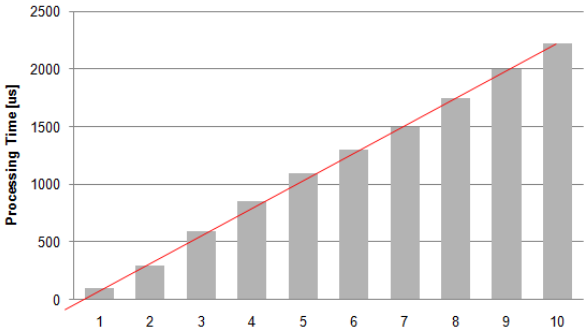


Figure 3.11: Cost vs. number of Operators

3.5.1 Framework Cost Analysis

In order to understand our system's performance characteristic, as well as calculate the benefits of a distributed deployment, we first evaluated how the different components that are used in the standard event flow contribute to the total processing cost of events. The event processing stages within the framework were described in Section 3.2 and shown in Figure 3.3. In the processing stages, each event is typically processed in 7 intervals, after it has been received:

1. Decoder
2. Routing
3. Translation
4. Complex Event Processing
5. (Back-)Translation
6. Routing
7. Encoder

As can be seen, the stages 1-3 correspond the incoming events, while stages 5-7 correspond to outgoing events. Furthermore, stage 4 is independent of our framework and has to be measured seperately. An exemplary benchmark for one engine is presented in Section 3.4.1.

We deployed a small evaluation network: an event source, the processing host, and an event sink. The processing host is a Windows PC running a single core CPU@2.0GHz. IBM's AMiT engine was connected via a wrapper to the DHEP Runtime Environment. The event source was simulating a smart meter sending *powerrequest* events continuously. In order to receive consistent, repeatable results, only one filter operator was deployed on the engine, filtering for an id, and thus always evaluating to true for every event sent.

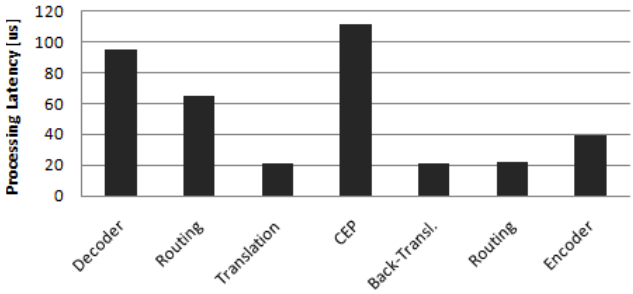


Figure 3.12: Processing Cost per Component

For the evaluation, we had a very low event rate of 1 event/second to ensure that there is only one event within the framework at every time. Figure 3.12 depicts the results we gained.

Several results are remarkable: First, the correlation (i.e., the filtering) has the highest processing cost. Second, the wrapper concept proves to be reasonable since translation of events is pretty fast compared to other tasks. Third, deserialization results in a considerable amount of processing

cost. In fact, the cost for retrieving and generating a DHEP object for an incoming event is almost as high as the filtering process of the engine. This is caused by the serialization methods of EMF, which seem to leave space for optimization.

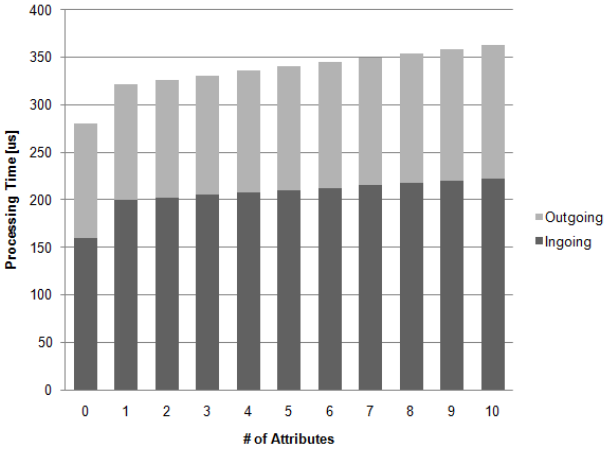


Figure 3.13: Framework Cost vs. Event Size

In a second evaluation, we varied the events produced from our event source in their size. We did this by adding additional attribute value pairs in the same settings. The result is shown in Figure 3.13. One can see that with an increasing size of the events, here the number of event attributes, the framework cost increases too. The reason for this is the additional overhead that is caused during the (de)serialization and translation of the event within the wrapper component. However, it can be seen that the cost increase is rather small and linear: About 1.5% increase for each additional attribute of the incoming and the outgoing events. The only exception is the low cost of an attribute-less event, which is based on the

fact that both (de)serialization and translation do not access the attribute list of the event at all. However, an event without any attribute is rather unusual in complex event processing scenarios. Therefore this effect only has a minimum influence.

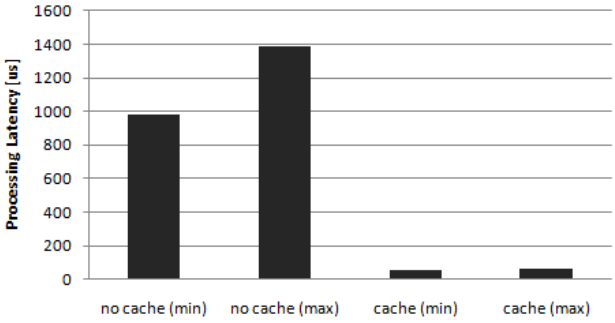


Figure 3.14: Event Context Enrichment Cost

3.5.2 Event Enrichment by the Query Engine

In our next evaluation we tackled the access of context and enrichment of events with the retrieved information. This is done with the query engine we briefly introduced in section 3.3.2. For the measurements, we used our basic network setup, and added a MySQL database to the processing host. We then created an operator that should retrieve some integer value from the database and add it to the input event. The results of our processing cost measurements are depicted in Figure 3.14. Context access produces a massive overhead to the system. The costs for accessing a database and querying for a value and adding the value to an event are about ten times higher than filtering for an attribute.

The expense of event enrichment can be reduced by enabling the caching mechanism as it is described in Section 3.2. As can be seen in the results, this can be a great benefit for applications, where the same attribute is retrieved for a large number of events. Furthermore, when caching comes into consideration, other factors like cache size and freshness of data should have a major influence on the cache behavior. Also, a type specific caching can further improve the caching mechanisms. However, it has to be noticed that these parameters are mostly application dependent and have to be set by application developers accordingly.

3.5.3 Scalability due to Distribution

Based on the previous evaluations (cf. Section 3.4.1 and 3.5.1), we are able to understand the processing latency within our system. As has been shown, the systems overall processing cost is caused by both the framework and the used CEP engine. Furthermore, we can state, that the framework costs per event are constant, whereas the cost for processing an event in a CEP engine are dependent on the used engine and the deployed operator set. In our last evaluations, we want to show the scalability of DHEP: by adding hosts to the network we are able to process an increasing event load.

We tested our framework with a simple deployment of our e-energy scenario which was introduced in Section 3.2: Smart meters calculate the current energy consumption of a household and send periodically consumption events to the energy provider. The energy provider first checks, whether the power consumption exceeds a predefined threshold. This is done with a simple *filter* operator ω_f. Afterwards, the exceeding consumption events are aggregated with ω_a (cf. operator definition in Section 3.3.1). The scenario's operator graph is depicted in Figure 3.15.

We chose this scenario based on four considerations: i) The deployed operators are the most basic operators one can get for a CEP engine. Therefore, we expect the correlation process to be very fast. This favors the CEP engine, and stresses our framework, because a high event throughput can be reached by the correlation process. ii) By setting the filter values we can adapt the event rates that are used within aggregation

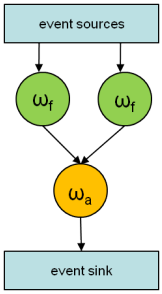


Figure 3.15: A simple operator graph with filtering and aggregation

operator. iii) With a low number of operators, the centralized deployment is favored, because there is not much space for splitting and distributing the operator set. iv) The setting reflects a typical distributed CEP application, which is often composed of operators processing frequent basic events close to the event sources, and more complex operators which process their results.

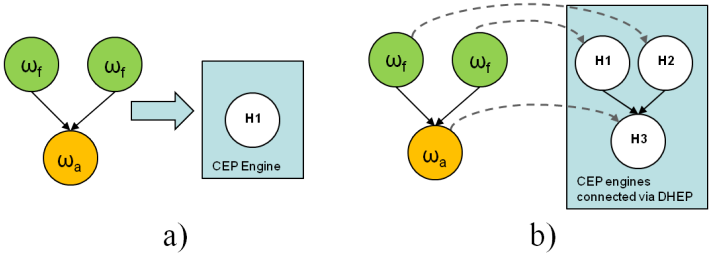


Figure 3.16: The two different deployment scenarios

To test the effects of operator distribution within DHEP, we deployed the set of operators (cf. Figure 3.16) on one centralized CEP engine as

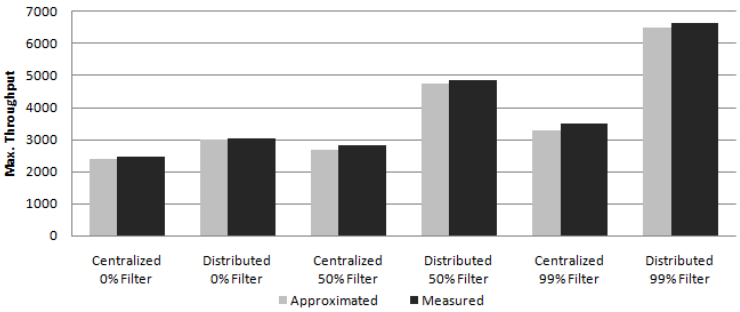


Figure 3.17: Maximum Event Throughput

well as three hosts running DHEP. While in the first, centralized case the operator results were directly processed further within the engine, in the second, distributed case DHEP overtakes the responsibility to connect the CEP engines. The results of the evaluation are shown in Figure 3.17. As can be seen, we deployed the scenario with three different filter settings, where we changed the restrictiveness of the filters. Moreover, we used the evaluation to verify our approximation functions presented in Section 3.4 (cf. the lighter bars in Figure 3.17). We approximated the values for the various component costs from our evaluation results presented before.

The results show three main conclusions: First, the more restrictive the filters are, the more events can be processed in both settings. This is self-evident, as less events are reaching the aggregation operator, hence producing less cost. Second, our lower bound approximations are pretty close to the actual values we got from our evaluations. Third, we benefit from a distributed deployment of operators although we deployed a scenario that is likely to favor a centralized one. Furthermore, the benefit increases, as the filters get more restrictive. The aggregation operator processes events much slower due to a more complex functionality, and consequently is the bottleneck in a deployment where no events are filtered out (0% Filter). In more realistic scenarios, where the events are filtered out, the aggregation receives less events. Hence, the benefit in distributing operators increases.

As can be seen, although our system produces additional costs, the advantage in distributing operators over multiple hosts is measurable even in very small scenarios. The advantage gets even more obvious, when bigger and more realistic business scenarios are deployed or when additional sources have to be processed by the CEP system. The increase of CEP costs might exceed the available processing power and network bandwidth of centralized CEP systems. With our framework, we can easily extend our setup by adding additional processing hosts to reduce both processing cost and network bandwidth usage of the CEP system.

3.6 Related Work

Since we combine in DHEP the expressiveness and power of mature CEP systems running on large servers, typically used nowadays, with the ideas of distributed CEP systems, a look in both directions is necessary.

While initial research focused on expressive languages and efficient centralized situation detection mechanisms (cf. [AE04, CM94, GD92]), academia has shifted its main focus on distributed CEP. Multiple approaches have been designed recently (cf. [Jak03, LJ05, LGL08, PSB03, KKR10]). Especially with the increasing popularity of Publish/Subscribe systems (cf. [TKKR10, BKR09]), event processing has become a highly distributed technology. CEP scales well in these systems, since it can benefit from the Publish/Subscribe paradigm by subscribing to specific events of interest, which can be extended to support complex events. A CEP engine attached to the Publish/Subscribe system takes care of the detection of complex events. Furthermore, the possibility to detect complex events on any node in the system opens a lot of optimization possibility, such as moving functionality closer to the source in order to reduce the overall network usage.

While we can learn from these approaches in terms of efficient network usage and operator migration, they lack the special industrial requirements we have defined in Section 3.2. They consider homogeneous engines for their migration and deployment techniques. Furthermore, they do not integrate context information of applications. Finally, distribution

techniques do not seem to be mature enough to be deployed in business applications. They take only static node resources into account, but ignore important aspects for business application, such as dynamic load, additional restrictions for the processing of operators, security aspects, and the proximity to context data sources that are necessary for event processing.

Attempts to interconnect heterogeneous components within a common distributed platform can be found in the context of service oriented architectures in form of the SCA or JBI specifications (cf. Service Component Architecture [BBB+05], Java Business Integration [THW05]). Similar as in DHEP, different services can interact and communicate with each other based on a common communication bus. Adapters are used to translate between the internal languages of the services and the communication language. However, the focus in these systems is to interconnect existing functionality encapsulated in services instead of distributing application parts among various configurable nodes.

First steps towards increasing the performance of CEP systems by combining heterogeneous correlation technology has been made by Chakravarthy et al. [CA08]. They manually combine a classical event processing engine and a stream processing engine in order to achieve an efficient situation detection.

Most closely to our goal in maintaining the scalability of mature correlation technology is the work of Biger et al. [BER08]. They integrate several centralized CEP systems into a distributed correlation network without modifying the system itself. In addition to connecting input and output of individual powerful correlation nodes and deciding which correlation tasks to perform at which node, the configuration and deployment of each correlation machine becomes a main issue. However, the approach has several characteristics contradicting our goal. First, the network setup as well as the CEP configuration has to be tailored towards the target application beforehand. Second, the intention of stratification is to achieve a maximum throughput by splitting a specific set of operators and pipelining it through multiples stratas. It is not intended for inherently distributed applications like the e-energy scenario.

3.7 Conclusion

In this chapter we presented the complex event processing system DHEP, which has been designed and created to close the gap between current CEP systems and business requirements. The concepts behind DHEP focus on providing a very modular architecture, that comprises various different event processing engines, and enables communication among them within a distributed system. In addition, DHEP comes along with a powerful object-oriented definition language, that enables efficient, tool-aided designing of big industrial CEP applications. The evaluation results show that, although the functionality provided by DHEP imposes additional cost, the system scales well by exploiting distributed detection of situations.

Future work in this research area might extend the evaluations with larger scenarios, which will provide knowledge about the weaknesses of the approach in large scale, long running application. To be able to efficiently use the system, we will provide and practically evaluate a variety of placement algorithms in the upcoming chapter.

4 Efficient Operator Placement in Constraint-Driven Environments

One of the core challenge towards scalable complex event processing is to efficiently distribute the event correlating operators within an event processing network. Although significant research has been made recently to allow an efficient operator placement, there remains a gap in supporting requirements that emerge from deploying CEP over heterogeneous and independent processing environments. Heterogeneity imposes strict *binary* constraints on the placement of operators, like the availability of a certain processing engine or the membership in a certain domain. This adds to the complexity of the underlying optimization problem and cannot be handled efficiently by existing solutions.

In the previous chapter, we presented the DHEP framework with its different elements, including a configuration tool which can be used to centrally calculate placements and deploy the operator set via configuration events. In distributed systems, maintaining central knowledge on a configuration tool is not always feasible. Consequently, our goal in this chapter is to present a distributed placement algorithm, which can find and optimize placement solutions locally on the nodes. Previously, we benchmarked the DHEP framework and learned how its various components behave under heavy load. This forms the foundation towards evaluating algorithms which handle the placement in a distributed network of DHEP hosts.

Based on the measurements presented so far, we examine the distributed placement, migration, and optimization of operators in heterogeneous environments. This is always done in the context of the optimization goal to minimize network usage as a commonly used measure for efficiency. We propose and evaluate a placement algorithm that efficiently finds valid solutions in scenarios where the solution space is heavily restricted by constraints. The algorithm operates in a decentralized way and is adaptive

to dynamic changes of processing hosts, operators, and load characteristics of the event processing network. The proposed operator migration policies resolve invalid placements and therefore counteract node failures. The evaluations show that the proposed algorithm is able to find efficient solutions within constraint-driven conditions that also include binary constraints.

In this chapter, we present a distributed solution to minimize the network usage in distributed CEP networks. In particular the contributions are

- a *placement algorithm*, which is capable of valid placements within constraint driven settings.
- an *optimization algorithm* which is capable of finding near optimal placements in terms of minizing the network usage within heavy constraint driven settings.

The presented approaches are implemented and evaluated. We discuss their applicability in detail.

4.1 Introduction

In distributed CEP the detection of a situation is typically performed by multiple cooperative operators which themselves are deployed on hosts within the CEP network. The distribution of event correlation enables scalable applications and can also result in a more efficient CEP. For example, by moving the correlation functionality closer to the event sources, the network usage of detecting situations may be improved. Furthermore, by distributing the same functionality among several hosts, CEP applications become more available and reliable.

The distribution of multiple operators within the *network of correlation hosts* (i.e., the *CEP system*) raises the question of placement. Depending on where the different operators receive, process, and produce events, the behavior of the whole CEP system will change in terms of detection time, resource consumption, and reliability. The challenge is to find an operator placement, such that CEP is efficient with respect to the applications optimization goals, such as network usage.

The placement problem in event processing systems has been tackled by many researchers recently (cf. [Fid06, LJ05, PSB03]). So far, distributed CEP is characterized by relying on a homogeneous system, where all hosts have the same capabilities and no binary constraints are imposed on the placement. In this context, a binary constraint is a capability need which must be matched by the host to be allowed to deploy an operator, like the membership in a certain security domain. Hence, handling the placement problem in heterogeneous systems, where it is often not possible to place operators on certain hosts due to constraints, is not considered in existing work. As we will discuss in the remainder of this chapter, today's placement solutions are not optimal in scenarios with many binary constraints as we face them with our DHEP framework.

Certainly, such constraints are of high relevance for real world deployments as we extensively discussed in Chapter 3. We face heterogeneous CEP systems hosted with differing configurations at various domains (cf. [SKPR09]). Throughout this Chapter, we will illustrate the placement problem by means of an energy & utility scenario. Consider a large power grid, where several participants (e.g., energy producers, brokers,

and customers), further referred to as *domains*, are producing, trading, or purchasing/consuming energy. Hence, a network of hosts is established. Due to the presence of several participants, the hosts are situated in various domains and may embrace heterogeneous CEP technology. In this scenario, a set of CEP operators is defined to detect energy load violations, regulate energy consumption, or steer the energy flow.

As a consequence of the heterogeneity, the placement of operators is often restricted to a small subset of hosts, that have the corresponding capabilities and permissions. Moreover, CEP systems require a high level of decentralization since a host that gains global knowledge might be a security and scalability problem in heterogeneous networks. Furthermore, the placement has to be adaptive, due to the dynamic behavior of CEP applications that stems from load variations, changes in the availability of correlation hosts, as well as to changes in the operators deployment. For example, energy consumption in our example scenario is likely to peak at certain times during a day, resulting in many concurrent energy shortage messages that have to be dealt with.

The rest of this Chapter is structured as follows. In Section 4.2, we detail the challenges of placement in heterogeneous CEP systems by means of the energy & utility example before presenting the system model and formal problem statement. We present our approach in Section 4.3, discuss related work in Section 4.4, and evaluate our approach in Section 4.5. Finally, we will have an outlook on possible extensions and conclusion in Section 4.6.

4.2 Problem Description

In this section, we first motivate the problem and present the challenges. After that, we describe the system model underlying our problem. Finally, we will formally define the problem.

4.2.1 Challenges

Constraints and Resources

The high number of collaboration among business partners in today's world results in a cooperative nature of the involved companies business processes. As an effect, the processing and execution of business events is performed in heterogeneous environments, where hosts have different characteristics, resources, domains and processing capabilities. This imposes two main challenges that need to be tackled by placement solutions: a constraint-based operator placement as well as the inclusion of resource usage. We exemplary show this with our energy and utilities scenario which was introduced in Chapter 3. In the scenario, the event processing is typically organized hierarchically among several domains, where information is processed and reacted on with the help of correlation engines. Smart meters act as event sources and processing hosts at the same time: they emit events about current consumption on the one hand but also steer energy consumption within the household on the other hand (e.g., based on energy prices). A functional overview of the scenario is given in Figure 4.1).

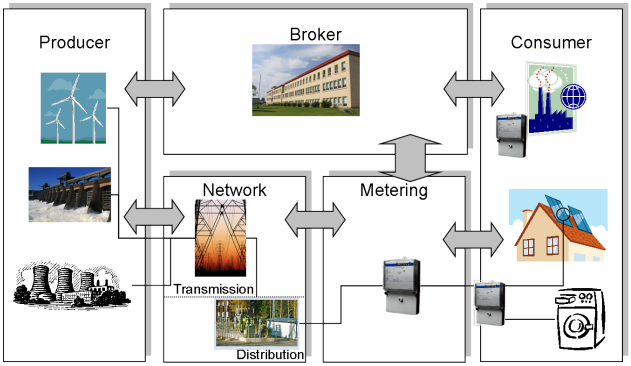


Figure 4.1: Participants in an energy network

It can be seen, that different situations are detected at various domains (e.g., Broker, Consumer) within a very large system. Thus, event correlation is favored to be deployed in a distributed way. However, multiple binary constraints increase the complexity of the placement process:

a) *Heterogeneous Engines*: The functionality needed at different levels within the power grid is of different complexity. While low-footprint event processing is sufficient in some places (e.g., filtering at power meters), major domains like the energy broker require more expressive processing capabilities. Therefore, different kinds of processing engines are installed within the domains of a large power grid. As a result, operators are not able to be processed at every host.
b) *Domain Restrictions*: Due to many participants in a large-scale power grid (e.g., different energy providers), security becomes a major issue (e.g., confidentiality). Domain restrictions can be specified in the operator descriptions and enforce a placement prohibition for operators on hosts outside the domain.
c) *Heterogeneous Resources*: The hosts within the network have differing resources. While energy providers are likely to possess large-scaled dedicated servers with high-end computing capabilities, small power meter units at the customer are equipped with less memory and processing power.

An algorithm which aims towards finding viable placement solutions for a heterogeneous network must be able to deal with these restrictions. This makes most placement algorithms used in CEP systems inappropriate, as they typically try to find the best placement (concerning a specific optimization goal) in the whole search space (cf. [PSB03, Fid06, RDR10]), e.g., by introducing a latency space. However, applications imposing binary constraints typically require the search space to be pruned to a subset of allowed placements. Consequently, a placement algorithm can only find valid solutions in a potentially small subset of the available hosts, depending on the given operator restrictions. Furthermore, since the solution space contains binary constraints it does not converge towards the valid solutions. This makes the available placement algorithms not applicable, as they rely on the converging solution spaces to find valid solutions (cf. [RDR11]).

Moreover, most current placement algorithms do not consider the actual resource usage of operators on the hosts. However, the presence of a wide range of different machines with different capabilities, processing power or memory makes the introduction of resource usage necessary (cf. [SKRR10]). Hosts can be overloaded when *their* operators are causing a lot of computational effort, resulting in falsified and unreliable results. To the best of our knowledge, no algorithm exists that both considers constraints and resource consumption at the same time.

Dynamics

Despite handling constraints and resource requirements, a placement algorithm has to respect the dynamic behavior of CEP in heterogeneous environments. For our approach, we assume the following application behavior:

a) *Static Operator Network*: Although adding new operators is often a typical example for dynamic behavior in CEP systems it is, in our experience, rather uncommon for industrial CEP applications. These applications are usually not changed frequently in terms of functionality. Once established, the set of operators is meant to run for a longer period and is going to report specific situations whenever they occur. Changes to the deployed set of operators often result from changes in the business logic of a company, for example when use cases are added or changed within a software. However, whenever a new operator is added, a quick deployment is desired.
b) *Dynamic Resource Availability*: Resource availability on a host affects our target applications more often than a changing set of operators. Resource consumption is often a consequence of changing event rates, which affects the stability and reliability of the processing: if the event processing is slower than the incoming event rate, the latency is increasing and events may be dropped. Once the available resources at a host get scarce, migration of one or more operators is necessary. Since operators are often based on real world events, the event rates are heavily dependent on uncontrollable sources and can therefore often not be predicted reliably.

As a consequence of these dynamics we have to be able to react on changing conditions properly:

a) Solutions have to be found quickly to minimize the time a system is in an *invalid state*, which is either that a host is overloaded, or an operator restriction is not fulfilled.
b) Due to changes in the system (which do not lead to an invalid state) a running operator placement may become inefficient over time. Consequently, the system needs to adapt itself in order to optimize the operator placement during runtime.

4.2.2 System Model

We consider a correlation network $N = \{n_1, n_2, ...\}$, and operator graph $G = (\Omega, S)$ as specified in Section 2.2.1. The hosts n are connected via a P2P network and together run a user-specified CEP application decribed by G.

A simple example of a typical operator description in our energy grid scenario is shown in Listing 4.1. Here, *powerConsumption* events from different households are aggregated and an *AggregatedPowerConsumption* event is forwarded for further processing.

OpDesc *AggregateCurrentConsumption*
WHEN *SEQ(pc1, pc2 : PowerConsumption)*
IF *pc1.meter* $\neq$ *pc2.meter*
RESTRICT *engine.type = amit*
EMIT *AggregatedPowerConsumption*
(amount = pc1.amount+pc2.amount)

Listing 4.1: Operator Description of a Simple Aggregation Operator

Both operators and hosts are dynamic: users may add, change, and remove operators as well as hosts may be added to, and removed from the CEP system during runtime.

As already indicated by the operator graph description in Section 2.2.1, operators can depend on each other. In particular, the outgoing event stream of an operator O_{ω_i} can serve as input to another operator, $i_(\omega_j) \subseteq I_\omega$. Sticking with the given example, the *AggregatedPowerConsumption* events are used in another operator that determines energy overload within a sub-network of the power grid. These interdependencies are described

in the directed, acyclic operator graph G, where we create an edge for every event stream between operators.

The hosts can be interpreted as network nodes in the correlation network. They are heterogeneous with respect to the capabilities they have and resources they provide. Resources, for example processing power or memory, are finite on every host and consumed by the operators placed on it. A capability is defined as a non-resource attribute of a host that might be required by an operator: for example a specific CEP engine, or membership in a domain. Capabilities are representing the binary constraints of our placement problem, as they only have two values (true, false). They are either fulfilled or not fulfilled. The host is modeled in our system such that every host n_j has p capabilities $\chi(n_j) = \{c_{j,1}, .., c_{j,p}\}$ and q resources $\gamma(n_j) = \{g_{j,1}, .., g_{j,q}\}$.

Sources and targets of events are not doing actual event processing in our network (i.e., they have no engine equipped and are therefore not capable of actually *hosting* operators). They are therefore not considered in the placement algorithm discussed in this chapter.

Operators can have different requirements with respect to resources and capabilities. Every operator ω_i has k capability requirements $\alpha(\omega_i) = \{a_{i,1}, ..., a_{i,k}\}$ (which we call binary *constraints*) and m average resource requirements $\beta(\omega_i) = b_{i,1}, ..., b_{i,m}$. The requirements are based on either the measured average resource consumption on a host (if the operator has been deployed before), or on a standard value based on the operator type (if the operator has not yet been deployed in the system). Resource requirements are measured on the nodes and updated on a regular basis. This is important as our proposed optimization process will later rely on resource consumption of operators and available resources on nodes. The measured average resource consumption is currently aggregated based on on the measurements of a specified time frame, which is currently set to 1 day.

Note: We chose the average resource consumption (instead of, e.g. the maximum value) since it was the most suitable value for most of our usecases, where the event rates within the system have not been subject to fluctuations and remained within a certain deviation threshold. In scenarios where the resource consumption varies heavily and has for

example peaks only during specific occurences, an average resource consumption measurement is unprofitable and other measurement metrics should be used. In our approach, we differ the resources of hosts (e.g. CPU consumption) and network (e.g. bandwidth consumption). Therefore, we additionally define the resource usage of an event stream (ω_i, ω_j) as $\delta(\omega_i, \omega_j) = [d_{i,j,1}, ..., d_{i,j,s}]$, where d is the average resource consumption of a network resource.

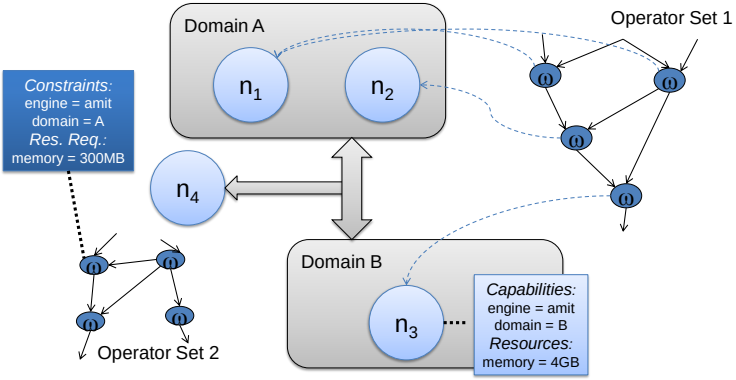


Figure 4.2: Elements of the System Model

Within this system, we now try to find a valid placement of operators on hosts in the correlation network with respect to constraints and resource requirements (cf. Figure 4.2). Therefore, we define the boolean matching function $\mu(a, c) : \alpha \times \chi \rightarrow \{true, false\}$, which is *true* if the constraint $a_{i,k}$ is included in the set of capabilities $\chi(n_j)$. A *valid placement* for an operator is given, if all constraints are fulfilled, i.e., $\forall a \forall c : \mu(a, c) = true$.

4.2.3 Problem Statement

Based on the given system model, we first formalize the general placement problem in distributed heterogeneous event processing networks (DHEP) and then state the optimization problem of minimizing network usage.

Definition 4 (The placement problem in DHEP) *Given an event processing network N consisting of j hosts n and a set of i operators $\omega \in \Omega$ that form an operator graph $G = (\Omega, S)$, find a placement P with the mapping function $\pi(r) = n$, which assigns an operator to a host, such that:*

$$\forall \omega : \mu(\alpha(\omega), \chi(\pi(\omega))) = true \tag{4.1}$$

$$\forall n, \forall c : \sum_{\pi(\omega)=n} (b_c) + \sum_{\pi(\omega_i)=n \vee \pi(\omega_j)=n} (d_{i,j,c}) \leq g_c \tag{4.2}$$

whereas $\delta(e_{i,j}) = 0$, if $\pi(\omega_i) = \pi(\omega_j)$

Condition 4.1 ensures, that all constraints of the placed operators are fulfilled. Condition 4.2 ensures that the resource usages of the operators and event streams do not exceed the host and network resources. This means, that for all operators placed on a host n and all resources c the sum of these operators' resource requirements as well as the incoming and outgoing event streams of these operators is less or equal to the number of provided resources.

Theorem 4.2.1 *The placement problem in heterogeneous systems is NP-hard.*

Proof sketch: Figure 4.3 illustrates how the problem can be reduced to *Bin Packing*. Here, hosts are mapped to bins, while operators are mapped to the values that going to be packed in the bins. The size of the bins is chosen by the free host resources, while the operators' resource consumption determines the values. Hence, if we could find an algorithm that solves the placement problem in deterministic polynomial time, the algorithm would also give a solution to pack the bins. □

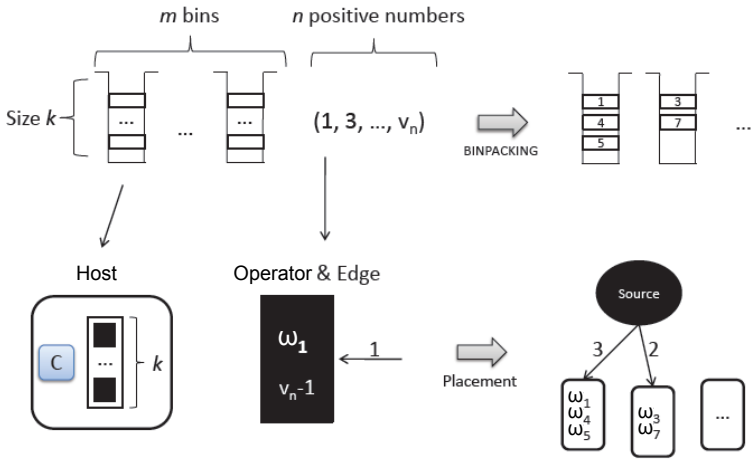


Figure 4.3: Reduction to *Bin Packing*

The formulated placement problem can be seen as a constraint satisfaction problem (CSP), where the operators constitute variables and the operator requirements are constraints that have to be matched by the hosts. Based on this CSP, we can formulate the constraint optimization problem of minimizing network usage. Network usage is the bandwidth-delay product, which denotes the load that is on the network at a certain point in time. Hence, we formally define the network usage σ of an event stream as

$$\sigma(\omega_i, \omega_j) = delay(\pi(\omega_i), \pi(\omega_j)) \times dataRate(\omega_i, \omega_j) \tag{4.3}$$

Furthermore, we define the placement cost ϕ of an operator ω_j on a host as the sum of network usage of its incoming event streams, i.e., its incoming edges in the operator graph:

$$\phi(\omega_j, n) = \sum_{\omega_x | (\omega_x, \omega_j) \in I_{\omega_j}} \sigma(\omega_x, \omega_j) \tag{4.4}$$

With Equation 4.3 and 4.4, we define our goal:

Definition 5 (Minimizing Network Usage in DHEP) *Given a cost function ϕ that determines the placement cost of an operator ω on a host n. If $\mathcal{P} = \{P_1, ..., P_k\}$ is the set of all possible placements solving the placement problem in DHEP (c.f. Definition 4) and $\pi_{P_i}(\omega) = n$ is the assignment of operator ω to host n done by P_i, $P_i \in \mathcal{P}$ is optimal iff for all $P_j \in \mathcal{P}$:*

$$\sum_{\Omega} \left(\phi(\omega, n)_{\pi_{P_i}(\omega)=n} \right) \leq \sum_{\Omega} \left(\phi(\omega, n)_{\pi_{P_j}(\omega)=n} \right) \tag{4.5}$$

With the given cost calculation, a placement is optimal in the sense of our optimization goal, network utilization, if it has the lowest placement costs in total. As a result, by reducing the placement cost the communication load generated by the operators is minimized. Furthermore we would like to stress that the definition of our cost function can be extended with the resource usage of the hosts, as it is demonstrated in [SKRR10]. By doing that, we would not only consider the event traffic in the network, but also resource usage on the hosts. This would change the optimization goal to minimizing system resource utilization.

4.3 Approach

Finding an optimal solution to the optimization problem (cf. 5) is expected to take significant processing time. This can lead to long phases in which the CEP system is inactive and events are unavailable. In order to ensure a high validity of the CEP system, our approach makes combined use of two algorithms. While the system is in an invalid state the algorithm aims to determine an *initial placement* for which all constraints are satisfied and no host suffers from overload (cf. 4). Here, we make use of concepts that aim towards finding a valid solution early by favoring hosts which both match the constraints and have the resources available needed by the operator. Once the initial solution is deployed and the system is running in a valid state, an *optimization* phase is started which aims at minimizing the network utilization.

During runtime, all hosts operate without central coordination. Continuously, a host checks for all of its operators whether the constraints are satisfied. To ensure the availability of the system during optimization, we perform logical migrations: Every host creates an *update list* where it stores potential migrations for the operator it wants to get optimized. Actual changes to the system are made in a *commit phase*, where updates with respect to an operator deployment are executed. The commit phase is entered, when the optimization algorithm has stopped. During the commit phase, every host that is participating in the running optimization deploys the new operators that are assigned to it.

Before presenting the details of the *initial placement algorithm* and the *optimization* phase in Section 4.3.2 and 4.3.3, we first concentrate in Section 4.3.1 on the common parts shared by the two algorithms.

4.3.1 Algorithm Basics

Our placement algorithm needs to react to the dynamics which stem from inserting new operators into the system or changing event rates. We therefore make use of a monitoring component at the host. If the changes result in an invalid placement, the affected host initiates a reconfiguration (i.e., *initial placement*). This may trigger subsequent migrations of operators, if necessary, to find a valid placement. If the placement stays valid, the continuously running optimization algorithm will eventually search for new configurations that would result in a better network usage. During this process, hosts go into a busy state while searching for alternative placements to prevent invalidations caused by parallelism. The generic course of action performed by a placement algorithm is characterized briefly in Algorithm 2.
Both the initial placement and the optimization of an operator run through several steps when they are started. The steps are sketched in procedure *FindPlacement* in Algorithm 1. After the placement algorithms are started for an operator, we define our solution space by searching for hosts that fulfill the constraints, i.e., are valid hosts for the operator. Then, each host checks whether it has enough free resources to deploy the operator and locally calculates the placement costs based on the expected network

Algorithm 1 Standard Routine of Placement Algorithm

```
procedure FindPlacement(ω)
    identify valid candidate hosts(ω)
    receive calculated deployment costs()
    repeat
        choose next best target host()
        tryLogicalMigrate(r)
    until migrationSuccess
end procedure
```

Algorithm 2 Generic Placement Algorithm

```
procedure monitorValidity
    while true do
        if (invalidState) then
            FindPlacement(ω)
            enterOptimizationState()
        end if
    end while
end procedure
```

utilization (c.f. Section 4.2.2). The result is sent back to the requesting host. Finally, among the replies a new host is chosen and the operator is migrated. Note that after migrating an operator, other subsequent migrations might be necessary to reach a valid state. Thus, we always perform these steps logically: The running system is only affected in the commit phase, where the changes are accepted and finally deployed on the hosts. Hence, optimization can run in parallel and is not continuously interrupting the CEP process.

Each of these typical steps will now be described in detail. However, although both the initial placement and the optimization algorithm follow this concept, they differ after the *identification of valid candidate hosts*. Therefore, we describe both mechanisms separately in Section 4.3.2 and 4.3.3.

Identifying Candidate Hosts

Candidate hosts are potential targets for an operator that needs to be migrated. Determining candidate hosts requires to deal with two difficulties: i) candidates must be determined fast and at low communication cost even though there is no central knowledge; ii) due to the presence of constraints the number of candidate hosts is typically restricted to a small subset of all available hosts.

To overcome these difficulties, our candidate selection uses a content-based Publish/Subscribe system that is deployed as part of the CEP system (cf. [TKKR10]). Target hosts are determined by publishing the operator by means of its binary constraints. The key is, that each host subscribes to the constraints it is capable to suffice (cf. Figure 4.4). In our system implementation, we specify the needed *engine type*, *domain*, and connections to *database* as constraints.

The course of finding candidate hosts is depicted in Figure 4.4. When an operator must be placed within the system, the initiating host (n_4) publishes a deployment request with the operator's constraints. Publish/Subscribe delivers the message to all subscribers, i.e., the hosts that can potentially deploy the operator. The deployment request contains

two pieces of information: the operator and an identifier. The operator is essential for the cost calculation, the identifier is needed for handling parallelism in the system, since multiple requests can occur concurrently.

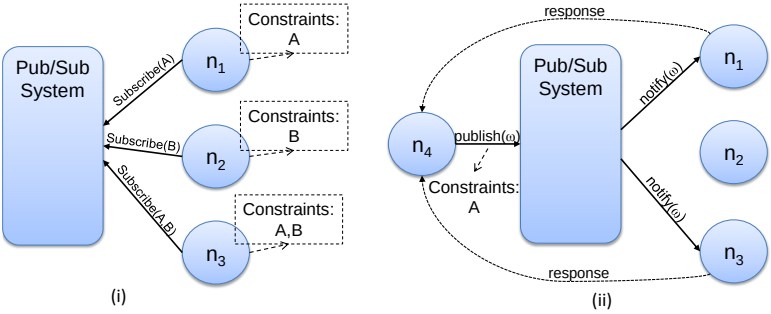
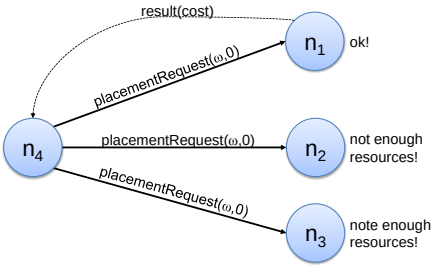


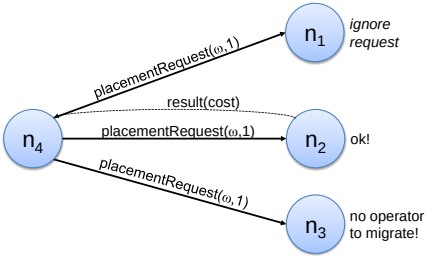
Figure 4.4: Subscribing to Operator Constraints

4.3.2 Initial Placement

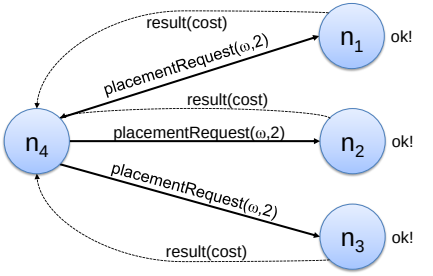
After receiving a placement request, every host first checks whether it can deploy the operator. Therefore, a host compares the estimated operator requirements with its own free resources. Basically there are two options: i) the host can deploy the operator; ii) the host cannot deploy the operator unless one or more other operators are migrated away from the host (that means, it meets the binary constraints, but not the resource requirements). If the host can deploy the operator he sends an answer containing the cost for the operator deployment (i.e., the network utilization it would cause). However, when migration of other operators is necessary, the operation of placing a operator becomes complex. The reason is that these migrated operators might cause high overall costs on other hosts themselves after



(a) Priority 0: Only Hosts with free Resources reply



(b) Priority 1: Hosts with easy migratable operators reply



(c) Priority 2: All hosts reply

Figure 4.5: Requesting Placements with the 3-Way Heuristic

moving them, hence compensating for a good placement of the originally placed operator. Furthermore, the migrated operators might cause even more migrations subsequently.

As a result, we use a 3-way heuristic which

a) aims at reducing the number of migrations,
b) favors placements with low costs in terms of the optimization goal,
c) guarantees to find a solution if there exists one.

In our approach every request is associated with a priority. The larger the priority value the more hosts will answer a placement request. The algorithm uses three priority levels, and its behavior is sketched in Figure 4.5.

At priority level 0, only hosts that can directly deploy the operator reply to the deployment request, i.e., they fulfill all constraints and have sufficient resources. The reply contains the result of the cost function. In our example, the requesting host n_1 waits for the results, and selects the host which results in the best placement cost, i.e., the lowest network utilization value. If no host can be found that can directly place the operator, the request message is assigned priority level 1. Now, every host replies that can deploy the operator after migrating one or multiple other operators whose migration cost are below the cost of the operator. The migration cost is used in order to prevent operators from migrating that require a lot of migration effort. With this cost, we are able to order operators based on their migration effort. The migration effort is a fixed value assigned to each operator at time of defintion and is determined by the amount of state information that has to be transferred in a migration process. Finally, priority level 2 is assigned to a request message if priority level 0 and priority level 1 messages did not succeed in finding a valid placement. Any host receiving a request message of priority 2 is answering the request, even if the migration cost of the already deployed operators is higher than that of the new operator. If there is no valid placement after priority 2 is reached, an error is sent to the host initiating the migration.

Replies to a deployment request are sorted by the initiator according to the deployment cost. For example, in Figure 4.5 the initial placement leads to a result list of n_2, n_4, n_3. Based on the result list, the requesting

host sends a placement request to place the operator to the best host in the list. The recursive component of the algorithm comes into account at this point: If the host receiving the placement request cannot deploy the operator without migrating one of its other operators (i.e., priority level 1 or higher), the placement process is initiated again.

This procedure is depicted in Figure 4.6. In (a) host n_2 is chosen to deploy operator ω_n. However, it is forced to migrate operators in order to do so. Therefore it creates a migration list containing the operators ordered by the migration cost. The host then tries to migrate operators based on the list, here ω_1 and ω_2. For both operators, a new placement procedure is initiated by n_2 subsequently in (b). If a placement fails, the next host in the result list is chosen (cf. ω_2 in (b) and (c)). As can be seen in (c), host n_3 may also be forced to migrate one of its operators in order to find a valid placement. After a better valid placement is found, and the algorithm finally terminates, the hosts can update their state and deploy the new operators (d).

Backtracking

If a placement attempt failed failed during the logical migration phase, an error message is sent to the host which initiated the placement request. All logical changes made during the placement attempt are revoked. This is done by sending a *rollback* message to every host participating in the placement attempt (in opposition to the already mentioned *commit* message). The hosts can be identified by a *tracepath*-list associated with the operator. Every host participating in the logical migration steps puts itself on the *tracepath*-list. The rollback mechanism resembles backtracking, i.e., it jumps back to a valid state if an entered path does not lead to a solution (in fact, it stays in the current state as the changes were only logically). The logical changes are then deleted on the hosts.

Termination

To ensure termination of our algorithm, we have to avoid infinite cycles during the logical migration phase. The cycles can occur in scenarios with

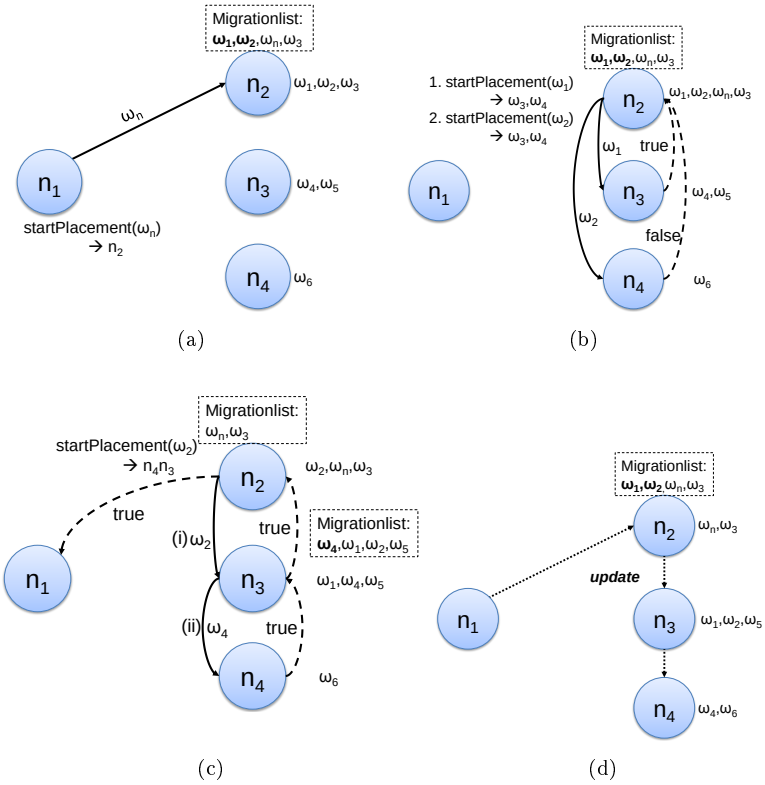


Figure 4.6: Cost Calculation during Placement

an overall heavy consumption of the available network and host resources . In these scenarios, a priority level 2 search is likely done (cf. Figure 4.5. That means, all hosts try to deploy the operator, possibly by migrating other operators away from itself. During this process, cycles can occur. To avoid cycles, we add an *operatorPath* variable to the placement request message. Every host receiving the request updates this variable by adding an identifier representing the host. A cycle can now be detected whenever an identical sub-path in the *operatorPath* is repeated. If a cycle is detected, the logical placement goes back the operator graph stepwise to the last logical placement before entering the cycle.

Furthermore, deadlocks might occur when two independent placement algorithms are running in parallel. Whenever a host is calculating a new deployment it gets into a *busy* state and will not accept other placement requests. To avoid the possible deadlocks, each requesting host waits for a random time until requesting again whenever a host is busy. If the host is still blocked, it has to cancel its running request if it has a lower id than the new request.

Properties

In this Section we discuss and prove properties of the presented algorithm. Lemma 4.3.1 states, that the algorithm is complete. Lemma 4.3.2 states, that the algorithm instantly terminates if there exists a host that allows a valid placement and has enough resources available.

Lemma 4.3.1 (Completeness) *Assume a system without node and network failures and the existence of a solution to the placement problem in DHEP (c.f. Section 4.2.3), where network conditions, hosts, and operators do not change. The initial placement algorithm will eventually find a solution.*

Proof sketch: Following algorithm 1, the first step is to find a valid target host, by using the 3-way heuristic. In the worst case scenario, no host with enough free resources can be found, which will cause the initiation of a placement request at priority 2. This will result in an answer of all

hosts which can validly deploy the operator. At this point, the initiating host has knowledge about all possible placements. By logically migrating the operator to a target, the target host will also use algorithm 1 for every operator it hosts until a solution is found. By doing this, the initial placement algorithm will, in the worst case, iterate over all valid placements combinations. In particular it will find any existing solution. □

Lemma 4.3.2 (Quick termination) *Assume a system without node and network failures. If there exists both an invalid operator placement and a host with enough free resources to satisfy the constraints of this operator. A valid placement for the operator is found after the first iteration step.*

Proof sketch: In the first iteration step, an event is published at priority level 0. The event contains the required resource types (as attributes). In a stable state all hosts have subscribed to their resource types they are able to provide. Consequently, any host capable of satisfying the constraints of the invalidated operator will respond to the event and eventually deploy the operator. □

Further remark: If such a host does not exist, the algorithm will always need more than one iteration step, since (multiple) migrations are done. The time spent on finding a solution is heavily dependent on the length of the search path, and therefore on the number of migrations done. While the different priority levels are used to keep the search path as short as possible, the highest priority results in a lot of communication. Yet it is required for completeness, as any host is eventually considered in the placement process.

4.3.3 Optimization Phase

The previously presented initial placement task is solving the placement problem in heterogeneous environments, as it guarantees to find a valid placement if one exists. To find a near optimal solution, we optimize this initial placement in the optimization phase described in the following. On a regular basis, every host tries to find alternative placements for

operators, calculates the cost difference of the new placement, and decides whether a migration should be initiated (as depicted in Figure 4.6).

Relating to the challenges stated in Section 4.2.1, the algorithm for minimizing the network latency in a heterogeneous event processing system needs to fulfill two major properties:

i) decentralized coordination,
ii) avoid to be stuck in local minima.

Decentralized coordination prevents us from maintaining global knowledge and forces us to make decisions based on local knowledge. This is especially problematic when we determine the next operator that should be optimized. Considering the optimization problem, the operator with the highest migration priority ideally should be the *worst* placed operator on the host. That means, the operator with the highest cost in terms of the optimization goal is tried to be migrated first. Ideally, there is a high disparity between its current and its optimal placement cost, and the overall placement cost in the system gets significantly better after migration. The operators with the lowest cost will be migrated with the lowest priority.

If a new target host with a better placement cost could be found during the logical migration steps, the actual migration will be triggered for the operator. At this point, another factor comes into play: Always choosing the host with the best result may lead to suboptimal solutions. The reason is, that the solution is found greedily, without being able to get back. That means, the algorithm tries to find the best placement for the operator. However, this placement may prevent deployments of other operators, thus leading to non optimal overall results. In other words, the algorithm will run into a local minimum for the migrated operator and may prevent solutions with an overall better cost in terms of the optimization goal. We approach this problem with two mechanisms: On the one hand, we calculate a migration probability for every logical migration. On the other hand, we may allow a migration with a worse placement cost in some cases (cf. Algorithm 3) . distribute the knowledge about the best cost within the system during the optimization process. *Avoiding local minima* is crucial for optimization algorithms.

We approach this problem by letting the hosts optimize *their* operators independently based on their local knowledge and applying techniques of simulated annealing. Our optimization algorithm is sketched in Algorithm 3. After entering the optimization phase, every host periodically starts the optimization algorithm. At first, it chooses the *worst* placed operator that has not been optimized so far (cf. *ChooseOperatorFromHost*). That means, the operator with the best optimization potential based on the host's local knowledge. It does so by comparing the cost of each operator with a *best known system cost* parameter that is distributed within the system during the optimization process. Thereafter, a migration probability is calculated which is determined by means of the current placement and the improvement potential (c.f. Section 4.3.1). The migration probability increases with the improvement potential. This value is used to add randomness in the process of choosing the next operator that is going to be optimized. After an operator has been selected, placement options are searched and the best replying host is selected. The host is selected if either:

i) the new placement cost is lower or,
ii) a randomly calculated number is higher than an acceptance function.

Algorithm 3 Pseudocode for the Optimization Algorithm

```
procedure initLSA( )
    r ← choose Operator from Host( )
    if searchAlternativePlacements(ω) then
        diff ← calculateCostDiff( )
        if migrationProbab(ω) ≥ RAND[0;1] then
            if diff < 0 or e^(-diff/T(n)) ≥ RAND(0..1) then
                updateHosts( )
            else discard( )
            end if
        end if
    end if
end procedure
```

It can be seen, that the basic principle of optimizing a current solution based on simulated annealing techniques shares similarities to a distributed greedy optimization:

i) every host has limited knowledge that is based upon the *tracePath* of a placement request;
ii) after predefined intervals, every host tries to find alternative placements for any of its operators.

However, there exist some differences which are motivated by the major requirements to coordinate the optimization decentrally as well as avoiding local minima.

Properties

The presented algorithm allows to find near best solutions decentralized by adopting techniques from simulated annealing in a decentralized network and avoiding to get stuck in local minima. In the following, we present the algorithms properties and discuss their correctness informally.

Property 4.3.1 (Improvement potential) *The optimization favors operators with a higher potential to improve the placement.*

Discussion: as in classic simulated annealing, our optimization algorithm does not always migrate the operator(s) when finding an alternative solution. Instead, there exists a certain migration probability P which is determined by means of the achieved improvement and the best possible improvement Δ_{max}. The best possible improvement describes the maximum difference between all operator's worst and best placement. The migration probability increases, when the achieved improvement is closer to the best possible improvement. Formally, the probability is defined as:

$$P(\omega) = \frac{\phi(\omega, n_{current}) - \phi(\omega, n_{new})}{\Delta_{max}} \tag{4.6}$$

However, in a distributed system Δ_{max} is unlikely to be known, since each host would have to know every possible placement. Thus, Δ_{max} is estimated locally. It is determined by the maximum cost difference of all

locally known operator costs. During the optimization process, various operator requests and replies reach the hosts and the values of Δ_{max} on each host adjust. In resource sparse systems, where many migrations are performed, the hosts finally have almost equal values for their local Δ_{max}. With that, a similar behavior like in the original simulated annealing algorithm is achieved, and the operators with the highest improvement potential are more likely to be chosen.

Property 4.3.2 (local minima) *The optimization algorithm can avoid getting stuck in local minima, given a non-zero temperature value.*

Discussion: as in classic simulated annealing, our optimization algorithm accepts a worse placement with a certain probability. This is done in order to be able to jump out of local minima, that would hinder a greedy algorithm from improving further. Therefore, we introduce an acceptance function that gives the algorithm a chance to accept worse placements too (c.f. Algorithm 3). The probability to accept a worse placement is dependent on two factors: the difference *diff* between the new solution and the current solution, as well as temperature value T, which is calculated by a temperature function. The probability to accept a worse placement increases with a higher T and a small *diff*. During the optimization process, the temperature function constantly reduces T to ensure the termination of the optimization process. Therefore, as long as T is non-zero, the algorithm may accept worse solutions and can escape local minima.

Property 4.3.3 (Termination) *The optimization algorithm running at each host terminates, when one of two conditions is met:*

i) an optimization run has been started for each (local) operator or,
ii) a user-defined termination criterion is fulfilled.

Discussion: While the first condition ensures, that an optimization run is finished after every operator has been considered, the second condition is introduced in order to compensate a probably long and unwanted runtime. Because no heuristic is used to find quick solutions (as with the initial placement), it can occur that the chosen path in the solution tree

is long. To overcome this problem, it is possible to add a user-defined termination criteria, which may for example stop the optimization run after a certain time or when the temperature value is below a certain threshold. Furthermore, the same mechanisms to avoid deadlocks and cycles are applied as in the *initial placement* algorithm (c.f. Section 4.3.2). After termination, the commit phase is started and the optimization results are deployed (if changes should be made). Note, that the optimization algorithm runs periodically, and a final termination does not exist.

4.4 Related Work

Placement in DCEP

With the increasing importance of distributed CEP, researchers have also tackled the placement of CEP functionality within a distributed CEP network. However, the existing solutions are unable to deal with heavy constraint settings, where only a small subset of hosts is appropriate for the placement of each operator. As a result, these systems are not able to solve the constraint satisfaction (optimization) problem.

For example, Pietzuch et al. describe a framework for event composition in distributed systems [PSB03]. Here, operators are defined as mobile CE detectors that move around in the network, searching for the best placement by means of a distribution policy. However, the system does not include any constraint respective behavior. Basically, placement is not restricted to a subset of hosts. Also, resource usage is no explicit variable in the system. As a result, the proposed framework is not able to solve the constraint satisfaction (optimization) problem.

Padres is a well-known distributed content-based Publish/Subscribe system which can deal with complex events [Fid06]. Migration of CEP functionality is provided and can be placed dynamically within the system. The provided algorithm reduces the bandwidth usage. Constraints are also not present in the system, since heterogeneity is not tackled.

Beside these approaches, many stream-based event processing system work with similar strategies (e.g., [SMW05]). This behavior comes along with the characteristics of current stream processing systems, where the focus lies in a high speed processing of huge amounts of events. As a consequence, *operators* in stream based systems are less complex and there are no constraints associated with them. Hence, placement is handled constraintless, for example by means of spring relaxation techniques (cf. [PLS+06] and [RDR10]). Here, a latency space is created in order to efficiently search for the best placement within the search space. However, it is not possible to restrict the search space based on constraints other than latency.

However, the CEP community has identified the need for placement algorithms that can handle operator restrictions. This is first mentioned in [KKR10], where the authors present a description language where users can attach restrictions to rule definitions. They also describe a simple mechanism which relies on proactively collected information to find a valid placement that satisfies all restrictions. However, there exists no concrete optimization goal and an adaption during runtime is not existent (as long as the restrictions are fulfilled). Moreover, the proposed algorithm is incomplete, since it relies on proactively collected information about the neighborhood.

(Distributed) Constraint Optimization

Algorithms that do handle placement problems in high constraint settings can be found within the constraint optimization community where a lot of research has been done in the last decades (c.f. [GL97, KJV83]). This resulted in many distributed algorithms, that try to be both efficient in terms of completeness and speed (c.f. [K. 95, CDM+91, YH00]). However, these algorithms are designed for static problems, making it unefficent to deploy them in a dynamically changing distributed system like ours. However, we still want to discuss some of the work in more detail, as basic principles are adopted in our solution: Weak Commitment Search [Yok95] was designed to overcome the problems of both backtracking and

hill climbing. The algorithm is based on a minimal-conflict backtracking. Instead of performing the backtracking step in case of a failure, the whole solution is discarded. The process then starts again with the old settings as the initial ones as well as additional constraints preventing the algorithm to enter wrong paths again. Hence, there only exists a weak commitment to the old (partial) solution. The weak commitment search is complete, meaning that it will always find a solution when there is one. However, to prevent an exponential increase of the number of constraints, it is proposed to cut the number of constraints to a value k. With this, a compromise is achieved between runtime and completeness. However, k is highly dependent on the problem and therefore it is unacceptable for dynamic scenarios as they are considered here.

Moreover, Branch-and-Bound optimizations have found a major consideration in the field of DCOP (cf. [EBB+08, MSTY05]). While these methods are able to provide quality guarantees, they are mostly slow as their backtracking based algorithm relies on synchronous communication. Also, they are not able to optimize previous solutions. Instead, when changes to the system are made, the algorithms start all over again to find a new solution, which makes them unacceptable for our problem.

Another recent approach tries to enhance the iterative improvement algorithm by improving the variable with the most violated constraints [CD01]. This, however, makes the algorithm incomplete, as not always a solution may be found.

In our work, we adopt mechanisms of DCOP, combine them with heuristics and integrate them into our distributed event system. This enables us to create a new placement algorithm for heavy constraint systems. The algorithm is able to find valid solutions with acceptable cost by restricting the search space and optimizing the current solutions by searching for alternative placements in the restricted search space.

4.5 Evaluation

Within the DHEP framework (cf. Chapter 3), we implemented the presented energy utility scenario from Section 4.2.1. Here, *powerConsump-*

tion events serve as the input of the CEP system. These events are filtered by (typically cheap and fast) different filter operators and aggregated by aggregation operators to *AggregatedPowerConsumption* events. Furthermore, more expensive sequence operators are used to calculate overload and underload situations. Constraints were assigned to all operators concerning required engine types, domains, and the connection to some specific database. At the same time, all our hosts are located in exactly one domain, provide at least one engine, and may have access to a database for context enrichment (cf. Chapter 3). Matching of these binary constraints is essential for our scenario.

In this setup, our goal was to find placements which match all binary constraints. First, the placement algorithm will run to find one valid placement which is deployed. Afterwards, the system will optimize itself to find better placements. We measured the overall placement cost by aggregating the placement cost of each deployed operator.

Before discussing our evaluations in detail, we describe how the placement optimization is behaving during runtime. As we were not able to find algorithms which were able to handle the binary constraints we face in our system, our primary goal is to verify the correctness of our approach. During our evaluations, we compare our proposed optimization algorithm with itself, by adapting its optimization behavior. Our optimization algorithm presented in Section 4.3.3 includes a temperature value based on the simulated annealing method. This algorithm is called *SA-based algorithm* in the remainder of this evaluation chapter. By removing the temperature value, we achieve a different optimization behavior. More specifically, if the temperature value is 0, the optimization algorithm will always only accept placement with overall better placement costs. Hence, it acts purely greedy and is therefore called the *greedy algorithm* in the remainder of this evaluation chapter.

To verify the validity of our approach, we want to compare the results of our distributed optimization algorithm with the theoretically optimal result. To be able to calculate the optimal result of a placement problem, an exhaustive search (i.e. brute-force) had to be made for a placement scenario. To be able to perform an exhaustive search in a reasonable low time, we chose a small placement scenario where 28 operators were

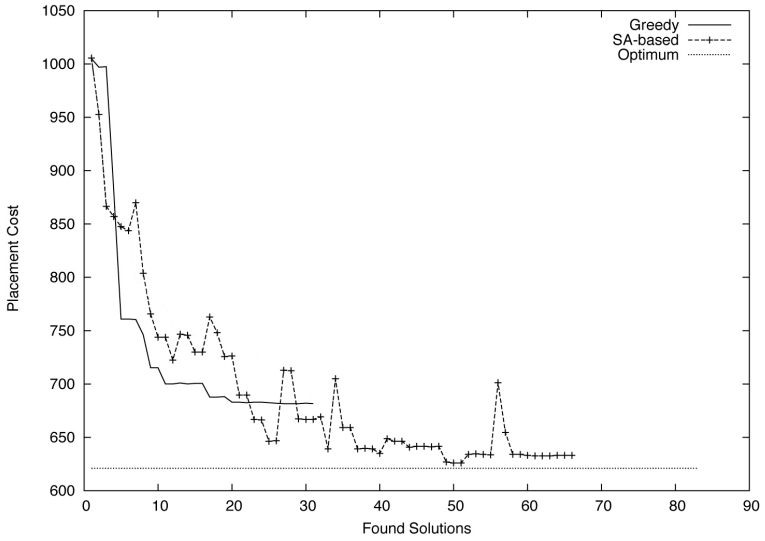


Figure 4.7: Optimization Improvements over Time

placed on 12 hosts. As described earlier, we used the smart energy scenario during our evaluations. Each operator performs the task of either filtering, aggregating, or measuring an event sequence. Every operator produces a different cost on its hosts. Among the 28 operators, every operator had a certain number of binary constraints which were randomly chosen. Likewise, the 12 hosts had randomly chosen capabilities matching those constraints.

Within this scenario, we measured the overall placement cost by aggregating the individual cost of all placed operators. The result of our exhaustive search marks the optimium solution which could theoretically be found by our distributed optimization algorithm. It is shown as the dotted line at the bottom in Figure 4.7. In comparison to that, we show the placement results of the greedy algorithm and the SA-based algorithm as they converge towards the optimum over time. Note: The X-axis of Figure 4.7 shows the various solutions found during the optimization steps, not the calculation time.

Several results are worth noting: 1) the greedy optimization algorithm gets better continuously while the SA-based optimization algorithm also accept worse solutions with a certain probability. 2) the greedy algorithm stops earlier than the SA-based algorithm, indicating that it converged to a local minimum without being able to get overall better results. 3) The SA-based algorithm was able to find an overall better solution than the greedy algorithm.

During long running applications, the behavior showed by Figure 4.7 is recurring: If the system changes, the current placement gets worse (or even invalid). Consequently, the sytem will be adapted. A new placement is calculated and optimized.

4.5.1 Comparing Different Constraint Levels

Since our algorithm was specifically designed for systems, in which binary constraints exist and restrict the amount of valid target hosts for each operators, we examined in our second evaluation how the binary constraints affect the placement results. Therefore, we evaluated the differences of

the algorithm variants under varying different scenarios. We measured both the achieved overall placement cost and the caused traffic in terms of messages sent. The latter is an indicator for the time and effort it took to find a valid solution. Similar to the first evaluation, we wanted to compare the optimization results with the theoretical optimium, found with an exhaustive search.

Table 4.1 lists the results for three scenario variants. For every scenario, we created multiple operator sets with varying constraints and deployed them on a network of 15 hosts. Scenario 1 is the most restrictive one. Out of the available hosts, only 10% were able to meet the binary constraints (i.e., the valid search space for an operator's target host is significantly lowered by the available capabilities). In Scenario 2, on average 18% of the hosts are possible placements for each operator. In scenario 3 about 35% of the hosts are on average valid for an operator placement.

Constraint Sat.	10%		18%		35%	
Algorithm	Cost	Msgs	Cost	Msgs	Cost	Msgs
Initial	1055	3.8	997	9.2	877	13.1
Greedy	1024.1	19.1	923.8	17.2	777.8	29.8
SA-based	1006.7	23.2	867.4	24.8	769.6	34.5
Optimum	949	-	789	-	721	-
Random	1219.3	13.1	1158.3	9.2	1178.4	13.1

Table 4.1: Optimization Results at different Scenarios

Table 4.1 lists the overall placement cost and the messages sent for each migration in the different scenarios. That means, for migrating an operator during the initial placement, an average of 9.2 messages had been sent in scenario 2. It can be observed, that reducing the constraints of a scenario leads to a different result of the algorithms: On the one hand, the overall quality that can be achieved is better, since there are more possibilities to find a better placement. On the other hand, this leads to

a higher number of messages that are processed during the optimization process.

We can derive some important results. First, all optimization algorithms improved the initially found placement result. In our experiments the convergence of the SA-based algorithm is 6.3% to 9.8% worse than the optimium depending on the scenario. The improvement is independent of the scenario's constraint satisfaction.

Second, the more restrictive the scenarios are, the more they profit from our distributed simulated annealing algorithm: Less messages are processed and the quality improvement is more promising. In scenarios that have less constraints like scenario 3, it is reasonable to favor a greedy algorithm. Here, the small difference between the optimization results does not justify the additional load imposed by the SA-based algorithm.

Third, the traffic caused by our algorithm is negligible. Even in worst case scenarios, where many hosts are involved in the placement process, we processed in average 35 messages for migrating an operator. In a system, which processes hundreds of event messages per second, the communication overhead of the placement algorithm is marginal.

Finally, it is worth noting that the average runtime of the initial placement algorithm was 331ms. This is significantly lower than the average runtime of the subsequent optimization phase of the SA-based algorith, which took 9.5s. We conclude that the differentiation between the two algorithms is of high importance for the availability of the system. This verifies our approach of achieving both a high availability by quick initial deployments as well as making logical optimization steps during runtime to improve the overall placement.

4.5.2 Insertion of Operators in Large Scenarios

In our third evaluation, the presented setup was changed in size by adding additional operators and hosts. In constrast to our previous evaluation, the binary constraints within our system are not changed but set fixed. More specifically, we simulated a network of 250 heterogeneous hosts. In this network, we continuously added operators, such that the resources

within the DHEP system get more and more scarce. Each added operator had a randomized constraint. However, it was guaranteed that at least 10 different hosts exist matching this constraint (at least 4%). After every operator insertion, we measured the overall placement cost. Then, we waited for the optimization algorithm to be completed, and measured the overall placement cost again. This procedure of adding operators was repeated until our system was not able to place the next operator. This whole scenario was repeated 10 times, the average results can be found in Figures 4.8 and 4.9. To put our results into perspective, we added the results of an algorithm which randomly chose a host as the target. The only condition was that it would need to match the binary constraints.

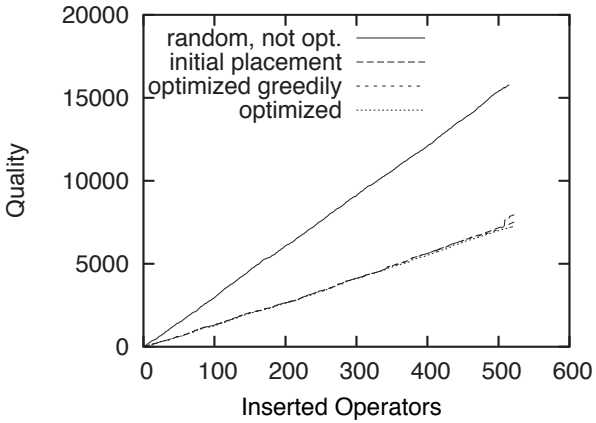


Figure 4.8: Quality Comparison in Large Scenarios under Growing Operator Sets

Figure 4.8 shows the achieved results of all algorithm variants. Several outcomes are apparent: First, as expected, the algorithms perform way better than a randomized operator deployment. Second, in the early and

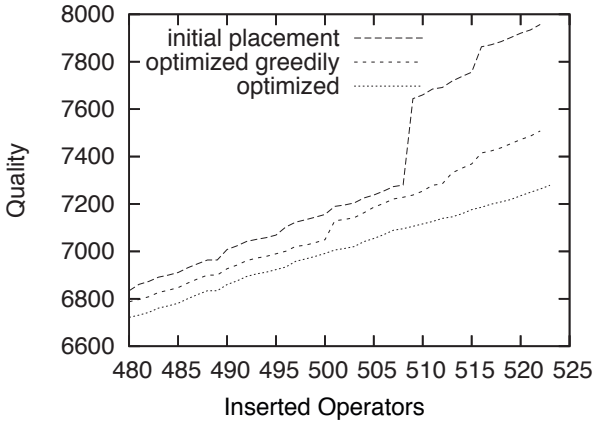


Figure 4.9: Quality Comparison in Large Scenarios in the Late Stages

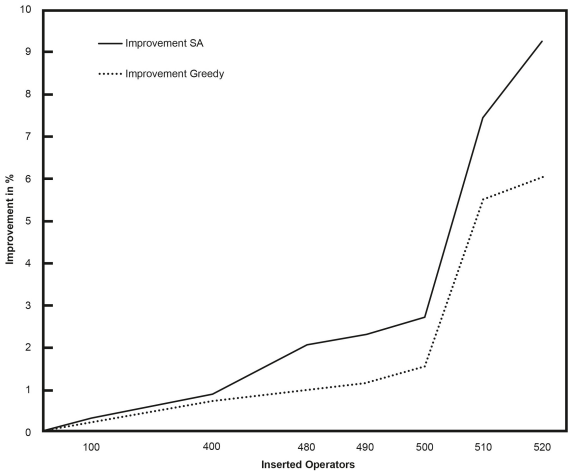


Figure 4.10: Improvement achieved by the Optimization Algorithms

mid stages of our evaluation, where the system resources are sufficient, the difference between an optimized placement and an initial placement is not as significant. This can be seen in Figure 4.10. Because there are enough free resources to place each operator directly on a host providing good placement results, without having to migrate other operators. Note that the system is already in an optimized state whenever the new operator is added. Hence, the improvement made during the optimization phase appear to be small.

However, this changes as soon as hosts resources get scarce and migrations might be needed to achieve a valid placement. This behavior can be seen in the late stages of the experiment, as depicted in Figure 4.9 as well as Figure 4.10. Based on our experiment, the use of the SA-based algorithm achieves the best optimization results especially when resources get scarce on the nodes.

4.5.3 Optimization under varying Temperature Values

In this evaluation we compare the behavior of our algorithm based on three different temperature values for the SA-based optimization: One *fast* converging temperature value, causing the algorithm to behave greedily soon; one *slow* converging temperature, allowing the algorithm to avoid local minima for a longer time; and one *mediocre* temperature in between. We compare the three algorithms with the greedy algorithm under a changing amount of available resources. By reducing the available resources, the overall resource utilization is increasing. This means, we increase the restrictiveness of the scenario stepwise, hence reducing the solution space of the placement problem and making it more difficult to find a valid placement.

Figure 4.11 shows the quality improvement achieved by all four optimization strategies relative to the scenario restrictiveness. Figure 4.12 shows the number of migrations needed during optimization to find the results relative to the restrictiveness of the scenario. In the depicted evaluation, the last valid placement could be found at an average resource usage of 97%. There were no valid solutions after reducing the resources even further.

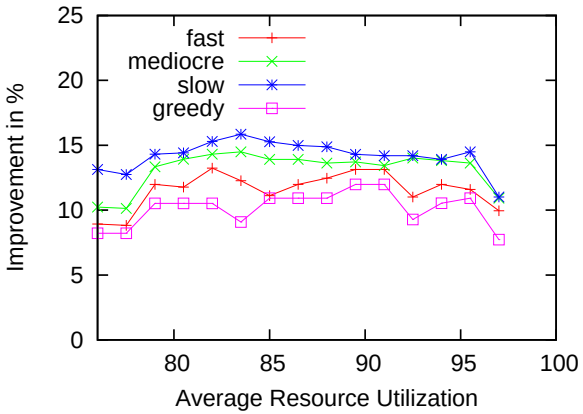


Figure 4.11: Improvement of Initial Solution in Restrictive Scenarios

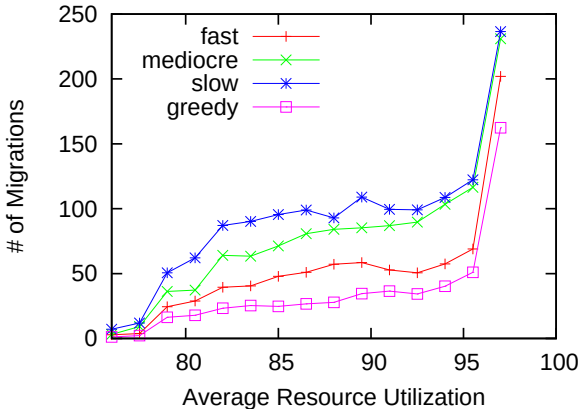


Figure 4.12: Migrations per Optimization in Restrictive Scenarios

Remarkably, it can be observed that finding valid solutions for highly restrictive scenarios (i.e., where almost all resources are consumed), results in more migrations (and therefore in a higher runtime). The reason is, that for optimizing the placement of an operator, it is more likely that another operator has to be migrated subsequently. Hence, the optimization under extreme resource requirements (>95%) resulted in a high number of migrations. This is especially true with slow converging temperature values, where additional migrations are performed due to accepting worse placements with a higher probability. This behavior can also be observed on other scenarios. As a consequence, we conclude: i) independent of the resource conditions on the hosts, there exists a tradeoff between the effort one is willing to put into optimizing the solution, and the quality of the result he will get; ii) Under extreme conditions, where all parts of the systems suffer from extreme resource consumption, our optimization strategies may require many migrations to eventually find valid solutions. This will highly increase the runtime of the algorithms. Here, we propose to prefer a greedy optimization to keep the runtime acceptable. However, we believe that these extreme conditions are rather unusual and should not happen regularly.

4.6 Conclusion

This chapter has addressed the placement of operators in heterogeneous event processing systems where it is important to cope efficiently with many constraints. The algorithm can adapt to dynamics that stem from load variations as well as to changes in the operator deployment. The analysis and evaluations show that the approach is in particular beneficial in heavy constraint settings. Here, it finds near optimal placements with low cost by efficiently restricting the search space of possible placements. Furthermore, the separation in two phases – the initial placement phase and the subsequent optimization phase – contributes to a fast operation of the event processing which is crucial in the practical operation of event processing networks. In our evaluations this was true until we reached at an average resource consumption of about 95%.

Future work to enhance the presented algorithms could concentrate on the selection of the temperature function and a more efficient use of Publish/-Subscribe to significantly reduce the communication cost in low constraint scenarios.

5 Access Policy Management in Distributed Multi-Domain CEP Systems

The goal of this part of our work is to ensure that in a multi-domain, large-scale CEP system, every producer of information can control the access to its information even over multiple correlation steps. In particular, our contributions are:

- An *access policy inheritance* mechanism to overcome the lack of security in multi-domain CEP systems. The inheritance ensures that the privacy of original information is guaranteed, even if the information is processed through a chain of operators.
- A mechanism to calculate *event obfuscation*. Here, we take into account the *obfuscation* of information during the correlation processes.
- A *scalable approach for secure access management* across multiple domains and correlation steps, by integrating event obfuscation into the access policy inheritance. We take into account the *obfuscation* of information during the correlation processes, which has an important influence whether access policies need to be respected or not.

In the upcoming Sections, we will present the aforementioned contributions, discuss our implementation as well as evaluate it's effectiveness.

5.1 Introduction

In today's business processes, it is very important to detect inconsistencies or failures early. For example, in manufacturing and logistic processes,

items are tracked continuously to detect loss or to reroute them during transport. To satisfy this need, complex event processing (CEP) systems are of increasing importance in today's business environments [HSB09]. The goal of CEP systems is to detect situations by observing and processing events which emerge from sensors all over the world, e.g., from packet tracking devices. CEP systems consist of operators which create complex events (representing a situation) based on other events they collect or retrieve.

CEP systems have seen a change of perspective recently. While, originally, powerful operators were used in a central way to efficiently correlate events and detect situations, the emerging increase of event sources and event consumers has lead towards a decentralized handling of events [Pie04, LJ05, KKR10, SKRR10]. In addition, the collaborative nature of today's economy results in large-scale networks, where different users, companies, or groups exchange events. As a result, event processing networks are heterogeneous in terms of processing capabilities and technologies, consist of differing participants, and are spread across multiple security domains. However, the increasing interoperability of CEP applications raises the question of security [HSB09]. It is not feasible for a central instance to manage access control for the whole network. Instead, every participant wants to control access to the data it produces. For example, a company wants to restrict some pieces of the information it provides to a subset of authorized users (i.e., that are registered in its domain), while other information should be publicly accessible.

While current work in providing security for event systems covers confidentiality of data, authorization of network participants as well as encryption of event data [TKAR10, PEB07, BESP08], this is not sufficient for CEP systems. Since event information is used to create complex events, the provider of the original event information loses its influence on access control, once another participant processes it. During the processing, a new event is created based on the incoming information. The original information is obfuscated. Although access to the original information is heavily restricted, the provider's business partners can provide their correlated information freely to everyone, even if it is heavily dependent on the original data. This constitutes a security problem, since it enables recipients of the correlated information to infer the original event data.

For example, think about a logistics process, where a manufacturer wants to deliver an item to a certain destination (see Figure 5.1). Based on the destination, a shipping company determines a warehouse close to the destination, where the item will be shipped to for further delivery. This is done in an event processing system, where operators are hosted in the domain of each party and communicate the relevant information between each other (e.g. the item's *destination* is transmitted to the shipping company). If now a third party receives the information about the *warehouse*, it might be able to draw some conclusions about the original event data (i.e., *destination*), which is undesired by the manufacturer. Hence, we have to take care that access control ensures the privacy of information even over multiple processing steps. In our case, the access policy for the warehouse is influenced by the access policy of the destination.

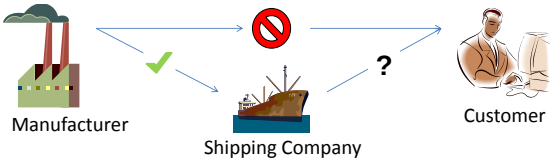


Figure 5.1: Access Control & Event Dependency

To ensure that in a multi-domain, large-scale CEP system, every producer of information can control the access to its information even over multiple correlation steps we specify an *access policy inheritance* mechanism to overcome this lack of security. The inheritance ensures that the privacy of original information is guaranteed, even if the information is processed through a chain of operators. Event producers can not only specify an *access policy* for the event information they provide, but also when the access policy can be ignored for certain events. Here, we take into account the *obfuscation* of information during the correlation processes, which has an important influence whether access policies need to be respected or not.

We define our system model and security goal in Section 5.2 and 5.3 respectively. Based on this, we present a general concept, how to achieve a policy consolidation algorithm respecting obfuscation of information in Section 5.4. Furthermore, we discuss the complexity of the general concept and, as a consequence, propose a local realization of the policy consolidation mechanism in Section 5.5. We present evaluation results in Section 5.6. Finally, we discuss related work in Section 5.7 and give a summary of the work in Section 5.8.

5.2 System Model

The system model used for the remainder of this Chapter is equivalent to the DHEP system model presented in Chapter 2.2.1. To ease the understanding of the chapter, the system model is recapped in the following.

We assume a distributed correlation network, where dedicated hosts are interconnected. On these hosts we deploy operators, which are executed to collaboratively detect situations and form the distributed CEP system. The cooperative behavior of the operators is modeled by a directed operator graph $G = (\Omega, S)$ which consists of operators $\omega \in \Omega$ and *event streams* $(\omega_i, \omega_j) \in S \subseteq (\Omega \times \Omega)$ directed from ω_i to ω_j. Thus, we call ω_i the event *producer* and ω_j the *consumer* of these events. Each event contains one or more event attributes which have discrete values. Every operator ω implements a function $f_\omega : I_\omega \rightarrow O_\omega$ that maps incoming event streams I_ω to outgoing event streams O_ω. In particular, f_ω identifies which events of its incoming streams are selected, how event patterns are identified (correlated) between those events, and finally how events for its outgoing streams are produced.

We illustrate the definition of a CEP system by means of our introduced logistic chain example, which will be used throughout this chapter. In the scenario, a manufacturer, a shipping company, and a customer each constitute a domain and each is providing an operator in its domain. The operators establish event streams as depicted in Figure 5.2. The manufacturer wants to send an item to one of its customers. Therefore, it sends events to a shipping company providing information about the

item's destination, its production place as well as the time when the product is ready. The shipping company receives these event attributes and uses them in a correlation function f_{sc}. The correlation creates a tuple of organizational information: the warehouse the item is going to be shipped to for further delivery, and the estimated day of delivery.

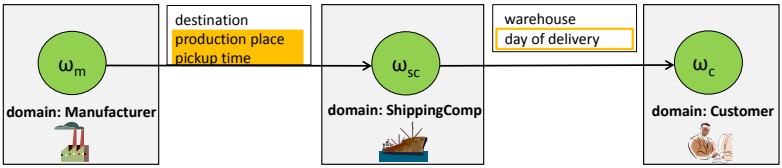


Figure 5.2: Attributes in Shipping Scenario

5.3 Access Control for CEP

The idea of our approach is to inherit access requirements, which are assigned to event attributes in an access policy, over a chain of event processing operators. During every processing of an event attribute, its obfuscation is calculated. Once a certain obfuscation threshold is reached during the further processing of the attribute, the attribute's access requirements can be ignored. In the following, we introduce the concepts we need to realize our approach, namely access policies and obfuscation policies, and define our security goal.

5.3.1 Access Policies

State of the art event processing systems provide basic mechanisms for access control. They allow to specify the access rights of subjects (operators) for the set of available objects (event attributes). These access

rights are usually provided by the owner of an object (e.g., the producer of an event stream) and may be granted to operators based on some *access requirement*. Such a requirement may be a role, a location, or a domain affiliation. As can be seen, the requirements are usually not direct *properties* of the operators, but of the hosts where the operators are deployed. Formally, we specify the access rights within an *access policy* AP for an operator ω as a set of (attribute, access requirement) pairs:

$$AP_\omega = \{(att_1, ar_1), ..., (att_n, ar_n)\}\,.$$

If there is no requirement specified for an attribute, it is accessible freely to any participant in the network. Note that we consider attributes to be distinct even if they use the same name, but are produced at two distinct operators.

An access requirement is a tuple of a property p, a mathematical operator op, and a value set val: $ar = (p, op, val)$, where $op \in \{=, <, >, \leq, \geq, \in\}$. val can be specified by a range or a set of values. For the sake of simplicity, in this paper access requirements are only referring to domain affiliation and have a structure like this:

$$ar_i = (domain, \in, \{domainA, domainB\}).$$

In our example scenario, the manufacturer's event attributes have different access requirements. While the information about the item's destination is accessible by the customer, information about where the item is produced and when it can be picked up is restricted to the shipping company.

Therefore, the attached AP is defined as follows:

$\mathrm{AP}_{manufacturer}$ =
{(destination, (domain,∈,{shippingComp,customer})),
(pickup time, (domain,=,shippingComp)),
(production place, (domain,=,shippingComp))}

If access policies are enforced and assured, a consumer is only eligible to receive an attribute if its properties match the access requirements defined for the attribute. In this case the consumer is trusted to use the attribute in its correlation function. We aim towards extending the semantics of this relationship. In our approach, the consumer is also trusted to adopt the requirements specified for the attribute in its own access policy for the correlated attribute resulting from the correlation function.

5.3.2 Obfuscation of Event Information

While access policies allow a producer to specify access requirements in a fine-grained manner, the inheritance of requirements to subsequent operators is also very restrictive and can limit the efficiency and applicability of the CEP system: If content is propagated or correlated over many steps, the number of access requirements will steadily increase and will prevent the access for more and more operators. This does not reflect the nature of event processing systems where basic events like single sensor readings may have only little influence on the outcome contained in a complex event representing a specific situation.

In our logistics example, the estimated day of delivery is calculated based on the *destination*, the *production place*, and the *pickup time*. As a consequence, the customer would have no access to the *estimated day of delivery* of the ordered item, since he does not fulfill the access requirements for *production place* and *pickup time*. Yet she has a reasonable interest in this information. And one may claim, that knowledge of the day of delivery does not necessarily allow to draw a relevant conclusion on the *production place* and *pickup time* attribute values. We say, the attribute values get *obfuscated* during the correlation process and depending on the achieved level of obfuscation, the access requirements of an attribute might no longer be needed. In our approach, the level of obfuscation is a measure, to which extent a consumer of the produced attribute (*estimated day of delivery*) can infer the value of the original attribute (*production place*). It can be easily seen in the example, that obfuscation is not only dependent on the values of the attributes, but also on the knowledge of the consumer. Since the *destination* value has led to the *day of delivery* as well,

knowledge of the destination would be of great help when trying to infer the restricted attribute *production place* because the delivery time of the item is probably related to the distance between destination and production place. In this work, we model the achieved obfuscation between two attributes as a function $obf(att_{old}, att_{new}, \omega_{req})$ returning the obfuscation achieved for a pair of attributes for a requesting operator. We elaborate event obfuscation and its theoretical background in Section 5.4.

Since attributes can and will get obfuscated during the correlation processes, every operator can also specify with its access policy an obfuscation policy. The obfuscation policy contains obfuscation thresholds for every attribute the operator produces. During the processing of an event attribute, its obfuscation is calculated. Once the threshold is reached during the further processing of the attribute, the attribute's access requirements can be ignored. Formally, we define the obfuscation policy OP for an operator ω as a set of (attribute, obfuscation threshold) pairs:

$$OP_{\omega} = \{(att_1, ot_1), ..(att_n, ot_n)\}\,.$$

For instance, the obfuscation policy

$$OP_{manufacturer} = \{(destination, 0.9)\}.$$

states, that the obfuscation threshold of 0.9 has to be exceeded for the *destination* attribute in order to ignore the access requirements for it, and therefore to be freely accessible. The semantics of the obfuscation value is further described in Section 5.4.

5.3.3 Security Goal

Let $att_{old} \rightarrow_{\omega} att_{new}$ denote that i) at some operator $\omega \in \Omega$, att_{old} is taken as input to the correlation function f_{ω}, and ii) f_{ω} produces att_{new} in dependence of att_{old}. Furthermore, let $att_{old} \rightarrow^{*} att_{new}$ denote the transitive closure of the dependency relation. For any pair of attributes with $att_{old} \rightarrow^{*} att_{new}$ we say that att_{new} is *dependent* on att_{old}. Our main goal is to preserve the privacy of event attributes over multiple correlation steps by respecting the dependency relationship between the attributes

produced by the CEP system. In particular, access requirements must not be applied solely to the attribute att_{old}, but have to be inherited to all dependent attributes (att_{new}) unless a sufficient obfuscation threshold for att_{new} has been reached.

More formally, given for each attribute att an initial set of access requirements denoted by $AR_{init}(att)$. We require for any policy consolidation algorithm two conditions to be met:

Condition 1 *For all attributes* $att \in O_\omega$ *produced at* ω

$$AR_{init}(att) \subset AP_\omega. \tag{5.1}$$

Condition 2 *For all dependent attribute pairs*

$(att_{old}, att_{new}) \in \rightarrow^*$ *with*

1. ω_i *has produced* att_{old} *with access requirement* $AR(att_{old})$ *and obfuscation threshold* $(att_{old}, x) \in OP_{\omega_i}$,
2. att_{new} *is produced by* ω_j
3. att_{new} *is consumed by* ω_k

the access requirement in AP_{ω_j} *yield*

$$AR(att_{old}) \subset AP_{\omega_j} \text{ iff } obf(att_{old}, att_{new}, \omega_k) < x. \tag{5.2}$$

A policy consolidation algorithm needs to ensure Condition 1 and Condition 2 in the presence of adversaries who try to derive event attribute values they are by policy not allowed to access directly.

An adversary can execute any operator of the CEP system. In doing so, the adversary is however limited to the behavior described in the system model. The adversary needs to be authenticated and can only access streams according to its properties. Furthermore, the derived event output follows the operator specification as well as follows the access requirements for each executed operator. Hence, each adversary is bound to analyzing outgoing event streams, which it is allowed to access, for inferring any additional information.

In the following, we first introduce our concept for distributed access policy composition in Section 5.4 which ensures that all access requirements are inherited for dependent attributes. In Section 5.5 we provide an implementation of our approach. We introduce means in measuring obfuscation and we show how measuring obfuscation can be used to reduce the number of access requirements in a scalable manner.

5.4 Policy Consolidation and Event Obfuscation

In this section we describe the basic concepts to achieve the security goal for a CEP system (as specified in Section 5.3.3). We assume a CEP system which is capable of establishing event streams between producer/consumer pairs. To establish an event stream, a consumer requests an event or event attribute. We assume that the request is always handled by the producer of an attribute. The consumer authenticates itself towards the producer. Based on this information, the producer handles the request, and we will elaborate the alternative request handlings in the following subsections. The assumed system behavior can, for example, be achieved by a Publish/Subscribe system.

We present our general concept by means of a new consumer requesting the *warehouse* attribute from operator ω_{sc}. The operator implements a correlation function $f_{\omega_{sc}}$, which maps events containing the attribute *destination* to outgoing events containing the attribute *warehouse*. The producer of the incoming attribute *destination* specified an access policy which requires that a recipient of *destination* must be within domain *shippingComp* or *customer* and an obfuscation policy which allows the requirement to be ignored for an obfuscation value of at least 0.9. Operator ω_{sc}, which is located in domain *shippingComp*, does not provide any additional access requirements for *warehouse* and thus no obfuscation threshold for it, but it has to respect the access policy from its predecessor. The new consumer wants to have access to the *warehouse* attribute. In general, we have to distinguish between two cases:

1. The consumer is within domain *shippingComp* or *customer*.
2. The consumer is not within domain *shippingComp* or *customer*.

In the first case, the consumer is allowed to access the *destination*, and since there is no additional requirement for the *warehouse*, it is also allowed to access the new attribute. Thus, it will receive every event produced by ω_{SC}. Furthermore, the new consumer has to ensure access policy inheritance if it further correlates the *warehouse* information.

In the second case, it will not have access to *warehouse*, unless the obfuscation of 0.9 is achieved. This is decided on a per-event basis, since obfuscation can depend on individual attribute values and differs for every event instance. Hence, the consumer will only receive a subset of the events containing the attribute produced by ω_{SC}. Apart from that, the consumer has no commitment for access policy inheritance, since every event it receives is freely available.

As can be seen, the two cases comprise the two main concepts we like to address: the inheritance of access policies as well as obfuscation calculation. These two concepts are now discussed.

5.4.1 Access Policy Inheritance

Even if access policies can be defined in a CEP system, information contained in the event stream of a producer may be inferred by hosts which do not fulfill the access requirements. The correlation function f of an operator can map data contained in an attribute to another attribute inside a newly produced event. That means the content provided by an event may be propagated over a chain of operators using event streams whose access cannot be controlled by the original producer.

In order to establish access control in complex event processing networks, it is essential that producers do not solely control the access of their own produced attributes, but also of the attributes which depend on them. We introduce an inheritance mechanism, which inherits access requirements of attributes to other attributes dependent on them. This ensures that every producer has full access control over the event information it produces, even if the information is used or transformed in a subsequent operator within the CEP network.

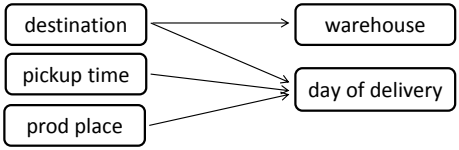


Figure 5.3: Dependency Graph of the Shipping Company Operator

Access policy inheritance consists of two basic conceptual steps: First, we have to identify the dependencies between attributes. Second, we have to map all access requirements specified for these attributes to their dependent outgoing attributes in the access policy. In the following, both concepts are discussed. We present our realization of these concepts in Section 5.5.

Creating a Dependency Graph

Since access policies are respecting access requirements from dependent attributes in our approach, these attribute dependencies have to be detected. We do that by analyzing the correlation function f_ω of every operator ω, once the operator is deployed in the system. For each outgoing attribute, our system looks up f_ω for the incoming attributes that are used for its creation.

Basically, all dependencies $att \rightarrow^* att_{new}$ for every outgoing attribute are modeled in a graph. In the policy creation phase, we can determine in the policy creation phase i) which access requirements need to be inherited and ii) how multiple access requirements need to be merged for the outgoing attributes. In our example, the warehouse is determined based on the item's destination (cf. Figure 5.3). The day of delivery of the item is estimated, based on the destination, the production place, and the pickup time.

Creating new Access Policies

In accordance with the dependency graph a policy composition mechanism is initiated, which generates the new access policy based on the received access requirements of *all* dependent objects. That means, the requirements have to be included in the newly generated access policy. Basically, the mechanism assigns the access requirements from all attributes $att \in att \rightarrow^* att_{new}$ to the new attribute att_{new}. Finally, the new access policy will be merged (i.e., optimized). Consolidating the policies will cause that new access policies steadily grow in size, since requirements of preceding operators are integrated. However, a new access policy may contain attributes with multiple associated access requirements, which will unnecessarily increase the access policy's size. Hence, we will merge access requirements associated to the same attribute and eliminate unnecessary requirements from the access policy. In our example scenario, the operator that determines the warehouse needs to map the requirement

$$(domain, \in, \{shippingCompany, customer\})$$

which is associated to the *destination* in $AP_{manufacturer}$ to the *warehouse* attribute, since the *warehouse* is dependent on the *destination*. Hence, only operators hosted by the shipping company or the customer could access the attribute. After creating the new access policy for the shipping company operator, the access policy contains the following entries:

$AP_{shipping}$ =
{(warehouse, (domain,∈,{shippingComp,customer})),
(day of delivery, (domain,=,shippingComp)),
(day of delivery, (domain,=,shippingComp)),
(day of delivery, (domain,∈,{shippingComp,customer}))}

Finally, the restrictions referring to the same attributes (*estArrivalTime*) are merged, resulting in the access policy:

$AP_{shipping}$ =
{(warehouse, (domain,∈,{shippingComp,customer})),
(estArrivalTime, (domain,=,{shippingComp}))}

5.4.2 Obfuscation Model

If a consumer does not fulfill the access requirements, she can still get access to an attribute. As defined in the security goal, we have to check, whether a specified obfuscation threshold is reached. This can only be decided if we take a closer look at i) the correlation function's in-/output and ii) the knowledge of the consumer. We have to analyze how likely it is that the consumer can infer on the incoming attribute.

Obfuscation is typically highly dependent on the correlation function, i.e., how it creates outgoing events based on the incoming events. We exemplary show this with two basic operators found in all major CEP systems: a *filter* and an *aggregator*. A filter is a common, often used operator and its correlation function is pretty simple: For every incoming event it is checked whether one or more attributes have a certain value or are within a certain value range. If this is the case, the events are forwarded to all requesters of the filter operator. Since the values of the incoming attributes match the output attributes, there is no obfuscation of event information. For every received attribute, the requester can directly infer the values of the original, incoming attributes.
An *aggregator* is also a very common, but more complex operator. Typically, it collects some events before producing any output. After a certain time window, or a certain amount of events have been collected, the aggregator combines the attribute values of the incoming events for a newly created output. As can be seen easily, the original values from the incoming attributes get obfuscated during the aggregation. The requesters of the aggregated output can not directly infer the incoming attribute values. Depending on the aggregation function she might still guess, though, that some values of incoming attributes are more likely to have occurred than others.

We already stated in Section 5.3.2, that the knowledge of the requester plays an important role for event obfuscation. If an attribute is correlated based on two or more incoming attributes, knowledge about one of these incoming attribute values would be of great help for a consumer of the correlated attribute when trying to infer another incoming attribute. As a result, we have to integrate the knowledge of every user when specifying

the inference of an event attribute. We model this knowledge as a function $known_{\omega_r}(I_\omega)$, which returns the set of input attributes from I_ω known to the recipient ω_r, i.e., $known_{\omega_r}(I_\omega) \subseteq I_\omega$.

Discussing the Semantics of Obfuscation

By dealing with obfuscation, our goal is not to influence or control the network participants in how they use the CEP system. In particular, we can not influence the domain of an attribute, i.e., its possible values. This is important for the semantics of attribute obfuscation. One may claim that obfuscation of an attribute is related to the number of possible values the attribute can have. The reason is, that the more attribute values are possible, the smaller is the chance of guessing the correct value. However, in a distributed, heterogeneous CEP system we have no influence on the number of possible values (e.g., increase it on purpose by dummy values). Furthermore, we have no control about knowledge of operators. For example, whether an operator knows the (semantics of a) correlation function implemented at another operator, or the possible values an attribute might have.

Therefore, we have to make strict assumptions for obfuscation measurement: We assume a requester of an attribute att_{out} has knowledge about i) the semantics of a correlation function producing att_{out} and ii) the possible values of the unknown event attributes att_{out} is dependent from. Based on these assumptions, we measure obfuscation of an unknown event attribute as the equality of likelihood of the different values it can have. That means, if a consumer cannot, based on its knowledge, make any conclusions on the correct value of an attribute, the attribute is perfectly obfuscated.

The obfuscation is perfect if the probability for all possible values of the incoming attribute is equal, i.e., the recipient can not infer on a value because it was *more likely* to have occurred. Then, maximum obfuscation is achieved. Consequently, if there is only a single possible value for the incoming attribute, and the user can directly infer on that value, the achieved obfuscation should be 0.

Formalizing Obfuscation

The discussed characteristics lead us to a formalization of attribute inference as a probability value. This inference probability is known as the Bayesian inference and gives us an answer to the question: *Given a certain output attribute, and a certain set of input attributes the consumer knows, how likely is a specific value for the incoming attribute we need to secure?*

The obfuscation is perfect if the probability for all possible values of the incoming attribute is equal, i.e., the recipient can not infer on a value because it was *more likely* to have occurred. In this case, the entropy of the distribution is 1. Consequently, if there is only a single possible value for the incoming attribute, and the user can directly infer on that value, the entropy is 0. That means, no obfuscation is achieved, hence it is also 0.

We define that $I^*(att_{new})$ is the set of att_{old} for which $att_{old} \rightarrow^* att_{new}$. Then the inference probability ip of an attribute $att_{old} \in I^*(att_{new})$ used to correlate att_{new} for a requester ω_r is the conditional probability distribution of att_{old}:

$$ip(att_{old}, att_{new}, \omega_r) = \\ P(att_{old}|known_{\omega_r}(I^*(att_{new}) \backslash I^*(att_{old})), att_{new})) \tag{5.3}$$

As one can see, the inference probability is not a single probability value, but a probability distribution over the value set of the inferred event attribute att_{old}. Furthermore, ip is not only dependent on the operator function, but also on the knowledge of the recipient.

Based on the inference probability, we can now measure the obfuscation value achieved for the incoming attribute att_{old} with respect to the outgoing event att_{new} by calculating the entropy of the inference probability distribution:

$$obf(att_{old}, att_{new}, \omega_r) = H(ip(att_{old}, att_{new}, \omega_r)) \tag{5.4}$$

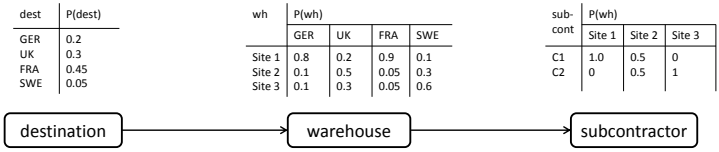


Figure 5.4: Bayesian Network consisting of Topology and Conditional Probability Tables

The entropy serves as the desired measure for obfuscation. If the probability for all possible values of the incoming attribute is equal, the entropy of the distribution is 1. Consequently, if there is only a single possible value for the incoming attribute, the entropy and therefore the achieved obfuscation is 0.

5.4.3 Measuring Obfuscation

To measure the obfuscation between two attributes a Bayesian Network is used, since it answers probabilistic queries about the attribute inference [RNC+95]. Before being able to query the Bayesian network, it needs to be trained.

Training a Bayesian Network is typically done in two phases: Structure learning and parameter learning. While parameter learning is done by observing the events of interest, structure learning tries to find the optimal network structure of the observed events, which is NP-hard. However, in our approach, we can skip the structure learning phase, as we model the structure of the Bayesian Network based on the event dependency graph. Every event attribute represents a variable (i.e., host) in the Bayesian Network and every dependency between attributes represents an edge. However, in addition to the dependencies of event attributes, every Bayesian Network associates a probability function with an event attribute. The probability distributions are created during the parameter learning phase. By analyzing processed events, a Bayesian Network *trains* about the probability distribution of attribute values. More specific, the training algorithm checks which particular attributes were used by the

creation of another attribute. Based on these observations, probability tables are created for every event attribute. (cf. Figure 5.4).

Once the Bayesian network is trained, it can be *queried* about the inference probability of certain attributes. Querying means to provide information about some known event attributes and to calculate the conditional probability distribution of the unknown event attributes. For example, by providing the occurrence of a certain event and its attributes, one can estimate the most probable attribute values that are going to be created in the next correlation. Fitting to our needs, we can also query the network the other way round: By providing information about the observed attribute outcome, we can query the probability distribution of the attribute values that have led to the observed outcome. In particular, we can query the inference probability $ip(att_{old}, att_{new}, \omega_r)$ by telling the Bayesian network the observed values for $known_{\omega_r}(I^*_\omega)$ and att_{new}.

5.4.4 On Parameter Learning

In this subsection, we briefly discuss parameter learning, as our approach is not able to work unless some parameter learning is done. However, parameter learning algorithms constitute a large research field for itself. And since it is not directly related to our actual contribution, we only give a short overview.

As indicated earlier in this section, parameter learning is used to create the conditional probability tables for each event's parameter values on every host in the network. Two driving factors play an important role in how parameter learning affects the the obfuscation calculation:

1. Accuracy of initial parameter learning
2. Adjusting probabilities in long-running environements

The initial learning has an effect on the timespan the system has to learn, until it has computed acceptable probabilities in order to run obfuscation

calculation. This is defined by a threshold value specifying the accuracy of the learning. This threshold is highly dependent on the application and what its users consider as acceptable. The point, when this acceptance level is reached, is furthermore highly dependent on the number of incoming events and the distribution of the parameter values. The calculation of the probability distributions is often following a maximum likelihood approach, which can be adapted in various ways dependent on application knowledge to enhance the results. The basic course of the approach is to look at the currently collected event data, and then searching for probability distributions that describe the collected data best. Many variations of this basic approach are existent. For example, to overcome the problem of defining the probability of unobserved parameter values, the expectation-maximization algorithm has been designed. It alternates between computing values for the unobserved parameters and maximizaing the likelihood. In our evaluation, we made use of the Weka data mining tool [HFH+09] which integrates multiple learning algorithms. However, it has to be stated that considering the exact distribution of parameter values is not known in our system, it is not possible to state concrete threshold values indicating the acceptable starting point for calculating obfuscation. Instead, this has to be an application specific value which needs to be set up by domain experts. For example, if the underlying system is working on business processes that occur on a daily basis, it might be needed to integrate the data of at least one working day in the learning process to achieve an acceptable knowledge base.

Once the application is running and working on the learned probability distribution, it may be necessary to adjust the values over time. This is especially true in long running applications, where the probabilities may shift over time. Keeping the probability tables up-to-date is typically solved by using a windowing mechnism, which rates newer data higher and finally forgets about outdated data. Those techniques are also integrated in the data mining tool we make use of in our evaluations.

5.4.5 Complexity Analysis

While it is easy to verify that the introduced mechanisms fulfill our security goals, the naive application of Bayesian networks does not allow for a scalable usage of CEP. Calculating the inference probability is NP-hard, adding a potentially infeasible amount of additional latency to the event processing, if the size of the Bayesian network is large.

To meet the large computational effort of calculating Bayesian inference, two different types of optimizations exist. On the one hand, approximation algorithms exist which use sampling techniques to estimate the conditional probabilities of the Bayesian network (e.g., [GG84, GS90]). Their precision depends strongly on the number of samples taken from the network, and it can be shown that no approximation scheme exists that allows to draw samples in polynomial time to achieve a certain precision, which makes the approximate algorithms infeasible for security applications (since no guarantees on security can be made in appropriate time) [RNC+95]. On the other hand, optimizations exist which try to reduce the complexity of calculating *exact* inference by storing partial results of the inference calculation which otherwise would have to be calculated multiple times (e.g., [ZP96, LS88]). Depending on the structure of the Bayesian network, the optimizations can reduce the time-complexity. This is especially beneficial for single connected Bayesian Networks (cf. Figure 5.5), where space- and time-complexity can be reduced to be linear to the number of attribute values. This is a great benefit for networks with a limited number of attributes, which themselves have possibly large domains.

Besides the computational effort, creating and querying a Bayesian Network also requires communication effort if the sources providing the needed information are spread among a network. All information about processed events need to be gathered since the training algorithm has to check which particular attributes were used by the creation of another attribute. Furthermore, in order to calculate obfuscation over a chain of operators, additional communication among operators will be needed, since the inference probability relies on the *knowledge* the consumer has (cf. Equation 5.3). Additionally, we would have to take a huge effort in order to identify the

exact events which have led to an attribute value for which we want to calculate obfuscation.

As can be seen, the time and communication needed to calculate the inference probability can be huge because for the measurement all dependent event attributes need to be considered ($I^*(att_{new})$). Hence, calculating the inference for the transitive closure of dependable attributes is not scalable. We tackle this by dividing the problem, and therefore the Bayesian Network, in multiple parts that allow to be treated independently.

5.5 Scalable Access Policy Consolidation

Instead of accounting for a global Bayesian network, we propose to exploit local knowledge available at each host. This allows us to reduce the number of relations of incoming and outgoing attributes and thus leads to a big reduction of processing costs. The idea of our approach is that a host in the CEP network creates a local Bayesian network for each of its deployed operators. The handling (i.e. forwarding) of the event is based on the locally achieved obfuscation. This limits the computational effort by accepting that obfuscation is not measured over multiple correlation steps. The consequence of this is, that some events may be treated more restrictive than actually needed. Since event information will only get more, but never less, obfuscated during the processing of the event, the locally measured obfuscation is always lower or equal than the actual global obfuscation.

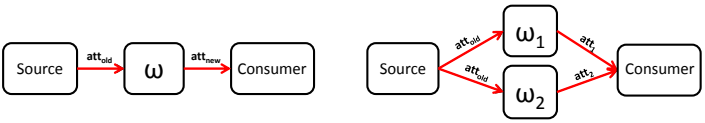


Figure 5.5: Single- and Multiply-Connected Correlation Networks

5.5.1 Measuring Local Obfuscation

In our approach, every host calculates obfuscation only for the locally known attribute dependencies (i.e., $att_{old} \rightarrow_{\omega} att_{new}$) in contrast to calculating the obfuscation for every pair of dependent attributes (i.e., $att_{old} \rightarrow^{*} att_{new}$). This has three major benefits: i) a smaller dependency graph, ii) less communication overhead, and iii) the local graph is not multiply connected because there exist only paths with length 1. As a consequence, every host can create a *local dependency graph* on its own instead of creating a global dependency graph for all dependent attributes. Furthermore, we can efficiently calculate the exact inference probabilities by applying optimizations for single connected networks to efficiently determine the obfuscation value (cf. Section 5.4.5).

However, even in a local approach to calculate obfuscation one has to consider the multi-path dependencies of attributes. These are attributes that might reach the recipient via multiple paths (i.e., parallel chains of operators in a multiply-connected correlation network, cf. Figure 5.5). An adversary that can subscribe to such attributes may be able to infer the original value by combining the event information received through the multiple paths.

Algorithm 4 illustrates the approach to determine local obfuscation. At initialization of the algorithm, multi-path dependencies between all possible attribute pairs are detected by analyzing the entire operator graph. For every attribute pair with multi-path dependencies the operators that reside on distinct paths exchange the dependency functions w.r.t. the attributes. For example, in a scenario as depicted in Figure 5.5, the inference probability is calculated as follows:

$$P(att_{old}|att_1, att_2) = \alpha * P(att_{old}) * P(att_1|att_{old}) * P(att_2|att_{old})$$

where α is the normalization constant $1/P(att_2)$.
Hence, $P(att_1|att_{old})$ is sent to operator ω_2 and $P(att_2|att_{old})$ to operator ω_1 vice versa.

After performing the initialization each operator can calculate the obfuscation value from local knowledge only. In the above example, if operator

Algorithm 4 Local Obfuscation Calculation

```
procedure INITIALIZE(ω)
    for all operator ω do
        D_ω ← FINDMULTIPATHOPERATORS(ω)
    end for
    for all ω ∈ D_ω do
        relAtts ←FINDRELATEDATTRIBUTES
        for all relAtts do
            TRANSMIT P(att_new|att_old) TO ω
        end for
    end for
end procedure


procedure UPONRECEIVEEVENT(e)
    for all att in e do
        if ∃ multPathDependency(att) then
            CALCULATEWORSTCASEOBFUSCATION(ATT)
        else
            CALCULATELOCALOBFUSCATION(ATT)
        end if
    end for
end procedure
```

ω_1 now calculates the obfuscation of an incoming attribute att_{old} for the outgoing attribute att_1, it uses the dependency functions received during the initialization phase. There, it searches for the outcome att_2 which has the highest chance for inferring att_{old}, i.e. the entry with the highest probability. This value is then used in the calculation of $P(att_{old}|att_1, att_2)$, as it results in the minimal achievable obfuscation.

Note that the initialization needs to be performed with each change of the correlation graph and follows the learning phase of the Bayesian networks. However, for many practical settings, changes to the operator graph are typically a result of changes to the business logic. Hence, we expect

only infrequent changes which might interrupt of the event processing service.

5.5.2 Correctness

As our work addresses mainly how to establish producer centric access policies in CEP in a scalable way, we give only informal correctness arguments under the limitations for the adversary introduced in Section 5.3. Three main properties guarantee, that the proposed approach is correct in terms of the defined security goal:

1. According to our assumptions in Section 5.3, an adversary tries to infer additional information by analyzing all event streams which it is allowed to access. The proposed algorithm considers the complete knowledge the recipient *might* have. That means, it is considered that every attribute influencing the requested local obfuscation ($obf(att_{old}, att_{new}, \omega_r)$) that is accessible to the consumer is known.
2. In accordance to property 1, every path from att_{old} to att_{new} is considered in the algorithm. That means, every piece of information an adversary may access in order to infer att_{old} is included when calculating the inference.
3. Locally unknown events (which may occur in multi-path dependency calculations) are always handled as a worst-case consideration. We always use the value in our calculations which would give an adversary the most inference information, i.e., the value resulting in the worst obfuscation.

Since all sources of event information which might influence the obfuscation value of any operator are considered in our approach, the obfuscation value calculated at an operator cannot be lowered further by any means.

Hence, with the presented approach, we guarantee: If the consumer does not fulfill the access requirements for an attribute att_{old}, it will also not be able to access any attribute att_{new} if the attributes depend on each other ($(att_{old} \rightarrow^* att_{new})$ unless a sufficient obfuscation threshold for att_{new}

has been reached. We do not guarantee, though, that the consumer will receive every attribute that has achieved a sufficient obfuscation.

5.6 Discussion and Evaluation

We implemented the presented approach within the DHEP framework [SKRR10] which enables complex event processing in a heterogeneous environment. That means, hosts may be spread among different security domains and have differing processing capabilities or use different correlation engines. Hence, using the framework allows us to create multi-domain distributed CEP networks.

To achieve policy consolidation, every operator receiving a request provides the requester with the information needed for further processing: the access policy as well as the obfuscation policy. The policies might be different depending on the consumer (see Section 5.4). The events a consumer receives as well as its adherence to access policy inheritance is dependent on whether it fulfills the access requirements. To realize the obfuscation measurement we make use of the Weka framework [HFH+09]. Weka is a data mining tool which comes with a Bayesian network implementation. Furthermore, it allows us to add hidden variables which is needed to compute multi-path inference as discussed in Section 5.5. Every host in our framework runs its own implementation of Weka. For every event attribute produced, we calculate the achieved obfuscation and forward it to potential recipients in accordance to the obfuscation. Weka does not provide any optimization for calculating the Bayesian inference. Instead, it uses the naive full calculation. To measure the computational effort of the *variable elimination* optimization (see Section 5.4.5), we provide an own algorithm implementation. In the following, we discuss the results of our evaluations.

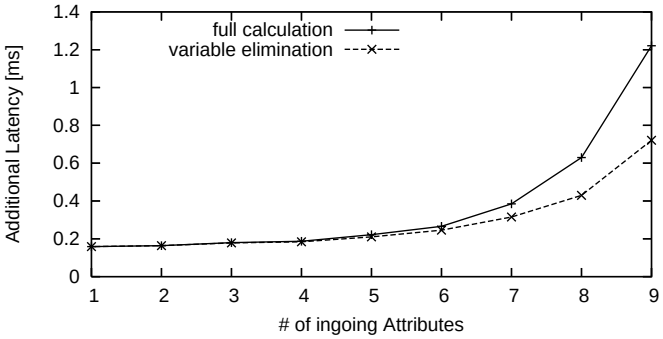
Influence on Network Usage and Latency

Introducing access policy consolidation will cause a reduction in network usage. This is due to the fact that both number and size of events decrease

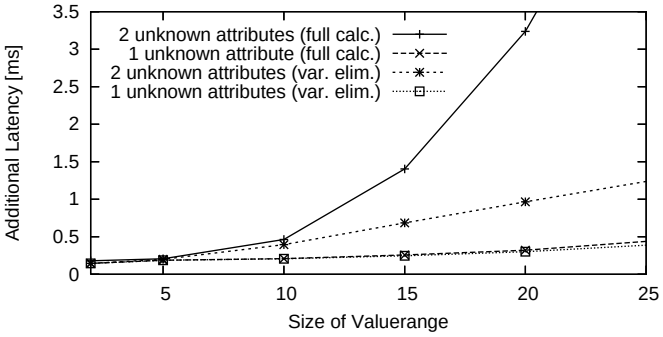
because not all events or event attributes will be received by an operator. However, it can be seen easily that this reduction is fully dependent on the application characteristics, especially on the access rights of the operators and the frequency distribution of event attribute values. Therefore it is not possible to provide meaningful evaluations and we focus on evaluations of the additional latency caused by our approach.

Despite reducing the network usage, policy consolidation will also cause additional latency for event processing on the network hosts. Although we can reduce the computational effort by only considering local obfuscation, the computation still takes a considerable amount of time. The computational effort is mainly dependent on the size of the Bayesian Network and the number of consumers, since different consumers can have differing obfuscation. We analyzed the additional latency caused by our policy consolidation mechanism both dependent on the number of input attributes as well as the number of attribute values. For our evaluations, we create a simple setup, where one operator receives events containing one attribute. In our evaluations, both the size of the attribute domain as well as the number of event sources vary. The operator is hosted on a machine with a 2GHz CPU and 3GB main memory, where the introduced Weka framework (as well as our external optimization algorithm) is deployed. The incoming events are processed by an ESPER correlation engine which creates an output event, containing one attribute, once events from all sources are received. For the newly created event, we calculate the achieved obfuscation for a consumer. We extracted and depicted only the time needed for calculating inference in the Bayesian Network, since it is the main source for additional latency caused in our approach.

Figure 5.6a depicts the additional latency depending on the number of event sources. The number of event sources has a direct influence on the size of the locally created dependency graph, hence on the size of the Bayesian Network. No incoming attribute was known to the consumer. The size of the attribute domain was fixed to two, meaning that every event attribute was boolean. The results show that the increase of the latency, caused by the computation of obfuscation values increases exponentially with the total number of attributes. This behavior is expected, as we already discussed in Section 5.4.5. However, computations are fast for networks with a small number of attributes, as they are common in



(a) Latency vs. Attributes



(b) Latency vs. Size of Domain

Figure 5.6: Measuring additional Latency

many CEP applications. Since security-related event systems have, depending on the network and event parameters, a processing time in the range of one millisecond and more per event [SL05, RR06, TKAR10], we consider a latency of up to 1ms as acceptable for our approach.

In our second evaluation, we leave the number of event sources fixed at two but varied the domain size (cf. Figure 5.6b). Furthermore, we calculate the achieved obfuscation for two different consumers. One consumer has no knowledge about any incoming attribute, while the other has knowledge about one incoming attribute. We calculate the obfuscation for the other event attribute, which both consumers might try to infer. As can be seen, the optimized algorithm proves to be significantly faster than the standard calculations, if there is more than one unknown event source. It can be seen, that the standard calculation causes a high amount of additional latency, when the size of the attribute domains increase. The variable elimination reduces the complexity to be linear dependent on the size of the attribute domain, and our approach benefits heavily from it.

Our results show that the additional processing time is highly dependent on the number of unknown attributes in the dependency graph as well as the number of potential values each of the unknown attributes might have. It can be seen, that the size of the attribute domain is less critical than the number of attributes. This fits well with many CEP systems, where it is unusual to correlate events from many different sources, but rather have a limited number of sources with potentially large attribute value ranges. We conclude that obfuscation calculation is reasonable if the application characteristics allow for it. The calculation may not be feasible for applications with very high event rates, since measuring the obfuscation would take too long and the processing of events would be slowed down. One solution for this kind of applications may be to calculate a static, worst case obfuscation instead of calculating the obfuscation for every new event.

5.7 Related Work

Due to the increasing popularity of event-driven systems, a lot of effort has been spent to make the systems secure. For example, a proposal of a role-based access control is stated in [Pie04]. Pesonen et al. and Bacon et al. discuss, how Publish/Subscribe systems can be secured by introducing access control policies in a multi-domain architecture [PEB07, BESP08]. They describe how event communication between the domains can be supported. Opyrchal et al. present the concept of event owners, that can be specified. These are used to provide access to *their* events [OP01]. Tariq et al. propose a solution to provide authentication and confidentiality in broker-less content-based Publish/Subscribe systems [TKAR10]. Our work is based on the previous work which make event communication secure among different entities in the system. We assume the presence of a system that can handle access control on events. Based on this, we use policy composition in order to derive the necessary access policies at any point during the event processing steps.

Access policy composition has found a lot of consideration in distributed systems. Bonatti et al. defined a well recognized algebra for composing access policies [BDCdVS02]. Especially in the area of web service composition, the composition of security policies plays an important role, as different policies have to be combined for every combination of web services (e.g., [ST11, LBOG06]). We adopt concepts from access policy composition into our distributed CEP system. This allows us to inherit access restrictions during the different processing steps in the various operators of our system.

To realize our concepts we make use of techniques from statistical inference. More specific, we calculate the Bayesian inference after creating a Bayesian network and learning the dependencies (e.g., [Lee12, RNC+95, HFH+09]). Since Bayesian inference is a complex calculation, several Monte-Carlo algorithms have been proposed to estimate the inference value(s). They all have in common to arbitrarily pick samples from the Bayesian network probability distribution, and estimate the values based on the samples. The precision of the estimated inference values is dependent on the number of samples. A commonly used technique is the Gibbs

sampler [GG84, GS90].

5.8 Conclusion

This chapter has addressed the inheritance and consolidation of access policies in heterogeneous event processing systems. We identified a lack of security in multi-hop event processing networks and proposed a solution to close this gap. More specific, we presented an approach that allows the inheritance of access requirements, when events are correlated to complex events. Our algorithm includes the obfuscation of information, which can happen during the correlation process, and uses the obfuscation value as a decision-making basis whether inheritance is needed. We implementated our approach, based on Bayesian Network calculations, to evaluate the overhead it imposes on CEP systems. Our analysis and evaluations show that the approach is computation-intensive, once the Bayesian Network grows, hence raising the processing time of an event. To deal with the calculation cost we introduced a local approach, where every participant calculates local obfuscation based on local knowledge achieved during the correlation process. Furthermore, we made use of the *variable elimination* optimization, to further reduce the computational effort for calculating obfuscation values. This approach constitutes a pioneer step for the enhancement of the security in multi-domain CEP systems by using obfuscation calculation. By increasing the Bayesian Network size or making use of even more suitable optimizations, one could be able to measure obfuscation efficiently over more than one correlation step.

6 Conclusion

This chapter gives a summary based on the results and contributions of this thesis. Furthermore, a brief outlook is given on possible future research directions.

6.1 Summary

With the constant rise in event generating and processing entities, complex event processing will be a key technology for many applications. Considering the interconnections resulting from modern, world wide collaboration of users, companies, or group of interests, the creation, management, and security of heterogeneous event processing solutions is mandatory.

In this thesis, we surveyed the current state of the art correlation systems and identified the lack of support as soon as event processing is established among multi-domain infrastructures. Especially the available optimization algorithms were unable to perform well in heavy constraint-driven networks. Furthermore, we discovered the lack of privacy guarantees for event information when events are processed over multiple hops in a multi-domain network.

Based on this observation we concluded the need to establish a secure, efficient distributed event processing system.

We developed the complex event processing system DHEP, which has been designed and created to close the gap between current CEP systems and business requirements. The concepts behind DHEP focused on providing a very modular architecture to support the variety and heterogeneity of current CEP solutions. This includes the possibility to interconnect various different event processing engines, and enables communication among

them within a distributed system. Moreover, DHEP comes along with a modeling component. The modeling is established by a powerful object-oriented definition language that enables efficient, tool-aided designing of big CEP applications. We extensively evaluated the DHEP system and showed that, although the functionality provided by DHEP imposes additional cost, the system scales well by exploiting the distributed detection of situations.

To allow for an efficient complex event processing in multi-domain infrastructures, we addressed the placement of operators in heterogeneous event processing systems. We focused on large-scale systems with many participants, where it is important to cope efficiently with many constraints. In particular, we presented an approach to minimize network utilization. The algorithm can adapt to dynamics that stem from load variations as well as to changes in the operator deployment. The analysis and evaluations show that the approach is particularly beneficial in heavy constraint settings. Here, it finds near optimal placements with low cost by efficiently restricting the search space of possible placements. Furthermore, the separation in two phases – the initial placement phase and the subsequent optimization phase – contributes to a fast operation of the event processing which is crucial in the practical operation of event processing networks. Moreover, in a setting where enough resources are available the algorithm ensures that a solution is found fast and at low overhead.

Finally, we addressed the security issues of complex event processing in multi-domain infrastructures. We proposed a solution to overcome the lack of privacy of event information by introducing inheritance and consolidation of access policies in heterogeneous event processing systems. Specifically, we presented an approach that allows the inheritance of access requirements, when events are correlated to complex events. This is established by investigating the obfuscation of event information during the correlation process. This obfuscation measure is then used as a decision-making basis whether inheritance is needed.

The implementation of our approach is based on Bayesian Network calculations. The analysis and evaluations show that the approach is computation-intensive, once the Bayesian Network grows, hence raising the processing time of an event. To deal with the calculation cost we introduced

a local obfuscation measure, where every participant calculates local obfuscation achieved during the correlation process. Furthermore, we made use of a *variable elimination* optimization, to further reduce the computational effort for calculating obfuscation values.

In conclusion, the presented approaches overcome the lack of support for multi-domain infrastructures current CEP solutions suffer from. We introduced a framework as a foundation for heterogeneous event processing. Based on this, we presented conceptual and algorithmic solutions to overcome the lack of efficiency and security which inherently comes along with CEP in multi-domain infrastructures.

6.2 Further Research Directions

There are several possibilities to extend the work presented in this thesis. This can be subdivided by the two main approaches presented. In the following, we will first address enhancements to increase the efficiency of the operator placement, and then discuss further extensions for the security solutions and access policy inheritance.

Operator Placement Our experiments showed that there is no unique temperature function for the optimization algorithms which guarantees best performance for all application scenarios. Hence, the adaptive selection of the temperature function gives further potential for improvements. An improvement of the the presented algorithms could be achieved by wisely selecting the temperature function depending on the underlying problem, i.e. the application, the resources consumed, the available network, and the available resources. This would highly involve the investigation of efficient problem evaluation and analysis.

Furthermore, the communication cost of our approach may not be optimal under certain conditions. This is, if the complexity of the placement decision of the underlying problem is not as high, i.e. the solution space is large. Here, other approaches are more efficient. Especially with a more

efficient use of Publish/Subscribe one may significantly reduce the communication cost in low constraint scenarios. However, this again would also involve the investigation of efficient problem evaluation and analysis, since we have to decide based on the problem parameters, which approach might be superior.

Security Our proposed mechanism to enhance the privacy of event information uses a limited obfuscation measurement. This is caused by the fact that calculating obfuscation in a distributed environment results in a high load, both for computation and communication. There exists a trade-off between the responsiveness of the correlation network and the accuracy of the obfuscation calculation. While this trade-off is (in our view) acceptable for obfuscation achieved in one correlation step, it results in a significant loss of responsiveness for obfuscation achieved over multiple correlation steps.

We consider the investigation of methods to increase the Bayesian Network size as a challenging, but also worthwhile enhancement to the proposed approach. With efficient methods, it might be acceptable to measure obfuscation over more than one correlation step.

Bibliography

[AC06] Raman Adaikkalavan and Sharma Chakravarthy, *Snoopib: Interval-based event specification and detection for active databases*, Data & Knowledge Engineering **59** (2006), no. 1, 139–165.

[ACV97] Eman Anwar, Sharma Chakravarthy, and Marissa Viveros, *An extensible approach to realizing advanced transaction models*, Advanced Transaction Models and Architectures, Springer, 1997, pp. 259–276.

[AE04] Asaf Adi and Opher Etzion, *Amit - the situation manager*, The VLDB Journal **13** (2004), no. 2.

[ASSC02] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci, *A survey on sensor networks*, Communications Magazine, IEEE **40** (2002), no. 8, 102–114.

[BBB+05] Michael Beisiegel, Henning Blohm, Dave Booz, J Dubray, Adrian Colyer, Mike Edwards, Don Ferguson, Bill Flood, Mike Greenberg, Dan Kearns, et al., *Service component architecture. building systems using a service oriented architecture*, Whitepaper [online] (2005), 1–31.

[BBKZ94] Holger Branding, Alexander Buchmann, Thomas Kudrass, and Jürgen Zimmermann, *Rules in an open system: The reach rule system*, Rules in Database Systems, Springer, 1994, pp. 111–126.

[BDCdVS02] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati, *An algebra for composing access control policies*, ACM Trans. Inf. Syst. Secur. **5** (2002), 1–35.

[BER08] Ayelet Biger, Opher Etzion, and Yuri Rabinovich, *Stratified implementation of event processing network*, Distributed event-based systems, 2008.

[BESP08] Jean Bacon, David M. Eyers, Jatinder Singh, and Peter R. Pietzuch, *Access control in publish/subscribe systems*, Proceedings of the second international conference on Distributed event-based systems (New York, NY, USA), DEBS '08, ACM, 2008, pp. 23–34.

[BKR09] Jorge A. Briones, Boris Koldehofe, and Kurt Rothermel, *Spine: Adaptive publish/subscribe for wireless mesh networks*, Studia Informatika Universalis (Hrsg) **7(3)** (2009), 320–353.

[BR04] Martin Bauer and Kurt Rothermel, *How to observe real-world events through a distributed world model*, Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on, IEEE, 2004, pp. 467–476.

[CA08] Sharma Chakravarthy and Raman Adaikkalavan, *Events and streams: harnessing and unleashing their synergy!*, DEBS '08: Proceedings of the second international conference on Distributed event-based systems (New York, NY, USA), ACM, 2008, pp. 1–12.

[CD97] C Samuel Craig and Susan P Douglas, *Responding to the challenges of global markets: change, complexity, competition and conscience*, The Columbia Journal of World Business **31** (1997), no. 4, 6–18.

[CD01] Philippe Codognet and Daniel Diaz, *Yet another local search method for constraint solving*, AAAI Fall Symp. on using Uncertainty within Computation, 2001.

[CDM+91] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al., *Distributed optimization by ant colonies*, Proceedings of the first European conference on artificial life, vol. 142, Paris, France, 1991, pp. 134–142.

[Cha97] Sharma Chakravarthy, *Sentinel: an object-oriented dbms with event-based rules*, ACM SIGMOD Record, vol. 26, ACM, 1997, pp. 572–575.

[CIM09] CIM, *Current cim model*, http://cimug.ucaiug.org, March 2009.

[CM94] Sharma Chakravarthy and Deepak Mishra, *Snoop: An expressive event specification language for active databases*, Data & Knowledge Engineering **14** (1994), no. 1, 1–26.

[Cou02] Simon Courtenage, *Specifying and detecting composite events in content-based publish/subscribe systems*, Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on, IEEE, 2002, pp. 602–607.

[CRHPP09] Ching-Hua Chen-Ritzo, C Harrison, J Paraszczak, and F Parr, *Instrumenting the planet*, IBM Journal of Research and Development **53** (2009), no. 3, 1–1.

[DBB+88] Umeshwar Dayal, Barbara Blaustein, Alex Buchmann, Upen Chakravarthy, Meichun Hsu, R Ledin, Dennis McCarthy, Arnon Rosenthal, Sunil Sarin, Michael J. Carey, et al., *The hipac project: Combining active databases and timing constraints*, ACM Sigmod Record **17** (1988), no. 1, 51–70.

[DHN+04] Frank Dürr, Nicola Hönle, Daniela Nicklas, Christian Becker, and Kurt Rothermel, *Nexus–a platform for context-aware applications*, Roth, Jörg, editor **1** (2004), 15–18.

[EBB+08] Redouane Ezzahir, Christian Bessiere, Imade Benelallam, Houssine Bouyakhf, and Mustapha Belaissaoui, *Dynamic backtracking for distributed constraint optimization.*, ECAI, vol. 8, 2008, pp. 901–902.

[EFGK03] Patrick Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, *The many faces of publish/subscribe*, ACM Computing Surveys (CSUR) **35** (2003), no. 2, 114–131.

[Fid06] Eli Fidler, *Padres: A distributed content-based publish/subscribe system*, Ph.D. thesis, University of Toronto, 2006.

[Fin10] Klaus Finkenzeller, *Rfid handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*, Wiley, 2010.

[GD92] Stella Gatziu and Klaus R. Dittrich, *Samos: An active object-oriented database system*, IEEE Data Eng. Bull. **15** (1992), no. 1-4, 23–26.

[GG84] Stuart Geman and Donald Geman, *Stochastic relaxation, gibbs distributions, and the bayesian restoration of images*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **PAMI-6** (1984), no. 6, 721 –741.

[GJS92] Narain Gehani, Hosagrahar V Jagadish, and Oded Shmueli, *Composite event specification in active databases: Model & implementation*, VLDB, vol. 92, Citeseer, 1992, pp. 327–338.

[GJS93] Narain Gehani, HV Jagadish, and Oded Shmueli, *Compose: A system for composite specification and detection*, 3–15.

[GL97] Fred Glover and Manuel Laguna, *Tabu search*, Kluwer Academic (1997).

[GS90] Alan E. Gelfand and Adrian F. M. Smith, *Sampling-based approaches to calculating marginal densities*, Journal of the American Statistical Association **85** (1990), no. 410, pp. 398–409 (English).

[Hag02] John Hagedoorn, *Inter-firm r&d partnerships: an overview of major trends and patterns since 1960*, Research policy **31** (2002), no. 4, 477–492.

[Han96] Eric N. Hanson, *The design and implementation of the ariel active database rule system*, Knowledge and Data Engineering, IEEE Transactions on **8** (1996), no. 1, 157–172.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten, *The weka data mining software: an update*, SIGKDD Explor. Newsl. **11** (2009), 10–18.

[HSB09] Annika Hinze, Kai Sachs, and Alejandro Buchmann, *Event-based applications and enabling technologies*, Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (New York, NY, USA), DEBS '09, ACM, 2009, pp. 1:1–1:15.

[HWLC93] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins, *Global positioning system. theory and practice.*, Global Positioning System. Theory and practice., by Hofmann-Wellenhof, B.; Lichtenegger, H.; Collins, J.. Springer, Wien (Austria), 1993, 347 p., ISBN 3-211-82477-4, Price DM 79.00. ISBN 0-387-82477-4 (USA). **1** (1993).

[IBM12] IBM, *The future of analytics. global technology outlook 2012*, www.research.ibm.com, IBM Research, 2012.

[Jak03] Gabriel Jakobson, *The technology and practice of integrated multiagent event correlation systems*, 568–573.

[jBo] *jbossdrools*, http://drools.jboss.org/, accessed April 2014.

[JCL+10] Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, and Reza Sherafat Kazemzadeh, *The padres publish/subscribe system*, Principles and Applications of Distributed Event-Based Systems (2010), 164–205.

[K. 95] K. Krishna et al., *Distributed simulated annealing algorithms for job shop scheduling*, Systems, Man and Cybernetics, IEEE Transactions **25** (1995), 1102–1109.

[KJV83] Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi, *Optimization by simmulated annealing*, science **220** (1983), no. 4598, 671–680.

[KKR10] Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel, *Cordies: expressive event correlation in distributed systems*, Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (New York, NY, USA), DEBS '10, ACM, 2010, pp. 26–37.

[LBOG06] Adam J. Lee, Jodie P. Boyer, Lars E. Olson, and Carl A. Gunter, *Defeasible security policy composition for web services*, Proceedings of the fourth ACM workshop on Formal methods in security (New York, NY, USA), FMSE '06, ACM, 2006, pp. 45–54.

[LCB99] Christoph Liebig, Mariano Cilia, and Alejandro Buchmann, *Event composition in time-dependent distributed systems*, Cooperative Information Systems, 1999. CoopIS'99. Proceedings. 1999 IFCIS International Conference on, IEEE, 1999, pp. 70–78.

[Lee12] Peter M Lee, *Bayesian statistics: an introduction*, John Wiley & Sons, 2012.

[LFA+06] Pablo Arozarena Llopis, Martijn Frints, David Ortega Abad, Javier González Ordás, Liam Fallon, Martin Zach, Hai Nguyen Thi Van, and Joan Serrat Fernández, *Madeira: A peer-to-peer approach to network management*, The Wireless World Research Forum (WWRF), Shanghai, China (April 2006), 2006.

[LGA96] BF Lieuwen, Narain Gehani, and Robert Arlein, *The ode active database: Trigger semantics and implementation*, Data Engineering, 1996. Proceedings of the Twelfth International Conference on, IEEE, 1996, pp. 412–420.

[LGL08] Yan Liu, Ian Gorton, and Vinh Kah Lee, *The architecture of an event correlation service for adaptive middleware-based applications*, J. Syst. Softw. **81** (2008), no. 12, 2134–2145.

[LJ05] Guoli Li and Hans-Arno Jacobsen, *Composite subscriptions in content-based publish/subscribe systems*, Proc of the 6th Int. Middleware Conf., 2005.

[LKS+10] Geetika Lakshmanan, Paul Keyser, Aleksander Slominski, Francisco Curbera, and Rania Khalaf, *A business centric end-to-end monitoring approach for service composites*, Services Computing (SCC), 2010 IEEE International Conference on, IEEE, 2010, pp. 409–416.

[LLS08] Geetika Lakshmanan, Ying Li, and Rob Strom, *Placement strategies for internet-scale data stream systems*, Internet Computing, IEEE **12** (2008), no. 6, 50–60.

[LM03] Ting Liu and Margaret Martonosi, *Impala: A middleware system for managing autonomic, parallel sensor systems*, ACM SIGPLAN Notices, vol. 38, ACM, 2003, pp. 107–118.

[LS88] Steffen L Lauritzen and David J Spiegelhalter, *Local computations with probabilities on graphical structures and their application to expert systems*, Journal of the Royal Statistical Society. Series B (Methodological) (1988), 157–224.

[LS08] David Luckham and Roy Schulte, *Event processing technical society*, Event Processing Glossary - Version **1** (2008).

[LS11] David Luckham and Roy Schulte, *Event processing glossary*, July 2011.

[Luc08] David Luckham, *The power of events: an introduction to complex event processing in distributed enterprise systems*, Rule Representation, Interchange and Reasoning on the Web (2008), 3–3.

[Luc11] David Luckham, *Event processing for business: organizing the real-time enterprise*, Wiley, 2011.

[McM07] Alan McMorran, *An introduction to iec 61970-301 & 61968-11: The common information model*, 2007.

[Mic06] Brenda Michelson, *Event-driven architecture overview*, Patricia Seybold Group **2** (2006).

[MLF+08] Emiliano Miluzzo, Nicholas Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane Eisenman, Xiao Zheng, and Andrew Campbell, *Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application*, Proceedings of the 6th ACM conference on Embedded network sensor systems, ACM, 2008, pp. 337–350.

[MSS99] Masoud Mansouri-Samani and Morris Sloman, *Gem: A generalized event monitoring language for distributed systems*, Distributed Systems Engineering **4** (1999), no. 2, 96.

[MSTY05] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo, *Adopt: Asynchronous distributed constraint optimization with quality guarantees*, Artificial Intelligence **161** (2005), no. 1, 149–180.

[OP01] Lukasz Opyrchal and Atul Prakash, *Secure distribution of events in content-based publish subscribe systems*, Proceedings of the 10th conference on USENIX Security Symposium - Volume 10 (Berkeley, CA, USA), SSYM'01, USENIX Association, 2001, pp. 21–21.

[PEB07] Lauri I. W. Pesonen, David M. Eyers, and Jean Bacon, *Encryption-enforced access control in dynamic multi-domain publish/subscribe networks*, Proceedings of the 2007 inaugural international conference on Distributed event-based systems (New York, NY, USA), DEBS '07, ACM, 2007, pp. 104–115.

[Pet04] Peter Pietzuch, Brian Shand and Jean Bacon, *Composite event detection as a generic middleware extension*, Network, IEEE **18** (2004), no. 1, 44–55.

[Pie04] Peter Pietzuch, *Hermes: A scalable event-based middleware*, Ph.D. thesis, 2004.

[PLS+06] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer, *Network-aware operator placement for stream-processing systems*,

Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on, IEEE, 2006, pp. 49–49.

[PSB03] Peter Pietzuch, Brian Shand, and Jean Bacon, *A framework for event composition in distributed systems*, Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Springer-Verlag New York, Inc., 2003, pp. 62–82.

[RDR10] Stamatia Rizou, F Dürr, and Kurt Rothermel, *Solving the multi-operator placement problem in large-scale operator networks*, Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on, IEEE, 2010, pp. 1–6.

[RDR11] S. Rizou, F. Durr, and K. Rothermel, *Fulfilling end-to-end latency constraints in large-scale streaming environments*, Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International, Nov 2011, pp. 1–8.

[Rid07] Kirstin Ridley, *Global mobile phone use to hit record 3.25 billion*, Reuters, June **27** (2007), 2007.

[RNC+95] Stuart Jonathan Russell, Peter Norvig, John Canny, Jitendra Malik, and Douglas Edwards, *Artificial intelligence: a modern approach*, vol. 74, Prentice Hall Englewood Cliffs, 1995.

[RR06] Costin Raiciu and David S. Rosenblum, *Enabling confidentiality in content-based publish/subscribe infrastructures*, Securecomm and Workshops, 2006, 28 2006-sept. 1 2006, pp. 1 –11.

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks, *Emf: Eclipse modeling framework*, Addison-Wesley Professional, 2008.

[Sch94] Douglas C. Schmidt, *Reactor - an object behavioral pattern for demultiplexing and dispatching handles for synchronous*

events, Proceedings of the First Pattern Languages of Programs conference in Monticello, Illinois, 1994.

[Sch96] Scarlet Schwiderski, *Monitoring the behaviour of distributed systems*, Ph.D. thesis, University of Cambridge, Computer Laboratory, 1996.

[SK91] Michael Stonebraker and Greg Kemnitz, *The postgres next generation database management system*, Communications of the ACM **34** (1991), no. 10, 78–92.

[SKPR09] Björn Schilling, Boris Koldehofe, Udo Pletat, and Kurt Rothermel, *Event correlation in heterogeneous environments*, Information Technology **51:5** (2009), 270–275.

[SKR11] Björn Schilling, Boris Koldehofe, and Kurt Rothermel, *Efficient and distributed rule placement in heavy constraint-driven event systems*, Proc. of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC) (Los Alamitos, CA, USA), vol. 0, IEEE Computer Society, 2011, pp. 355–364.

[SKRR10] Björn Schilling, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran, *Distributed heterogeneous event processing: Enhancing scalability and interoperability of cep in an industrial context*, Proc. of the 4th Int. Conf. on Distributed Event-Based Systems, 2010.

[SKRR13] ———, *Access policy consolidation for complex event processing*, IEEE Conference on Networked Systems (NetSys), 2013.

[SL05] Mudhakar Srivatsa and Ling Liu, *Securing publish-subscribe overlay services with eventguard*, Proceedings of the 12th ACM conference on Computer and communications security (New York, NY, USA), CCS '05, ACM, 2005, pp. 289–298.

[SMW05] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom, *Operator placement for in-network stream query processing*, Proceedings of the twenty-fourth ACM

SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, 2005, pp. 250–258.

[SPC96] Robert Simon, Mark Pullen, and Woan Sun Chang, *The a-rdl system*, Active Database Systems, Citeseer, 1996.

[ST11] Fumiko Satoh and Takehiro Tokuda, *Security policy composition for composite web services*, Services Computing, IEEE Transactions on **4** (2011), no. 4, 314 –327.

[THW05] Ron Ten-Hove and Peter Walker, *Java business integration (jbi) 1.0*, Java Specification Request **208** (2005).

[TIB] TIBCO, *Tibco business events 3.0*, http://www.tibco.com/products/event-processing/complex-event-processing/businessevents/default.jsp, accessed April 2014.

[TKAR10] Adnan Tariq, Boris Koldehofe, Ala Altaweel, and Kurt Rothermel, *Providing basic security mechanisms in broker-less publish/subscribe systems*, Proceedings of the 4th ACM Int. Conf. on Distributed Event-Based Systems (DEBS), 2010.

[TKKR10] Adnan Tariq, Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel, *Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints*, International Conference on Parallel Computing (EURO-PAR), 2010.

[TOK08] Wei Tai, D. OSullivan, and J. Keeney, *Distributed fault correlation scheme using a semantic publish/subscribe system*, Network Operations and Management Symposium, 2008. NOMS 2008. IEEE (2008), 835–838.

[VVL07] Jussi Vanhatalo, Hagen Völzer, and Frank Leymann, *Faster and more focused control-flow analysis for business process models through sese decomposition*, Service-Oriented Computing–ICSOC 2007 (2007), 43–55.

[WCEW02] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander Wolf, *Security issues and requirements for internet-scale publish-subscribe systems*, System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on, IEEE, 2002, pp. 3940–3947.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons, *The active badge location system*, ACM Transactions on Information Systems (TOIS) **10** (1992), no. 1, 91–102.

[WJ07] Alex Wun and Hans-Arno Jacobsen, *A policy management framework for content-based publish/subscribe middleware*, Middleware 2007 (2007), 368–388.

[YB05] Eiko Yoneki and Jean Bacon, *Determination of time and order for event-based middleware in mobile peer-to-peer environments*, Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on, IEEE, 2005, pp. 91–96.

[YH00] Makoto Yokoo and Katsutoshi Hirayama, *Algorithms for distributed constraint satisfaction: A review*, Autonomous Agents and Multi-Agent Systems **3** (2000), no. 2, 185–207.

[Yok95] Makoto Yokoo, *Asynchronous weak-commitment search for solving distributed constraint satisfaction problems*, Principles and Practice of Constraint Programming - CP'95, Springer, 1995, pp. 88–102.

[ZOTW06] Yongluan Zhou, Beng Ooi, Kian-Lee Tan, and Ji Wu, *Efficient dynamic operator placement in a locally distributed continuous query system*, On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE (2006), 54–71.

[ZP96] Nevin Lianwen Zhang and David Poole, *Exploiting causal independence in bayesian network inference*, CoRR **cs.AI/9612101** (1996).