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

Diplomarbeit Nr. 3180

Konzeption und Realisierung eines Sicherheitskonzepts zur sicheren Datenstromverarbeitung in einer verteilten Ausführungsumgebung

Oliver Dörler

Studiengang:	Informatik
Prüfer:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Nazario Cipriani
begonnen am:	2. November 2010
beendet am:	4. Mai 2011
CR-Klassifikation:	H.2.0, H.2.4, H.3.4

Kurzfassung

Vorliegende Diplomarbeit entwickelt ein Sicherheitskonzept für NexusDS. Das Konzept definiert Sicherheitsmechanismen um den Zugriff und die Verarbeitung von sensiblen und geschützten Daten zu kontrollieren. Die Mechanismen beinhalten Funktionen um Daten zu Anonymisieren und zu Verschleiern. Die Entwicklung des Konzeptes beinhaltet die Abgrenzung von Sicherheit im Kontext von NexusDS, Erhebung von Anforderungen aus Anwendungsszenarien und NexusDS selbst, die Entwicklung entsprechend dienlicher Maßnahmen, deren Umsetzung in eine Sicherheitsarchitektur und eine prototypische Implementierung in NexusDS.

Abstract

This diploma thesis develops a security concept for NexusDS. The concept defines a set of mechanisms to limit the access on data with the purpose to control access on sensitive information. The mechanisms also include functions to mask and anonymize data. For the development of the concept, the thesis analyses the actual situation of the NexusDS and considers several scenarios of use. Based on this analyze, a security architecture for NexusDS was developed and integrated as a prototype implementation into NexusDS.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Vision von Nexus [41, 5], ein Sonderforschungsbereich der Universität Stuttgart, ist eine Zukunft in der die Mehrzahl der eingesetzten Anwendungen von Kontextinformation Gebrauch machen. Basis der Vision ist die Idee, dass Kontextinformation direkt oder indirekt von nahezu jeder Anwendung benutzt und erzeugt werden kann. Kontextinformationen sind Informationen, die einen Bezug zu Objekten in der Realwelt modellieren. Das kann zum Beispiel die aktuelle Position eines mobilen Gerätes sein, auf dem eine ortsabhängige Anwendung ausgeführt wird oder stationäre Anwendungen, die zum Beispiel auf Arbeitsplatzrechner laufen. Auf einem Arbeitsplatzrechner könnte zum Beispiel aus dem Verhalten des Benutzers Kontextinformation gewinnen, die eine Müdigkeitserkennung realisiert, die den Benutzer auffordert, aus Gesundheitsgründen eine Pause einzulegen.

Angenommen Millionen von Nutzern würden ihren aktuellen Standort alle 30 Sekunden mithilfe eines GPS-Empfängers in Nexus einspeisen. Ergebnis wären Millionen von kontinuierlichen Datenströmen, die von Orts-bezogenen Diensten verwendet werden könnten. Die Verarbeitung einer großen Zahl an Datenströmen kann von gewöhnlichen Systemen, die Daten im ersten Schritt speichern und offline aufbereiten, häufig nicht effizient durchgeführt werden. Um diese Lücke zu schließen, wurde zur Verarbeitung von Datenströmen NexusDS [13, 30, 14] entwickelt. Besonderheit von NexusDS ist die verteile Ausführungsumgebung, die eine Zerlegung und Verteilung der Datenverarbeitung auf mehrere Rechenknoten ermöglicht. Bei einer Verteilung auf eine Infrastruktur es sein, dass die Gestalt der Rechenknoten sehr heterogen ausfällt. Verwandte Konzepte zu NexusDS, wie zum Beispiel *Aurora* [1], sind auf eine homogene Infrastruktur von Rechenknoten angewiesen. NexusDS hingegen erlaubt nicht nur die Ausführung in einer heterogenen Infrastruktur, sondern ermöglich auch das gezielte Ausnutzen besonderer Eigenschaften der Rechenknoten durch die Verwendung von *Constraints*. Constraints sind Meta-Daten, mit denen sich die Verarbeitung von Daten in NexusDS gezielt steuern lässt. Unterstützt ein Rechenknoten beispielsweise OpenCL[1], können gezielt Berechnungen diesem Rechenknoten zugewiesen werden, die von dessen besonderer Eigenschaft profitieren.

[1]OpenCL steht für ein Framework namens Open Computing Language, das die Ausführung von Programmcode auf CPU's und GPU's gleichermaßen unterstützt.

In der vorliegenden Ausprägung kennt NexusDS keinen kontrollierten Datenzugriff. Es steht jedem Benutzer der Plattform frei, alle verfügbaren Daten, sofern ein physischer Zugriff möglich ist, zu nutzen und auf in beliebiger Art und Weiße zu verwenden. Soll die Plattform zur Verarbeitung von sensiblen Informationen eingesetzt werden, ist es zwingend notwendig Mechanismen einzuführen, die den Datenzugriff gemäß definierbarer Bedingungen einschränken. Gleichzeit muss die Gestaltung der Restriktionen flexibel genug sein, dass eine uneingeschränkte Weiterentwicklung von NexusDS möglich ist.

1.1 Motivation

Wesentlich für die Bereitstellung einer Plattform, die verschiedensten Anwendungen zu Verfügung stehen soll, ist eine für dritte offene Struktur. Dritte sind zum Beispiel Anwendungsentwickler, Unternehmungen, öffentliche Dienste oder private Benutzer die Anwendungen oder Daten bereitstellen. Nur mit einer offenen Struktur kann den unterschiedlichsten Benutzergruppen die Möglichkeit gegeben werden, neue Daten und neue Dienste in NexusDS zu integrieren. Um nicht das Einsatzgebiet von NexusDS einzuschränken, sollte es daher möglich sein, auch sensible und geschützte Daten "sicher" Verarbeiten zu können. Benutzer überlassen, beziehungsweise akzeptieren NexusDS, im Allgemeinen nur dann, wenn verlässliche Zugriffskontrollen zu Verfügung stehen, um sensible Daten vor Missbrauch zu schützen. Weitergehend ist es wünschenswert, Fähigkeiten zur Anonymisierung beziehungsweise einer Verschleierung, von Daten vorzusehen. Derartige Funktionen sind zum Beispiel bei Anwendungen mit Positionsinformationen von großem Wert. Sie ermöglichen, dass Benutzer und Dienste den Detaillierungsgrad von privaten Informationen je nach Einsatz gezielt steuern können. Die gezielte Verfeinerung eines kontrollierten Datenzugriffes hebt die Beschränkung auf, dass ein Zugriff entweder nur vollständig auf alle Details von Daten gestattet oder vollständig verboten werden muss.

Für NexusDS wurde bisher keine Untersuchung oder Implementierung zur Einführung von Sicherheitsmechanismen durchgeführt, diese Lücke soll durch die Diplomarbeit geschlossen werden. Trotz eines steigenden Interesses der Forschungsgemeinschaft an Datenstrom Management Systemen (DSMS) und eine immer stärkeren Durchdringung kommerzieller Anwendungen mit Kontextinformation, gerade im aktuellen Smartphone Boom, wurde das Thema Sicherheit in DSMS bisher nur wenig untersucht [31]. Die Tatsache, dass bestehende Sicherheitskonzepte, zum Beispiel aus den relationalen Datenbanken, nicht oder nur teilweise für DSMS eigenen [11, 23] und das die bereits entwickelten Sicherheitskonzepte nur Teile von NexusDS abdecken, motiviert die vorliegende Diplomarbeit.

1.2 Gliederung

Um ein Sicherheitskonzept für NexusDS zu entwickeln, ist im ersten Schritt, der von Kapitel 2 umgesetzt wird, eine Definition und Abgrenzung des Begriffs Sicherheit notwendig. Zusätzlich stellt

das Kapitel ein Vorgehensmodell vor, anhand dessen die Entwicklung des Sicherheitskonzeptes in der Diplomarbeit strukturiert wird.

Kapitel 3 führt die Grundlagen von Nexus und NexusDS ein. Erläutert werden die Architektur, eingesetzte Datenmodelle, die verschiedenen Komponenten aus denen Nexus und NexusDS aufgebaut ist und deren Funktionsweise. Das Kapitel schließt mit einer kurzen Einordnung von NexusDS im Bezug zu verwandten Arbeiten.

Das Kapitel 4 umreist zu Beginn kurz Zugriffskontrollen in relationalen Datenbanken und stellt bisher vorgestellte Sicherheitskonzepte für DSMS vor. Nach der Vorstellung werden die Konzepte zusammengefasst und parallel erörtert, inwieweit die vorgestellten Konzepte auf NexusDS übertragbar sind.

Um einen Anforderungsrahmen für das zu Entwickelende Sicherheitskonzept aufzuspannen, führt Kapitel 5 verschiedene Anwendungsszenarien ein. Die Untersuchung der Anwendungsszenarien bezieht sich auf Anforderungen, die das Sicherheitskonzept erfüllen muss, um einen kontrollierten Datenzugriff sicherzustellen. Den Anwendungsszenarien folgt eine Untersuchung von NexusDS auf Anforderungen bezüglich der technischen Details von NexusDS.

Nach der Erhebung der zu berücksichtigen Anforderungen folgt in Kapitel 6 der Entwurf eines Sicherheitskonzeptes für NexusDS. Der Fokus liegt auf der Entwicklung und der Beschreibung von Maßnahmen, die in der Lage sind, jede gestellte Anforderung umzusetzen.

Kapitel 7 entwickelt aus den Maßnahmen eine Sicherheitsarchitektur, die in NexusDS implementiert werden kann. Ergebnis sind konkrete Komponenten, die mit ihren Aufgaben und dem jeweiligen Verhalten detailliert beschrieben werden.

Besonderheiten der prototypischen Implementierung stellt Kapitel 8 vor. Der Fokus liegt auf der Vorstellung der wichtigsten Details zur Umsetzung der vorgestellten Architektur

Geschlossen wird die Diplomarbeit mit Kapitel 9, dass eine Zusammenfassung der wichtigsten Punkte enthält und einen Ausblick auf zukünftige Arbeiten gibt.

Danksagung

Dank gilt allen, die mich bei der Erstellung der Diplomarbeit und im Laufe meines Studiums unterstützt haben. Insbesondere meiner Eltern, die mich jederzeit in jeder Hinsicht unterstützt haben und meinem Betreuer Nazario Cipriani, der mich als Gesprächspartner immer wieder zu neuen Ideen inspirierte.

Kapitel 2

Sicherheit

Sicherheit ist ein vielgestaltiger Begriff. Um überhaupt ein Sicherheitskonzept für NexusDS entwickeln zu können, müssen zuerst die grundlegenden Begrifflichkeiten für die Diplomarbeit definiert werden. Dafür gibt der erste Abschnitt 2.1 eine kurze Einführung zum Thema Sicherheit, die den Begriff Sicherheit für den weiteren Verlauf der Diplomarbeit abgrenzt.

Abschnitt 2.2 geht kurz auf *Security Engineerings*, dass verschiedene Ansätze zur Entwicklung von sicheren System beschreibt. Die darin vorgestellten Grundlagen werden im weiteren Verlauf der Ausarbeitung aufgegriffen.

Um einen ersten Eindruck zu erhalten, wie ein kontrollierter Datenzugriff umgesetzt werden kann, stellt Abschnitt 2.3 drei verbreitete Modelle zur Zugriffsteuerung vor.

In Abschnitt 2.4 wird das Vorgehensmodell vorgestellt, das die Struktur der Erarbeitung des Sicherheitskonzeptes für NexusDS illustriert. Das Vorgehensmodell strukturiert die Entwicklung einer Sicherheitsarchitektur und liefert eine klare Schrittfolge für die Diplomarbeit.

2.1 Grundlagen der Sicherheit

Die Bedeutung des Wortes Sicherheit hängt von seiner Einbettung ab, dass bedeutet, es ist wichtig zu beachten, in welcher Umgebung und Kontext wird Sicherheit betrachtet. Eine Einbettung kann zum Beispiel Sicherheit im Straßenverkehr sein oder wie in der vorliegenden Diplomarbeit die Informationstechnik. Wichtige Einflussgröße ist die Wahrnehmung von Sicherheit der einzelnen Personen, denn verschiedene Personen haben nicht notwendigerweise die gleiche Wahrnehmung von Sicherheit. Der Benutzer eines Computers kann der Meinung sein, dass der einmalig installierte Virenscanner seinen Computer ausreichend vor allen Gefahren schützt und weitere Maßnahmen unnötig sind. Andere Nutzer können der Überzeugung sein, dass ein Virenscanner ohne regelmäßig durchgeführte Aktualisierungen der Virendefinitionen langfristig nutzlos ist. Das Beispiel untermauert, dass der Begriff Sicherheit stark von der Subjektivität und des Hintergrundwissens eines Betrachters abhängen kann.

Eine sehr wichtige Einsicht für die Sicherheit ist, dass ein Sicherheitskonzept nur so stark ist wie seine schwächste Komponente. So führt ein Einbruch über die schwächste Komponente zu einer Ausbreitung über das gesamte System, ein typisches Problem in verteilten Systemen [17, 16]. Weitere sich häufig wiederholende Beispiele sind fahrlässig simple Passwörter oder unveränderte Standardpasswörter im Auslieferungszustand von beispielsweise Routern oder Hardwarefirewalls. Es ist also nicht nur wichtig technische Mechanismen zu implementieren, sondern auch den Humanteil, die Nutzer eines Systems ausreichend für Sicherheit zu sensibilisieren [2].

Einflussreiche Größe für Sicherheit ist die Zeit. Werden beispielsweise Anwendungen wie Internetbrowser betrachtet, die von einer großen Zahl an Benutzer eingesetzt werden, spielt die Zeit zwischen Erkennen und schließen einer Sicherheitslücke eine bedeutende Rolle. Je mehr Zeit verstreicht, in der die Anwendung durch die Sicherheitslücke angreifbar ist, desto wahrscheinlicher ist es, dass diese von einem Angreifer ausgenutzt wird. Stellen zum Beispiel die Daten auf dem Computer, der den Browser ausführt, einen hohen Wert dar, kann eine hohe Anziehungskraft für kriminelle Energie bestehen und Lücken werden besonders schnell für Angriffe ausgenutzt.

Ist eine Infrastruktur[1] vielen Parteien zugänglich, ist Sicherheit schwerer zu gewährleisten als in isolierten Systemen mit einem definierten, engen Benutzerkreis. Ebenfalls ist für ein Sicherheitskonzept wichtig das technische Wissen der Benutzer zu berücksichtigen. Gerade in einer Umgebung mit einer hohen Anzahl an verschiedenen Benutzergruppen kann das Benutzerwissen vom Laien bis zum Experten reichen, was einen differenzierten Umgang mit den Nutzern erfordert. Differenzierter Umgang bedeutet, wie viele komplexe Sicherheitsmechanismen können einem Benutzer zugemutet werden, sodass das System eine ausreichende Nutzerakzeptanz und Ergonomie erzielt. Auf der anderen Seite kann es besonders in sensitiven Anwendungsbereichen wie Banken oder öffentlichen Einrichtungen sein, dass die Erfüllung von gesetzlichen Regeln der Ergonomie übergeordnet werden muss.

2.1.1 Begriffe der Sicherheit

Um den Begriff Sicherheit für die Diplomarbeit zu spezifizieren, muss die Einbettung definiert werden, in der Sicherheit betrachtet wird. In der vorliegenden Arbeit ist die Einbettung bezüglich eines Informationstechnologie-Systems (IT-System) gegeben. Nach Eckert [16] wird ein IT-System wie folgt definiert:

> "Ein IT-System ist ein geschlossenes oder offenes, dynamisches technisches System mit der Fähigkeit zur Speicherung und Verarbeitung von Informationen."

In Abschnitt 1.1 wurde bereits angesprochen, dass es sich bei NexusDS um ein offenes System handelt. Es kann mit unterschiedlichster Hardware und Software verschiedenster Benutzer erweitert werden. Ein geschlossenes System wäre inkompatibel zu Technologien anderer Hersteller und auf einen definierten Teilnehmerkreis beschränkt.

[1] Infrastruktur wird von dem lateinischen *infra* für unten, unterhalb abgeleitet. In der vorliegenden Arbeit steht Infrastruktur für die allgemeinen Basiskomponenten, die notwendig sind, um ein IT-System zu betreiben.

In der Literatur [16, 18, 2] werden im Allgemeinen die Teilnehmer eines IT-Systems als **Objekt** und **Subjekt** bezeichnet. Diese Bezeichnungen sollen auch hier zur Anwendung kommen und sei wie folgt definiert.

Objekt: Ein Objekt bezeichnet Entitäten wie zum Beispiel eine Datei, ein Datenbankeintrag oder ein ausführbares Stück Quellcode.

Subjekt: Benutzer von Objekten werden Subjekte genannte. Ein Subjekt kann zum Beispiel ein Mensch sein, der ein Smartphone bedient oder auch ein Prozess auf dem Smartphone der im Auftrag des Benutzers handelt.

Um unterschiedliche Sicherheitsanforderungen in einem IT-System zu klassifizieren, werden **Schutzziele** definiert. Die Einteilung von Sicherheitsanforderungen erleichtert das Gruppieren verschiedener Anforderungen und die Zuordnung an Komponenten eines Sicherheitssystems. Gleichzeitig gewährleistet die Betrachtung der Schutzziele, dass die maßgeblichen Eckpunkte für ein Sicherheitssystem abgedeckt werden. Die folgende Auflistung von Schutzzielen basierend auf Eckert [16]. Weitere Verfeinerungen und Erweiterungen der Schutzziele sind möglich [6], verlassen aber den Rahmen der Diplomarbeit, der sich auf die Herstellung eines kontrollierten Zugriffes auf Daten bezieht. Die Schutzziele von Eckert zeichnen sich durch eine klare Zerlegung aus und fördern eine stimmige Strukturierung und Analyse von Sicherheitsanforderungen im weiteren Verlauf der Diplomarbeit.

Authentizität: Um die Echtheit beziehungsweise Glaubwürdigkeit von **Objekten** und **Subjekten** zu bestimmen, muss deren Identität überprüft werden. Dieser Prozess des Abgleiches von einer behaupteten Identität und der tatsächlichen hinterlegten Identität wird als Authentifikation bezeichnet. Das Ergebnis ist ein bejahen oder verneinen der behaupteten Identität. Sollte das Ergebnis positiv sein, wird das Objekt oder Subjekt als authentisch bezeichnet.

Datenintegrität: Gewährleistet, dass es im System den **Subjekten** nicht möglich ist, geschützte Daten unautorisiert und/oder unbemerkt zu verändern. Voraussetzung für Datenintegrität ist die Festlegung von Zugriffsrichtlinien für Daten, die definieren, welche Zugriffe gestattet sind.

Informationsvertraulichkeit: Sicherstellen von Informationsvertraulichkeit bedeutet, es besteht keine Möglichkeit, dass Informationen von **Subjekten** gelesen werden können, für die das **Subjekt** keine Freigabe besitzt. Dies gilt ebenfalls für Indirektionen über andere **Subjekte** oder **Objekte** im Datenfluss. So soll ausgeschlossen werden, dass die Schutzmechanismen unterlaufen werden können.

Verfügbarkeit: Stellt sicher, dass autorisierte Aktionen von **Subjekten** durchgeführt werden können, ohne von nicht autorisierter Stellen in irgendeiner Form beeinträchtigt zu werden. Beispielsweise mit gezielter Monopolisierung von CPU-Zeit oder Bandbreite, ohne das eine entsprechende Berechtigung vorliegt.

Verbindlichkeit: Bedeutet, dass ausgeführte Aktionen eindeutig dem **Subjekt** zugeordnet werden, das die Aktion angestoßen hat. So wird eine Historie aufgebaut die gewährleistet, dass im Nachhinein ausgeführte Aktionen nicht abgestritten werden können.

Anonymisierung: Die Veränderung personenbezogener Daten, sodass gar nicht, oder nur mit sehr großem Aufwand, persönliche Verhältnisse zugeordnet werden können. Eine schwächere Form ist die Pseudonymisierung, die lediglich eine Zuordnungsvorschrift umfasst, um zum Beispiel Realnamen durch Pseudonyme zu ersetzen. Ein klassisches Beispiel dafür sind die Nummernschilder an Fahrzeugen, die als Pseudonym für den Halter stehen.

2.2 Security Engineering

Security Engineering beschreibt die Anwendung des Ingenieur-Leitbildes auf den Aspekt IT-Systeme sicher zu gestalten und konzentriert sich auf die dazu notwendige Nutzung von **Methodiken**, **Werkzeugen** und **Prozessen**. Gestaltung umfasst sowohl die Entwicklung neuer, sicherer IT-Systeme von Grund auf, als auch die Überarbeitung bereits vorhandener Systeme. Dies umfasst die Entwicklung von Sicherheitskonzepten, deren Implementierung und Tests der IT-Systeme. Laut Eckert [16] handelt es sich dabei noch um keine methodisch ausgearbeitete Disziplin. Anderson [2] sieht in Security Engineering interdisziplinäre Anforderungen, sowohl in Software und Hardware als auch in Psychologie, Evaluierungsmethoden und der Rechtswissenschaften. Der Abschnitt ist lediglich eine kurze Einführung in ein sehr weitreichendes Thema. Zu Details sei der Leser angehalten, in der referenzierten Literatur nachzuschlagen.

2.2.1 Methodiken

Grundlegende, allgemeine Konstruktionsprinzipien, wurden von Saltzer und Schroeder bereits 1975 in [34] definiert. Zu diesen Prinzipien gehören zum Beispiel *Fail-safe defaults* die fordern, dass grundsätzlich jeder Zugriff vorerst verboten ist. Für einen Zugriff muss zuerst eine explizite Erlaubnis erteilt werden. Das weitere Prinzip *Least privilege* sagt aus, dass nur die minimalen Rechte zur Erfüllung der Aufgabe vergeben werden sollten. Für die vollständige Übersicht und Details wird in die originale Arbeit verwiesen.

Methoden in Form von Empfehlungen und Richtlinien zur Erstellung eines Sicherheitskonzeptes für die Informationstechnik (zum Beispiel IT-Systeme, Netzwerke, Anwendungen) werden vom *Bundesamt für Sicherheit in der Informationstechnik* (BSI) mit den *IT-Grundschutz-Kataloge* [9] herausgegeben. Inhalt der Kataloge sind Sammlungen von Standardsicherheitsmaßnahmen, die anhand von Eintrittswahrscheinlichkeiten und Schadensumfang von pauschalen Gefährdungen verwendet werden. Die Methodik, die seit 1994 stetig weiterentwickelt wurde, stellt einen Baukasten zur Etablierung und Aufrechterhaltung von Mechanismen zum Schutz von Information einer Institution bereit. Auf Basis des Konzeptes kann auch eine Zertifizierung nach ISO 27001 vergeben werden.

Eine Methodik, um Authentizität und Integrität von Daten zu sicherzustellen, ist eine *Public-Key Infrastruktur (PKI)*. Grundlage ist ein asymmetrisches Kryptosystem, in dem Subjekten kryptografische Schlüssel zugeordnet werden. Jedes Subjekt besitzt ein Schlüsselpaar, dass aus einem privaten Schlüssel, der geheim gehalten wird, und einem öffentlichen Schlüssel, der jedem Subjekt frei verfügbar ist, besteht. Der private und der öffentliche Schlüssel stehen in einer mathematischen Beziehung, sodass Daten, die mit dem öffentlichen Schlüssel verschlüsselt wurden, mit dem privaten Schlüssel entschlüsselt werden können. Mit der mathematischen Beziehung können Daten sicher zwischen Kommunikationspartnern ausgetauscht werden, ohne dass vorher ein gemeinsamer geheimer Schlüssel über gegebenenfalls unsichere Wege austauscht werden müsste. Will ein Absender Daten für einen Empfänger verschlüsseln, wählt er zur Verschlüsselung der Daten den der Identität zugeordneten öffentlichen Schlüssel zur Verschlüsselung der Daten. Nur der Besitzer des privaten Schlüssels kann die so geschützten Daten entschlüsseln und damit wieder lesbar machen. Um echte Schlüssel von gefälschten Schlüsseln zu unterscheiden, werden die Schlüssel in Zertifikate eingebettet. Die Zertifikate bestätigen die Vertrauenswürdigkeit der behaupteten Identität eines eingebetteten Schlüssel. Ausgestellt werden die Zertifikate von einer Zertifizierungsstelle, bei der sich der Teilnehmer zuvor korrekt ausgewiesen hat. Zertifikate können auch dazu benutzt werden, um die Identität von Subjekten zu überprüfen. Dazu wird dem Zertifikat-Inhaber eine zufällig generierte Zahl übermittelt, die dieser mit seinem privaten Schlüssel verschlüsselt und an den Absender zurückschickt. Nun kann der Wert mit dem öffentlichen Schlüssel entschlüsselt werden, stimmt das Ergebnis mit dem verschickten Wert überein, wurde die Identität des Zertifikats-Inhabers bestätigt. Es gilt jedoch zu beachten, dass ein Zertifikat nur so vertrauenswürdig ist, wie die Zertifizierungsstelle vertrauenswürdig ist.

2.2.2 Implementierung und Werkzeuge

Neben der konzeptionellen Arbeit, eine Sicherheitsarchitektur zu entwickeln, die die gewünschten Schutzziele realisiert, muss die Architektur mit einer Implementierung realisiert werden. Bei der Implementierung der Sicherheitsarchitektur, und auch des zu schützenden IT-Systemes kann es vorkommen, dass Anforderungen nicht korrekt umgesetzt werden. Ergebnis kann ein ungewolltes Verhalten der Software sein, dass sich im Gegensatz zu Syntaxfehlern nicht direkt bei der Übersetzung der Anwendung, sondern erst während der Laufzeit bemerkbar macht. Ein derartiges ungewolltes Verhalten wird als logischer Fehler bezeichnet und kann weitreichende Folgen haben. Beispielsweise könnte ein logischer Fehler bei der Implementierung der Funktionalität zu Authentifizierung von Subjekten auftreten, der bei der Prüfung der behaupteten Identität auftritt und einigen Objekten eine falsche Identität bestätigt. Wird ein Subjekt fälschlicherweise mit einer Identität mit umfangreicheren Rechten als vorgesehen ausgestattet, kann dies zu seiner Kompromittierung des Sicherheitskonzeptes führen.

Zum Auffinden der genannten Fehler können unterstützend Werkzeuge bei der Programm Analyse eingesetzt werden. Eine Klasse wird als *dynamische Programmanalyse* bezeichnet, die zur Laufzeit einer Anwendung durchgeführt wird. Zur Durchführung wird das Programm auf Testdaten ausgeführt und das Verhalten untersucht. Beispielsweise unterstützt die von Intel ver-

marktete Software *Parallel Studio* [21] die Analyse von Anwendungen, die auf mehrere Threads[2] verteilt sind, nach möglichen *data races* oder *deadlocks*. Eine weitere Klasse von Werkzeugen wird für die *statische Programmanalyse* eingesetzt. Die Analyse wird durchgeführt ohne das betreffende Programm auszuführen, indem der Quellcode als Eingabe zu Untersuchungen verwendet wird. Untersucht wird der Quellcode zum Beispiel nach Puffer Überläufen, Dereferenzierung von Null-Pointer oder nach unerfüllbaren Vergleichen. Für die statische Suche nach Programmierfehlern existierten sowohl frei Verfügbare als auch kommerzielle Software [4].

2.2.3 Prozesse

Microsoft adressiert die Entwicklung sicherer Software mit dem *Security Development Lifecycle* (SDL) [29]. Im SDL definiert Microsoft einen Softwareentwicklungsprozess, der aus drei Elementen besteht: vorbildliche Lösungen aus der Praxis (best practise), Prozessverbesserungen und Metriken. Ziel ist die Verwundbarkeit des Softwaredesigns, der Codierung und Dokumentation so früh und so weit wie möglich zu reduzieren. Damit soll ein pragmatischer Software Entwicklungszyklus bereitgestellt werden, der unter anderem auch die bereits angesprochenen Prinzipien von Saltzer und Schroeder [34] berücksichtigt.

Internationaler Standard zur Bewertung und Zertifizierung von IT-Systemen ist die *Common Criteria for Information Technology Security Evaluation* (CC). Eine Bewertung wird durchgeführt, indem Funktionalität und die Vertrauenswürdigkeit von IT-Systemen anhand von Kriterienkatalogen geprüft wird. Je nach Ergebnis und Umfang der Prüfung können unterschiedlich starke Zertifikate vergeben werden. Die aktuelle Version 3.1 ist mit ISO/IEC 15408 standardisiert und löst den europäischen ITSEC-Standard[3] und amerikanischen TCSEC-Standard[4] ab, um einen weltweit einheitlichen Standard zu schaffen.

Das BSI schlägt im BSI-Standard 100-2 [8] einen *Informationssicherungsprozess* vor, der auf Basis der bereits unter Methodiken angesprochen *IT-Grundschutz-Kataloge* eine Auswahl und Realisierung von Maßnahmen auf drei Ebenen begleitet. Die erste Ebene ist die strategische Ebene, sie beinhaltet die Initiierung des Sicherheitsprozesses mit der Erstellung einer Sicherheitsleitlinie, bestehend aus den angestrebten Informationssicherheitszielen und der verfolgten Sicherheitsstrategie sowie der Einrichtung eines IT-Sicherheitsmanagements. Auf der zweiten, der taktischen Ebene, wird ein Sicherheitskonzept nach den IT-Grundschutzkatalogen erstellt. Letzte, die operative Ebene sorgt für die Sensibilisierung und Schulung des Personals zur IT-Sicherheit und der Aufrechterhaltung der Sicherheit im laufenden Betrieb. Unter Aufrechterhaltung wird ein sich wiederholender Prozess verstanden, der nicht nur bei der Entwicklung, sondern auch während der Systemaktivität eine Überwachung durchführt, um neue Schwachstellen zu erkennen. In folgenden Wiederholungen des Prozesses können die neu erkannten Schwachstellen eliminiert werden. Im Wesentlichen

[2] Wird in Deutsch auch als *Aktivitätsträger* bezeichnet und beschreibt eine Folge von Arbeitsschritten eines Programmes, die sowohl sequenziell als auch parallel ausgeführt werden können.

[3] Information Technology Security Evaluation Criteria, 1991 von der Europäischen Kommission verabschiedet [19]

[4] Trusted Computer System Evaluation Criteria, wird auch als Orange Book bezeichnet, wurde von der US-Regierung herausgegeben [15]

	DAC	MAC	RBAC
Nachteile	Erzeuger der Daten zwangsläufig Besitzer, hoher Verwaltungsaufwand	Schreibzugriff kann nachfolgende Lesezugriffe ungewollt beschränken	Aufwendiges Modell durch Indirektion
Vorteile	Einfaches Modell	Effizientes Regelsystem	Leichte Verwaltung

Tabelle 2.1: Die Tabelle gibt eine Übersicht über die wichtigsten Vor- und Nachteile der vorgestellten Modelle zur Zugriffskontrolle.

bezieht sich das BSI-Modell auf Infrastrukturen von Institutionen. Darunter fallen beispielsweise Büroräumen, einzelne Server oder die Installation von Standardanwendungen wie Microsoft Outlook.

2.3 Modelle zur Zugriffskontrolle

Zur Umsetzung von Zugriffskontrollen existieren mit *Discretionary Access Control*, Abschnitt 2.3.1, und *Mandatory Access Control*, Abschnitt 2.3.2 zwei häufig verwendete Grundmodelle. Eine abgeschwächte Form des Mandatory Access Control, welches eine höhere Flexibilität ermöglicht, ist als *Role Based Access Control* bekannt und ist Thema des Abschnitt 2.3.3.

2.3.1 Discretionary Access Control (DAC)

Ein häufig eingesetztes Modell ist das *Discretionary Access Control (DAC)* Modell, veröffentlicht von der TCSEC [15]. Bekanntes Einsatzbeispiel ist Microsoft Windows, dass eine leicht veränderte Version, das als *Discretionary Access Control List* (DACL) bezeichnet wird, einsetzt [28]. In DAC werden für jedes Subjekt s Berechtigungen festgelegt. Rechte r können aus einer Menge von Zugriffsrechten wie zum Beispiel löschen, lesen oder schreiben sein. Ein Prädikat p legt ein Zugriffsfenster, eine Einschränkung auf bestimmte Objekte, für das jeweilige Objekt fest. Daraus entsteht das Tupel (s, o, t, p), welches mit einer Eigenschaft $f \in \{true, false\}$ erweitert wird um zu entscheiden, ob s die Rechte an ein weiteres Subjekt s' weitergeben darf. Ergebnis ist das Quintupel (s, o, t, p, f), das die Rechtevergabe beschreibt.

DAC ist ein einfach gehaltenes Modell, das Berechtigungen explizit zuordnet. Allerdings leidet es unter einigen Schwächen. Erstens entsteht aus den Quintupeln unter Umständen eine sehr große Matrix, die einen hohen Verwaltungsaufwand erfordert. Zudem geht das Konzept davon aus, dass der Erzeuger von Daten als Eigner für die Sicherheit verantwortlich ist, was die Flexibilität einschränkt.

2.3.2 Mandatory Access Control (MAC)

Mandatory Access Control (MAC) [15] markiert alle Subjekte und Objekte mit einer injektiven Abbildung auf eine Sicherheitseinstufung. Die Markierung kann aus einer beliebig definierten Menge X sein. Voraussetzung ist, dass die Menge geordnet ist, sodass das Abgleichen der Markierungen zu einem wohldefinierten Ergebnis führt. Jedes Subjekt s erhält eine Markierung zur Stufe der Vertrauenswürdigkeit: $vertrauenswuerdigkeit(s) \rightarrow x$. Alle Objekte o erhalten einen Markierung, die die Sicherheitseinstufung definiert: $klassifizierung(o) \rightarrow x$. Das Objekt o kann von Subjekt s nur gelesen werden, wenn die Stufe der Markierung ausreichend ist: $klassifizierung(o) \leq vertrauenswuerdigkeit(s)$. Wird ein Objekt geschrieben, muss es mindestens die Berechtigungsstufe des schreibenden Subjekt s erhalten. Die Regel verhindert, dass nach einem Schreibvorgang das Objekt von Subjekten gelesen werden kann, die zuvor keine Berechtigung hatten: $vertrauenswuerdigkeit(s) \leq klassifizierung(o)$.

Im Gegensatz zum DAC Modell verwendet das MAC Modell Regeln und definiert nicht für jede Beziehung zwischen Subjekt und Objekt eine Relation. Dies erspart sehr große Matrizen und deren hohen Verwaltungsaufwand. Nachteilig sind einige organisatorische Mängel. Mit der letzten Regel, zur Steuerung des Datenflusses, wird die Einstufung von geschriebenen Objekten unter Umständen auf ein zu hohes Level angehoben. So verlieren Subjekte mit niedriger Einstufung ihre Leserechte. Zudem bedeutet die Notwendigkeit alle Objekte einstufen zu müssen einen nicht unerheblichen Aufwand.

2.3.3 Role Based Access Control (RBAC)

Eine implizite Autorisierung wird mit dem *Role Based Access Control (RBAC)* [20] beschrieben. Die grundlegende Struktur besteht aus Subjekten, Rollen und Zugriffsrichtlinien. Rollen können verschiedene Zugriffsrichtlinien zugewiesen werden, die Berechtigungen zur Ausführung von bestimmten Operationen vergeben. Werden Rollen hierarchisch angeordnet, können Organisationsstrukturen abgebildet werden. So kann beispielsweise die Rolle Mitarbeiter der Rolle Abteilungsleiter untergeordnet werden. Werden der Mitarbeitergruppe zusätzliche Berechtigungen gewährt erhalten automatisch die Abteilungsleiter ebenfalls die Berechtigung. Subjekten können Rollen zugewiesen, die so Sammlungen von Berechtigungen erhalten. Dies erleichtert die Verwaltung von Zugriffsrichtlinien erheblich, da bei zur Aktualisierung nicht für jedes betroffene Subjekt und Objekt die Änderungen einzeln ausgeführt werden müssen.

2.4 Vorgehensmodell für NexusDS

Zur Entwicklung einer verlässlichen Software ist eine strukturierte Vorgehensweise ratsam [26]. Mit Strukturierung lässt sich der Prozess der Entwicklung in Schritte unterteilen, die jeweils klare und überprüfbare Ziele definieren. Dies erleichtert nicht nur die Entwicklung, sondern unterstützt maßgeblich das Erzielen einer höheren Qualität des gesuchten Sicherheitskonzeptes für NexusDS.

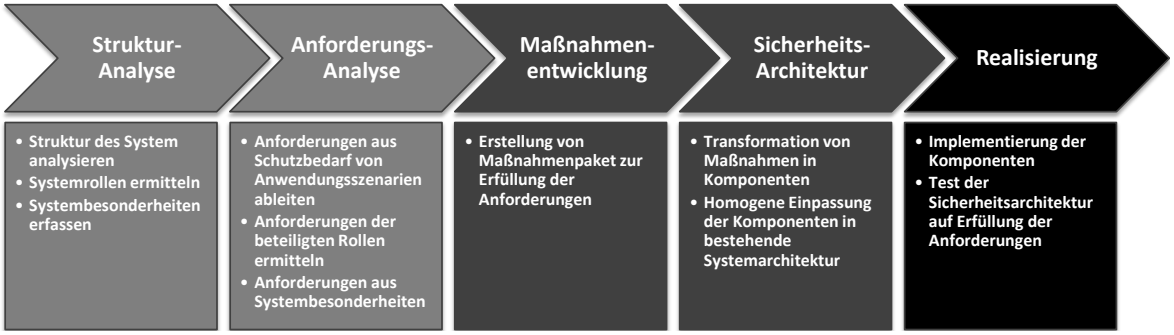

Abbildung 2.1: Schritte und Teilaufgaben des Vorgehensmodells zur Entwicklung des Sicherheitskonzeptes für NexusDS.

In der Literatur zum Thema Sicherheit in der Informationstechnik werden die Vorgehensweisen in der Regel auf Angriffsszenarien ausgerichtet. Für NexusDS soll in der Diplomarbeit erstmals ein Sicherheitskonzept entwickelt werden, das einen **kontrollierten Zugriff** auf Information einführt. Die Analyse, mit welchen speziellen Angriffen, wie zum Beispiel *Buffer Overflow* Angriffe, die neu umzusetzenden Kontrollmechanismen ausgehebelt werden könnten, lässt der der begrenzte Umfang der Diplomarbeit nicht zu. Deshalb soll ein Vorgehensmodell für die Diplomarbeit verwendet werden, dass den Fokus auf die Sicherstellung eines kontrollierten Zugriff auf Informationen legt und nicht auf die Abweisung verschiedener Angriffsszenarien.

Unter Auslassung der strategischen Ebene, zu den verschiedenen Ebene siehe Abschnitt 2.2.3, bietet der BSI-Standard [8] eine solide Grundlage, um ein Vorgehensmodell für NexusDS abzuleiten. Das BSI-Modell bezieht sich im Wesentlichen auf Infrastrukturen von Institutionen, was eine Anpassung auf NexusDS notwendig macht. Im Folgenden werden sowohl die Schritte vorgestellt, als auch auf kurz, falls nötig, auf Veränderungen zum Original BSI-Modell eingegangen. Für die detaillierte Beschreibung, wie die einzelnen Schritte vom BSI definiert werden, sei auf die Quelle des BSI-Standards verweisen. Abbildung 2.1 stellt die Schrittfolge des Vorgehensmodell dar.

Das Vorgehensmodell beginnt mit der **Strukturanalyse**. In der Strukturanalyse müssen zuerst die funktionalen Eigenschaften, die Systemumgebung und der Verwendungszweck analysiert werden. Im BSI-Modell wird die Analyse hauptsächlich zur Zerlegung in Komponenten (Anwendungen, Informationen, Räume, IT-Systeme, Kommunikationsnetze) passend für die Grundschutz-Kataloge durchgeführt und läuft im Regelfall auf einen Netztopologieplan hinaus. Für NexusDS, soll eine Topologie der verschiedenen Komponenten des Systems und deren Zusammenspiel entwickelt werden. Ebenfalls müssen Kommunikationswege betrachtet werden, die in verteilten Systemen eine elementare Grundlage bilden und als systemkritisch anzusehen sind. Zu beachten ist, dass die Erhebung nicht im Kontext einer Sicherheitsarchitektur durchgeführt wird, sondern Aufdecken soll, welche Komponenten von NexusDS in welcher weise untereinander Interagieren beziehungsweise verbunden sind. Komponenten sind dabei sowohl von technischer, als auch von menschlicher Natur, wie zum Beispiel die Rolle eines Benutzers.

In der **Anforderungsanalyse** werden aus Anwendungsszenarien Sicherheitsanforderungen, darunter kann beispielsweise das Einschränken von Leserechten verstanden werden, an das Si-

cherheitskonzept extrahiert. Anhand von Anwendungsszenarien lässt sich der Rahmen für die Diplomarbeit abstecken und eine anschauliche Erhebung durchführen. Um die verschiedenen Anforderungen der Szenarien zu homogenisieren, werden die Anforderungen anhand der in Abschnitt 2.1.1 vorgestellten Schutzziele klassifiziert. Zur Vervollständigung müssen ebenso die spezifischen Eigenschaften von NexusDS im Kontext von Sicherheit betrachtet werden. Dazu dient als Grundlage die im vorherigen Schritt durchgeführte Analyse der Struktur von NexusDS, die unter dem Gesichtspunkt von Zugriffskontrollen nach Anforderungen untersucht wurde.

Der Schritt **Maßnahmenentwicklung** ist im Original BSI-Modell der Schritt *Auswahl von Maßnahmen*. Die Auswahl im Original Modell bedeutet Maßnahmenempfehlungen aus den IT-Grundschutz-Katalogen passend zu den definierten Gefahrenlagen zu entnehmen. Für den vorliegenden Fall sind die fertigen Komponenten aber nur begrenzt anwendbar, da sich das BSI-Modell mit Standardmaßnahmen, wie zum Beispiel Feuerlöscher installieren, für NexusDS kaum eignet. Deshalb wird der Schritt in die Entwicklung von angepassten Maßnahmen für NexusDS transformiert. Entwicklung von Maßnahmen bedeutet, dass auf Basis der bis zu diesem Schritt vorgenommenen Analysen, Maßnahmen entwickelt und vorgestellt werden, die die definierten Anforderungen erfüllen und auf NexusDS anwendbar sind.

Im vorletzten Schritt **Sicherheitsarchitektur** werden die gewonnenen Maßnahmen aus dem vorherigen Schritt in eine Sicherheitsarchitektur für NexusDS überführt. Schwerpunkt der Aufgabe ist, konkrete Systemkomponente und Erweiterungen zu entwickeln, die die definierten Maßnahmen zuverlässig umsetzen. Dabei sollten sich die Umsetzungen möglichst harmonisch in das Zielsystem einfügen und von bereits vorhandenen Strukturen, soweit möglich, Gebrauch gemacht werden.

Abgeschlossen wird das Vorgehensmodell mit der **Realisierung** der Sicherheitsarchitektur. Dabei handelt es sich im Wesentlichen um die Implementierung der Architektur in NexusDS. Diese sollte unter den Aspekten der in Abschnitt 2.2 erwähnten Gesichtspunkte des *Secure Engineering* durchgeführt werden. Die Implementierung sollte dabei von Tests begleitet werden, die sicherstellen, dass die definierten Anforderungen sachgerecht umgesetzt wurden.

Abschließend noch ein kurzer Überblick über die Veränderungen der Schrittfolge zum Original BSI-Modell. Der Schritt der *Schutzbedarfsfeststellung* wurde in eine Anforderungsanalyse umgewandelt, die eine Schutzbedarfserhebung für Anwendungsszenarien enthält. Dadurch lässt sich die erstmalige Erhebung von Anforderungen und Schutz freier gestalten als im Original. Die Schritte *Bedrohungsanalyse* und *Risikoanalyse* wurden ausgelassen, da wie bereits angesprochen spezielle Angriffsszenarien nicht berücksichtigt werden sollen. Grundlegende Sicherheitskonstruktionen, die einen Schutz vor Angriffen im Allgemeinen bieten, werden bereits von der Anforderungsanalyse erhoben. Das vorgestellte Modell ist im Gegensatz zum Original nicht iterativ. Nach der Realisierung einer ersten Sicherheitsarchitektur sollte zur Aufrechterhaltung der Sicherheit ein iteratives Modell angewandt werden, dass auf der Basis des von der Diplomarbeit initiierten Sicherheitskonzeptes kontinuierlich Verbesserungen umsetzt.

Kapitel 3

Nexus und NexusDS

Das Kapitel über Nexus und NexusDS führt im ersten Abschnitt Grundlagen zu Nexus und dem darauf aufsetzenden NexusDS ein. Es erläutert die Zielsetzung von Nexus, illustriert kurz dessen Architektur und stellt das gemeinsam von Nexus und NexusDS verwendete Datenmodell vor.

Abschnitt 3.2 bezieht sich auf eine detailliertere Vorstellung von NexusDS. Vorgestellt werden die Architektur von NexusDS, das Service- und Operator-Modell und die Gestaltung von Anfragen. Aufsetzend auf der Vorstellung von NexusDS, führt Abschnitt 3.3 die vom Vorgehensmodell vorgesehene Strukturanalyse von NexusDS aus.

Geschlossen wird das Kapitel mit Abschnitt 3.4, der eine kurze Übersicht zu verwandten, bereits vorgestellten Systemen zur Datenstromverarbeitung gibt.

3.1 Einführung in Nexus

Ziel von Nexus ist eine ganzheitliche Modellierung der Real-Welt und den darin entstehenden Bezug von Kontextinformation zu erfassen. Zur Abbildung werden lokale Kontext-Modelle erstellt, die ein Teilgebiet der Real-Welt in einer je nach modellabhängigen weise Abbilden. Um die definierten Kontextmodelle in Korrelation mit der sich ständig im Fluss befindlichen Real-Welt zu halten, ist es notwendig eine Möglichkeit vorzusehen, stetige Aktualisierung über den Zustand der Real-Welt einzuholen. Dazu greift Nexus auf die immense Zahl an Sensoren, die in der modernen Welt kontinuierlich Aktualisierungen über den Zustand der Real-Welt liefern, zurück. Solche Sensoren können zum Beispiel Mobiltelefone, an das Internet angebundene Webcams oder Webservices [1] sein, die Informationen aller Art generieren. Aktuelle Smartphones verfügen häufig in der Standardausstattung über GPS-Empfänger und WLAN-Adapter. Diese Geräte können mit wenig Aufwand genutzt werden, um eine exakte GPS-Standortinformation des Benutzers zu sammeln. Verfügt der Benutzer dazu noch über eine Flatrate für mobiles Internet oder einen anderen preiswerten Datenzugang, ist es ein Leichtes die aktuelle Position

[1] Ein Webservice definiert eine Standardisierung für verschiedene Softwareagenten um Informationen auszutauschen

des Besitzers über große Zeiträume in Nexus einzuspeisen. Die Positionsinformation kann dann von vielfältigen Kontextmodellen genutzt werden um Orts-bezogene Dienste zu realisieren. Neben Mobiltelefonen gehört zur Vielfältigkeit moderner elektronischer Hilfsmittel heute eine sehr große Anzahl an Navigationsgeräten, die ebenfalls als Sensoren für Nexus dienen können. Immer häufiger werden die Modelle mit Onlineverbindungen ausgestattet, um aktuelle Verkehrsinformationen zu beziehen. Gleichzeitig Versorgen diese ihrerseits die Betreiber von Verkehrsservices mit anonymisierten Statusinformationen des Fahrzeuges [40]. Sodass zum Beispiel anhand der Bewegungsmuster, übermittelt von den Navigationssystemen, und das in Bezug setzen zu Straßenkarten und weiteren Verkehrsinformationen verbesserte Verkehrsinformationen generiert werden können. Das Verkehrsszenario ist nur eine Möglichkeit zur Verwendung der Informationen, so könnte die Information auch für verkehrsfremde Szenarien eingesetzt werden. Die Information muss lediglich mit einem Nexus Kontextmodell in den gewünschten Bezug gesetzt werden.

3.1.1 Nexus Architektur

Die Architektur [5] von Nexus besteht aus drei Schichten, **Application Tier**, **Federation Tier** und **Service Tier**.

Application Tier: Anwendungen, die die Nexus Plattform verwenden, sind in der Application Tier angesiedelt. Sie können Kontextabfragen an den Federation Tier richten, Ereignisse registrieren, um bei deren Eintreten benachrichtigt zu werden und gemeinsame Services nutzen. Ein Service ist eine Anwendung, die im Gegensatz zu Anwendungen im Application Tier, ihrerseits von anderen Anwendungen benutzt werden kann.

Federation Tier: Beherbergt sogenannte Nexus Nodes, in welchen Funktionen, Ereignisregistrierung und Services der Nexus Plattform ausgeführt werden. Eingehende Anfragen aus dem Application Tier werden analysiert und angeforderte Information aus den zuständigen Quellen abgefragt. Die Schicht vereinigt zudem alle lokalen Kontextmodelle zu einem globalen Modell.

Service Tier: Schicht in der die lokalen Kontextmodelle vorgehalten werden. Die Kontextmodelle werden von Servern bereitstellt, indem standardisierte Schnittstellen implementiert werden. Die Datenstruktur wird in der **Augmented World Model Language (AWML)** und Abfragen in der **Augmented World Query Language (AWQL)** formuliert.

3.1.2 Das Augmented World Model

Die Modellierung [5] des Kontext-Modells ist so gestaltet, dass es möglich ist, jede Art kontextabhängiger Anwendungen zu unterstützen. Ein globales, gemeinsam genutztes Kontext-Model, namens **Augmented World Model (AWM)** modelliert die Bezüge von Objekten der Real-Welt. Einige Beispiele für Objekte der Real-Welt sind Personen, Häuser oder Flüsse. Aber es existieren

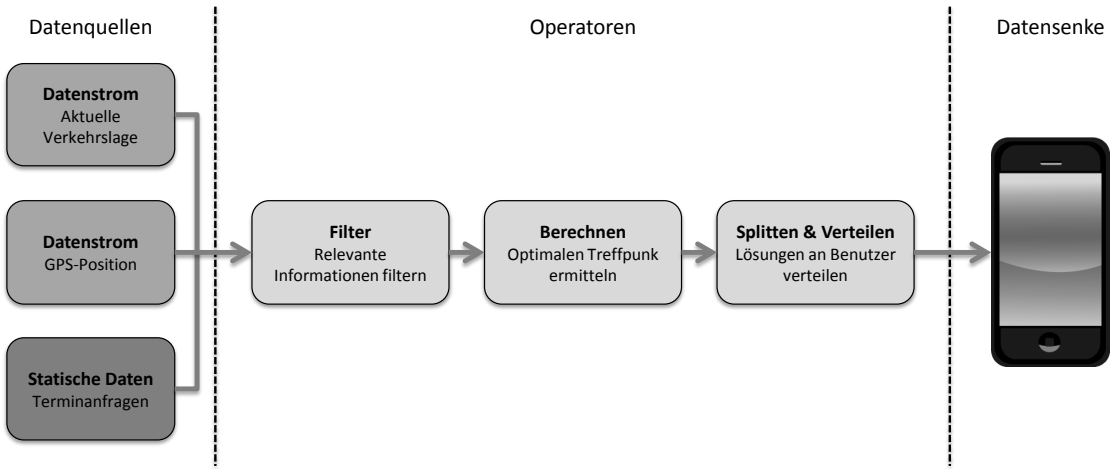

Abbildung 3.1: Illustration einer Anfrage für NexusDS die mit Positionsdaten, Terminen und Verkehrslage einen optimalen Zeit- und Treffpunkt für Benutzer von Smartphones berechnet und verteilt.

nicht nur physische Objekte, sondern auch virtuelle Objekte, das können zum Beispiel Verweise auf Dokumente oder auch Relationen zwischen Objekten sein.

Grundlage für das AWM ist die **Augmented World Model Language (AWML)** zur Datenmodellierung und die **Augmented World Query Language (AWQL)** um Abfragen zu definieren. Beide Sprachen basieren auf XML [43], der Extensible Markup Language, definiert vom World Wide Web Consortium (W3C). XML ist eine Auszeichnungssprache um Daten zu strukturieren und den Datenaustausch zwischen Computern zu erleichtern. Um eine einheitliche Struktur zu erhalten, definiert Nexus verschiedene Schemas [42]. Das globale Schema definiert sich aus dem **Nexus Standard Class Schema (NSCS)**, die Basistypen des AWM mit den Nexus **Standard Attribute Types (NSAT)** und dem **Nexus Standard Attribute Schema (NSAS)** um mehrere Attribute zu gruppieren.

3.2 NexusDS

NexusDS ist ein **Data Stream Processing System (DSMS)**, ein System zur Verarbeitung von Stromdaten ohne vorheriges Zwischenspeichern von Daten. Ein Datenstrom ist ein kontinuierlicher Fluss an Information, der unendlich sein kann und auf den kein wahlfreier Zugriff möglich ist. Beispiel für einen Datenstrom ist eine Folge aus Positionsdaten, die von einem GPS-Sensor ermittelt werden. Bisher wurden verschiedene Vorschläge, wie zum Beispiel *Borealis* [7] oder *Aurora* [1], zur Verarbeitung von Datenströme vorgestellt. Unter den Vorschlägen finden sich Konzepte, die die Datenverarbeitung wie NexusDS auf mehrere Rechenknoten verteilen. Dabei werden Anfragen zerlegt und auf verschiedene Rechenknoten verteilt, dies kann zum Beispiel aus Gründen der Lastverteilung geschehen. Alle bisherigen DSMS Konzepte haben hinsichtlich der

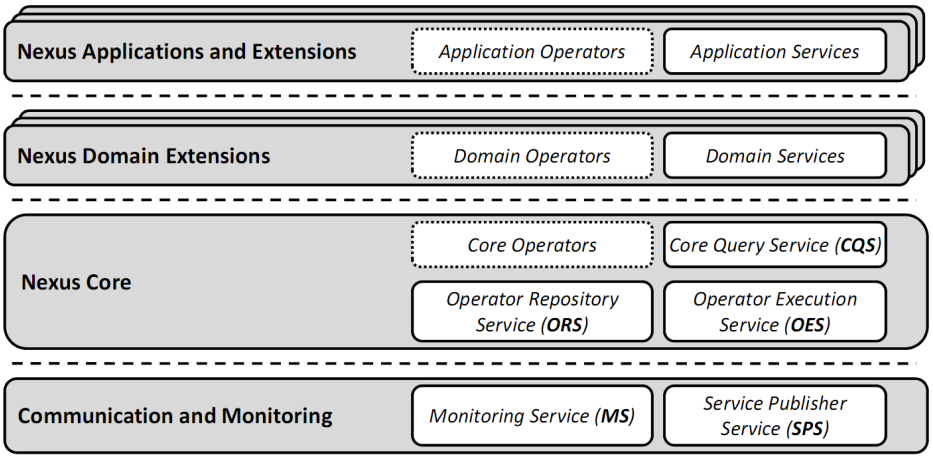

Abbildung 3.2: Schichtarchitektur von NexusDS. Operatoren sind mit durchgezogenen Boxen und Services mit gestrichelten Boxen dargestellt [13].

Verteilung eine gemeinsame Einschränkung gegenüber NexusDS, die bereits in der Einleitung angesprochene Fähigkeit, die Ausführung in einem heterogenen Umfeld zu unterstützen. NexusDS adressiert neben der generellen Ausführbarkeit Szenarien, in denen es notwendig, oder von Vorteil sein kann, dass Operationen spezifische Anforderungen an die Ausführungsumgebung stellen. Zum Beispiel kann eine Anforderung spezielle Hardware sein, die OpenCL ermöglicht. Ein weiteres Beispiel kann ein Operator sein, der besonders hohe Leistungsansprüche stellt, die nur von ausgewählten Rechenknoten erfüllt werden können.

3.2.1 Architektur

Ziel der Architektur ist eine flexible und nahtlose Erweiterbarkeit zu gewährleisten. **Services** definieren einen Rahmen um Dienste einzubringen, die dem klassischen Anfrage-Antwort Paradigma folgen. **Operatoren** hingegen definieren Erweiterungen für die Verarbeitung der Datenströme. Im Folgenden eine kurze Vorstellung der in Abbildung 3.2 dargestellten Schichtarchitektur von NexusDS.

Communication and Monitoring Layer: Die Schicht implementiert Kommunikationsfunktionen die sich in Sachen Flexibilität und Skalierbarkeit an Peer2Peer[2] Netzwerke orientiert. Wichtige bereitgestellte Services sind der **Monitoring Service (MS)** und ein **Service Publish Service (SPS)**. Der Monitoring Service überwacht Rechenknoten und deren speziellen Charakteristiken. Nutzung und Veröffentlichung der Services in NexusDS wird über den SPS gesteuert. Es ist außerdem möglich, mehrere Instanzen der Services zu starten, um die Verfügbarkeit zu verbessern.

[2]Unter Peer2Peer werden Netzwerke verstanden, deren Rechner im Gegensatz zur Client-Server Architekturen im Regelfall gleichberechtigt sind [27].

Nexus Core Layer: Beinhaltet die verteilt ausgeführten, zentralen Services von NexusDS. Der **Core Query Service (CQS)** ist für die Annahme, Verteilung und Ausführung der Anfragegraphen zuständig. Im **Operator Repository Service (ORS)** werden die **Core Operators (CO)** mitsamt Metadaten vorgehalten, dazu gehören auch Datenquellen und Datensenken. Zugewiesene Anfragefragmente werden vom **Operator Execution Service (OES)** ausgeführt. Dazu interagiert der Service mit den Core Services, um zum Beispiel für die Ausführung fehlende Operatoren aus Repositories zu laden.

Nexus Domain Extension Layer: Verfügbare Operatoren und Services werden je nach Bedarf in logischen Clustern zusammengeführt um unwichtige Details zu verstecken. So wird ein Zuschnitt auf die benötigte Funktionalität einer Domain erzielt.

Nexus Applications and Extensions Layer: Spezielle Operatoren oder Services, die nur in einzelnen Anwendungen zum Einsatz kommen, können mit dieser Schicht in die NexusDS Infrastruktur ausgelagert werden. Das ist zum Beispiel für rechenintensive Teiloperationen hilfreich, die nicht auf Endgeräten wie Smartphones ausgeführt werden können.

3.2.2 Service-Modell

Bei der Vorstellung der Architektur wurden bereits vorhandene Services in NexusDS erläutert. Beispielsweise wurde erläutert, dass der *Operator Repository Service* (ORS) alle Operatoren und Meta-Daten für das Operator-Modell vorhält. Der ORS bildet so einen integralen Bestandteil der NexusDS Architektur und realisiert einen Teil der Funktionalität. Services stellen außerdem den Verbindungspunkt für Anwendungen dar, die NexusDS zur Verarbeitung von Datenströmen nutzen, um mit NexusDS zu kommunizieren. Soll NexusDS mit zusätzlicher Funktionalität erweitert werden, können dazu neue Services implementiert und in NexusDS registriert werden. Das Operator-Modell ist für die Verarbeitung von Datenströmen vorgesehen und Operatoren werden lediglich benutzt. Services hingegen interagieren untereinander und mit angebundenen Anwendungen.

3.2.3 Operator-Modell

NexusDS verfügt über ein flexibles Operator-Modell, das für beliebige Domänen und Anwendungen angepasst werden kann. Es bildet die Basis zur Umsetzung von Datentransformationen und Auswertungen in Anfragen, indem Operatoren zur Datenverarbeitung im Rahmen des Modells implementiert werden. Operatoren sind entweder aus der Gruppe logischer oder physischer Operatoren. Ein logischer Operator kann mit unterschiedlichen physischen Implementierungen realisiert sein und definiert nur eine gewisse Semantik der Operation. Jede physische Implementierung kann eine Spezialisierung sein, zum Beispiel angepasst an Hardwareeigenschaften unterschiedlicher Rechenknoten.

Zur Erklärung des Modells ist beispielhaft eine Anfrage für NexusDS in Abbildung 3.1 dargestellt. Daten strömen nach dem push Paradigma entlang der Verbindungskanten von **Quellen** zur Verarbeitung in Operatoren. Quellen können sowohl Stromdaten, etwa ein GPS-Sensor, der

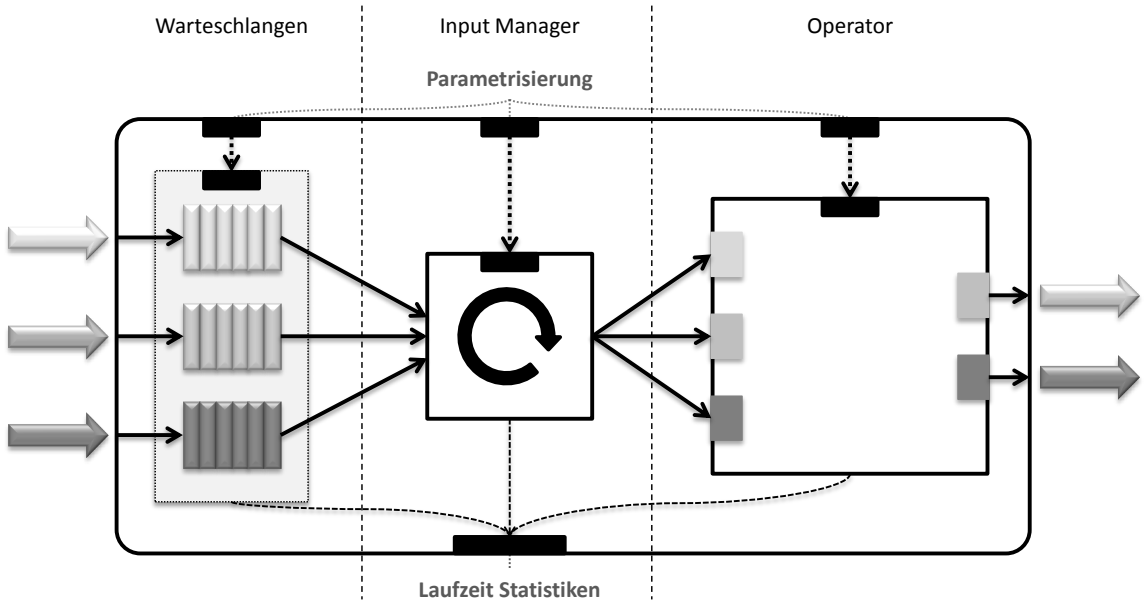

Abbildung 3.3: Darstellung des Operator Modell mit Warteschlangen, Input Manager und Operator, die mögliche Parametrisierung und das Sammeln von Laufzeitstatistiken.

stetig eine Position misst, als auch statische Datenbanken mit Busfahrplänen sein. Operatoren können nicht nur in Reihe geschaltet werden, sondern in verschiedenster Form untereinander verkettet werden. Nach Abschluss der definierten Verarbeitungsschritte fliesen die Daten an ihr Ziel, sogenannte **Senken**. Senken können zum Beispiel Dateien oder beliebige andere Ziele wie ein Smartphone Display sein. Eine Anordnung von Quellen, Operatoren und Senken wird als Anfrage bezeichnet und mithilfe des **Nexus Plan Graph Model (NPGM)** definiert.

Das von NexusDS eingesetzte Operator-Modell, basiert auf einem Boxen Paradigma. Abbildung 3.3 zeigt, aus welchen Komponenten eine Operator-Box aufgebaut ist. Direkt an den Dateneingängen sind **Warteschlangen** vorgesehen um den kontinuierlichen Datenfluss zu puffern. Sie sind zusätzlich in der Lage die Daten nach Kriterien zu sortierten, beziehungsweise zu priorisieren, falls der Puffer nicht mehr ausreicht, was einem gleitenden Datenfenster entspricht. Vorgehaltene Daten werden von einem **Input-Manager** aus den Warteschlangen entnommen und in einer für die **Operation** passenden Kombination, sofern mehrere Dateneingänge vorhanden sind, in die Operation gegeben. Offensichtlich ist in der vorliegenden Architektur der Operator ein passives Objekt, das mit Daten bestückt werden muss. Ein Beispiel für einen Operator ist eine Select-Operation, die Datenelemente nach bestimmten Kriterien entweder weiterleitet oder verwirft. Besonderheit des von NexusDS verwendetet Modell sind die bereits angesprochenen Meta-Daten, bezeichnet als **Constraints**. Diese beschreiben das Verhalten der Komponenten, wie zum Beispiel welche Datentypen an Ausgängen und Eingängen verarbeitet werden können, als auch die Anforderungen (zum Beispiel spezielle Hardware oder Speicherplatz) die die Komponente an die Ausführungsumgebung stellt. Operationen können parametrisiert werden um das Verhalten der Komponente zu Laufzeit gezielt zu steuern. Zur Vereinfachung der Handhabung verschiedenster Parameter von Komponenten kann der Entwickler **Presets** liefern, die zum Beispiel

Standardeinstellungen beinhalten. Jede der Komponenten kann innerhalb der Spezifikationen frei implementiert werden und zu einer individuellen Box zusammengestellt werden. **Quellen** und **Senken** sind einfacher aufgebaut. Eine Quelle besteht nicht aus mehreren Komponenten und bei einer Senke ist jedem Dateneingang eine Warteschlange zur Datenpufferung vorgelagert.

3.3 Strukturanalyse von NexusDS

Nach dem Vorgehensmodell aus Abschnitt 2.4, soll NexusDS zuerst in seine Komponenten zerlegt werden. Die Zerlegung und Vorstellung der einzelnen Komponenten wurde bereits im vorherigen Kapitel mit der Einführung von NexusDS abgeschlossen. Es bleibt die Aufgabe das Zusammenspiel der einzelnen Komponenten zu analysieren. Gefragte Analyse lässt sich anhand des Ausführungsmodells, dass in Abbildung 3.4 vereinfacht dargestellt wird, ableiten.

Der Anfragegraph, in Form eines NPGM, muss zunächst von einem **Extension Developer** erstellt werden. Sollten von der Anfrage Operatoren benötigt werden, die nicht bereits im **Operator Repository (ORS)** verfügbar sind, müssen diese ebenfalls entwickelt und im ORS verfügbar gemacht werden.

3.3.1 Das NexusDS Ausführungsmodell

Die Ausführung eines logischen Anfragegraphs beginnt, indem einem **Core Query Service (CQS)** über das **Query Interface (QI)**, ein ausgewählter logischer Anfragegraph zugeführt wird. Das Interface nimmt die Anfrage entgegen und gibt dem Absender ein Feedback. War der Start erfolgreich, ist das Feedback eine eindeutige Query-ID, die die Anfrage identifiziert. Trat im Startvorgang ein Fehler auf, ist die Antwort eine Fehlermeldung und die Anfrageplanung wird nicht weiter ausgeführt. War der Startvorgang erfolgreich, werden im folgenden **Query Optimizer (QO)** Schritt Optimierungen der Anfrage durch Umschreiben durchgeführt. Der **Query Fragmenter (QF)** ist für Aufbereitung der Anfrage zu einer physischen Anfrage zuständig. Das bedeutet, dass logischen Operatoren, die nur eine gewisse Semantik definieren durch passende physische Operatoren ersetzt werden. Dabei wird eine Optimierung nach den physischen Belangen der Anfrage durchgeführt und Rechnenknoten ermittelt, die nach statischen Auswertungen und aktueller Verfügbarkeit, für die entsprechenden Operatoren geeignet sind. Die Statistiken werden während der Ausführung vom **Statistics Collector (SC)** gesammelt. Basierend auf dieser Untersuchung, wird die Anfrage passend zu einer optimierten Auswahl von Rechenknoten zerlegt. Mithilfe des **Execution Manager (EM)** werden die Anfragefragmente verteilt. Wichtige Aufgabe des EM ist mit dem **Operator Execution Service (OES)** in Verbindung zu bleiben, um bei Veränderungen der Ausführungsumgebung neue Optimierungen vorzunehmen. Zur Überwachung auf Veränderungen ist der EM mit dem Query Fragmenter und dem Query Optimizer verbunden.

Wie in der Abbildung 3.4, in Form von drei dicken Pfeilen dargestellt, werden die Fragmente der Anfrage in verschiedene Execution Manager (EM) übergeben. Die EM werden im Beispiel auf

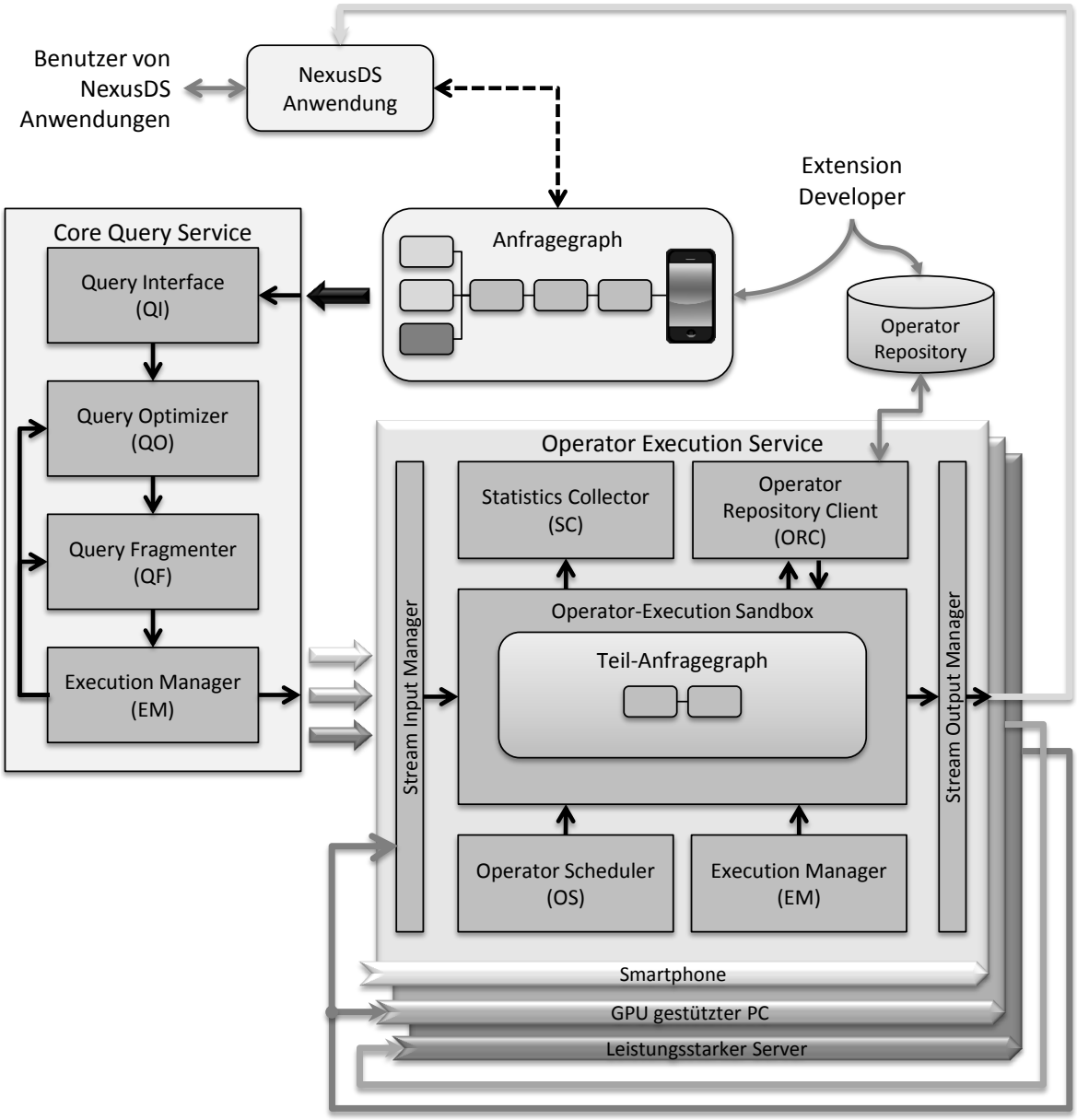

Abbildung 3.4: Vereinfachte Darstellung der Ausführung einer Anfrage, gezeigt ist die Interaktion zwischen den verschiedenen Komponenten und Rollen im NexusDS Ausführungsmodell.

verschiedenen Rechenknoten ausgeführt, was durch die Sockel der EM ausgedrückt wird. Jeder EM prüft bei der Initialisierung, ob alle nötigen Operatoren im lokalen **Operator Repository Client (ORC)** vorliegen, sollte dies nicht der Fall sein, werden sie aus dem OR geladen. Sobald alle verfügbar sind, werden die Operatoren geladen, parametrisiert und in der **Operator-Execution Sandbox** ausgeführt. Während der Ausführung sammelt an dieser Stelle der SC Statistiken über das Laufzeitverhalten der Operationen und Rechenknoten. Sollte sich eine oder mehrere der Ausführungsumgebungen in ihren Eigenschaften verändern, sind Anpassungen bei der Ausführung notwendig. Änderungen werden entweder nach dem **Lightweight Adaptation (LA)** oder **Heavyweight Adaptation (HA)** Schema durchgeführt. LA sieht nur eine Umsortierung der Ausführungsreihenfolge innerhalb des eigenen Anfragefragmentes und eine Verlangsamung der vorgelagerten Anfragefragmente vor. Sollten diese Maßnahmen nicht ausreichen, um Pufferüberläufe und einen allgemeinen Zusammenbruch der Anfrage zu verhindern, kann mit der HA über den CQS nochmals der Prozess zu einer vollständigen Neuverteilung der kompletten Anfrage angestoßen werden. Sind die Vorbereitungen auf allen Rechenknoten für jeden beteiligten OES abgeschlossen, wird die Ausführung als stabil bezeichnet.

In der Abbildung ist die Kommunikation der einzelnen Anfragefragmente mit dünnen Pfeilen dargestellt. Die Kommunikation ist unabhängig davon, ob die Fragmente auf verschiedenen oder auf demselben Rechenknoten ausgeführt werden. Ausgehende Daten werden vom **Stream Output Manager** an das oder die Ziele weitergeleitet. Zum Beispiel werden in der Illustration Daten in einem Zyklus vom *leistungsstarken Server* in den **Stream Input Manager** des *GPU gestützten PC* geleitet. Selbige Daten gehen in ein Anfragefragment zu dem *Smartphone* und final aus diesem an die NexusDS Anwendungen, die zu Beginn die Anfrage initiierte. Ausgeführt wird die Anfrage, bis eine vorher definierte Lebensdauer abläuft oder die Anfrage explizit vom Absender, mit der der Anfrage eindeutig zugeordneten Query-ID, beendet wird.

3.3.2 Rollen in NexusDS

In NexusDS lassen sich mehrere Rollen abgrenzen. Ausgehen von Abbildung 3.4 sind bereits die Rolle des Benutzers und Extention Developer dargestellt. Die **Benutzer**rolle definiert den üblichen Anwender, der mit einer Anwendung interagiert, die NexusDS verwendet. Unter Interaktion wird verstanden, dass der Benutzer sowohl Informationen von der Anwendung erhält als auch Information, wie zum Beispiel Änderungen an den Sicherheitseinstellungen, über die Anwendung in NexusDS eingibt.

Die zweite Rolle des **Extention Developer** wurde bereits in der Einführung kurz angerissen. Bei der Entwicklung einer Anfrage müssen von der Rolle alle notwendigen Operatoren entwickelt werden, die die notwendigen Funktionen für die Anfrageverarbeitung realisieren. Dies gilt natürlich nur, falls nicht schon entsprechende Operatoren zu Verfügung stehen auf die zurückgegriffen werden könnte. Die Zusammenstellung der Anfrage, definiert den Datenfluss und die verschiedenen Schritte, die notwendig sind, um ein gefordertes Ergebnis zu erzeugen. Des weiteren müssen Anfragen und Operatoren mit allen nötigen Parametern und Spezifikationen versehen werden, sodass sie korrekt arbeiten.

Nicht in der Abbildung aufgeführte Rollen sind der Datenbesitzer und der Systemadministrator. Ein **Datenbesitzer** speist Informationen in NexusDS ein, das heißt, es wird eine Quelle, zum Beispiel ein GPS-Sensor, bereitgestellt. Der **Systemadministrator** erfüllt die klassische Rolle, die aus Wartungs- und Konfigurationsaufgaben von NexusDS besteht.

3.4 Verwandte Arbeiten zu NexusDS

Neben NexusDS wurden weitere DSMS zur Verarbeitung von Datenströmen vorgestellt. Die ersten Vorschläge beinhalteten keine verteilte Ausführungsumgebung, wie die über die NexusDS verfügt. Vorteil ist, dass diese Systeme Probleme wie Netzwerkverzögerungen reduzieren und im allgemeinen Fragen der Verteilung von Verarbeitungsschritten nicht zu lösen sind. Zu dieser Gruppe gehören zum Beispiel *Aurora* [1] und *STREAM* [3]. Angesprochene, zentralisierte Konzepte sind bei hohem Benutzeraufkommen anfällig für Überlastungen und eine Skalierbarkeit ist nur eingeschränkt möglich. Aktuelle Konzepte, wie *Borealis* [7] oder *StreamGlobe* [38] verteilen die Last der Anfrageverarbeitung über mehrere Rechenknoten, um zentrale Flaschenhälse zu vermeiden.

NexusDS ermöglicht die individuelle Erweiterung mit neuer Funktionalität mit zusätzlichen Services und Operatoren. Die Erweiterungen durch Operatoren ist nicht nur offline möglich, sondern auch zur Laufzeit. In Borealis sind nur begrenzte Erweiterungen möglich. Zusätzliche Operatoren müssen auf jedem Rechenknoten manuell installiert werden und können nicht wie in NexusDS zur Laufzeit dynamisch aus Repositories geladen werden. StreamGlobe ermöglicht das Mitsenden von notwendigen Operatoren in Anfragen, die die vorinstallierten Operatoren der Rechenknoten erweitern. Was jedoch auch kein Laden von nicht mitgelieferten oder installierten Operatoren zur Laufzeit ermöglicht.

Das Ausnutzen von spezifischen Eigenschaften der Rechenknoten, das bereits in Abschnitt 3.2 erwähnt wurde, wird von keinem der zur Zeit der Erstellung der Diplomarbeit bekannten Konzepte umgesetzt. Lediglich StreamGlobe nutzt im begrenzten Maße die Eigenschaften unterschiedlicher Rechenknoten aus. Dazu klassifiziert es Rechenknoten in *Speaker-Peers*, die zusätzliche Query-Optimierungen durchführen, *Thin-Peers* mit geringer Leistungsfähigkeit und *Thick-Peers*, die komplexere Berechnungen ausführen können. Das steht jedoch in keinem Verhältnis zu den umfangreichen Möglichkeiten in NexusDS, das durch gezielte Definitionen von Constraints nahezu beliebige Eigenschaften klassifiziert und ausnutzen kann.

Zusammenfassend sind die wichtigsten Alleinstellungsmerkmale gegenüber verwandten Arbeiten zu NexusDS das **Ausnutzen heterogener Rechenknoten**, die **individuelle Erweiterbarkeit** mit neuer Funktionalität und das **zur Laufzeit dynamische Einbinden von Operatoren**.

Kapitel 4

Verwandte Sicherheitskonzepte

Das Kapitel stellt zur Einführung im ersten Abschnitt kurz die Zugriffskontrolle in relationalen Datenbanken vor. Die darauf folgenden Abschnitte bilden den Hauptteil und besprechen bereits vorgestellte Sicherheitskonzepte für DSMS.

Überraschend zeigt sich, dass seit Beginn der Untersuchungen zu DSMS nur wenige ernsthafte Sicherheitskonzepte bezüglich DSMS erarbeitet wurden. Mit einem der ersten Konzepte aus 2005 befasst sich Abschnitt 4.2, gefolgt von *ACStream* aus dem Jahre 2007, das Inhalt von Abschnitt 4.3 ist. Die aktuellste Entwicklung ist das Sicherheitskonzept *FENCE*, dessen Entwicklung 2008 begann und in Abschnitt 4.4 vorgestellt wird.

Abgeschlossen wird das Kapitel mit Abschnitt 4.5, dass die wichtigsten Punkte der vorgestellten Konzepte zusammenfasst. In der Zusammenfassung wird zugleich kurz erörtert, inwieweit sich die Sicherheitskonzepte auf NexusDS übertragen lassen.

4.1 Zugriffskontrolle in DBMS

Um Daten strukturiert zu speichern, werden häufig relationale Datenbanken eingesetzt und legen Informationen in Tabellen ab, die Relationen definieren. Die Spalten der Tabellen bilden die Attribute beziehungsweise Felder, während jede Zeile ein Datensatz ist. Zur Definition, Manipulation und Abfrage von Daten in relationalen Datenbanken ist SQL[1] ein weitverbreitet Standard. Für die Datenabfrage steht zum Beispiel das **SELECT** Statement zur Auswahl, dass Datensätzen unter Berücksichtigung bestimmter Kriterien Ausgewählt und **INSERT** um neue Datensätze in die Datenhaltung einzufügen. Für eine detaillierte Ausführung wird der Leser gebeten, in der entsprechen Standardliteratur, wie [22], nachzuschlagen.

Zur Zugriffskontrolle sieht SQL die Befehle **grant**, um Rechte zu gewähren, und **revoke** um Rechte zu entziehen, vor. Beispiele für Datenbankhersteller, die Versionen ihrer DBMS mit MAC-Modell, siehe Abschnitt 2.3.2, zur Zugriffskontrolle anbieten, sind Oracle, Ingres und Informix

[1]SQL ist der Nachfolger von SEQUEL, die aktuellsten Version ist in der ISO-Norm ISO/IEC 9075:2008 beschrieben

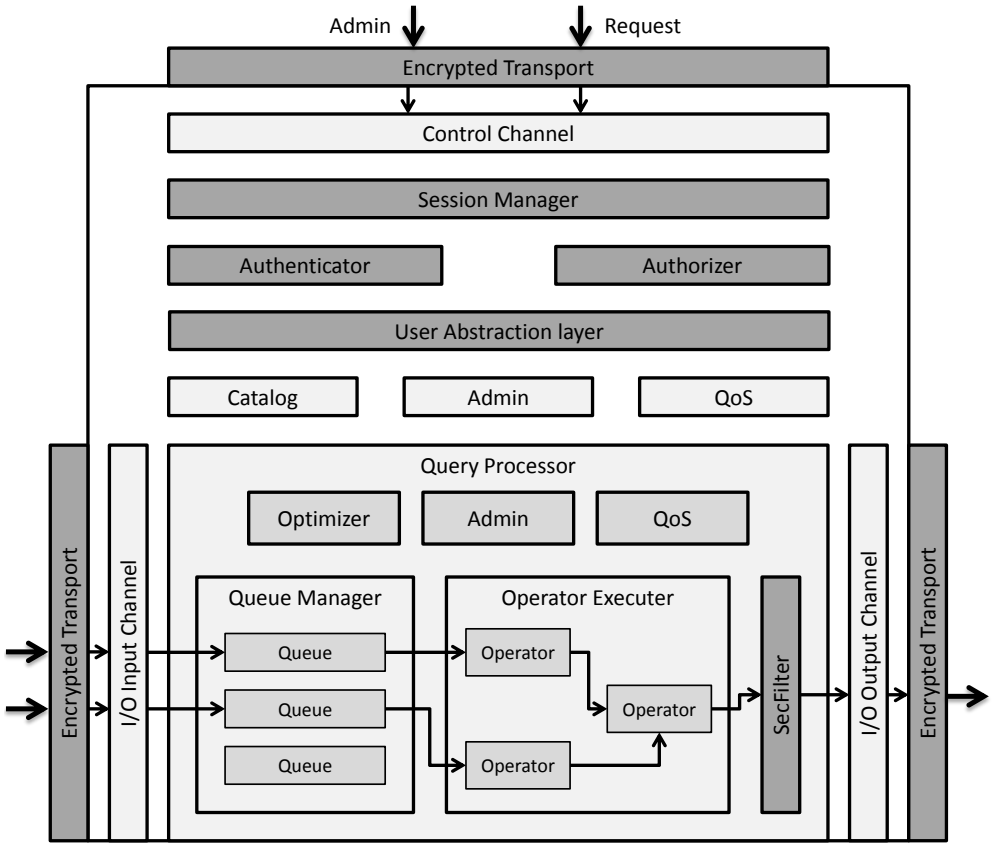

Abbildung 4.1: Generelle Sicherheitsarchitektur für DSMS nach Lindner und Meier [23]

[22]. Jedem Subjekt werden Nutzungsrechte für Operationen auf definierten Objekten eingeräumt. Beispielsweise sind in PostgresSQL [36] eine freie Anzahl von Subjekten in *Rollen* zusammengefasst. Rollen können wiederum Gruppen definieren, die ihrerseits andere Rollen als Mitglieder enthalten. So lässt sich mit dem Befehl ''`CREATE ROLE donald;`'' eine neue Rolle `donald` erzeugen. Mit ''`GRANT SELECT,INSERT ON geldspeicher TO donald WITH GRANT OPTION;`'' wird die Rolle `donald` mit der Berechtigung ausgestattet, SELECT und INSERT auf der Tabelle `geldspeicher` auszuführen und die Rechte an weitere Rollen weiterzugeben.

4.2 Secure Borealis

Eine der ersten Arbeiten, die zu Sicherheit in DSMS veröffentlicht wurde, stammt von den Autoren Lindener und Maier [24]. Darin wird ein Konzept vorgestellt, das auf eine generische Verwendbarkeit in unterschiedlichen DSMS ausgerichtet sein soll. Um dieses Ziel zu erreichen, leiten die Autoren eine allgemeine Architektur für DSMS ab und erweitern diese um Komponenten zur Zugriffskontrolle. In einer aufsetzenden Arbeit [23], wird das Konzept für *Borealis* realisiert und auf seine Performance gegenüber einem ungeschützten System untersucht.

Abbildung 4.1 stellt die von den Autoren angenommene allgemeine Architektur für DSMS mit hellgrauen Komponenten dar. Die Architektur wurde von verschiedenen DSMS Konzepten abgeleitet, unter anderem auch *Aurora*, einem Vorgänger von Borealis. In der Architektur wird zwischen administrativen Eingaben und Benutzereingaben unterschieden, wobei beide Typen von Eingaben über den **Control Channel** in Empfang genommen werden. Der *Catalog* dient als Meta-Daten Ablage, wie zum Beispiel Query-Beschreibungen. Eine Überwachung der Systemleistung findet mit der **QsQ** Komponente statt. Der Kern des Systems, der **Query Processor** QP, wird durch das **Admin** Modul kontrolliert. Im QP befindet sich der **Optimizer**, der anhand von Informationen der **Scheduler** und **Monitor** Komponente Optimierungen der Anfrage vornimmt. Die **Queues** befinden sich unter der Kontrolle des **Queue Manager** und beziehen ihre Daten aus dem **I/O Input Channel**, an den die eingehenden Datenströme angebunden werden. Zur Verarbeitung wird im **Operator Executor** die Anfrage ausgeführt. Final findet die Ausgabe über den **I/O Output Channel** statt.

Die vorgeschlagene Sicherheitsarchitektur erweitert die vorhandene Architektur und kommt ohne Modifikationen von bestehenden Komponenten aus. Die Erweiterungen sind in Abbildung 4.1 dunkelgrau hervorgehoben. Der **Session Manager** bindet jede Anfrage an eine Session, die einem Subjekt zugeordnet ist. Um Subjekte korrekt zu identifizieren, wird ein **Authentificator** angedacht, der über einen beliebigen Mechanismus, wie zum Beispiel Benutzername und Passwort, zur Authentifikation verfügt. Mit der **Autorizer** Komponente wird entschieden, ob ein Subjekt berechtigt, ist eine Aktion auszuführen. Zur Abbildung von Berechtigungen wird ein Modell namens OxRBAC (owner-extendet RBAC) verwendet. Es ist ähnlich dem in Abschnitt 2.3.3 erläuterten RBAC Modell, es verfügt lediglich über zusätzliche Relationen um den Besitzer von Objekten abzubilden. Hat ein Subjekt keine Rechte an Objekte, werden diese mit dem **User Abstraction Layer** versteckt. Das ist der erste Teil der Sicherheitsstrategie, der als *Object Level Security* bezeichnet wird. Zweiter Teil ist die *Data Level Security* ist mit der **SecFilter** Komponente umgesetzt, die aktiv wird, sobald eine Anfrage startet. Dieser nachgelagerte Filter sorgt dafür, dass Datenelemente, auf die kein Zugriff gestattet ist, vor Auslieferung eliminiert werden. Zur Behandlung von Datenströmen, die verbunden oder aggregiert werden, wird jedes Datenelement mit einer Markierung versehen, sodass diese auch später noch einem Datenstrom zugeordnet werden können. Sind Datenelemente aus verschiedenen Datenströmen kombiniert, folgt ebenfalls eine Kombinierung der Markierung, sodass eine Zuordnung zu den originalen Datenströmen möglich ist und ein nachgelagertes Filtern auch von kombinierten Datenelementen möglich ist. Um die Informationsübertragungen nach außen zu schützen, sind die Transportwege von Anfragen, Steuerungsbefehlen und Datenströme über einen **Encrypted Transport** verschlüsselt.

4.3 ACStream

Ein weiteres Konzept wurde von Carminati, Ferrari und Tan in [11, 12] entwickelt, dass auf Aurora [1] aufsetzt und auf dem Umschreiben von Anfragen basiert. In einer weiterführenden Veröffentlichung wird das Konzept als *ACStream* [10] bezeichnet und für einen kommerziellen Nachfolger von Aurora, für *StreamBase* [39], vorgestellt.

Subjekt	Streams	Attribute	Ausdrücke	Recht	GTC	WTC
Offizier	Position	Pos, SID	Position.Platoon = self.Platoon	read	-	-
Soldat	Position	Pos	Pos $\geq$ Target(a)$-\delta \wedge$ Pos $\leq$ Target(a)$+\delta$	avg	[S(a), E(a)]	1,1
Doktor	Health, Position	Heart, SID	Position.SID = Health.SID $\wedge$ Pos $\geq$ Target(a)$-\delta \wedge$ Pos $\leq$ Target(a)$+\delta$	read	[S(a), E(a)]	-

Tabelle 4.1: Beispiele für Zugriffsrichtlinien eines militärischen Beispielszenario in ACStream, entnommen aus [12]

Zur Motivation wurde als Anwendungsszenario die Überwachung von militärischen Operationen gewählt. Für Soldaten werden zwei Datenstromschemas definiert, die Informationen zur Position mit `Position(TS, SID, Pos, Platoon)` und der Gesundheitsstatus mit `Health(TS, SID, Platoon, Heart, BPressure)`. Jedes Datenelement eines Datenstromes hat die in Klammern aufgeführten Attribute. Im Beispielszenario ein *Zeitstempel* (TS), eine eindeutige *Identifikationsnummer* für jeden Soldaten (SID) und für die *Kampfgruppe* (Platton), die *Position* (POS) und zur Gesundheitsüberwachung zusätzlich *Herzfrequenz* (Heart) und *Blutdruck* (BPressure).

Zugriffskontrollen können für Attribute, Datenelemente und Datenströme vergeben werden. Tabelle 4.1 zeigt drei Beispiele. Beispielsweise werden für das Subjekt *Offizier* Leserechte, *read*, auf dem Datenstrom *Position* für die Attribute *Pos* und *SID* vergeben. Anhand von Ausdrücken wird eingeschränkt, für welche Datenelemente die Rechte angewendet werden. Zugriffe auf Datenelemente für die kein expliziter Zugriff gestattet wurde sind grundsätzlich nach dem *least-privilege* Prinzip verboten. Im Beispiel kann der Offizier nur Datenelemente lesen, die zu Soldaten gehören, die seinem Kommando unterstehen.

Ferner erlaubt das Konzept die Berücksichtigung von *temporalen Beschränkungen*. Temporale Beschränkungen ermöglichen, den Zugriff auf Datenelemente auf ein Zeitfenster zu limitieren. Möglich ist sowohl ein generelles Fenster mit Obergrenze und Untergrenze *General Time Constraint* (GTC), als auch ein Fenster das in Größe und Schrittweite beschränkt wird, bezeichnet als *Window Time Constraint* (WTC). Anwendungsbeispiele sind ebenfalls in Tabelle 4.1 dargestellt. So können die Subjekte *Soldat* und *Doktor* auf Datenelemente zugreifen, die sich auf Soldaten beziehen, die in einem bestimmten Bereich um ein Ziel a befinden. Das Beispiel für das Subjekt *Soldat* autorisiert die Durchschnittsfunktion *avg* in einem definierten Zeitintervall GTC relativ zu einem Ziel a mit Startzeitpunkt *S(a)* und Endzeitpunkt *E(a)*. Mit der Richtlinie darf die Durchschnittsfunktion über den Tupel eines Fenster von der Größe und Schrittweite von einer Stunde ausgeführt werden, definiert als WTC. Sehr ähnlich verhält es sich mit dem Beispiel des Subjektes *Doktor*. Hier ist das Lesen von Datenelemente auf einen Zeitintervall begrenzt, der im Beispiel die Dienstzeit eines Zieles a umfasst.

Realisiert wird das Konzept durch Umschreiben von Anfragen im Einklang mit den definierten Zugriffsrichtlinien. Abbildung 4.2 stellt die Architektur auf Basis des kommerziellen Ablegers

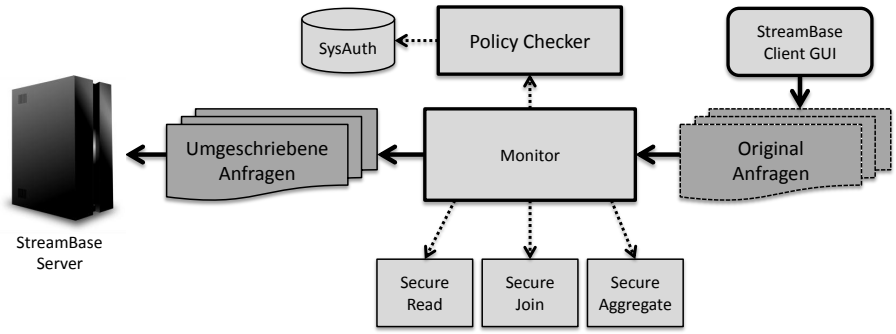

Abbildung 4.2: Architektur von ACStream in StreamBase in Anlehnung an [10]

StreamBase vor. Über die **StreamBase Client GUI** werden Anfragen ohne spezielle Berücksichtigung von Zugriffskontrollen erstellt. Die Anfragen in StreamBase bestehen aus einem gerichteten Graphen der Operatoren, in Form von Boxen, kreisfrei mit Kanten für den gewünschten Datenfluss anordnet. Bevor ein Benutzer erstmals Anfragen in das System einstellen kann, muss dieser sich in ACStream als Subjekt registrieren, beziehungsweise Authentifizieren. Die **originalen Anfragen**, die mit der regulären GUI erstellt wurden, werden vor Ausführung von der **Monitor** Komponente überarbeitet. Dazu wird jede Box der Anfrage durch den **Policy Checker** unter Zuhilfenahme der Zugriffsrichtlinien, wie in Tabelle 4.1 gezeigt, aus der zentralen **SysAuth** Komponente geprüft. Sind Einschränkungen der Daten notwendig, kann die Anfrage mit zusätzlichen Sicherheitsoperatoren versehen werden. Die notwendigen Sicherheitsoperatoren sind in drei Klassen eingeteilt. **Secure Read** Operatoren filtern Datenelemente und Attribute, **Secure Join** Operatoren ermöglichen Daten zu filtern, die aus mehreren Strömen zusammengesetzt sind und **Secure Agregate** Operatoren kontrollieren Aggregatsfunktionen. Ist die Umschreibung der Anfrage abgeschlossen, kann die Anfrage auf einem **StreamBase Server** zur Ausführung gebracht werden.

4.4 FENCE

Der *FENCE* Ansatz basiert auf der Einflechtung von Zugriffsrichtlinien in die Datenströme auf Basis von Interpunktionen. Die Idee wurde erstmalig von Nehme, Rundensteiner und Bertino in [33] vorgestellt. Es folgte 2009 *StreamShield*, dass das Konzept verfeinerte und 2010 das Framework *FENCE* [32]. Das Framework zielt auf zentralisierte DSMS ab, die die üblichen Select, Projektion und Join Operationen ausführen.

Unter *Sicherheits Interpunktion* beziehungsweise Security Punctations (SP), wird eine Methode verstanden, die Zugriffsrichtlinien direkt zwischen die Datenelemente in Datenströme integriert. Die Abbildung 4.3 zeigt die Einbettung von Interpunktionen in den Datenstrom und aus welchen Komponenten eine Interpunktion aufgebaut ist. Eine SP kann nicht nur Richtlinien für Datenelemente definieren, sondern ebenso Richtlinien für Anfragen. Zur Unterscheidung definiert die erste Komponente den **Typ**, die die Interpunktion entweder als *Data Security Predicates* (DSP) oder *Query Sicherheit Prädikate* (QSP) ausweist. Die zweite Komponente der Interpunktion, die

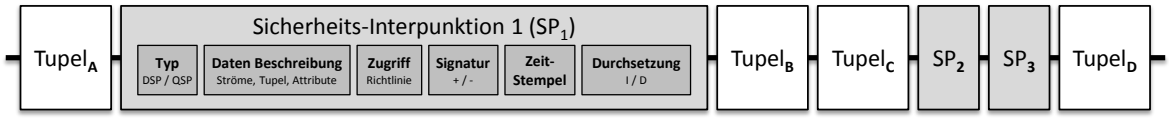

Abbildung 4.3: Komponenten und Einbettung der Sicherheits-Interpunktion in FENCE [32]

Daten Beschreibung, definiert, auf welche Objekte die Richtlinie anzuwenden ist. Objekte können Datenströme, Datenelemente oder Attribute sein. Unter **Zugriff** wird das Zugriffsmodell, wie zum Beispiel RBAC und die betroffenen Subjekte definiert. Eine **Signatur** gibt an, ob die Richtlinie angewendet wird, "+", oder mit "-" wieder zurückgenommen wird. Der **Zeitstempel** speichert, wann die SP erzeugt wurde und in der letzten Komponente wird die **Durchsetzung** definiert. Die Durchsetzung legt fest, wie streng die Richtlinie angewendet wird. Mit *Immediate* (I) kommt es zur sofortigen Anwendung der SP auf alle Datenelemente, auch auf die, die noch mit einer anderen Richtlinie gesendet wurden und sich noch in der Verarbeitung befinden. *Deferred* (D) wendet die Richtlinie nur auf der SP folgenden Datenelementen an.

Umgesetzt wird das Interpunktionskonzept mit der in Abbildung 4.4 abstrakt dargestellten Architektur. FENCE sieht drei Rollen vor, die erste Rolle kommt dem **Daten Provider** zu, der Datenquellen bereitstellt. Beispielsweise ein Benutzer, der die Positionsdaten eines GPS-Empfänger für Orts-bezogene Dienste einspeist. Betritt der Träger des GPS einen Bereich, in dem er den Zugriff auf seine Position beschränken will, flechtet er eine SP vom Typ DSP in den Datenstrom ein, dass die Einschränkung spezifiziert. Ziel des Datenstroms ist der Kern des Sicherheitsframework, die **Security-Aware Continuous Query Processing (SA-CQP)** Komponente. Die zweite Rolle ist dem **Anfrage Ersteller** zugeordnet, der den Kontext in Form einer Anfrage definiert. Die Anfrage wird ebenfalls in Form eines Datenstromes in den **Security Analyzer** übertragen. Die letzte Rolle, der **DSMS Administrator** stellt sicher, dass die in der Anfrage definierten Richtlinien korrekt umgesetzt werden. Ergebnis sind Zugriffsrichtlinien für Anfragen, vom Typ QPS, die aus dem Security Analyzer in das SA-CQP übertragen werden. Im SA-CQP werden unter Berücksichtigung der gesammelten Zugriffsrichtlinien die Operationen auf die Datenströme angewendet.

Zur Anwendung der Zugriffsrichtlinien in den Anfragen erörtert FENCE verschiedene Ansätze. Zunächst wird ein **naiver Ansatz** betrachtet, der die Datenverarbeitung und die Anwendung der Zugriffsrichtlinien getrennt ausführt. Diese Strategie bestehen aus je einem vor- und einem nachgeschalteten Filter. Nachteil ist, dass Datenelemente, die am Ende der Anfrage ausgefiltert werden umsonst verarbeitet wurden. Vorteil ist die einfache Integration, die die bestehende Architektur der Anfrageverarbeitung unberührt lässt. Zweite Strategie wird als **Security Filter Approach (SFA)** bezeichnet und wendet die Richtlinie mit einem von den Autoren entwickelten *SecurityShield Plus* (SS$^+$) Operator [33] an. Der Operator filtert gemäß der Zugriffsrichtlinien Datenelemente aus, um sie vor unberechtigtem Zugriff zu schützen. Vorteil ist, dass mit den direkt physisch in die Anfrage eingebrachten Operatoren, die Datenelemente zuverlässig und ohne hohe Performanceverluste ausgefiltert werden. Es sind jedoch unter Umständen größere Änderungen am DSMS notwendig um die korrekte Einbringung der Operatoren sicherzustellen. Die letzte Strategie ist ein Umschreiben der Anfrage, bezeichnet als **Query Rewrite Approach**

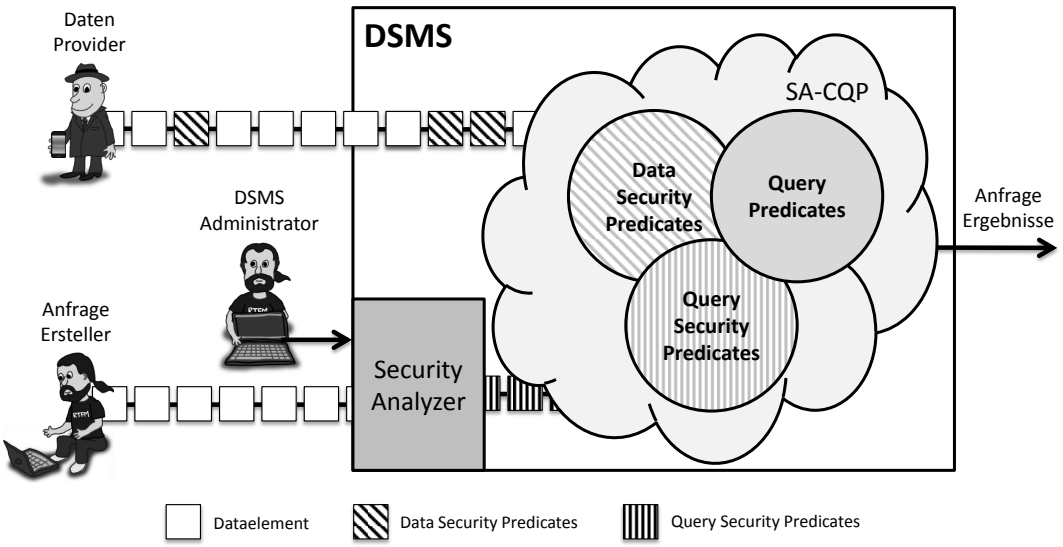

Abbildung 4.4: Abstrakte Architektur von FENCE mit den berücksichtigen Rollen [32]

(QRA). Der Kern ist eine speziell angepasste Komponente zur Umschreibung der Anfragen, die je nach Zugriffsrichtlinien einen sicheren Anfragegraph erzeugt. Vorteilhaft ist, dass zur Umsetzung auf gewöhnliche Select Operatoren zurückgegriffen werden kann. Nachteil ist, dass bei jeder Veränderung der Richtlinien die Anfrage gegebenenfalls neu angepasst werden muss.

4.5 Zusammenfassung und Anwendbarkeit in NexusDS

Die Umsetzung von Zugriffskontrollen in *DBMS* basiert in der Regel auf einer fest definierten Menge von Bedingungen, die einmalig überprüft werden, sobald ein Subjekt eine Operation auf Daten ausführen will. Ist das Subjekt authentifiziert und verfügt über die notwendigen Rechte für die Operation der Anfrage, wird die Anfrage ausgeführt. Im Fall von Datenströmen muss die Anfrage nicht nur zu Beginn kontrolliert werden, sondern über die gesamte Ausführungszeit der Anfrage. Ändert sich während der Ausführungszeit die Berechtigung für Subjekte, muss die Verarbeitung während der Ausführung angepasst werden. Dieser markante Unterschied erschwert die Einsetzbarkeit der DBMS Konzepte in einem DSMS und begründet eine gesonderte Untersuchung von DSMS nach Sicherheitskonzepten [24, 11, 33].

Im Konzept *Secure Borealis* wird der Fokus auf einzelne Rechenknoten gelegt und keine verteilte Ausführung betrachtet. Kern des Konzeptes ist es, nach Verarbeitung einer Anfrage alle Datenelemente der Datenströme zur Laufzeit zu filtern, die nicht die gesetzten Berechtigungen erfüllen. Unter Umständen wird zur Erzeugung von Ergebnissen sehr viel Rechenleistung durch teurere Operationen verschwendet, die später eliminiert werden. Um Datenströme zu identifizieren, versieht die Architektur jedes Datenelement mit einem Identifikator. Sollten verschiedene Datenströme aggregiert werden, kann durch die Beibehaltung der Markierungen der ursprünglichen Datenströme auch nach deren Vereinigung für jedes Datenelement entschieden

werden, ob es gefiltert werden muss oder nicht. Diese Strategie kann sich als nachteilig erweisen, wenn einzelne Datenelemente mit mehrfachen Markierungen versehen werden müssen, unter Umständen folgt daraus ein großer Datenüberhang. Für NexusDS ist das vorgestellte Konzept nur begrenzt geeignet. Zwar könnten die Ergebnisse vor Auslieferung an Senken in Abhängigkeit der Zugriffsrichtlinien gefiltert werden, aber der Leistungsverlust verursacht durch unnötige Berechnungen ist nicht wünschenswert.

ACStream hingegen schreibt Anfragen unter Berücksichtigung der gegebenen Bedingungen um und setzt *Secure Read*, *Secure Join* und *Secure Aggregate* Operatoren ein um bereits vor der Verarbeitung die Datenelemente vorzufiltern. Daher leidet der Ansatz nicht wie der vorherige unter einer Verschwendung von Rechenleistung. Der Vorgang zum Umschreiben berücksichtigt frühzeitig, dass gewisse Berechnungen verworfen werden müssen. Voraussetzung ist, dass die Semantik der Operatoren klar definiert ist, sodass algorithmisch ermittelt werden kann, welche Änderungen der Anfrage notwendig sind. In NexusDS ist eine definierte Semantik der Operatoren nur schwierig zu gewährleisten. Die Möglichkeit Operatoren individuell zu entwickeln erfordert eine Beschreibung der Semantik, sodass für jede Anfrage eine Lösung algorithmisch berechenbar ist. Statten die Entwickler die Operatoren mit Metadaten bezüglich der Semantik aus, müsste die Korrektheit jeder Angabe vor Einsatz geprüft werden, was in einem offenen System ein erheblichen Aufwand bedeuten würde. Hervorstechend im Konzept ist die Einführung der temporalen Beschränkungen, dass Operationen auf einen Intervall von Datenelementen beschränken kann. Soll die gleiche Fähigkeit im Sicherheitskonzept für NexusDS ermöglicht werden, muss sichergestellt sein, dass die individuell entwickelten Operator-Boxen, siehe Abschnitt 3.2, in NexusDS keine eigene Datenhaltung aufbauen und die Sicherheitseinstellung zum Intervall aushebeln.

Das Interpunktionskonzept in *FENCE* zeichnet sich dadurch aus, dass es sich direkt in die Datenströme integriert, und stellt den homogensten und flexibelsten Ansatz dar. Die Flexibilität entsteht aus der direkten Integration von Richtlinien in Datenströme, was eine dynamische Reaktion auf veränderte Zugriffsrichtlinien möglich macht. Ebenso vermeidet die Interpunktion einen unnötigen Datenüberhang durch Zusatzinformation in Datenelementen, wie dies im Konzept von Lindener und Meier der Fall ist. Interpunktion ist ein vielversprechender Ansatz, der sich zur nahtlosen Integration von Zugriffsrichtlinien auch für NexusDS eignet.

Nach Vorstellung der Konzepte und Bezugnahme auf NexusDS zeigt sich, dass allen eine Betrachtung der Auswirkungen einer verteilten Ausführung, insbesondere auf heterogenen Rechenknoten, fehlt. Die Ermittlung möglicher Lücken der Zugriffskontrollen, die durch eine Verteilung auftreten können, wurde ebenso wenig betrachtet. Weiter fehlt die Möglichkeit, sensible Daten in verschiedenen Detailstufen zu filtern. So können zum Beispiel Positionsdaten lediglich verarbeitet oder nicht verarbeitet werden, eine Anonymisierung oder Verschleierung wird nicht in Erwägung bezogen. Damit ist klar, dass kein vorhandenes Sicherheitskonzept ohne Weiteres auf NexusDS übertragen werden kann und eine weiterführende Untersuchung notwendig ist.

Kapitel 5

Anforderungen

Bevor ein Sicherheitskonzept für NexusDS entworfen werden kann, muss klar sein, welche Anforderungen das Sicherheitskonzept erfüllen soll. Dies soll in erster Linie eine Zugriffskontrolle für in NexusDS verarbeitete Daten sein. Die Zugriffskontrolle soll den Datenzugriff nicht nur verbieten oder erlauben, sondern eine feingranulare Zugriffssteuerung mit gezielter Anonymisierung und Verschleierung von Daten realisieren. Nicht betrachtet werden Anforderungen um NexusDS vor Angriffen zu schützen. Zu Angriffen zählt zum Beispiel das Ausnutzen von *Buffer-Overflows* oder *Social Engineering*. Zum Thema Angriffe findet der interessierte Leser einen guten Einstieg in [16].

Der erste Teil von Anforderungen wird in Abschnitt 5.1 erhoben, indem Anwendungsszenarien vorgestellt und analysiert werden. Die Analyse besteht aus einer Untersuchung der Anwendungsszenarien anhand der in Abschnitt 2.1 vorgestellten Schutzziele, wie zum Beispiel *Authentizität* und *Informationsvertraulichkeit*. Die Schutzziele definieren wichtige Eigenschaften eines Sicherheitskonzeptes, die notwendig sind, um eine zuverlässige Zugriffskontrolle für Daten sicherzustellen. Die Abarbeitung der Schutzziele gewährleistet, dass die Untersuchung der Anwendungsszenarien alle wichtigen Aspekte einer Zugriffskontrolle berücksichtigt.

Nachdem Anforderungen aus verschiedenen Anwendungsszenarien erhoben wurden, adressiert ein zweiter Schritt die technischen Aspekte des zu entwickelnden Sicherheitskonzeptes. Abschnitt 5.2 betrachtet und analysiert die Arbeitsweise und die besonderen Eigenschaften von NexusDS. Die daraus gewonnenen Anforderungen definieren Bedingungen, die erfüllt werden müssen, um die von den Anwendungsszenarien aufgestellten Anforderungen technisch in NexusDS umsetzen zu können.

Um eine Deckung zwischen Anforderungen aus den Anwendungsszenarien und den Anforderungen aus den technischen Eigenschaften von NexusDS sicherzustellen, wird in der Zusammenfassung der erhobenen Anforderungen in Abschnitt 5.3 wieder auf die Schutzziele zurückgegriffen.

5.1 Anforderungen aus Anwendungsszenarien

Der vorliegende Unterabschnitt stellt drei Anwendungsszenarien mit jeweils unterschiedlichem Fokus vor. Jedes Szenario wird zuerst aus der Perspektive der Anwendung erläutert. Das beinhaltet, wie die Anwendung aussieht, welche Funktionen realisiert werden und welche Erwartung die Nutzer, beziehungsweise Entwickler, an das Verhalten der Anwendung stellen. In der Beschreibung werden besonders relevante Bezeichner hervorgehoben und als Subjekte oder Objekte identifiziert. Zur Erinnerung, als **Subjekt** wird bezeichnet, wer Aktionen in einer Anwendung anstößt oder durchführt. Das kann zum Beispiel ein Benutzer sein oder ein Prozess oder Service, der im Auftrag eines Benutzer handelt. Werden Daten adressiert, wie zum Beispiel ein Datenstrom von Bildern einer Kamera oder eine Liste von Subjekten, wird von einem **Objekt** gesprochen.

Gefolgt wird die Beschreibung der Anwendungsszenarien von der Erhebung der Anforderungen. Für jedes Schutzziel wird das beschriebene Anwendungsszenario nach Anforderungen untersucht, die den Eigenschaften des Schutzzieles entsprechen. Daraus ergibt sich eine Menge von Anforderungen, die nach Schutzzielen klassifiziert sind und den **Schutzbedarf** der einzelnen Anforderungsszenarien abbilden.

5.1.1 Börsenkurse von SuperQuotes

Die fiktive Firma *SuperQuotes* bietet einen Datendienst für Börsenkurse (**Objekte**) an. Den Nutzern dieses Dienstes werden Lizenzen verkauft, deren Gültigkeit an ein definiertes Land gebunden ist. Das bedeutet es ist nicht gestattet, die Daten in anderen Ländern als Deutschland zu verarbeiten. Die vergebene Lizenz an den Kunden (**Subjekte**) ist nicht kommerziell. Es dürfen weder die Daten noch die Ergebnisse aus deren Verarbeitung weiterverkauft, veröffentlicht oder an Dritte weitergegeben werden.

Der **Schutzbedarf** wird durch die Erfüllung der ersten vier Schutzziele gewährleistet, das Schutzziel **Anonymisierung** spielt in diesem Szenario keine Rolle. **Authentizität** von Subjekten und Objekten ist notwendig um Kunden und Datenlieferant korrekt zu identifizieren. Es sollen nur zahlende Kunden beliefert werden und der Kunde möchte sich sicher sein, die korrekten und authentischen Daten zu beziehen. **Datenintegrität** stellt sicher, dass Börsenkurse vertrauenswürdig sind und nicht von dritter Stelle manipuliert wurden. Mit **Informationsvertraulichkeit** wird sichergestellt, dass nur authentifizierte Kunden die angebotenen Börsenkurse lesen können. So soll ein Abzweigen der Daten an Dritte, die keine Lizenz besitzen, unterbunden werden. Das Schutzziel muss ebenfalls das Lesen oder Verarbeiten der Daten auf Rechenknoten außerhalb von Deutschland verhindern. **Verfügbarkeit** spielt für viele Börsenanwendungen eine wichtige Rolle. Der Service muss zu jeder Zeit verfügbar sein und über eine ausreichende Leistungsfähigkeit verfügen, sodass auch in Zeiten starker Volatilität der Märkte eine zuverlässige Ausführung der Anfragen sichergestellt ist. Die **Verbindlichkeit** von Aktionen ermöglicht SuperQuotes, illegale Nutzung der Daten aufzudecken und verantwortliche Benutzer zu identifizieren. Voraussetzung zur Erfüllung der verschiedenen Ziele ist, dass Subjekte und Objekte mit Bedingungen versehen werden, die die erlaubten Aktionen definieren.

5.1.2 Orts-bezogener Dienst Squebber

Mit dem fiktiven Dienst *Squebber*, für Smartphones mit GPS, können Benutzer (**Subjekt**) virtuelle Markierungen an Orten (**Objekte**) hinterlassen. Eine virtuelle Markierung (**Objekt**) kann beliebigen Inhalt tragen, zum Beispiel eine Textnachricht (**Objekt**) oder eine beliebig andere Orts-bezogene Funktionalität (**Objekt/Subjekt**). Solche Orts-bezogenen Funktionen können auch von dritten Diensten, wie zum Beispiel Facebook oder Google Maps bereitgestellt werden. Jeder Benutzer von Squebber hat ein Profil in dem verschiedene persönliche Daten (**Objekte**) gespeichert sind, zum Beispiel Lieblingsfilme oder Beziehungsstatus.

Benutzer Tom hat heute Geburtstag und eine Favoritenliste (**Objekt**) von Orten, die von seinen Freunden (**Subjekte**) täglich besucht werden. Tom will eine spontane Geburtstagsfeier organisieren und hinterlässt an verschiedenen Orten aus der Favoritenliste virtuelle Markierungen. Die virtuellen Markierungen enthalten Einladungen (**Objekt**) für seine Geburtstagsfeier. Die Einladung wird automatisch auf dem Smartphone eingeblendet, wenn einer seiner Freude bei Squebber den Ort passiert. Tom ist ein geselliger Mensch und hinterlässt die Einladung nicht nur für Freunde, sondern auch für weitere Benutzer, die in ihrem Profil den gleichen Lieblingsfilm eingetragen haben. Passiert ein Squebber Mitglied einen der Orte, wird ein Abgleich der persönlichen Daten durchgeführt und ein positives oder negatives Ergebnis entscheidet über die Anzeige der Einladung.

Speist ein Benutzer permanent seine Positionsinformation (**Objekt**) ein, kann er durch Freunde zu einem spontanen Treffen eingeladen werden. Zur Vereinfachung von spontanen Treffen berechnet der Dienst einen optimalen Treffpunkt und eine optimale Route für alle eingeladenen Benutzer. Die berechnete Route wird danach mit Hilfe von Google Maps visualisiert, sodass der Benutzer sicher zum Treffpunkt geleitet werden kann. Die Positionsdaten können aber auch von dritten Diensten, wie Facebook, benutzt werden. Zur Kontrolle der persönlichen Daten kann die Einspeisung der Positionsinformation in unterschiedlicherweise eingeschränkt werden. Die Verfügbarkeit kann auf bestimmte Zeitintervalle und bestimmte Benutzergruppen beschränkt werden. Zusätzlich kann der Detailgrad der Positionsinformation je nach Empfänger eingestellt werden, sodass zum Beispiel Freunde eine genaue Position und Arbeitskollegen (**Subjekte**) nur den Landkreis erfahren, in dem sich der Benutzer befindet. Alle persönlichen Daten, die in Squebber hinterlegt sind, können auch mit Einstellungen zur Privatsphäre versehen werden. Die Einstellungen können zu jeder Zeit direkt vom Smartphone aus verändert werden.

Dieses Szenario beansprucht einen höheren **Schutzbedarf** als das vorhergehende und erfordert die Berücksichtigung aller Schutzziele. Wie im vorherigen Szenario wird ebenso eine Vergabe von Bedingungen zum Datenzugriff an Subjekt und Objekte vorausgesetzt. Um die verschiedenen Subjekte zu identifizieren und nach ihrer Berechtigungsstufe zu behandeln, ist das Sicherstellen von **Authentizität** notwendig. Nur so können sensible persönliche Informationen fehlerfrei und ohne Verwechslungen zugeordnet werden. **Datenintegrität** schützt die Subjekte vor manipulierten Objekten und **Informationsvertraulichkeit** ist unerlässlich, um die persönlichen Daten vor unbefugtem Zugriff zu schützen. Die **Verfügbarkeit** des Services kann Einfluss auf die Beliebtheit nehmen und so sollten nach Möglichkeit größere Unzulänglichkeiten vermieden werden. Gegebenenfalls will der Service Verstöße gegen eine Benutzerordnung ahnden, dazu

ist eine **Verbindlichkeit** nötig, um zum Beispiel einen Ausschluss von Nutzern hinreichend zu begründen. Entscheidend ist im Szenario die **Anonymisierung** zum Schutz persönlicher Daten. Es darf beim Abgleich der persönlichen Profile und im Allgemeinen bei allen Funktionen des Dienstes zu keiner Verletzung der Privatsphäre der Benutzer kommen. Zusätzlich sollen die Positionsdaten in Abhängigkeit des abrufenden Benutzers verschleiert werden, indem zum Beispiel der Detailgrad der Positionsinformation in Abhängigkeit des Datenempfängers reduziert wird. Mit der Möglichkeit dritte Dienste, wie zum Beispiel Facebook einzubinden, muss bei den genannten Anforderungen bedacht werden, dass die Anonymisierung, auch bei der Interaktion mit dritten Diensten, zuverlässig umgesetzt wird.

5.1.3 Fehlerszenario in intelligenten Fabriken

Der fiktive Konzern *EM* (**Subjekt**) produziert Elektroautos in einer Kette intelligenter Fabriken [25]. Nicht alle Fabriken gehören dem Konzern selbst, viele Zwischenschritte der Produktion werden von Zulieferer (**Subjekte**) in konzernfremden intelligenten Fabriken ausgeführt. Um eine umfassende Überwachung der Produktion in Echtzeit durchzuführen, müssen Daten (**Objekte**) aus verschiedensten Quellen zusammengeführt werden. Quellen für die Daten sind die verschiedenen Fabriken der unterschiedlicher Konzerne (**Subjekte**), in denen Mitarbeiter (**Subjekte**) in Organisationseinheiten arbeiten und Prozessen zugeordnet sein können.

Zur effektiven Überwachung der gesamten Produktionskette ist es erforderlich, die genierten Daten zeitnah zusammenzuführen und auszuwerten. Die integrierte Auswertung von Daten aus unterschiedlichen Konzernen, Organisationseinheiten, Prozessen und verschiedenen Mitarbeitern, müssen Kompetenzen beachtet und geheime Daten geschützt werden. Beispielsweise haben Abteilungsleiter erweiterte Kompetenzen für die Verarbeitung der Daten aus ihrer Abteilung aber nur einen beschränkten oder gar kein Zugriff auf die Daten einer Organisationseinheit eines konzernfremden Zulieferers. Zur Abgrenzung des Datenzugriffs werden Bedingungen an Daten gebunden, die definieren wer und in welchem Detailgrad zugreifen darf. Bedingungen sollen auch an Subjekte gebunden werden können, die definieren, welche Zugriffe dem jeweiligen Subjekt gestattet sind. In Konzernen mit einer hohen Anzahl an Mitarbeitern ist es wünschenswert, dass Bedingungen nicht nur an einzelne Subjekte, sondern an ganze Organisationseinheiten gebunden werden können, in denen eine Menge an Subjekten zusammengefasst werden.

In einem Hochtechnologiebereich, wie im vorliegenden Fall, wollen die verschiedenen Zulieferer ihr technisches Wissen schützen. Deshalb dürfen bestimmte Daten nur für ausgesuchte Organisationseinheiten verfügbar sein. Für die Fehlersuche, die über mehrere Organisationseinheiten hinweg durchgeführt wird, müssen Informationen in Echtzeit nach definierten Regeln verschleiert werden, sodass eine Fehlersuche möglich ist, aber Betriebsgeheimnisse geschützt sind. Besonders sensible Daten dürfen nur von ausgewählten konzerneigenen Rechenknoten verarbeitet werden und die lokale Infrastruktur nicht verlassen. Um ein Einhalten von Verträgen zu gewährleisten, sollen alle Aktionen der Subjekte im System protokolliert werden.

Auch in diesem Szenario müssen für den **Schutzbedarf** alle Schutzziele betrachtet werden und eine Zuordnung von Zugriffsbedingungen sichergestellt werden. **Authentizität** muss erfüllt werden, sodass alle Subjekte, die das System benutzten sicher identifiziert werden und der korrekten Organisationseinheit und Prozessen zugeordnet werden können. Das gilt auch für Objekte, die ebenfalls zweifelsfrei identifiziert werden müssen. Die **Datenintegrität** sorgt dafür, dass Objekte nur dann geändert werden, wenn die entsprechende Organisationseinheit oder ein Subjekt tatsächlich eine entsprechende Zuständigkeit besitzt. **Informationsvertraulichkeit** gewährleistet den Schutz der Betriebsgeheimnisse und von Informationen, die nur ausgewählten Subjekten oder Organisationsgruppen vorenthalten sind. In einer Produktionskette von mehreren miteinander verzahnten Fabriken muss die **Verfügbarkeit** jederzeit sichergestellt sein. Der Ausfall von einzelnen Gliedern in der Produktionskette hat Auswirkungen auf die effektive Überwachung der gesamten Produktionskette. Zur Rechtssicherheit der verschiedenen Unternehmungen sollte eine nachvollziehbare **Verbindlichkeit** für alle Aktionen bestehen. Der **Anonymisierung** kommt in diesem Fall weniger ein Verbergen von persönlichen Daten zu, sondern es soll eine gezielte Verschleierung und Vereinfachung von Daten geleistet werden. Dabei muss die Reduktion des Detailgrades der Information gerade so stark sein, dass genug Aussagekraft für über Zugriffsgrenzen hinwegreichende Aufgaben wie die Suche nach Produktionsfehlern erhalten bleibt. Gleichzeitig aber interne Informationen, insbesondere Betriebsgeheimnisse ausreichend, geschützt werden. Das erfordert die Möglichkeit, dass individuelle Transformationen zur Datenverschleierung und Vereinfachung definiert werden können.

5.2 Anforderungen aus NexusDS

Nachdem im vorhergehenden Kapitel Anforderungen erhoben wurden, die sich auf die Anwendungsseite von NexusDS beziehen, müssen noch die Anforderungen erhoben werde, die sich aus der technischen Beschaffenheit von NexusDS ergeben. Aus den Anforderungen der technischen Seite werden im weiteren Verlauf der Diplomarbeit konkrete Funktionen und Komponenten abgeleitet, zum Beispiel Funktionen zur gezielten Unterbindung und Zulassung von Zugriffen auf geschützte Informationen oder Komponenten zur Authentifizierung von Subjekten. Vorerst werden jedoch nur Anforderungen gewonnen, ohne auf die technische Realisierung einzugehen. Die Gewinnung der Anforderungen setzt auf den Abschnitt 3.2 auf. Dieser Abschnitt hat die technischen Eigenschaften und Besonderheiten, wie zum Beispiel das Operator-Modell von NexusDS, bereits ausführlich erläutert.

Die Erhebung ist in zwei Teilen gegliedert. Der erste Teil untersucht jede Basis-Rolle von NexusDS auf rollenspezifische Anforderungen. Jedes Subjekt das NexusDS benutzt, kann nach Verhalten in eine der Basis-Rollen eingeteilt werden. Der zweite Teil untersucht NexusDS nach Anforderungen, die sich aus der Architektur und den besonderen technischen Eigenschaften von NexusDS ergeben.

5.2.1 Basisrollen von NexusDS

Mit einer gezielten Erhebung von Anforderungen aus den Basis-Rollen in NexusDS, finden die unterschiedlichen Bedürfnisse von Benutzern im Sicherheitskonzept Beachtung. Wie bereits in Kapitel 2 ausgeführt, spielt die Erfüllung der Benutzerbedürfnisse eine bedeutende Rolle, um die Nutzerakzeptanz von NexusDS zu erhöhen.

Benutzer: Die Rolle des Benutzers ist direkt an die eingesetzte Anwendung gebunden. Er erwartet eine sichere und zuverlässige Ausführung seiner Anwendung, die eine zugesicherte Servicequalität erfüllt. Sicherheitseinstellungen, die der Benutzer setzt, sollen zuverlässig umgesetzt werden und ohne Verzögerung in Kraft treten. Nach Möglichkeit erwartet der Benutzer außerdem eine benutzerfreundliche Darreichung der Sicherheitseinstellungen, die sein technisches Verständnis für NexusDS nicht übersteigt.

Datenbesitzer: Werden Daten in NexusDS bereitgestellt, will der Besitzer die Verwendung der Daten kontrollieren. Grundlage zur Umsetzung ist die Möglichkeit, jeder Datenquelle Besitzer und Bedingungen für den Zugriff zuordnen zu können. Die Zuordnung muss zuverlässig und eindeutig sein, sodass die Bedingungen bei der Verarbeitung der Daten korrekt beachtet werden können. Bei der Gestaltung der Bedingungen sollte der Besitzer die Möglichkeit haben, den erlaubten Zugriff in der Granularität flexibel bestimmen zu können. Beispielsweise den Detailgrad von Positionsinformationen oder den Zugriff auf eine Teilmenge von Attributen eingespeister Datenelementen zu beschränken. Die Definition soll sowohl für strukturierte als auch für unstrukturierte Daten möglich sein.

Extension Developer: Entwickler von neuer Funktionalität und Anfragen für NexusDS wünschen Bedingungen zur Verwendung und Ausführung der von ihnen bereitgestellten Objekte vergeben. Beispielsweise will der Entwickler eines Input-Manager diesen nur für ausgewählte Benutzer zu Verfügung stellen, die eine Lizenzgebühr entrichten. Die Gestaltung der Bedingungen muss flexibel genug sein, dass Entwickler in der Freiheit individuelle neue Funktionen in NexusDS einzubringen nicht eingeschränkt werden. Ein flexibles Modell erleichtert außerdem bereits entwickelte Regelsysteme von vorhandene Anwendungslandschaften zu integrieren und die Migration bereits bestehenden Anwendungen auf NexusDS wird erleichtert. Des Weiteren sollten zur Unterstützung von Entwickler Möglichkeiten vorgesehen werden, Werkzeuge an das Sicherheitskonzept so anzubinden, dass Informationen über die Zugriffseinschränkungen bezogen werden können. Zum Beispiel bei der Erstellung von Anfragen ist ein vorzeitiger Abgleich von Beschränkungen hilfreich, sodass keine Anfragen entwickelt werden, die aufgrund unbekannter Einschränkung während der Entwicklung von Anbeginn nicht lauffähig sind.

Systemadministrator: Bedingungen, die für das gesamte IT-System oder auch nur für einzelne Rechenknoten gelten, werden mit der Basisrolle Systemadministrator definiert. Darunter fallen zum Beispiel Einschränkungen, dass nur ausgewählte Operatoren zur Ausführung auf einem Rechenknoten zugelassen sind. Bei der Erstellung und Vergabe erwarten Administratoren die Möglichkeit, Bedingungen detailliert auf heterogene Rechenkonten zuschneiden

zu können. Dem Systemadministrator fallen im Zuge der Einführung einer Sicherheitsarchitektur für NexusDS auch Aufgaben zu, die Sicherheitsarchitektur zu überwachen. So muss es der Rolle möglich sein, Protokolle und einen aktuellen Zustand der einzelnen Komponenten der Sicherheitsarchitektur zu überwachen.

5.2.2 Eigenschaften von NexusDS

Die Anforderungen, die sich aus der technischen Seite von NexusDS ableiten, sind im vorliegenden Unterabschnitt in die Hauptaspekte von NexusDS eingeteilt. Hauptaspekte leiten sich aus der Struktur, die in Abschnitt 3.2 vorgestellt wurde und dem Ausführungsmodell von NexusDS, das in Abschnitt 3.3.1 vorgestellt wurden, ab. Ziel der Gliederung von Anforderungen anhand der Hauptaspekte ist, dass das Sicherheitskonzept die Besonderheiten von NexusDS, wie die Ausnutzung von Eigenschaften heterogener Rechenknoten, angemessen im Sicherheitskonzept berücksichtigt werden.

Asynchrone Ausführung von Anfragen: Ein DSMS führt eine Anfrage über einen potentiell unbeschränkten Zeitraum T aus, in dem Änderungen an den Zugriffsbedingungen erfolgen können. Gewöhnliche Anfragen, zum Beispiel in einem DBMS, werden bezüglich der Zugriffseinstellungen zu einem festen Zeitpunkt $t_1 \in T$ ausgewertet und unter den ermittelten Bedingungen ausgeführt und beendet. Bei einer unter Umständen sehr langen Ausführungszeit einer Anfrage in einem DSMS, kann eine Änderung von Zugriffsbedingungen zum Zeitpunkt $t_2 \in T$, wobei $t_2 > t_1$, eintreten. Diese Änderung kann dazu führen, dass die Auswertung von Zeitpunkt t_1 ungültig wird und eine erneute Auswertung der Bedingungen erzwingen. Um Änderungen der Bedingungen zu erkennen, ist eine durchgehende Überwachung der Bedingungen über die gesamte Ausführungszeit hinweg notwendig. Sollten sich Änderungen ergeben und eine erneute Auswertung zeigt, dass der vorliegende Zustand der Ausführung nicht mehr bedingungskonform ist, muss die Anfrage zur Laufzeit angepasst werden. Die Mechanismen zur Überwachung müssen in der Lage sein, alle für die Anfrage relevanten Bedingungen über die Ausführungszeit überprüfen zu können. Dennoch sollten die Mechanismen sich möglichst gering auf die Leistungsfähigkeit von NexusDS auswirken, um die Einsetzbarkeit des DSMS, zum Beispiel in Echtzeitanwendungen, nicht einzuschränken.

Individuelle Erweiterbarkeit: In NexusDS ist es möglich, individuelle Operatoren zur Erweiterung der Funktionalität zu entwickeln. Dazu steht ein Operator-Modell bereit, dass Operator-Boxen aus verschiedenen Komponenten, wie bereits in Abschnitt 3.2 und Abbildung 3.3 beschrieben, zusammensetzt. Das führt zu dem Problem, dass die individuelle Programmierung der Komponenten ein Abzweigen oder anderes missbräuchliches Verwenden der Daten ermöglicht. Zur Unterbindung muss ein Mechanismus vorgesehen werden, der alle Komponenten der Operator-Box, die mit zu schützenden Informationen in Berührung kommen, kontrolliert. Die Kontrolle muss so gestaltet sein, dass die Verarbeitung der Daten nur im Einklang der definierten Bedingungen aller beteiligten Objekte und Subjekte geschieht. Neben der Verarbeitung muss ebenso sichergestellt werden, dass keine Informationen unbemerkt weitergeleitet, unberechtigt zwischengespeichert, gelesen oder verändert

werden. Ist zum Beispiel eine Weiterleitung möglich, könnten die Daten nicht berechtigten Zielen verfügbar gemacht werden. Trotz der notwendigen Kontrolle von individuellen Entwicklungen müssen die Zugriffsbedingungen und prüfenden Mechanismen so gestaltet sein, dass die Erweiterbarkeit von NexusDS nicht eingeschränkt wird.

NexusDS kann nicht nur mit Operatoren, sondern auch mit Services erweitert werden. Steht ein Service in Bezug zu geschützten Daten, für die eine Zugriffskontrolle erzwungen wird, muss der Service ebenfalls kontrolliert werden. Die Kontrolle erstreckt sich darüber, ob ein Service auf bestimmte Daten zugreifen darf und auf die Interaktion mit weiteren Services oder Subjekten. Ein Service darf nur dann auf geschützte Daten zugreifen, wenn er authentifiziert ist und alle relevanten Bedingungen den Zugriff gestatten. Relevant sind die Bedingungen, die mit alle beteiligten Subjekte und Objekte der Aktion verbunden sind.

Einbindung von Operatoren zur Laufzeit: Ein Nachladen von Operatoren zur Laufzeit erfordert, dass alle zu berücksichtigten Bedingungen dynamisch mit dem einzubindenden Operator abgeglichen werden. Möglicherweise kann der Operator aufgrund bestehender Bedingungen der betroffenen Anfrage oder Rechenknoten nicht verwendet werden. Ist das der Fall, muss ein Ersatz gefunden werden und gegebenenfalls eine Umplanung der Anfrage durchgeführt werden, sodass alle relevanten Bedingungen eingehalten werden.

Verteilte Architektur: Die Ausführung von Anfragen und Services in NexusDS kann über mehrere Rechenknoten verteilt werden. Diese Knoten sind nicht notwendigerweise lokal, das heißt, die Ausführung kann über ein Netzwerk, wie zum Beispiel über das Internet, auf Rechenknoten verteilt werden. Aus dieser Begebenheit entsteht die Anforderung, dass alle Kommunikationswege im Sicherheitskonzept gegen Abhören und Manipulation geschützt werden müssen. Bei der verteilten Verarbeitung von Anfragen muss sichergestellt werden, dass die Bedingungen der Anfrage an allen Rechenknoten erfüllt sind, bevor es zu einer Verarbeitung von Objekten kommt. Die Bedingungen müssen daher nicht nur rechtzeitig vor der Verarbeitung betroffener Objekte verteilt werden, sondern es muss auch in Erwägung gezogen werden, dass unter Umständen die Verbindung zwischen Rechenknoten unterbrochen wird. Sollten in dieser Zeit Bedingungen verändert werden, muss nach Wiederherstellung der Verbindung dafür gesorgt werden, dass die neuen Bedingungen dem Rechenknoten bekannt gemacht werden.

Heterogene Rechenknoten: Ein heterogenes Umfeld an Rechenknoten betrifft nicht nur die Architektur der Recheneinheiten hinsichtlich der Hardware oder Softwareausstattung, sondern kann sich ebenso in der Ausgestaltung der Zugriffseinstellungen niederschlagen. Unterliegen die Rechenknoten bestimmten Einschränkungen, dass aufgrund der Eigenschaften des Rechenknotens gewisse Bedingungen nicht erfüllt werden können, muss die Einschränkung bei der Verteilung der Anfrage berücksichtigt werden. Stellt zum Beispiel eine Bedingung der Anfrage die Forderung, dass auf Rechenknoten zur Ausführung eine spezielle Betriebssystemüberwachung installiert ist, müssen Rechenknoten mit dieser Eigenschaft gefunden werden. Auf der Seite der Rechenknoten können auch Bedingungen definiert werden, die Verteilung von Anfragen unter Umständen beeinflusst. Zum Beispiel könnte ein Systemadministrator eines Rechenknoten, die Ausführung bestimmter

Operatoren ausschließen, was den Rechenknoten für Anfragen mit betroffenen Operatoren unbrauchbar macht.

5.3 Zusammenfassung der Anforderungen

Die Anforderungen aus Anwendungsszenarien, Basis-Rollen und den Eigenschaften von NexusDS wurden in den vorhergehenden Abschnitten ausgearbeitet. Vorliegender Abschnitt fasst die zentralen Aspekte der Anforderungen zusammen und konsolidiert die Aspekte in die Schutzziele.

Authentizität: Für die korrekte Zuordnung von Bedingungen ist eine zuverlässige Authentifizierung von Subjekten und Objekten notwendig. Denn es kann nur dann eine eindeutige Zuordnung sichergestellt werden, wenn jedes Subjekt und Objekt eindeutig identifiziert werden kann. Das heißt jedes Subjekt und Objekt, dass an dem durch das Sicherheitskonzept geschützten Teil von NexusDS teilnehmen will, muss eine eindeutige Identität zugeordnet werden. Die Authentizität wird dann sichergestellt, indem Mechanismen die von einem Subjekt oder Objekt behauptete Identität prüfen und die Identitäten nur dann als echt bestätigen, falls es der wahren Identität des Subjektes oder Objektes entspricht.

Datenintegrität: Objekte dürfen nur mit entsprechender Berechtigung modifiziert werden, das setzt voraus, dass für alle Subjekte und Objekte Zugriffsbedingungen für den Zugriff zugeordnet sind. Wünscht ein Subjekt die Veränderung von Objekten, wird vor Veränderung zuerst die Zulässigkeit des Zugriffes geprüft. Ein Zugriff kann zum Beispiel die Erstellung und Zuordnung neuer Bedingungen für den Zugriff auf ein Objekt sein. Zulässig ist der Zugriff nur dann, wenn alle an der Veränderung beteiligten Subjekte und Objekte korrekt authentifiziert wurden und alle zugeordneten Zugriffsbedingungen erfüllt werden. Ist der Zugriff nicht zulässig, muss sichergestellt werden, dass die Sicherheitsarchitektur den Zugriff unterbindet.

Informationsvertraulichkeit: Die Informationsvertraulichkeit setzt eine korrekte Authentifizierung von Subjekten und Objekten voraus. Weiter müssen Zugriffsbedingungen an die zu schützende Informationen gebunden sein. Kern des Schutzzieles ist, dass Objekte in unterschiedlicher Granularität vor unberechtigter Einsicht geschützt werden. Auch dann, wenn sie von individuell entwickelten Operator-Boxen verarbeitet werden. Das umfasst die Prüfung aller beteiligten Objekte und Subjekte, zum Beispiel die Anfrage selbst, und die eingesetzten Operatoren und Services die den Zugriff ausführen. Die Informationsvertraulichkeit auch dann zu gewährleisten, wenn Komponenten individuell entwickelt wurden, macht Mechanismen erforderlich, die trotz einer freien und individuellen Entwicklung eine zuverlässige Kontrolle über geschützte Daten sicherstellten.

Verfügbarkeit: Kommt es bei der Ausführung von Anfragen auf Rechenknoten zu Konflikten bei der Ressourcenverteilung ist unter Umständen die Zuverlässigkeit sensibler Anfragen gefährdet. Für die Diplomarbeit wird die Annahme getroffen, dass die Betreiber von Diensten in NexusDS dafür Sorge tragen, dass ausreichend Ressourcen für die Ausführung aller

gestatteten Anfragen vorhanden sind. In diesem Fall kann ein Ressourcenkonflikt nur dann auftreten, wenn nicht berechtigte Anfragen auf einem geschützten Rechenknoten des Dienstbetreibers zur Ausführung gebracht werden. Das Sicherheitskonzept muss dafür Sorge tragen, dass nur solche Anfragen auf geschützten Rechenknoten zur Ausführung kommen, die vom Besitzer des Rechenknotens gestattet sind. Eine detaillierte Vorgehensweise zur Zuweisung von Prioritäten im Falle von Ressourcenkonflikten, sind kein unmittelbares Thema der Diplomarbeit und werden nicht weiter vertieft.

Anonymisierung: Anonymisierung bedeutet, dass persönliche Daten so verfälscht oder gesperrt werden, dass danach keine Rückschlüsse auf persönliche Verhältnisse möglich sind. Die gezeigten Anwendungsszenarien stellen höhere Anforderungen als ein einfaches Sperren oder Zufälliges verfälschen von Informationen. Die Anonymisierung in NexusDS muss eine Filterung und Verschleierung von Objekten in Abhängigkeit von individuellen Bedingungen und Anwendungsszenarien leisten. Zum Beispiel, dass der Detailgrad von Positionsinformation in Abhängigkeit des Empfängers der Information variiert, sodass als Freunde definierte Datenempfänger einen genauen Standort und Arbeitskollegen nur einen Landkreis empfangen. NexusDS stellt als Plattform keine Einschränkung an Datenstrukturen oder im Allgemeinen wie Informationen dargestellt werden. Deshalb muss das Sicherheitskonzept eine Möglichkeit vorsehen, beliebige Tranformationen auf mindestens den Datentypen zu unterstützen, die auch NexusDS unterstützt.

Kapitel 6

Grundlagen des Sicherheitskonzeptes

Der nächste Schritt in der Entwicklung des Sicherheitskonzeptes ist, nach dem Vorgehensmodell aus 2.4, Maßnahmen zur Erfüllung der in Kapitel 5 aufgestellten Anforderungen zu entwickeln.

Das Kapitel stellt im ersten Abschnitt die zentralen Maßnahmen des Sicherheitskonzeptes vor, ohne bereits detaillierte Komponenten der Sicherheitsarchitektur zu entwerfen. Zur kontrollierten Verarbeitung von Anfragen erläutert Abschnitt 6.2 drei Sicherheitszonen, die die kontrollierte Anfrageverarbeitung in unterschiedliche Strenge umsetzen.

6.1 Basisstruktur des Sicherheitskonzeptes

Grundlage des Sicherheitskonzeptes ist eine Sammlung von Maßnahmen, die die zuvor aufgestellten Anforderungen umsetzen können. Es gilt zu beachten, dass die Maßnahmen sich gegenseitig ergänzen und nicht alleinstehend betrachtet werden dürfen.

6.1.1 Übersicht der Maßnahmen für das Sicherheitskonzept

Um Bedingungen zum Zugriff auf Objekte zu strukturieren, wird ein Meta-Daten-Modell auf Basis der *Augmented World Model Language (AWML)* eingeführt, dass die Spezifikation von **Zugriffsrichtlinien** erlaubt. Operatoren werden geschützt, indem sie in einer **zentralen Datenhaltung für Operatoren** in der Sicherheitsarchitektur verfügbar gemacht werden. Die Datenhaltung ersetzt das ursprüngliche *Operator Repository* von NexusDS. Alle Operatoren aus dieser Datenhaltung können nur noch unter der Kontrolle der Sicherheitsarchitektur ausgeführt werden und jeder Zugriff muss explizit mit einer an den Operator gebundenen Zugriffsrichtlinie erlaubt werden. Das gilt ebenfalls für die Datenströme, die die geschützten Operatoren generieren. Zum Beispiel eine geschützte Quelle kann nur unter Beachtung der an die Quelle gebundenen Zugriffsrichtlinien ausgeführt werden. Die Datenströme, die die Quelle erzeugt, können ebenfalls nur noch unter Beachtung der mit der Quelle verbundenen Zugriffsrichtlinien gelesen werden. Zugriffsrichtlinien definieren, unter welchen Bedingungen Zugriffe erlaubt sind, alle nicht von Zugriffsrichtlinien explizit definierten Zugriffe sind nicht erlaubt.

Zugriffsrichtlinien müssen in der verteilten Umgebung von NexusDS so vorgehalten werden, dass Services zur Entscheidung ob ein Zugriff gestattet ist, die aktuell gültigen Zugriffsrichtlinien abrufen können. Dazu wird ein **zentraler Service zur Verwaltung von Zugriffsrichtlinien** eingeführt, der alle Zugriffsrichtlinien vorhält. An diesem zentralen Service werden Zugriffsrichtlinien erstellt, geändert und gelöscht. Alle Subjekte, Services und Anwendungen die Zugriffsrichtlinien definieren wollen, verbinden sich direkt mit diesem Service. Die weitere Verteilung der Zugriffsrichtlinien wird von der Sicherheitsarchitektur selbst abgewickelt.

Um Subjekte in der Sicherheitsarchitektur korrekt zu identifizieren, ist eine Zuordnung von eindeutigen Identitäten und einem Mechanismus zur Bestätigung von Identitäten notwendig. Diese Aufgabe wird von einem **zentralen Service zur Verwaltung von Identitäten** realisiert. Alle Subjekte, die von der Sicherheitsarchitektur geschützte Operatoren verwenden wollen, müssen sich über den zentralen Service mit einer eindeutigen Identität registrieren. Zudem trifft der Service mit dem jeweiligen Subjekt eine Vereinbarung, wie die Identität bestätigt werden kann. Handelt es sich bei der Identität um einen Benutzer, wird eine eindeutige Kombination aus Benutzername und Passwort vergeben. Zur Prüfung der Identitäten eines Rechenknotens werden je Rechenknoten eindeutige Identifikatoren vergeben, die den Rechenknoten eindeutig charakterisieren. Identitäten von Operatoren bestehen aus einem Bezeichner, der bei der Verfügbarmachung des Operators im zentralen Operator Repository der Sicherheitsarchitektur zugewiesen wird. Bestätigt wird die Identität durch einen eindeutigen Hashwert, der über die Implementierung des Operators gebildet wird.

Wird die Ausführung einer Anfrage angestoßen, durchläuft die Anfrage verschiedene Schritte im bereits vorgestellten Ausführungsmodell, Abschnitt 3.3.1, von NexusDS. Verantwortlicher Service für die Aufgabe ist der *Core Query Service (CQS)*, der für das Sicherheitskonzept so erweitert wird, dass während der Planung alle für die Anfrage relevanten Zugriffsrichtlinien berücksichtigt werden. Relevant sind alle die Zugriffsrichtlinien, die an Operatoren der auszuführenden Anfrage gebunden sind. Zum Beispiel falls Datenelemente einer Quelle nur von bestimmten Operatoren gelesen werden dürfen. Daraus entsteht eine **kontrollierte Anfrageplanung**, die nur die Anfrage zur Ausführung bringt, deren Zugriffe durch Zugriffsrichtlinien gedeckt sind.

Wird eine Zugriffsrichtlinie zum Zeitpunkt t_D definiert, muss diese für alle durch die Richtlinie betroffenen Datenelemente, die zum Zeitpunkt $t_V \geq t_D$ in die Verarbeitung gehen, durchgesetzt werden. Würde das Sicherheitskonzept nur eine zentrale Datenhaltung für Zugriffsrichtlinien vorsehen, kann aber nicht ausgeschlossen werden, dass bei der verteilten Ausführung ein Zeitraum $t_V - t_D > 0$ bis zur Durchsetzung verstreicht. Denn die Zugriffsrichtlinien müssen zuerst von der definierenden Stelle an den zentralen Speicher übertragen werden. Von der zentralen Stelle wiederum an die verarbeitende Stelle, was eine gewisse Übertragungszeit beansprucht. In der verstrichenen Übertragungszeit $T_{delay} = t_V - t_D$, wobei $T_{delay} > 0$, könnten Datenelemente bereits mit veralteten Zugriffsrichtlinien verarbeitet worden sein, was die Gefahr von Verletzungen von Zugriffsrichtlinien birgt. Das in Abschnitt 4.4 vorgestellte Framework *FENCE* löst das Problem durch **Einflechten von Zugriffsinterpunktionen in den Datenstrom**. Es arbeitet unabhängig von möglichen Verzögerungen bei der verteilten Ausführung und transportiert die Zugriffsrichtlinien direkt an die Stelle, an der die Verarbeitung stattfindet. Würde eine globale Verteilung

stattfinden, indem eine zentrale Komponente alle Zugriffsrichtlinien an die betroffenen Operatoren weiterleitet, dann ist eine globale Zeitreferenz notwendig. Anhand der Zeitreferenz könnten Zeitstempel für Zugriffsrichtlinien definieren, ab wann eintreffende Datenelemente mit der neuen Zugriffsrichtlinie zu verarbeiten sind. Jedoch kann in einer verteilten Ausführungsumgebung, wie bereits besprochen nicht davon ausgegangen werden, dass die Zugriffsrichtlinien an alle Stellen mindestens zu dem Zeitpunkt eintreffen, wenn betroffene Zugriffsrichtlinien angewandt werden müssen. Es wäre dann möglich, dass ein Teil der Operatoren, verteilt über verschiedene Rechenknoten, die Zugriffsrichtlinie rechtzeitig erhalten, der verbleibende Teil der Operatoren aber nicht. Damit würde ein Teil der Anfrage auf Basis aktueller und ein weiterer Teil auf Basis veralteter Zugriffsrichtlinien arbeiten. Die Einflechtung vermeidet derartige Probleme, da sich die Zugriffsrichtlinien auf dem gleichen Weg wie die betroffenen Datenelemente fortbewegen.

Aus diesen Gründen wird das Konzept der Interpunktionen für die Diplomarbeit übernommen und mit einer zentralen Datenhaltung kombiniert. Die zentrale Datenhaltung bildet den zentralen Punkt zur Bearbeitung von Zugriffsrichtlinien. Gleichzeitig dienen sie als Quelle für Zugriffsrichtlinien bei der kontrollierten Planung der Anfrage und zur Synchronisation von Zugriffsrichtlinien, falls Verbindungsabbrüche bei der verteilten Ausführung auftreten. Soll eine Zugriffsrichtlinie unmittelbar gültig werden und sich auch auf die bereits in der Verarbeitung befindlichen Datenelemente auswirken, kann die sofortige Anwendung erzwungen werden. Dies muss dann explizit in den Zugriffsrichtlinien definiert werden, die unmittelbar auf alle Datenelemente angewendet werden sollen. Wird eine Zugriffsrichtlinie definiert, die die sofortige Anwendung erzwingt, verteilt die Sicherheitsarchitektur die Zugriffsrichtlinien an alle in der Ausführung betroffenen Operatoren.

Die Services des Sicherheitskonzeptes erhalten ein System zur **Protokollierung**. Das Protokoll enthält alle Informationen über die durchgeführten Veränderungen, sodass nachvollzogen werden kann, von welchem Subjekt welche Änderungen zu welcher Zeit getätigt wurden. Zum Beispiel falls Veränderungen am Datenbestand der Zugriffsrichtlinien durchgeführt werden. Ebenso werden ausgeführte Anfragen zu Protokoll gebracht. Hierzu hält die kontrollierte Anfrageplanung die Anfrage selbst, Zeitpunkt und die Identität des Subjektes fest, dass die Anfrage abgeschickt hat. Dadurch lässt sich die Verwendung von Operatoren und Datenströmen von Subjekten nachvollziehen.

Die Services, aus denen die Sicherheitsarchitektur besteht, müssen mit weiteren Services der Sicherheitsarchitektur kommunizieren. Jeder Service der Sicherheitsarchitektur verfügt, sofern dieser mit Services kommunizieren will, über ein digitales Zertifikat, das zu Beginn einer Kommunikation überprüft wird. Die Erstellung, die Verwaltung und die Zuordnung von Zertifikaten werden über eine **Public-Key-Infrastruktur**, siehe Abschnitt 2.2.1, realisiert. Die Zertifikate ermöglichen eine zuverlässige, gegenseitige Authentifizierung von Services. Will ein Subjekt eine Verbindung mit einem Service aufbauen, zum Beispiel um neue Zugriffsrichtlinien zu erstellen, überprüft es ebenfalls zuvor das digitale Zertifikat des Services. Der Service seinerseits kann über den zentralen Service zur Verwaltung von Identitäten überprüfen, ob das Subjekt authentisch ist. So ist sichergestellt, dass nur dann sensitive Information ausgetauscht wird, wenn beide Seiten verlässlich authentifiziert wurden.

Bisher vorgestellte Sicherheitskonzepte für NexusDS ähnliche Systeme, in Abschnitt 4 wurde eine Auswahl vorgestellt, erlauben oder verbieten den Zugriff auf Datenelemente in Abhängigkeit der definierten Zugriffsbedingungen. Die Sicherheitsarchitektur für NexusDS ermöglicht nicht nur das einfache Verbieten eines Zugriffes, sondern auch eine Transformation der Datenelemente um Zugriffsbedingungen zu erfüllen. Da NexusDS grundsätzlich jeden Typ von Daten unterstützt, muss das Sicherheitskonzept flexibel genug sein, um beliebige Transformationen durchführen zu können. **Zur Unterstützung von beliebigen Datentypen und Transformationen erlaubt die Sicherheitsarchitektur das Implementieren von Filtern**. Filter werden von *NexusDS Extention Developer* implementiert und in der Sicherheitsarchitektur in einer zentralen Datenhaltung verfügbar gemacht. Filter müssen explizit in Zugriffsrichtlinien definiert werden. Bezieht sich eine Zugriffsrichtlinie auf eine Quelle, muss definiert werden, auf welche Datenausgänge der Filter angewendet werden soll. Handelt es sich um eine Operation oder Senke, müssen die Dateneingänge definiert werden, auf die der Filter angewendet wird. Für einen Datenausgang oder Dateneingang ist möglich, mehrere Filter zu definieren. Sollte das der Fall sein, werden die Filter nach einer in den Zugriffsrichtlinien definierten Reihenfolge hintereinander ausgeführt.

Zur Sicherung der Kommunikationswege werden **alle Datenkanäle verschlüsselt**. Eine Verschlüsselung der Datenströme schützt vor unberechtigtem Lesen und Manipulation der Daten. Das betrifft sowohl Datenströme, die in Anfragen verarbeitet werden, als auch Kanäle zum Übertragen von zum Beispiel neuen Zugriffsrichtlinien in die Datenhaltung.

Mit einer expliziten Definition in Zugriffsrichtlinien, **können geschützte Datenströme für den ungeschützten Teil von NexusDS freigegeben** werden. Zum Beispiel könnte für eine Quelle, die GPS-Positionsinformationen erzeugt eine Zugriffsrichtlinie definiert werden, die unter Anwendung eines Filters für das ursprüngliche NexusDS freigegeben wird. Der Filter kann eine Transformation realisieren, die den Detailgrad der Positionsinformation reduziert, die dem Besitzer der Positionsinformation als Anonymisierung ausreicht.

Das Anwendungsszenario der intelligenten Fabriken aus Abschnitt 5.1.3 deutete an, dass NexusDS in komplexen Anwendungslandschaften eingesetzt werden kann. Häufig bestehen in umfangreichen Anwendungslandschaften vielfältige Datenhaltungen, die für die Entscheidung einer Zugriffsberechtigung ausgewertet werden müssen. Zur Vereinfachung der Integration bestehender Systeme ermöglicht die Sicherheitsarchitektur den Einsatz von **Evaluatoren**. Evaluatoren werden von *NexusDS Exention Developer* entwickelt und in einer zentralen Datenhaltung der Sicherheitsarchitektur verfügbar gemacht. Mit der Referenzierung auf Evaluatoren in Zugriffsrichtlinien wird die Auswertung von Zugriffsberechtigungen nicht mehr von der Sicherheitsarchitektur selbst, sondern von den individuellen Evaluatoren vorgenommen. Dadurch vereinfacht sich die Integration der Sicherheitsarchitektur in bereits vorhandene Anwendungslandschaften wesentlich.

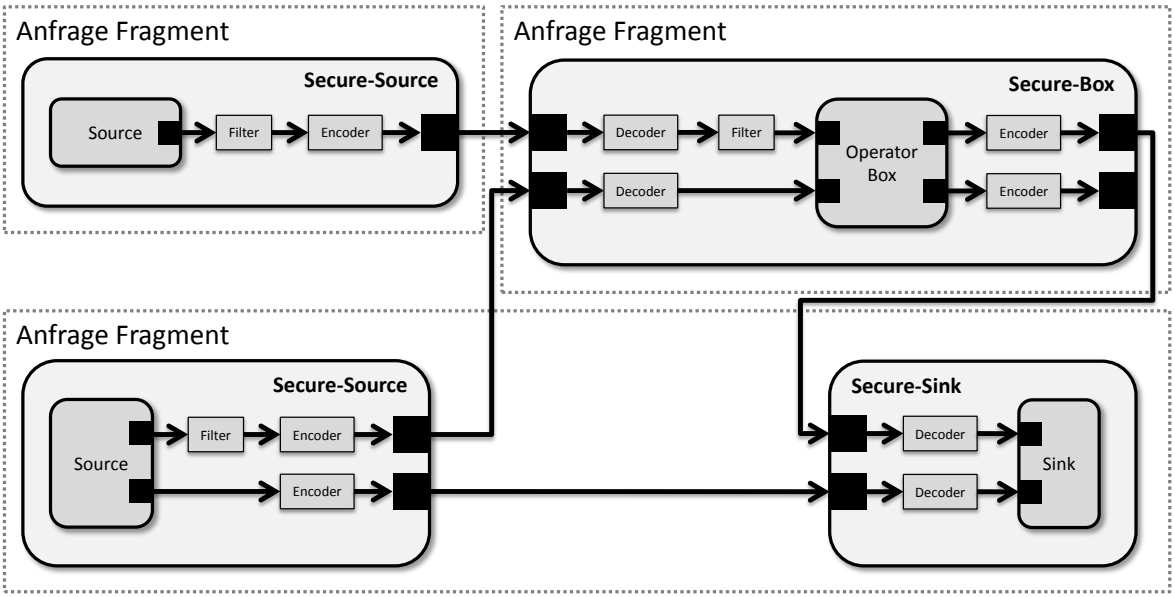

Abbildung 6.1: Vereinfachtes Modell der Verarbeitung von Datenströmen in Anfragen der *Sicherheitszone-Mittel* und *Sicherheitszone-Hoch*.

6.2 Kontrollierte Datenstromverarbeitung in drei Sicherheitszonen

Das Sicherheitskonzept teilt die Ausführung von Anfragen in drei Sicherheitszonen ein, die im Folgenden vorgestellt werden. **Sicherheitszone-Null** entspricht dem bestehenden NexusDS ohne Zugriffskontrollen. Die **Sicherheitszone-Mittel** realisiert eine kontrollierte Verarbeitung von Anfragen, die es erlaubt, bestehende Quellen, Operatoren und Senken von NexusDS weiterzuverwenden. Um Risiken durch unkontrolliertes Verhalten des Operator-Modells auszuschließen, erzwingt die **Sicherheitszone-Hoch** die Nutzung von zertifizierten Komponenten des Operator-Modells.

6.2.1 Sicherheitszone-Null

NexusDS in der vorliegenden Form, ohne Sicherheitskonzept, repräsentiert *Sicherheitszone-Null*. Es werden grundsätzlich keine Kontrollmechanismen vorgesehen, die den Datenzugriff in Anfragen einschränken. Datenelemente von geschützten Operatoren, können in der *Sicherheitszone-Null* nur mit klar definierten Ausnahmen verwendet werden. Die Ausnahme muss explizit in Zugriffsrichtlinien definiert werden, indem ein Datenausgang als ungeschützt deklariert wird. Falls gewünscht können dem Datenausgang Filter vorgelagert werden, sodass die Daten nur in einem durch die Filter bestimmten Detailgrad ungeschützt weiterverarbeitet werden.

6.2.2 Sicherheitszone-Mittel

Sicherheitszone-Mittel betten die Ausführung von Quellen, Operator-Boxen und Senken in eine kontrollierte Umgebung ein. Abbildung 6.1 illustriert die Einbettung der Boxen, die unverändert aus dem ursprünglichen NexusDS übernommen werden können. Für jeden Boxentyp existiert ein spezifischer Baustein der Sicherheitsarchitektur, der als Einbettung bezeichnet wird. Für Quellen die **Secure-Source**, für Operatoren die **Secure-Box** und für Senken die **Secure-Sink**. Die Datenströme zwischen den Einbettungen werden mit einem symmetrischen Schlüssel verschlüsselt, was gegenüber asymmetrischen Kryptosystemen einen Geschwindigkeitsvorteil birgt [16]. Da der Schlüssel nicht an Stellen außerhalb der Sicherheitsarchitektur gegeben wird, ist ein asymmetrisches Verfahren aus privatem und öffentlichem Schlüssel nicht notwendig. Jede Anfrage erhält einen eigenen geheimen Schlüssel, der von dem Service verwaltet wird, der die betroffene Anfrage geplant hat. Der Schlüssel wird mit einer Verfallszeit von 48 Stunden versehen, um die Wahrscheinlichkeit eines Bekanntwerdens des geheimen Schlüssels zu reduzieren. Läuft ein Schlüssel zur Ausführungszeit einer Anfrage ab, kümmern sich die Einbettungen um eine Erneuerung, sodass die Ausführung unterbrechungsfrei auf einen neuen geheimen Schlüssel umgestellt werden kann. Zur Geheimhaltung des Schlüssels ist dieser nur dem Service zur Verwaltung der Anfrageplanung und den jeweiligen Einbettungen bekannt, er wird nicht den eingebetteten Operatoren oder anderen Teilen von NexusDS verfügbar gemacht. In Abbildung 6.1 ist zu sehen, wie ausgehende Datenströme durch Encoder verschlüsselt werden und eingehende Datenströme mit Decoder entschlüsselt werden. So werden die Datenströme zwischen Boxen direkt verschlüsselt und damit unlesbar zwischen verschiedenen Anfragen und für alle Subjekte außerhalb der Sicherheitsarchitektur.

Jede geschützte Quelle wird in eine **Secure-Source** eingebettet. Filter sind, wie in Abbildung 6.1 zu sehen, direkt der eingebetteten Quelle nachgelagert und transformieren Datenelemente der Quelle. Welche Filter zur Anwendung kommen, wird von den mit der Quelle verbundenen Zugriffsrichtlinien definiert. Neben der optionalen Transformation von Datenelementen durch Filter stellt die Secure-Source Einbettung sicher, dass nur dann Datenelemente von der Quelle weitergeleitet werden, wenn der Zugriff auf die Datenelemente gestattet ist. Wird eine Anfrage zur Ausführung gebracht, wird dies von der kontrollierten Anfrageplanung sichergestellt. Ändern sich jedoch zur Ausführungszeit Zugriffsrichtlinien, die die Quelle betreffen und eine weitere Ausführung verbieten, sperrt die Quelle die Datenausgänge. Dass Zugriffsrichtlinien sich gegenseitig ausschließen, ist nicht möglich, da jede Zugriffsrichtlinie einer Erlaubnis entspricht. Wird während der Ausführungszeit eine Zugriffsrichtlinie zurückgenommen, die die Ausführung einer Secure-Source erlaubt, sperrt die Secure-Source sofort die Weiterleitung der Datenelemente der eingebetteten Quelle. Gleichzeitig meldet die Secure-Source der Ausführungsumgebung, dass die Anfrage abgebrochen werden muss. Bevor Datenelemente die Quelle verlassen, werden diese mit dem Encoder verschlüsselt. Neben dem Verschlüsseln von Datenelementen interpunktiert der Encoder alle für die Quelle definierten Zugriffsrichtlinien, die ebenfalls verschlüsselt werden, in den Datenstrom.

Um die Datenelemente von geschützten Operatoren verarbeiten zu können, müssen die Operatoren der betroffenen Anfragen in jeweils eine **Secure-Box** eingebettet werden. Die Secure-Box erhält Zugriff auf den geheimen Schlüssel, um die eingehenden verschlüsselten Datenströme zu entschlüsseln und die interpunktierten Zugriffsrichtlinien zu entfernen, sodass der eingebettete Operator die Datenelemente verarbeiten kann. Sollten in Zugriffsrichtlinien Filter für die eingebettete Operator-Box definiert sein, werden sie nach dem Encoder wie in Abbildung 6.1 zu sehen den Eingängen der Operator-Box vorgelagert. Werden neue Zugriffsrichtlinien bekannt, entweder über eingehende Interpunktionen oder die Secure-Box wird von dem zentralen Service zur Verwaltung der Zugriffsrichtlinien benachrichtigt, überprüft die Secure-Box Bedingungen der Zugriffsrichtlinie. Zum Beispiel kann der Fall eintreten, dass eine neue Zugriffsrichtlinie die Ausführung eines Operators O auf wenige ausgewählte Rechenknoten beschränkt und alle anderen Zugriffsrichtlinien, die die Ausführung des Operators vorher erlaubten, zurück genommen werden. Ist eine Secure-Box betroffen, da diese O einbettet, überprüft die Secure-Box, ob die Bedingungen der neuen Zugriffsrichtlinie erfüllt sind. Ist das nicht der Fall, sperrt die Secure-Box alle Datenausgänge und beendet die Anfrage analog zur Secure-Sink.

Über die Dateneingänge eingehende Zugriffsrichtlinien müssen auch den nachfolgenden Boxen zur Verfügung gestellt werden, was eine Wiederinterpunktion notwendig macht. Die Wiederinterpunktion einer Zugriffsrichtlinie muss in alle Datenausgänge erfolgen, die von dem Datenstrom abhängen, über den die Zugriffsrichtlinie eingetroffen ist. Zu welchem Zeitpunkt die Wiederinterpunktion erfolgt, hängt von der Relation zwischen der Anzahl der eingehenden und ausgehenden Datenelemente ab. Die Details zur Wiederinterpunktion werden im weiteren Verlauf erörtert. Bevor die verarbeiteten Datenelemente und Interpunktion die Secure-Box verlassen, wird der Datenstrom analog zur Secure-Source wieder mit dem geheimen Schlüssel der Anfrage verschlüsselt.

Das Senken die Ergebnisse von geschützten Anfragen lesen können, müssen sie jeweils in eine **Secure-Sink** eingebettet werden. Zentrale Aufgabe ist die eingehenden Datenströme zu entschlüsseln, gegebenenfalls Filter den Dateneingängen vorzulagern und die Datenelemente an die eingebettete Senke weiterzureichen. Parallel nimmt die Secure-Sink noch die Aufgabe wahr, die für die Senke relevanten Zugriffsrichtlinien zu überprüfen. Sollte sich eine Änderung in den Zugriffsrichtlinien ergeben, die die eingebettete Senke betrifft, wird analog zur Secure-Sink und Secure-Box vorgegangen.

6.2.3 Sicherheitszone-Hoch

Die vorgestellte *Sicherheitszone-Mittel* bettet Quellen, Operator-Boxen und Senken so ein, dass nur auf die Datenelemente zugegriffen wird, für die eine Zugriffsrichtlinie den Zugriff erlaubt. Das Verhalten der eingebetteten Komponenten kann jedoch nicht automatisch von der Sicherheitsarchitektur überprüft oder überwacht werden. Beispielsweise könnte ein Input-Manager nach wie vor einen Datenkanal öffnen, alle eingehenden Datenelemente kopieren und an eine unautorisierte Stelle weiterleiten. Ein weiteres Beispiel wäre der Fall, dass es Operatoren lediglich erlaubt sein soll, dass Durchschnitte nur über maximal x Datenelemente gebildet werden dürfen. Ein Operator könnte aber eine Datenhaltung führen die $x + n$, wobei $n > 0$, Datenelemente

speichert und darüber einen Durchschnitt bilden. Um der offenen Lücke zu begegnen, erzwingt die *Sicherheitszone-Hoch* eine digitale Signierung von Operatoren von ausgewählten Prüfern nach einem definierten Kriterienkatalog. Dazu können Zugriffsrichtlinien Signaturen definieren, die von der Anfrageplanung und Einbettungen überprüft werden. Die höhere Sicherheit hat aber einen Preis. Alle Komponenten, die in der Sicherheitszone eingesetzt werden sollen, müssen manuell überprüft und signiert werden.

Jeder Prüfer der Komponenten signieren möchte, muss als Prüfer an einem zentralen Service der Sicherheitsarchitektur registriert werden. Die Signierung basiert auf einem asymmetrischen Kryptosystem wie in Abschnitt 2.2.1 vorgestellt. Der öffentliche Schlüssel wird in der zentralen Datenhaltung für Zertifizierungen abgelegt, sodass der öffentliche Schlüssel zur Prüfung von Signaturen frei verfügbar ist. Hat ein Prüfer ein Operator inspiziert, berechnet er die Signatur der Komponente, verschlüsselt sie mit seinem privaten Schlüssel und legt die verschlüsselte Signatur in der zentralen Datenhaltung für geschützte Operatoren ab. Fordern Zugriffsrichtlinien digitale Signaturen für Operatoren, sorgt die kontrollierte Anfrageplanung dafür, dass nur Operatoren zur Ausführung kommen, die digitale Signaturen nach den Angaben in den betreffenden Zugriffsrichtlinien aufweisen. Zur Laufzeit von Anfragen stellen die Einbettungen sicher, dass die Angaben von Signaturen erfüllt werden. Bei Änderungen zur Laufzeit wird analog zur vorherigen Sicherheitsstufe eine Prüfung durchgeführt und bei einem Fehlschlag die Anfrage abgebrochen. Es ist keine Einschränkung vorgesehen, dass eine Box mit nur einer digitalen Signatur versehen werden darf. Werden durch die Zugriffsrichtlinien mehrere Signaturen definiert, wird die betroffene Komponente auf die korrekte Signatur von allen definierten Prüfern untersucht. Nur falls alle Zertifizierungen erfüllt sind, wird der Zugriff gestattet. Es ist zu beachten, besteht die Box wie im Falle eines Operators und Senke aus mehreren Komponenten, muss die Signatur für alle Komponenten gelten.

Das Stellen von Prüfern und wie eine Prüfung zur Zertifizierung abläuft, hängt von den Anwendern von NexusDS ab. Als Grundlage für Kriterien zur Zertifizierung wurde in Abschnitt 2.2.3 die *Common Criteria for Information Technology Security Evaluation* vorgestellt, die als internationaler Standard eine Referenz sein könnten. Die Möglichkeit verschiedene digitale Signaturen und damit verschiedene Prüfer zu wählen, eröffnet die Möglichkeit verschiedene Vertrauensinstanzen zu schaffen. Beispielsweise könnten alle Komponenten, die von Google entwickelt werden, mit einer Google Standardsignatur versehen werden. Die Prüfungsrichtlinien hinter der Signatur verspricht womöglich keine vollständige und detaillierte Überprüfung des Programmcodes, mag aber für einige Anwendungsszenarien als Garantie ausreichen.

Das Versehen von Komponenten der Operator-Boxen mit Signaturen zur Zertifizierung darf nicht mit den bereits besprochenen Identitäten verwechselt werden. Jede Komponente einer Operator-Box hat auch ohne Zertifizierung eine Identität. Dies sagt jedoch nichts darüber aus, ob die Komponente in irgendeiner Form überprüft wurde, sondern ist lediglich dazu da, die Objekte eindeutig und zweifelsfrei zu identifizieren.

Kapitel 7

Architektur des Sicherheitskonzeptes

Auf die Vorstellung des Sicherheitskonzeptes im vorherigen Kapitel folgt die Umsetzung des Konzeptes zu einer Sicherheitsarchitektur. Die Sicherheitsarchitektur besteht aus mehreren Services und Bausteinen, die im vorliegenden Kapitel ausführlich erläutert werden. Zur Erläuterung gehören eine detaillierte Beschreibung des Zusammenspiels mit den weiteren Teilen der Sicherheitsarchitektur und die Funktionsweise und Aufgaben jedes Services und Bausteines.

Im ersten Abschnitt 7.1 werden die Grundlagen zur Kommunikation in der Sicherheitsarchitektur erläutert. Die Definition und Auswertung von Zugriffsrichtlinien wird in Abschnitt 7.2 ausführlich erläutert. Abschnitt 7.3 adressiert die Wiedereinflechtung von Zugriffsrichtlinien in Datenströme nach Operator-Boxen.

Der Secure Core Query Services (SCQS) ist für die kontrollierte Planung von Anfragen verantwortlich und wird in Abschnitt 7.4 vorgestellt. Die bereits erwähnten Einbettungen sind je in Abschnitt 7.5 zur Secure-Source, Abschnitt 7.6 zur Secure-Box und Abschnitt 7.7 zur Secure-Sink erläutert. Die folgenden Abschnitte erläutern die Services der Sicherheitsarchitektur. Neben dem SCQS existiert noch eine weitere Zahl an Services, die für die Ausführung der Sicherheitsarchitektur verantwortlich sind und werden in Abschnitt 7.8 erläutert.

7.1 Kommunikation in der Sicherheitsarchitektur

In der Einleitung zum Kapitel wurde angedeutet, dass die Sicherheitsarchitektur aus mehreren Services besteht. Jeder Service realisiert eine Aufgabe, sodass zur Aufgabenerfüllung die Notwendigkeit besteht, dass Services Informationen austauschen. Um zu verhindern, dass der Informationsaustausch eine Schwachstelle der Sicherheitsarchitektur ist, muss sichergestellt sein, dass die Kommunikationspartner sich authentifizieren können. Was nur dann möglich ist, wenn jeder Service eine eindeutige Identität erhält, die von den Kommunikationspartner verifiziert werden kann. Für diesen Zweck erhält jeder Service der Sicherheitsarchitektur ein **digitales Zertifikat**. Mit dem zugewiesenen Zertifikat bestätigt ein Service seine Mitgliedschaft in der Sicherheitsarchitektur und kann sich für die Interaktion mit anderen Services ausweisen. Die

Vergabe von Zertifikaten erfolgt nach einer **Public-Key Infrastruktur**, die eine hierarchische Vergabe von Zertifikaten ermöglicht. Die generelle Funktionsweise wurde bereits in Unterabschnitt 2.2.1 erläutert und wird an dieser Stelle nicht weiter ausgeführt.

Als **Zertifizierungsinstanz**, die das Signieren von Zertifikatsanträgen übernimmt, und **Validierungsdienst**, der die ausgestellten Zertifikate enthält, dient der zentrale Service **Certificate Autority Point (CAP)**. Der Service wird in Unterabschnitt 7.8.5 noch im Detail vorgestellt. Vorliegende Ausprägung der Authentifizierung erlaubt mit der Vergabe von Zertifikaten nur zu definieren, ob ein Klient eine Verbindung mit Services der Sicherheitsarchitektur aufbauen darf. Die Entscheidung, welche Informationen ein Klient von einem Service beziehen darf, entscheidet jeder Service selbst. Zur Entscheidung kann ein Service alle oder nur eine Teilmenge der ausgestellten Zertifikate des CAP akzeptieren. Zur Vereinfachung definiert die Diplomarbeit, dass alle Services die über ein digitales Zertifikat des CAP verfügen, unbeschränkt Informationen austauschen. Auf die Diplomarbeit folgende Erweiterungen der Sicherheitsarchitektur können aber auch Zertifikatsketten bilden, die je nach Wurzelzertifikat unterschiedlichen Informationsaustausch gestattet. Der interessierte Leser findet einen Einstieg zu dieser Thematik in [16].

Ein weiterer Fall, der zu betrachten ist, dass Subjekte Kontakt mit Services der Sicherheitsarchitektur aufnehmen. Das ist zum Beispiel dann der Fall, wenn ein Subjekt eine neue Zugriffsrichtlinie erstellen will. In diesem Fall erlauben die Zertifikate dem Subjekt zu verifizieren, dass es sich mit einem authentischen Service der Sicherheitsarchitektur verbindet. Das stellt sicher, dass Subjekte sensible Informationen nur an vertrauenswürdige Services der Sicherheitsarchitektur übermitteln. Welchen Subjekten Zugriffe auf den Service erlaubt sind, legt jeder Service individuell fest und wird jeweils für jeden Service später erörtert.

Grundsätzlich gilt für die Sicherheitsarchitektur, die Datenkanäle zur Kommunikation zwischen Services und von Subjekten zu Services sind verschlüsselt. Das erschwert ein Abhören von sensiblen Informationen, wenn Verbindungen mit Services aufgebaut werden. Datenströme in der kontrollierten Anfrageverarbeitung, werden ebenso verschlüsselt, was noch im Verlauf des Kapitels erörtert wird.

7.2 Definition und Auswertung von Zugriffsrichtlinien

Zugriffsrichtlinien definieren, unter welchen Bedingungen, ein Zugriff auf von der Sicherheitsarchitektur geschützte Operatoren gestattet wird. Ein Zugriff in der Sicherheitsarchitektur ist zum Beispiel die Ausführung eines Input-Manager oder die Verarbeitung von Datenelementen, die von einer Quelle während der Ausführung in einer Anfrage generiert werden. Operatoren werden geschützt, indem sie nicht in dem ursprünglichen NexusDS verfügbar gemacht werden, sondern nur in der Sicherheitsarchitektur von NexusDS. Ist das der Fall, sind auch alle Datenelemente, die von einem geschützten Operator in einer Anfrage zur Ausführungszeit generiert werden, geschützt. Das **Secure Operator Repository (SOR)** bildet den zentralen Ort der Sicherheitsarchitektur, die alle zu schützenden Operatoren beherbergt. Es ist nur noch dann die Ausführung und die Verarbeitung der von dem Operator erzeugten Datenelemente möglich, wenn an den Operator gebundene Zugriffsrichtlinien den Zugriff explizit gestatten.

Der vorliegende Abschnitt erläutert die Verwendung und den Aufbau von Zugriffsrichtlinien. Abschnitt 7.2.1 führt ein, wie Zugriffsrichtlinien erstellt und verteilt werden. Das Abbilden von Bedingungen, die den Zugriff auf Operatoren definieren, führt Abschnitt 7.2.2 ein. Die detaillierte Struktur von Zugriffsrichtlinien folgt in Abschnitt 7.2.3 gefolgt von den Abschnitten 7.2.4 und 7.2.5, die auf Details zu Evaluatoren und Filter eingehen.

7.2.1 Administration und Verteilung von Zugriffsrichtlinien

Administriert werden Zugriffsrichtlinien an einem zentralen Service der Sicherheitsarchitektur, der als **Policy Administration Point (PAP)** bezeichnet wird. Über den Service können neue Zugriffsrichtlinien eingebracht und bestehende Zugriffsrichtlinien verändert oder gelöscht werden. Der Service ist die zentrale Schnittstelle für Subjekte um die Zugriffsrichtlinien für die Sicherheitsarchitektur zu bearbeiten. Der Service sorgt für eine konsistente Datenhaltung und für die Verteilung von Zugriffsrichtlinien an alle Punkte der Sicherheitsarchitektur, die von einer Änderung betroffen sind. Wird eine Anfrage geplant, werden zur kontrollierten Anfrageplanung alle relevanten Zugriffsrichtlinien aus dem PAP bezogen. Sind von Änderungen an Zugriffsrichtlinien auch in der Ausführung befindliche Operatoren betroffen, dann müssen die neuen oder veränderten Zugriffsrichtlinien aktiv den Operatoren zugestellt werden. Für die Weiterleitung arbeitet der PAP mit dem **Secure Core Query Service (SCQS)** zusammen. Der genaue Ablauf wird im weiteren Verlauf der Diplomarbeit erörtert.

7.2.2 Abbilden von Zugriffsbedingungen

Ein Zugriff geht immer von einem Subjekt auf ein Objekt. Zur Erinnerung, Objekte sind passive Daten, zum Beispiel auf einer Festplatte abgelegte Dateien. Subjekte sind Benutzer oder Operatoren, die im Auftrag eines Benutzers handeln. Zum Beispiel ist die Implementierung einer Quelle ein Objekt. Befindet sich die Quelle in der Ausführung, dann ist sie ein Subjekt, da es sich nicht mehr um ein passives Stück Information handelt. Deshalb besteht die Ausführung einer Anfrage aus den Zugriffen ''`execute`'' und ''`read`''. Im ersten Schritt müssen die Implementierungen der Operatoren zur Ausführung gebracht werden, das beschreibt der Zugriffstyp ''`execute`''. Im zweiten Schritt sollen die Daten, die von Operatoren erzeugt werden, von weiteren Operatoren verarbeitet werden. Das erfordert die Prüfung auf den Zugriffstyp ''`read`'', der definiert, ob ein Operator die von einem anderen Operator erzeugten Datenelemente lesen darf oder nicht. Eine explizite Betrachtung eines Zugriffstyps ''`write`'' findet im Kontext von der Verarbeitung von Anfragen nicht statt. Per Definition wird festgelegt, dass in Operatoren eingehende Datenelemente gelesen werden und es sich bei den ausgehenden Datenelementen um neue Datenelemente handelt. Das heißt, es werden Datenelemente aus Sicht der Sicherheitsarchitektur an Operatoren grundsätzlich neu erstellt und nicht überschrieben. Jedoch hängen die neu erzeugten Datenelemente von den Datenströmen ab, aus denen sie erzeugt wurden. Dies wird noch im weiteren Verlauf genauer erörtert.

Der Zugriffstyp ''`execute`'' ist abhängig von dem Subjekt, dass eine Anfrage zur Ausführung bringen will. Die Operatoren der Anfrage sollen nur dann ausgeführt werden, wenn dem

Subjekt die Ausführung gestattet ist. Das ermöglicht die Einschränkung der Nutzung von Operatoren auf ausgewählte Benutzergruppen. Eine weitere Einschränkung ist eine Restriktion, die definiert auf welchen Rechenknoten Operatoren von einem Subjekt ausgeführt werden dürfen. Die Ausführung auf bestimmte Rechenknoten zu beschränken ergibt sich aus der Nutzung der Rechenknotenressourcen durch Operatoren. Daraus ergibt sich die Möglichkeit einzuschränken, dass nur ausgewählte Subjekte, die Ressourcen von Rechenknoten mit den Fähigkeiten von Operatoren nutzen können, um Ergebnisse in einer Anfrage zu erzeugen. Zum Beispiel ein Operator, der den GPS-Sensor eines Mobiltelefons ausliest. Dieser nutzt als Ressource den GPS-Sensor des Mobiltelefons, wenn er auf einem Rechenknoten des Mobiltelefones ausgeführt wird. Würde nur beschränkt, ob ein Subjekt einen Operator ausführen darf oder nicht, könnte das Subjekt den Operator auf jedem Mobiltelefon zur Ausführung bringen und Positionsdaten auslesen. Dadurch würde der Besitz an der Positionsinformation nicht ausreichend berücksichtigt, um den Zugriff auf die Positionsinformation zu kontrollieren. Die fehlende Bindung wird hergestellt, indem die Ausführung des Operators auch von dem auszuführenden Rechenknoten abhängt. Das heißt, sowohl der Besitzer des Operators, zum Beispiel der Entwickler, als auch der Besitzer des Rechenknotens, im Beispiel der Besitzer des Mobiltelefons, müssen der Verarbeitung ihrer Ressourcen mit einer Zugriffsrichtlinie zustimmen.

Nachdem die generierten Datenelemente an den erzeugenden Operator und dessen ausführenden Rechenknoten gebunden sind, muss die Weiterverarbeitung der generierten Datenelemente reglementiert werden. Dies geschieht über die Auswertung des Zugriffstyps ''`read`'', der festlegt von welchen Operatoren, auf welchen Rechenknoten Datenelemente verarbeiten werden dürfen. Der Bezug auf die zu verarbeitenden Datenelemente wird hergestellt, indem Bezug auf den Operator und Rechenknoten genommen wird, der der Ursprung der Datenelemente ist. Will nun ein Operator A die Datenelemente eines vorgelagerten Operators B lesen, muss für den Operator A eine Menge von Zugriffsrichtlinien vorliegen, die den Zugriff auf den vorgelagerten Operator B erlauben. Sollte der vorgelagerte Operator A wiederum Operatoren vorgelagert haben, dann muss A auch für diese Operatoren ein Leserecht besitzen. Denn jeder Operator, der an der Verarbeitung eines Datenelementes teilnimmt, erlangt einen Besitz an dem Datenelement, somit muss Operator A ein Leserecht an allen vorgelagerten Operatoren besitzen.

Die Auswertung der genannten Zugriffstypen wird in der Sicherheitsarchitektur von dem **Policy Decision Point (PDP)** durchgeführt. Der Service wird später noch im Detail erörtert.

7.2.3 Definition von Zugriffsrichtlinien im Meta-Daten-Modell

Die Ausgestaltung der Zugriffsrichtlinien wird mit einem Meta-Daten-Modell auf Basis der AWML, vorgestellt in Abschnitt 3.2, definiert. Die AWML bietet als XML Dialekt die Möglichkeit beliebige Daten strukturiert abzubilden und integriert sich nahtlos in die vorhandene NexusDS Architektur. Des Weiteren lassen sich über Attribute, Klassen und Schemas flexibel Datenstrukturen definieren und erweitern. Das erleichtert einen zukünftigen Ausbau des Meta-Daten-Modells zur Abbildung der Zugriffsrichtlinien, wenn das Sicherheitskonzept neuen Anforderungen angepasst werden soll.

Attribut	K	Beschreibung
policyID	1	Eindeutige Identität der ZR
timestamp	1	Definiert ab wann die ZR gültig ist
role	1..n	Rollen für die die ZR gilt
roleInherit	1	Ob die ZR auf untergeordnete Rollen vererbt werden soll
operator	1..n	Menge Operatoren für die ZR gilt
access	1	Zugriffstyp aus der Menge `{read,execute}`
node	1..n	Operatoren dürfen auf diesen Rechenknoten ausgeführt werden
signatureK	0..1	Öffentlicher Schlüssel der geforderten Signaturen
immediate	1..1	Boolescher Wert, der die sofortige Anwendung der ZR definiert
use	0..1	Definiert mit ''+'' oder ''-'' Anwendung der ZR
policyS	0..1	Digitale Signatur der ZR

Attribut	K	Beschreibung
slotID	1	Gilt für Datenausgang mit dieser SlotID
with	1..n	Zugriff auf Datenelemente ist diesen Operatoren gestattet
on	1..n	Zugriff auf die Objekte darf auf diesen Rechenknoten erfolgen
unprotected	1	Boolescher Wert um Datenstrom ungeschützt freizugeben

Tabelle 7.1: Basisattribute einer Zugriffsrichtlinie (ZR), Spalte K steht für die Kardinalität der Attribute.

Tabelle 7.1 zeigt die Grundmenge an Attribute, die für eine Zugriffsrichtlinie definiert werden können. Die Kardinalität (K) gibt an, ob ein Attribut option, verpflichtend oder beliebig häufig vorkommen kann. Das Attribut `policyID` definiert einen eindeutigen Identifikator für die Zugriffsrichtlinie und ist verpflichtend. Der Identifikator der Zugriffsrichtlinie entspricht der Identität der Zugriffsrichtlinie. Der Zeitpunkt, zu dem die Zugriffsrichtlinie definiert wird und damit gültig ist, ist als Zeitstempel im Attribut `timestamp` festgehalten, was ebenfalls eine verpflichtende Angabe ist. Für welche Subjekte die Zugriffsrichtlinie gilt, definiert das Attribut `roles`. Die Kardinalität fordert mindestens ein Eintrag, es können jedoch endlich viele Einträge zugewiesen werden, sodass die Zugriffsrichtlinie für alle Eintragungen gilt. Im weiteren Verlauf wird noch erörtert werden, dass Zugriffsrichtlinien vererbt werden können. Um eine Vererbung für die Zugriffsrichtlinie auszuschließen, wird das Attribut `roleInherit` mit `false` belegt, sonst mit `true`. Für welche Operatoren die Zugriffsrichtlinie gilt wird über das Attribut `operator` festgelegt. Auch hier sind Mehrfacheinträge erlaubt, es muss jedoch mindestens ein Operator definiert werden. Als Angabe sind sowohl eindeutige Identitäten einzelner Operatoren möglich, als auch die Angabe von Domänen, die Mengen von Operatoren umfassen. Der mit der Zugriffsrichtlinie adressierte Zugriff definiert das verpflichtende Attribut `access`, das mit `{''read'',''execute''}` belegt werden kann. Die Bedeutung der einzelnen Zugriffskonstanten wurde bereits erläutert. Soll die Zugriffsrichtlinie sofort an allen Stellen durchgesetzt werden, kann das boolesche Attribut `immediate` mit `true` belegt werden. Dann wirkt sich die Zugriffsrichtlinie

Attribut	K	Beschreibung
evaluatorURI	1	Identität des Evaluators
evaluatorV	1	Nummerische Angabe zur Version des Evaluators
evaluatorS	1	Signatur des Evaluators zur Prüfung der Authentizität
rule	0..1	Individuelle Meta-Daten für den Evaluator

Attribut	K	Beschreibung
filterURI	1	Falls ein Filter angewendet werden soll, die Identität des Evaluators
filterV	1	Numerische Angabe zur Version des Filters
filterS	1	Signatur des Filters zur Prüfung der Authentizität
slotID	1..n	`slotID`s der Datenausgänge, auf die der Filter anzuwenden ist
rule	0..1	Individuelle Meta-Daten für den Filter
order	1	Relative Reihenfolge falls mehr als ein Filter pro `slotID` definiert ist

Tabelle 7.2: Optionale Attributmenge zur Definition eines Evaluators (oben) und eines Filters (unten), Spalte K steht für die Kardinalität der Attribute.

auch auf die Datenelemente aus, die sich bereits in der Verarbeitung befinden und sonst noch zu den vorherigen Bedingungen ausgewertet werden würden.

Im vorherigen Abschnitt wurde erörtert, dass Rechenknoten zur Ausführung von Operatoren explizit zu definieren sind. Hierzu wird das Attribut `node` mit einer Menge von Rechenknoten-Identitäten belegt, auf denen die Ausführung des Operators gestattet ist. Es muss mindestens ein Rechenknoten benannt werden, sonst wäre der Operator nicht ausführbar. Anstatt einzelner Identitäten können auch Domänen angegeben werden, die Mengen von Rechenknoten enthalten.

Ist als Zugriffstyp ''`read`'' definiert, bezieht sich die Zugriffsrichtlinie auf die von dem Operator erzeugten Datenelemente. Der untere Teil der Tabelle 7.1 definiert die Menge von Attributen, die je Datenausgang der refrenzierten Operatoren vergeben werden können. Eine `slotID` ist aus dem ursprünglichen Operator-Modell von NexusDS und indexiert eindeutig Dateneingänge und Datenausgänge. Das Attribut `slotID` in der Zugriffsrichtlinie definiert den Index des Datenausgangs, auf die sich die Attribute `with` und `on` beziehen. Sie erlauben die Einschränkungen anzugeben, von welchen Operatoren, Attribut `with`, auf welchen Rechenknoten, Attribut `on`, die ausgehenden Datenelemente verarbeitet werden dürfen. Das boolesche Attribut `unprotected` legt fest, ob der Datenausgang in von Anfragen außerhalb der kontrollierten Ausführungsumgebung lesbar ist. Wird dieses mit wahr belegt, ist der Datenausgang unverschlüsselt und es können Operatoren des ursprünglichen NexusDS an den Datenausgang angebunden werden. Für jeden Datenausgang muss eine Definition vorhanden sein, wenn die Datenelemente für andere Operatoren lesbar sein sollen.

Soll bestimmt werden, dass die ausgehenden Datenelemente der in Attribut `operator` definierten Operatoren nur von zertifizierten Objekten verarbeitet werden, kann mit dem Attribut `signatureK`

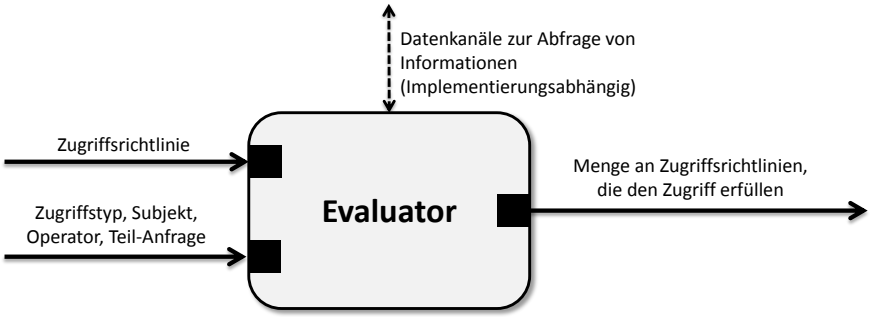

Abbildung 7.1: Schema eines Evaluators mit Eingabe und Ausgabe.

der öffentlicher Schlüssel des gewünschten Prüfers definiert werden. Ein Eintrag für dieses Attribut impliziert die Ausführung in Sicherheitszone-Drei.

Tritt der Fall ein, dass eine Zugriffsrichtlinie zur Ausführungszeit einer Anfrage zurückgenommen wird. Müssen alle sich in der Ausführung befindlichen Operatoren, die von der Zugriffsrichtlinie referenziert sind, über die Änderung benachrichtigt werden. Hierzu setzt die Sicherheitsarchitektur unter Anderem auf die Interpunktion von Zugriffsrichtlinien in Datenströme. Für die Operatoren, die die Interpunktionen empfangen, muss erkennbar sein, ob die interpunktierte Zugriffsrichtlinie zurückgenommen oder angewendet werden soll. Für diesen Zweck existiert ein Attribut `use`, dass mit einem Zeichen ''+'' oder ''-'' definiert, ob die interpunktierte Zugriffsrichtlinie angewendet oder zurückgenommen werden soll. Verwendet wird das Attribut nur dann, wenn die Zugriffsrichtlinie in einen Datenstrom eingeflochten wird. Das Konzept wurde bereits, wie auch die Interpunktion mit Zugriffsrichtlinien in den Datenstrom von *FENCE* in 4.4 vorgestellt. Ebenfalls nur für die Interpunktion genutztes Attribut ist `policyS`, indem eine digitale Signatur der Zugriffsrichtlinie eingetragen wird. Diese wird vor der ersten Interpunktion erzeugt und berechnet sich aus dem eindeutigen Hashwert der Zugriffsrichtlinie, auf den der geheime Schlüssel der Anfrage angewendet wurde. Der Signaturwert wird von jeder Einbettung vor Verwendung der Zugriffsrichtlinie überprüft, um sicherzustellen, dass ein eingebetteter Operator keine selbst erzeugten Zugriffsrichtlinien in die Datenströme einschleust. Ein Kopieren der Signatur ist für die Operatoren nicht möglich, denn sie Verfügen nicht über den geheimen Schlüssel der Anfrage.

7.2.4 Optionale Auswertungen von Zugriffsrichtlinien mit Evaluatoren

Es wurde bereits erwähnt, dass Zugriffsrichtlinien entschieden werden müssen, was einen Mechanismus zu deren Auswertung notwendig macht. Zur Auswertung betrachtet der **Policy Decision Point (PDP)** die zu einem Operator definierten Zugriffsrichtlinien und vergleicht den geforderten Zugriff mit dem von den Zugriffsrichtlinien erlaubten Zugriff. In Abschnitt 5.1.3 wurde das Anwendungsszenario der intelligenten Fabriken vorgestellt, in dem komplexe Zugriffe kontrolliert werden müssen. Um zu vermeiden, dass bereits bestehende Systeme zur Entscheidung von Zugriffen in die Zugriffsrichtlinienarchitektur von NexusDS vollständig übertragen werden

müsse, erlaubt die Sicherheitsarchitektur die Definition von **Evaluatoren**. Evaluatoren sind ausführbare Objekte, die einen beliebigen Algorithmus zur Entscheidungsfindung implementieren. Um den PDP anzuweisen, zur Entscheidung eines Zugriffes einen Evaluatoren zu verwenden, wird eine Zugriffsrichtlinie zusätzlich mit den in Tabelle 7.2 gezeigten Attribute versehen. Die Angabe muss die eindeutige Identität über Attribut `evaluatorURI`, eine Versionsangabe über Attribut `evaluatorV` und die digitale Signatur über Attribut `evaluatorS` des Evaluator definieren. Unterschiedliche Versionen erleichtern den Entwicklern von Evaluatoren einen gezielteren Einsatz von unterschiedlichen Versionen, was zum Beispiel die Einführung neuer Evaluatoren erleichtert. Die Signatur stellt sicher, dass der Evaluator genau die Implementierung aufweist, die die Zugriffsrichtlinie fordert. Sodass die sicherheitskritische Auswertung genau von der Evaluator-Implementierung vorgenommen wird, der gefordert ist. Berechnet wird die Signatur vom zentralen **Secure Operator Repository (SOR)**, dass Evaluatoren vorhält. Attribut `rule` ist ein frei definierbares Feld für Meta-Daten, das die Implementierung des Evaluators selbst auswertet. Das erlaubt die flexiblere Nutzung von einer Implementierung für verschiedene Aufgaben, frei nach der Realisierung des Entwicklers.

Für jede Zugriffsrichtlinie kann nur ein Evaluator definiert werden. Weiter ist zu beachten, dass Zugriffsrichtlinien mit Evaluatoren immer Zugriffsrichtlinien ohne Evaluatoren nachgeordnet werden. Das heißt, wenn eine Entscheidung getroffen werden soll, ob ein Zugriff gestattet ist oder nicht und es werden mehrere Zugriffsrichtlinien mit und ohne Evaluator für den Zugriff gefunden. Dann werden zuerst die Zugriffsrichtlinien ohne Evaluator ausgewertet, erfüllt eine davon den Zugriff, werden die Zugriffsrichtlinien mit Evaluator nicht mehr beachtet. Nur falls keine Zugriffsrichtlinie ohne Evaluator den Zugriff erfüllt, werden Zugriffsrichtlinien mit Evaluator ausgewertet. Dadurch wird eine gegebenenfalls sehr aufwendige Prüfung durch Evaluatoren vermieden, wenn gewöhnliche Zugriffsrichtlinien den Zugriff erfüllen. Die genaue Rangfolge der Auswertung, sofern mehrere Zugriffsrichtlinien mit Evaluator vorhanden sind, wird später noch erläutert.

Als Eingabe erhält ein Evaluator die Identität der Zugriffsrichtlinie, die ihn definiert, die Art des gewünschten Zugriffes aus den Zugriffskonstanten `{read,execute}`, die Identität des Subjektes das den Zugriff wünscht und den Operator, auf den der Zugriff erfolgen soll. Soll der Zugriffstyp ''`read`'' ausgewertet werden, erhält der Evaluator noch einen Anfrageteilgraph, der dem zu prüfenden Operator in der Anfrage vorgelagert ist. Die Details dazu werden im Abschnitt 7.4, zur kontrollierten Anfrageplanung, erläutert. Als Ergebnis muss der Evaluator den Zugriff gestattet oder ablehnen. Sollte der Zugriff erlaubt werden, liefert der Evaluator eine nicht leere Menge von Zugriffsrichtlinien zurück, die die Anfrage erfüllen. Mit den zurückgelieferten Zugriffsrichtlinien kann die Sicherheitsarchitektur weitere Auswertungen vornehmen und hat die Bestätigung das der Zugriff erlaubt ist.

Abbildung 7.1 zeigt das Schema eines Evaluators mit den genannten Eingaben. Mit der gelieferten Information kann die Sicherheitsarchitektur einen Abgleich durchführen, ob der Zugriff gestattet ist oder nicht. Wie die Implementierung zu einem Ergebnis kommt, ist dem Entwickler des Evaluators überlassen. Denkbar ist zum Beispiel der Zugriff auf anwendungsinterne Datenbanken, die NexusDS nicht bekannt sind. Im Anwendungsszenario von *Squebber*, siehe Abschnitt 5.1.2, könnten das zum Beispiel Freundeslisten sein, die definieren, dass Subjekte befreundet

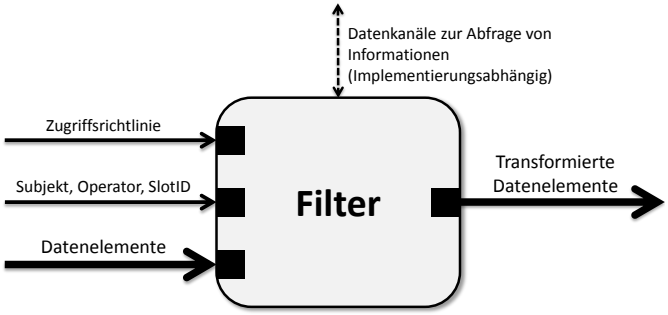

Abbildung 7.2: Schemas eines Filters mit Eingängen und Ausgängen.

sind und so gegenseitig auf Daten zugreifen dürfen. Die Implementierung von Evaluatoren erleichtert bestehenden Anwendungen, die Sicherheitsarchitektur effizient zu nutzen. Will die Implementierung auf die Services der Sicherheitsarchitektur zur Entscheidungsfindung zugreifen, benötigt die Implementierung eine Zertifizierung des zentralen Service **Certificate Autority Point (CAP)**. Nur dann kann die Implementierung eine Verbindung herstellen. Wo die Evaluatoren zur Ausführung gebracht werden, wird später erläutert.

7.2.5 Transformation von Datenströmen mit Filter

Um den Zugriff auf Datenelemente in Anfragen zu verfeinern und nicht den Zugriff nur vollständig zuzulassen oder vollständig zu blockieren, können Operatoren mit **Filter** versehen werden. Dazu werden analog zum Evaluator Filter in Zugriffsrichtlinien definiert. Tabelle 7.2 zeigt die dazu notwendigen Attribute. Zusätzliches Attribut ist die `slotID`, die definiert, an welchen Datenausgang, im Falle einer Quelle und an welchen Dateneingang, im Fall einer Operation oder Senke, der Filter zu verwenden ist. Die Attribute bilden eine Gruppe und können mehrfach in einer Zugriffsrichtlinie definiert werden um je `slotID` mehrere Filter anzubringen. Für Warteschlangen und Input-Manager können ebenfalls Zugriffsrichtlinien mit Filter definiert werden. Ist das der Fall, werden diese mit den Filtern der Operation, der die Warteschlangen und der Input-Manager vorgelagert ist, vereinigt. Ein Zwischenschalten in die Kette von Warteschlange, Input-Manager und Operation wäre aus Komplexitätsgründen nicht sinnvoll. Die Reihenfolge der Ausführung von Filter wird später noch erläutert.

Sollten für eine `slotID` mehrere Filter definiert sein, wird das Attribut `order` ausgewertet. Es ist mit einer natürlichen Zahl zu belegen, nach deren aufsteigender Reihenfolge die Filter angeordnet werden. Der Filter mit der kleinsten Zahl führt die erste Transformation durch und der Filter mit der höchsten Zahl die letzte Transformation. Befinden mehrere Filter auf der gleichen Ordnungsstufe, aus verschiedenen Zugriffsrichtlinien, wird die Ordnung nach Alter der Zugriffsrichtlinie ausgewertet, von der ältesten zur jüngsten Zugriffsrichtlinie. Die Subjekte, die Zugriffsrichtlinien mit Filter spezifizieren, müssen darauf achten, dass die Transformationen von hintereinander ausgeführten Filtern kompatibel sind. Lediglich das Datenformat ist festgelegt und entspricht genau dem, das durch den zu filternden Dateneingang beziehungsweise Datenausgang des Operators gegeben ist.

Wird eine Menge von Operatoren im Attribut `operator` definiert, dann bezieht sich der definierte Filter für jeden Operator auf dieselbe `slotID`. Muss eine Unterscheidung festgelegt werden, weil die Operatoren zum Beispiel unterschiedliche Datentypen an der gleichen `slotID` verarbeiten, müssen getrennte Zugriffsrichtlinien definiert werden. Ob ein Filter mit dem jeweiligen Datenausgang beziehungsweise Dateneingang kompatibel ist, legen mit Implementierung des Filters assoziierte Meta-Daten fest. Abbildung 7.2 illustriert das Schema eines Filters. Es handelt sich analog zum Evaluator um ein ausführbares Objekt. Welche Transformation der Filter implementiert, ist dem jeweiligen Entwickler überlassen. Soll der Filter mit Services der Sicherheitsarchitektur interagieren, ist wie für den Evaluator eine Zertifizierung des zentralen **Certificate Autority Point (CAP)** Service notwendig. Die Vorhaltung der Filter ist analog zum Evaluator im zentralen **Secure Operator Repository (SOR)**.

Zu Ausführungsbeginn eines Filters erhält der Filter die Identität der Zugriffsrichtlinie, die den Filter definiert. Anhand der eindeutigen Identität kann die Implementierung des Filters sich, falls nötig, die Zugriffsrichtlinie zur Auswertung beschaffen. Zudem die Identität des Subjektes, dass die Anfrage ausführt, in der der Filter eingesetzt werden soll und die Identität des Operators und die `slotID`, der der Filter vor- oder nachgelagert ist. Mit diesen zusätzlichen Informationen kann die Implementierung des Filters individuell auf die Einsatzumgebung reagieren. Daneben besitzen Filter genau einen Dateneingang, an den der zu transformierende Datenstrom angebunden wird, und genau einen Datenausgang, der den transformierten Datenstrom ausgibt. Über diesen Dateneingang und Datenausgang wird der Filter in den betroffenen Datenstrom zwischengeschaltet. Aus Gründen der Effizient werden die Filter immer auf demselben Rechenknoten ausgeführt wie der referenzierte Operator.

7.3 Abhängigkeit von Datenströmen und Wiedereinflechtung von Zugriffsrichtlinien

In den vorherigen Abschnitten wurde bereits angedeutet, dass es notwendig ist, den Fluss von Datenelementen zu verfolgen. Das ist dann der Fall, wenn für den Zugriffstyp ''`read`'' ermittelt werden muss, von welchen Operatoren ein Datenstrom erzeugt wurde. Zur Ermittlung des Pfades in der Anfrage, der die Entstehung und Bearbeitung des Datenstromes beschreibt, ist es notwendig zu wissen, welche Dateneingänge für welche Datenausgänge relevant sind. Abbildung 7.3 zeigt die drei möglichen Fälle, die auftreten können, wenn Datenströme von einer Operator-Box verarbeitet werden. Abschnitt 7.3.1 erörtert die Möglichkeiten, eine zuverlässige Zuordnungen von Dateneingängen auf Datenausgänge vorzunehmen.

Neben der Aufgabe der Zuordnung muss für die Wiederinterpunktion von Zugriffsrichtlinien ein korrekter Zeitpunkt bestimmt werden. Wiederinterpunktion beschreibt die Aufgabe, Zugriffsrichtlinien, die über die Datenströme interpunktiert verteilt werden, nach dem Verarbeiten des Datenstromes durch eine Operator-Box wieder in Datenströme einzubringen. Das erfordert zum Ersten eine korrekte Zuordnung, aus welchen eingehenden die ausgehenden Datenströme erzeugt werden und zum Zweiten die Ermittlung des korrekten Zeitpunktes zur Wiederinterpunktion. Diese Fragestellung erörtert Abschnitt 7.3.2.

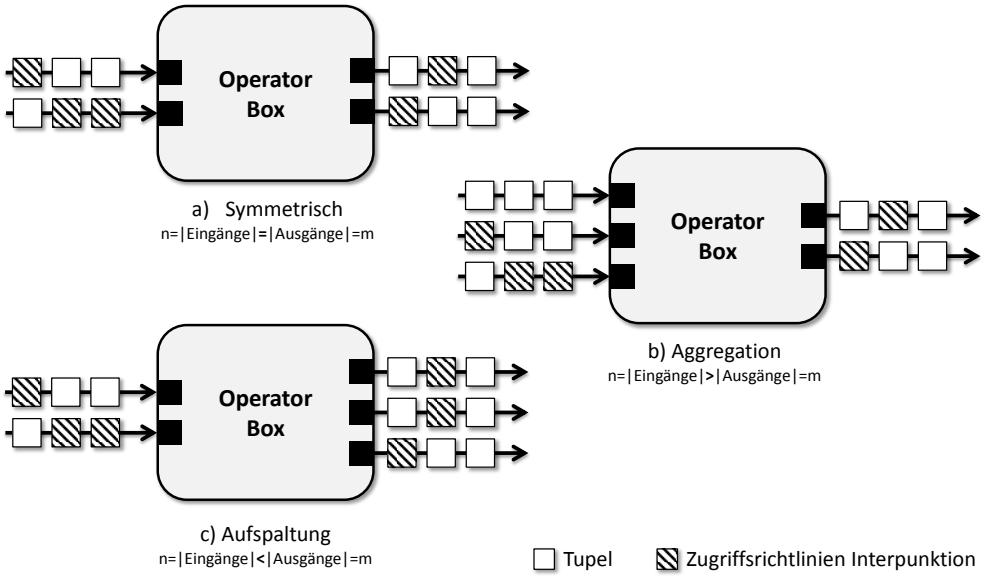

Abbildung 7.3: Mögliche Fälle zur Zuordnung von Dateneingänge auf Datenausgänge.

7.3.1 Zuordnung von Dateneingänge auf Datenausgänge

Die Zuordnung von eingehenden Datenströmen auf die ausgehenden Datenströme hängt von dem Ursprung der Datenströme und der durchgeführten Operation der Operator-Box ab. Der Ursprung der Datenströme spielt insoweit eine Rolle, dass ermittelt werden kann, welche Operatoren für die Information eines Datenstromes verantwortlich sind. Ist ein Datenstrom mit Zugriffsrichtlinien interpunktiert, müssen alle ausgehenden Datenströme, die mithilfe des geschützten Datenstromes berechnet wurden, ebenfalls mit den ursprünglichen Zugriffsrichtlinien versehen werden. Welche ausgehenden Datenströme welchen eingehenden Datenströmen zugeordnet werden müssen, ist abhängig von der Operation, die die Operator-Box durchführt.

Wie bereits in der Erläuterung zu den drei Sicherheitszonen in Abschnitt 6.2 beschrieben, kann die Sicherheitsarchitektur nicht ohne Weiteres eine zuverlässige Annahme treffen, wie die von der Box durchgeführte Operation gestaltet ist. Für alle Fälle die in Abbildung 7.3 gezeigt sind, bestehen zahlreiche Möglichkeiten der Abbildung von Dateneingängen auf die Datenausgänge. Beispielsweise könnte in Fall **a)** für $n > 1$, jeder zweite Dateneingang auf den letzten Datenausgang abgebildet werden oder alle Dateneingänge auf alle Datenausgänge. Deshalb muss entweder bei der Zuordnung eingehender Datenströme auf die ausgehenden Datenströme eine Heuristik zum Einsatz kommen, oder der Operator muss mit ausreichend Information versehen werden, sodass die Zuordnung zuverlässig durchgeführt werden kann.

Zum Versehen der Operation mit Informationen zur Zuordnung von eingehenden Datenströmen auf ausgehende Datenströme, wird das Meta-Daten-Modell von NexusDS zur Beschreibung der Operationen erweitert. Tabelle7.3 zeigt die zusätzlichen Attribute, die dem Operator Meta-Daten-Modell hinzugefügt werden. Die Erweiterung erlaubt für jeden Datenausgang zu definieren, von welchen Dateneingängen der generierte Datenstrom abhängt. Zwar wäre auch eine Klas-

Attribut	K	Beschreibung
outputSlotID	1	Index des zu beschreibenden Datenausgangs
inputSlotID	0..n	Indexmenge der abhängigen Dateneingänge
independent	1	Boolean ob Datenausgang unabhängig ist

Tabelle 7.3: Attribute zur Beschreibung der Abhängigkeit von Datenausgängen zu Dateneingängen, Spalte K steht für die Kardinalität der Attribute.

sifizierung von Operatoren möglich, sodass zum Beispiel ein Operator der Klasse *Aggregation*, wie in der Abbildung mit Fall **b)** dargestellt, dennoch wäre auch dann nicht genau klar, welche Dateneingänge in welche Datenausgänge aggregiert werden. Somit ist eine bloße Klassifizierung nicht ausreichend, um eine exakte Zuordnung zu definieren. Sollte eine Operation einen neuen Datenstrom erzeugen, der unabhängig von allen eingehenden Datenströmen ist, muss der Sachverhalt explizit vermerkt werden. Sollte die Beschreibung unvollständig sein, das heißt, ein oder mehrere Datenausgänge haben weder Verweise auf Dateneingänge noch einen expliziten Ausschluss, kommt eine Heuristik zum Einsatz. In *Sicherheitszone-Hoch* sind die Angaben zur Zuordnung von eingehenden Datenströmen auf ausgehende Datenströme und eine explizite Definition von zugriffsrichtlinienfreien Ausgängen verpflichtend. Die verantwortlichen Prüfer, die die Operatoren verifizieren, haben das Vorhandensein und die korrekte Zuordnung zu prüfen.

Sollten in der *Sicherheitszone-Mittel* keine oder nur unvollständige Angaben vorhanden sein, kommt eine Heuristik zum Einsatz. Sie geht grundsätzlich davon aus, dass ausgehende Datenströme, die keine eingehenden Datenströme referenzieren und nicht als unabhängige Datenströme markiert sind, von allen Dateneingängen abhängig sind. Das heißt, für die Wiederinterpunktion, dass in den Datenausgang die Zugriffsrichtlinien aller Dateneingänge wiederinterpunktiert werden. Würde die Heuristik umgekehrt arbeiten und alle keinerlei Zugriffsrichtlinien wiederinterpunktieren, dann würde dieser Ausgang aufgrund des generellen Zugriffsverbotes nicht mehr weiterverarbeitet werden können. Da das Sicherheitskonzept aber die Möglichkeit vorsehen will, dass bereits vorhandene Operatoren von NexusDS verwendet werden können, ist die Vorgehensweise nicht sinnvoll. Sonst müsste jeder Operator über die erweiterten Meta-Daten verfügen, dass nachfolgende Operatoren Datenelemente verarbeiten dürfen. Nicht erkennbar für die Heuristik ist, ob die Menge der Zuweisungen auf Datenausgänge vollständig ist. Beispielsweise könnte ein Datenausgang Z mit Zuweisungen auf Eingang A und B versehen sein, es müsste aber, dass die Zuweisung vollständig ist, auch Dateneingang C auf Z verweisen. Soll diese mögliche Lücke ausgeschlossen werden, muss eine Zugriffsrichtlinie signierte Operationen der *Sicherheitszone-Hoch* erzwingen.

7.3.2 Zeitpunkt der Einflechtung

Trifft eine Zugriffsrichtlinie zum Zeitpunkt t an einer Operator-Box ein, muss diese für alle Datenelemente, die ab Zeitpunkt t verarbeitet werden auch für nachgelagerte Boxen gelten. Die Wiederinterpunktion in die ausgehenden Datenströme der Box darf deswegen nicht zu früh und nicht zu spät stattfinden. Wird zu früh interpunktiert, rutscht die Zugriffsrichtlinie in die

Attribut	K	Beschreibung
outputSlotID	1	Index des zu beschreibenden Datenausgangs
outgoing	1	Zahl ausgehender Datenelemente, k
incoming	1..n	Menge aus {inputSlotID, n} für abhängige Dateneingänge

Tabelle 7.4: Attribute zur Beschreibung der eingehenden Datenelemente im Verhältnis der ausgehenden Datenelemente, Spalte K steht für die Kardinalität der Attribute.

Zukunft und gilt bereits für Datenelemente, für die die Zugriffsrichtlinie noch keine Gültigkeit haben sollten. Eine zu späte Interpunktion verletzt den Schutz der Datenelemente, da die Datenelemente vor der Interpunktion noch mit veralteten Zugriffsrichtlinien verarbeitet werden. Analog zur Betrachtung der Zuordnung von Dateneingängen auf Datenausgänge gilt, dass der Sicherheitsarchitektur nicht ohne Weiteres klar ist, wann eine Zugriffsrichtlinie interpunktiert werden muss.

Für die korrekte Einflechtung von Zugriffsrichtlinien zwischen die Datenelemente muss bekannt sein, aus wie vielen eingehenden Datenelementen, wie viele ausgehende Datenelemente entstehen. Ebenfalls muss klar sein, auf welche Dateneingänge und Datenausgänge sich die Angaben beziehen. Wenn zum Beispiel bekannt ist, dass aus drei eingehenden Datenelementen in Dateneingang A und zwei eingehende Datenelemente in Dateneingang B genau ein Datenelement an Datenausgang C entsteht, kann die Einflechtung exakt vorgenommen werden. Dann muss lediglich berücksichtigt werden, wie viele Datenelemente sich in den jeweiligen Warteschlangen aufhalten und ob momentan eine Datenverarbeitung stattfindet. Auf Basis der Informationen wird dann berechnet, ab welchem Datenpaket das die Operation verlässt, die Zugriffsrichtlinie in den Datenstrom interpunktiert werden muss. Zur Spezifikation wird wie bei der Verteilung der Zugriffsrichtlinien auf Datenausgänge das Meta-Daten-Modell des Operatoren-Modell erweitert. Tabelle 7.4 zeigt die Erweiterung. Zu jedem Datenausgang, der nicht unabhängig ist, muss definiert werden, wie viele Datenelemente von welchen Dateneingängen notwendig sind. Das Attribut `incoming` beschreibt für jeden eingehenden Dateneingang, identifiziert mit `inputSlotID`, die Anzahl eingehender Datenelemente n, sodass k Datenelement aus dem Operator in den spezifizierenden Datenausgang gegeben werden. Die Anzahl der ausgehenden Datenelemente k muss deshalb definiert werden, da ein Operator aus einer beliebigen Anzahl an eingehenden Datenelementen eine beliebige Anzahl an ausgehenden Datenelementen generieren kann. Die Zuverlässigkeit der Angaben hängt von den Zugriffsrichtlinien gewählten Sicherheitszonen ab. Nur in *Sicherheitszone-Hoch*, kann von einer vollständigen und korrekten Spezifikation ausgegangen werden.

Analog zur Verteilung der Zugriffsrichtlinien auf Datenausgänge muss geklärt werden, wie sich die Sicherheitsarchitektur verhält, falls keine vollständigen Angaben vorhanden sind. Für Datenausgänge, die nicht als zugriffsrichtlinienfrei definiert sind oder für die keine Angaben vorhanden sind, werden Zugriffsrichtlinien sofort interpunktiert. Denn es kann keine zuverlässige Annahme getroffen werden, wie viele Datenelemente von der Operation aus der Warteschlange entnommen werden und wie viele Datenelemente nach einer Entnahme generiert werden. Deshalb ist zur Gewährleistung der Zugriffskontrolle eine Verschiebung der Zugriffsrichtlinie in

die Zukunft der Möglichkeit vorzuziehen, dass Datenelemente mit veralteten Zugriffsrichtlinien verarbeitet werden. Der sichere Ausschluss einer Verschiebung der Zugriffsrichtlinie auf der Zeitachse kann nur mit dem Erzwingen der *Sicherheitsstufe-Hoch* gewährleistet werden.

7.4 Kontrollierte Planung von Anfragen

Die Planung von Anfragen muss unter Berücksichtigung der Zugriffsrichtlinien durchgeführt werden. Im Falle einer Planung ohne Beachtung kann nicht davon ausgegangen werden, dass eine Anfrage lauffähig ist und sofort abgebrochen werden muss. Dafür wird der ursprüngliche *Core Query Service* zum **Secure Core Query Services (SCQS)** erweitert. Dieser führt die Planung von Anfragen unter Berücksichtigung der aktuellen Zugriffsrichtlinien durch.

Die Ausführung geschützter Operatoren kann nur über den SCQS erfolgen. Dessen Komponenten, die im vorliegenden Abschnitt erläutert werden, erhalten Zugang zu dem zentralen SOR. Nur im SOR sind geschützte Operatoren verfügbar, sodass die ursprüngliche Anfrageplanung keinen Zugriff auf die Operatoren hat. Auf das ursprüngliche *Operator Repository (OR)* besteht vom SCQS ebenfalls Zugriff, da eine Anfrage sowohl aus geschützten Operatoren als auch aus ungeschützten Operatoren bestehen kann. In diesem Fall können ungeschützte Operatoren aber nur an Datenausgänge verbunden sein, die explizit über Zugriffsrichtlinien als ungeschützt deklariert wurden. Der Austausch von Information zwischen den Komponenten des SCQS ist durch digitale Zertifikate geschützt. Das heißt, andere Services, die nicht über entsprechende Zertifikate verfügen, können sich nicht in die Anfrageplanung einmischen.

7.4.1 Secure Query Interface (SQI)

Die geschützte Variante des *Query Interfaces* unterscheidet sich gegenüber der ungeschützten Variante dadurch, dass nicht jede Anfrage akzeptiert wird. Es können nur die Subjekte Anfragen absetzen, die über eine registrierte Identität verfügen, die als authentisch bestätigt werden kann

Im ersten Schritt wird vom Absender (Subjekt) der Anfrage eine gesicherte Verbindung zum **Secure Query Interface (SQI)** aufgebaut **(1)**. Nach Aufbau der Verbindung überprüft das SQI den Absender der Anfrage, der zum Verbindungsaufbau Identität und Passwort übertrug. Die Prüfung wird über den **Identity Administration Point (IAP)** durchgeführt. Bestätigt dieser die Identität des Subjekts, ist es authentifiziert. Zum Abschluss gibt das SQI die Anfrage mit der Identität des Subjektes, das die Anfrage ausführen will, an den **Secure Query Optimizer (SQO)**.

7.4.2 Secure Query Planer (SQP)

Der **Secure Query Planer (SQP)** überprüft Anfragen mit geschützten Operatoren nach den in der Sicherheitsarchitektur gegebenen Zugriffsrichtlinien. Die Überprüfung besteht aus dem Abgleich, ob das Subjekt, dass die Anfrage ausführen will, die geschützten Operatoren der Anfrage zur

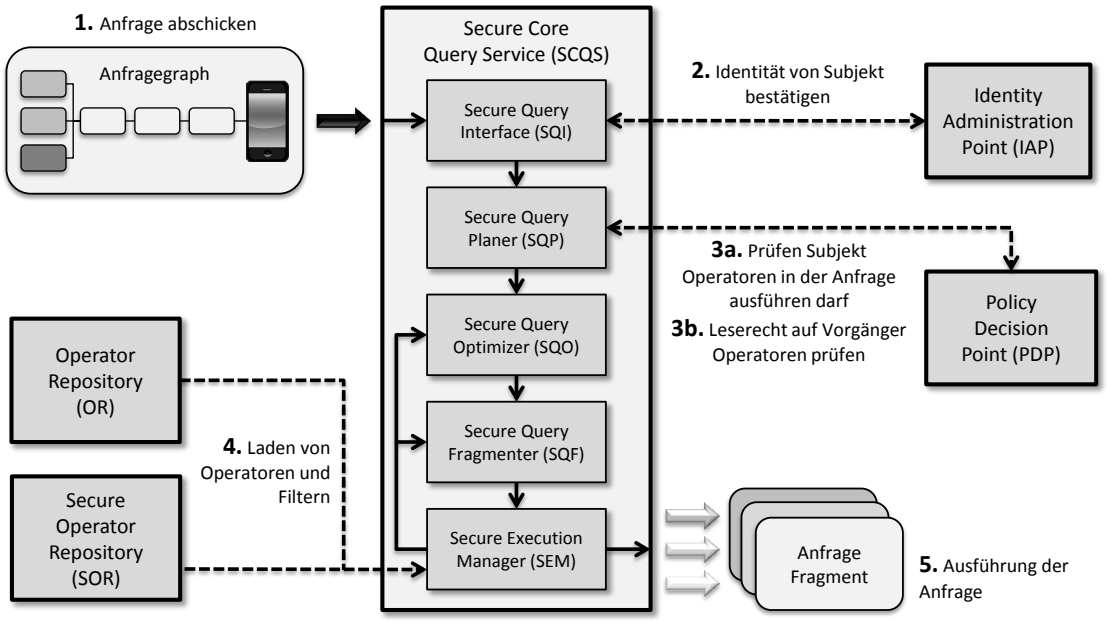

Abbildung 7.4: Veränderte Anfrageplanung in der Sicherheitsarchitektur, die Zugriffsrichtlinien für die Verteilung der Anfrage berücksichtigt.

Ausführung bringen darf. Des Weiteren wird bei der Überprüfung bereits eingeschränkt, auf welchen Rechenknoten die Ausführung gestattet ist. Darauf folgt die Überprüfung, ob jeder Operator die Datenelemente seiner vorgeschalteten Operatoren verarbeiten darf. Nur wenn für beide Teile eine erfolgreiche Überprüfung stattfand, leitet der SQP die Anfrage in den nächsten Schritt weiter. Die Anfrage selbst ist dann mit zusätzlichen Meta-Daten aufbereitet, sodass die nachfolgenden Schritte der Anfrageplanung, die Planung und Verteilung im Einklang der Zugriffsrichtlinien durchführen können.

Die Prüfung ob eine Anfrage ausgeführt werden darf findet in drei Schritten statt. Im **ersten Schritt** wird geprüft, ob das Subjekt, dass die Anfrage ausführen will jeden Operator der Anfrage ausführen darf. Das betrifft Zugriffsrichtlinien mit dem Zugriffstyp ''`execute`''. Im **zweiten Schritt** wird sichergestellt, das die hintereinander angeordneten Operatoren, die Datenelemente ihrer Vorgänger lesen dürfen. Das betrifft den Zugriffstyp ''`read`''. In jedem Schritt kann eine Menge von Rechenknoten definiert sein, für die die Ausführung gestattet ist. Sollte im **dritten Schritt** aus der Kombination der möglichen Rechenknoten, auf denen die Operatoren ausgeführt werden dürfen, keine Lösung gefunden werden, kann die Anfrage nicht ausgeführt werden und die Anfrageplanung wird Abgebrochen. Allgemein gilt, wird wird die Anfrage abgebrochen, erhält der Absender eine Fehlermeldung.

Schritt Eins, Prüfung der Ausführbarkeit aller Operatoren. Für jeden Operator wird über den zentralen **Policy Decision Point (PDP)** abgefragt, ob dem Subjekt die Ausführung des Operators gestattet ist. Dazu wird eine verschlüsselte Verbindung zum PDP hergestellt und für jeden Operator die folgende Anfrage stellt. Als Zugriff ''`execute`'', die Identität des Subjektes, dass die Anfrage ausführt und die Identität des zu prüfenden Operators. Ist die Ausführung gestattet,

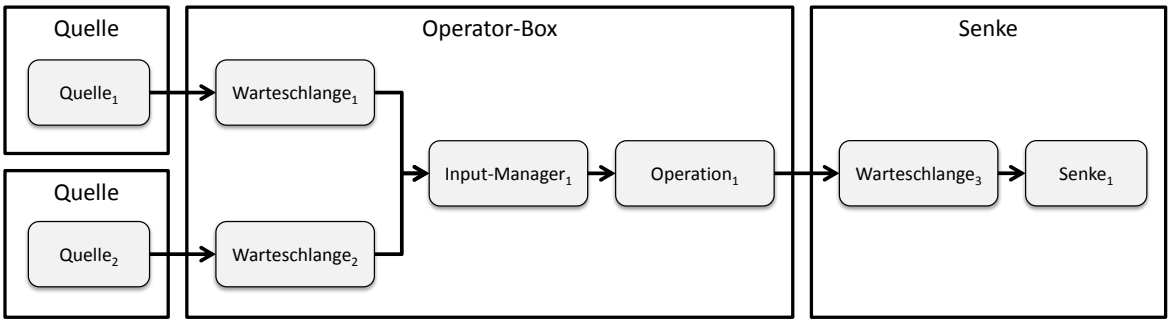

Abbildung 7.5: Vereinfachte Darstellung einer Anfrage als Graph, in der jeder Operator ein Knoten darstellt.

erhält der SQP die Menge an Zugriffsrichtlinien, die den Zugriff erfüllen **(3a)**. Die Menge wird dem Operator zugeordnet und mit dem nächsten Operator der Anfrage in gleicher Weise verfahren. Die Auswertungsreihenfolge der Operatoren spielt keine Rolle, sie können einzeln betrachtet werden, da für den Zugriffstyp ''`execute`'' keine Abhängigkeit zwischen den Operatoren besteht. Sollte für einen Operator kein Zugriff erlaubt sein, wird die Planung abgebrochen.

Schritt Zwei, Prüfung, ob der lesende Zugriff auf die dem Operator vorgelagerten Operatoren gestattet ist. Vorgelagert sind genau die Operatoren, die auf dem Pfad liegen, der in einen Dateneingang des zu überprüfenden Operators eingeht. Abbildung 7.5 illustriert die Betrachtung einer Anfrage als Graph. So ist in der Abbildung $\texttt{Senke}_1$ auf jeden Fall die Operator-Menge $\texttt{Warteschlange}_3$, $\texttt{Operation}_1$ und $\texttt{Input-Manager}_1$ vorgelagert. Die weiteren Vorgänger unterliegen einer Fallunterscheidung, je nachdem wie die Dateneingänge auf die Datenausgänge von $\texttt{Operation}_1$ verzweigen. Ist in der Abbildung der Datenausgang von $\texttt{Operation}_1$ sowohl von dem Dateneingang mit $\texttt{Quelle}_1$ und $\texttt{Quelle}_2$ abhängig, wäre die Menge der vorgelagerten Operatoren $\texttt{Warteschlange}_3$, $\texttt{Operation}_1$, $\texttt{Input-Manager}_1$, $\texttt{Warteschlange}_1$, $\texttt{Warteschlange}_2$, $\texttt{Quelle}_1$ und $\texttt{Quelle}_2$. Ist der Datenausgang nur von $\texttt{Quelle}_1$ abhängig, wäre es die Menge $\texttt{Warteschlange}_3$, $\texttt{Operation}_1$, $\texttt{Input-Manager}_1$, $\texttt{Warteschlange}_1$ und $\texttt{Quelle}_1$, der Fall mit nur $\texttt{Quelle}_2$ ist symmetrisch. Die Menge der Knoten (Operatoren), mit den jeweiligen Kanten (Verbindungen) sind ein Teilgraph des Anfragegraphs. Der Teilgraph enthält sowohl die Operatoren also auch die jeweils abhängigen Verbindungen von Datenausgängen zu Dateneingängen. Wie die Verzweigungen im Detail aufgelöst werden, entscheidet entweder eine Heuristik oder es bestehen Angaben in den Meta-Daten der Operation, die der verzweigende Knoten ist. Abschnitt 7.3.1 hat die Handhabung erläutert und soll hier nicht mehr erläutert werden.

Anfragen an den PDP werden nun wie folgt formuliert. Zugriffstyp ist ''`read`'', Subjekt das die Anfrage ausführt analog zum vorherigen Schritt, der zu prüfender Operator und der Teilgraph V der Vorgänger, der oben Beschreiben wurde. Zu jedem Knoten des Teilgraphs v_i, wird noch die Menge an Rechenknoten mitgeliefert, auf denen der Operator des Knotens ausgeführt werden darf. Die Menge der möglichen Rechenknoten kann mit den Zugriffsrichtlinien aus dem vorherigen Schritt mit Zugriffstyp ''`execute`'' berechnet werden, die jedem v_i zugeordnet wurden. Mit diesen wird die Vereinigung des Attributes `node` gebildet, welche dann genau alle die Rechenknoten enthält, auf denen Operator v_i ausgeführt werden darf **(3b)**. Ergebnis ist im

Erfolgsfall wieder eine Menge an Zugriffsrichtlinien für den überprüften Operator, die ebenfalls für den Operator vermerkt wird. Der Schritt wird für jeden Operator im Graph vorgenommen. Wird für einen Operator keine erfüllenden Zugriffsrichtlinien gefunden, wird die Planung abgebrochen.

Schritt Drei sucht für jeden Operator nach einer nicht leeren Menge an Rechenknoten, auf denen die Ausführung gestattet ist. Jeder Operator hat in beiden vorherigen Schritten eine Menge an Zugriffsrichtlinien zugeordnet bekommen, die wie folgt verarbeitet wird.

Die Menge der Zugriffsrichtlinien vom Zugriffstyp ''`read`'' wird über dem Attribut `on` geschnitten. Ist die Menge nicht leer und die übrig gebliebenen Zugriffsrichtlinien enthalten alle v_i, existiert mindestens ein Rechenknoten auf dem alle Vorgänger gemeinsam verarbeitet werden dürfen. Sonst wird die Verarbeitung abgebrochen. Danach wird die ermittelte Menge an Rechenknoten, auf denen die Verarbeitung aller Vorgänger erlaubt ist, mit dem Attribut `node` der Zugriffsrichtlinien vom Zugriffstyp ''`execute`'' geschnitten. Übrig bleibt eine Menge von Rechenknoten, auf denen das Subjekt den Operator ausführen darf und der Operator alle vorgelagerten Operatoren lesen darf.

Aus den erläuterten Teilschritten muss nun eine Kombination gefunden werden, die jedem Operator einen Rechenknoten zuordnet, sodass jeweils die Datenelemente der vorgelagerten Operatoren gelesen werden können. Ist das nicht möglich, wird die Anfrage abgebrochen. Sollte für einen Operator mehr als eine Zugriffsrichtlinie den Zugriff je Zugriffstyp ''`execute`'' und ''`read`'' gestattet, wird für jeden Operator die Menge der Zugriffsrichtlinien nach Rang geordnet. Höchsten Rang haben die Zugriffsrichtlinien, die weder Filter noch digitale Signaturen definieren. Es folgen absteigend sortiert die Ränge Zugriffsrichtlinien mit Filter, mit digitalen Signaturen und als letztes Zugriffsrichtlinien mit Filter und digitale Signaturen. Unterscheidungen in jedem Rang werden getroffen über die Anzahl der Filter und digitalen Signaturen. Das heißt, es ist die Zugriffsrichtlinie höher im Rang, die eine kleinere Anzahl an Filter oder digitale Signaturen definiert. Die Vorgehensweise wurde gewählt, weil sonst in einem offenen System wie NexusDS, es sehr schnell zu Einschränkungen kommen kann. Das folgt daraus, dass eine hohe Anzahl an Subjekten für verschiedenste Operatoren Zugriffsrichtlinien vergeben können. Die Auswahl von Zugriffsrichtlinien von nicht maximalem Rang, würde mit zusätzlichen Filtern den Detailgrad von Informationen einschränken, obwohl weitere Zugriffsrichtlinien eine detailliertere Verarbeitung zulassen würden. In zukünftigen Erweiterungen der Sicherheitsarchitektur könnte noch eine Auswertung vorgesehen werden, die anhand einer zusätzlichen Beschreibung von Filtern abschätzt, wie stark der Detailgrad von Information durch einen bestimmten Filter beschränkt wird. Dadurch würde eine noch feinere Auswahl möglich, denn in der vorliegenden Rangfolge wäre eine Zugriffsrichtlinie mit einem Filter der alle Datenelemente löscht höher, als eine Zugriffsrichtlinie mit zwei Filtern, die keine Veränderungen vornehmen. Aus Gründen des Umfanges muss jedoch für die Diplomarbeit auf eine feinere Unterscheidung verzichtet werden.

Nach Beendigung von Schritt Drei steht für jeden Operator eine Auswahl von nach Rang sortierten Zugriffsrichtlinien bereit. Auf Basis der Zugriffsrichtlinien, die den höchsten Rang besitzen, wird die Anfrage mit zusätzlichen Meta-Daten angereichert. Die Meta-Daten beschreiben für jeden Operator, auf welchen Rechenknoten dieser ausgeführt werden darf und definieren

die Filter, die an die jeweiligen Dateneingänge und Datenausgänge in welcher Reihenfolge angeordnet werden müssen. Die Reihenfolge der Filter besteht zum einen aus dem in den Zugriffsrichtlinien definierten Attribut `order`, dem Zugriffstyp ''`read`'' oder ''`execute`'' und den Filtern die aufgrund der Forderungen vorgelagerter Operatoren eingebunden werden müssen. Zuerst sortiert der SQP alle Filter der Zugriffsrichtlinien in der Reihenfolge des `order` Attributes, von der kleinsten natürlichen Zahl zur höchsten. Tritt der Fall auf, dass auf einer Zahl mehr als nur ein Filter definiert ist, kommt es zu einer Fallunterscheidung. Filter von Zugriffsrichtlinien mit Zugriffstyp ''`read`'' stehen vor Filter des Zugriffstyp ''`execute`'', sind immer noch Filter auf der gleichen Ordnung, wird nach Zeitstempel der Zugriffsrichtlinien unterschieden. Filter in Zugriffsrichtlinien mit älteren Zeitstempeln stehen vor jüngeren, sind die Zeitstempel gleich, was sehr unwahrscheinlich ist, ist die Reihung nicht deterministisch und folgt aus der zufälligen Reihenfolge der Datenstruktur. Anhand der definierten Reihenfolge kann bei der Definition von Zugriffsrichtlinien eine gewisse Abschätzung getroffen werden, in welcher Reihenfolge Filter von der Anfrageplanung angeordnet werden. Als weiteres Meta-Datum wird die jeweilige Zugriffsrichtlinie angefügt, die den `execute` Zugriff gestattet und den `read` Zugriff für jeden der jeweilig vorgelagerten Operatoren. Dies wird noch für die Interpunktion mit Zugriffsrichtlinien eine Rolle spielen und später erörtert. Der Schlüssel, mit dem die Datenströme der Anfrage verschlüsselt werden, berechnet ebenfalls der **Secure Query Planer (SQP)** und wird als Meta-Datum der gesamten Anfrage hinzugefügt.

7.4.3 Secure Query Optimizer (SQO)

Bisher werden von NexusDS keine Optimierung durchführt, weswegen dieser Schritt übersprungen wird und die Erläuterung der kontrollierten Anfrageplanung direkt am **Secure Query Fragmenter (SQF)** anschließt. Es seien jedoch einige Einschränkungen erwähnt, die bei der Optimierung auf jeden Fall berücksichtigt werden müssen.

Jede durchgeführte Optimierung muss sich im Rahmen der definierten Zugriffsrichtlinien bewegen. Das heißt, Operatoren können nur dann durch Optimierungen ersetzt werden, wenn das Subjekt das Recht hat, die Optimierung auszuführen. Des Weiteren können Veränderungen der Operatoren weitreichende Folgen auf die Zugriffsrechte zum Verarbeiten von Datenelementen haben. Sollte zum Beispiel ein Operator in der Anfrage definiert sein, weil das Subjekt nur mit diesem Operator Datenelemente einer verbundenen Quelle verarbeiten darf, dann kann der Operator nicht durch eine Optimierung ersetzt werden. Ähnliches gilt für die Auswahl von Rechenknoten, diese Optimierung muss ebenfalls unter den Einschränkungen der jeweilig geltenden Zugriffsrichtlinien betrachtet werden. Es kann also nur dann eine erfolgreiche Optimierung durchgeführt werden, wenn sie in enger Koordination mit den vorhandenen Zugriffsrichtlinien der Sicherheitsarchitektur abgestimmt wird.

7.4.4 Secure Query Fragmenter (SQF)

Die Fragmentierung der Anfrage darf nun nur noch auf Basis der Einschränkungen des **Secure Query Planer (SQP)** durchgeführt werden. Dazu sind zu jedem Operator die Angaben

vorhanden, auf welchen Rechenknoten der Operator ausgeführt werden darf. Zum Abschluss der Fragmentierung wird noch ein Eintrag mit Zeitstempel im Protokoll des SQF erstellt, der eine Kopie der Anfrage und die Identität des Subjektes enthält, dass die Anfrage abgeschickt hat. Damit lässt sich nachvollziehen, welche Subjekte, welche Anfragen zu welcher Zeit ausgeführt haben. Sonst ist die Fragmentierung keinen weiteren Einschränkungen unterlegen.

7.4.5 Secure Execution Manager (SEM)

Nachdem die Anfrage in Fragmente zerlegt wurde, geht sie an die zuständigen **Secure Execution Manager (SEM)**. Die SEM erfüllen grundsätzlich die gleiche Aufgabe wie die ursprünglichen *Execution Manager (EM)*, verfügt aber über zusätzliche Funktionen zur Initialisierung und Ausführung von kontrollierten Anfragen. Das Verhalten zur Ausführungszeit unterscheidet sich vom ursprünglichen EM insofern, das er eine Benachrichtigung über einen Zugriffsfehler in der Ausführung erhalten kann. Dann leitet der SEM den Abbruch der Anfrage über den **Secure Core Query Service (SCQS)** ein, der wiederum eine Fehlermeldung an den Absender der Anfrage absetzt und die gesamte Abfrage abbricht. Zudem ist die Aufgabe des SEM neue oder veränderte Zugriffsrichtlinien an die Einbettungen weiterzuleiten, die der SEM aus dem **Secure Core Query Service** erhält. Es werden immer nur auf die eingebettete Operation passende Zugriffsrichtlinien an die Einbettungen weitergeleitet. Passend sind sie dann, wenn sie die Operator-Identität im Attribut `operator` referenzieren.

Zu jedem Operator der aus dem **Secure Operator Repository (SOR)** bezogen wird, muss die jeweilige Einbettung aufgebaut werden **(5)**. Bei der Einbettung handelt es sich um eine logische Konstruktion, die aus verschiedenen Komponenten zusammengesetzt wird. Die Komponenten für alle drei Einbettungstypen, Secure-Source, Secure-Box und Secure-Sink sind gleich. Sie werden lediglich in unterschiedlicher Form zusammengesetzt. Deren genauer Aufbau wird im folgenden Abschnitt erläutert. Die Komponenten sind ebenfalls im SOR vorgehalten und werden zum Start des Fragmentes vom SEM bezogen. Für jeden geschützten Operator baut der SEM die jeweilige Einbettung zusammen, versieht die Komponenten der Einbettungen mit Verbindungen zur Datenübertragung und instantiiert diese in der ursprünglichen Ausführungsumgebung von NexusDS. Während der Instanziierung wird jede Komponente der Einbettung mit den notwendigen Informationen versehen, die diese für die Ausführung benötigt. Das heißt Decoder und Encoder erhalten den geheimen Schlüssel zur Ent- und Verschlüsselung der Anfrage. Der **Local Policy Administration Point (L-PAP)** erhält alle Zugriffsrichtlinien, die die Ausführung des Operators ermöglichen und nach dem höchsten Rang ausgewählt wurden. Neben den Zugriffsrichtlinien erhält der L-PAP alle digitalen Signaturen, über die der Operator verfügt. Ebenso die Identität des Operators und dessen Meta-Daten sowie alle Identitäten der dem Operator vorgelagerten Operatoren. Die Meta-Daten des Operators werden auch an die Encoder gegeben, da diese unter Umständen Angaben zum Zeitpunkt der Wiederinterpunktion von Zugriffsrichtlinien beinhalten. Filter werden mit der Zugriffsrichtlinie parametrisiert, die sie definieren, dem Subjekt das die Anfrage ausführt, der zugeordneten Operator-Identität und der jeweiligen `slotID`. Die Informationen für den SEM sind bereits in Meta-Daten der Anfrage enthalten und wurden von

dem **Secure Query Planer (SQP)** erzeugt. Es folgt die Parametrisierung und Ausführung der Operatoren, nach dem Schema des ursprünglichen *Execution Managers (EM)* **(6)**.

Während der SEM ein Fragmente zur Ausführung bringt, wird in einer Datenhaltung des SCQS vermerkt, welche Operator-Identität der SEM zur Ausführung bringt. Die Information ist notwendig um neue und veränderte Zugriffsrichtlinien in sich in der Ausführung befindlichen Operatoren weiterzuleiten. Nach der Eintragung wird überprüft, ob in der Zwischenzeit der Planung am SCQS geänderte Zugriffsrichtlinien eingetroffen sind, die einen der Operatoren der geplanten Anfrage betrafen. Ist das der Fall, werden diese direkt an die zuständigen Einbettungen übertragen, sodass die neuen Zugriffsrichtlinien direkt zu Ausführungsbeginn der Anfrage interpunktiert werden und der aktuelle Stand der Zugriffsbedingungen abgebildet wird.

7.5 Secure-Source, Architektur und Ausführungsmodell

Die logische Konstruktion der Secure-Source kapselt eine Quelle, wie sie bisher in NexusDS verwendet wird, um sie in der ursprünglichen Ausführungsumgebung von NexusDS kontrolliert auszuführen. Logisch heißt, dass sie vom **Secure Execution Manager (SEM)** vor der Ausführung aus den hier erläuterten Komponenten zusammengesetzt wird. Die Secure-Source kommt sowohl in *Sicherheitszone-Mittel* als auch in *Sicherheitszone-Hoch* zum Einsatz, integriert Filter in die Datenströme und schützt die Datenströme der Datenquelle vor unkontrolliertem Zugriff durch Verschlüsselung. Die Erläuterung zur Architektur und der Funktionsweise der Secure-Source geht nur auf die Komponenten innerhalb der Secure-Source ein. Die Komponenten außerhalb der Secure-Source, mit denen die Secure-Source interagiert, werden noch in den nachfolgenden Abschnitten im Detail erläutert. Abbildung 7.6 illustriert den Aufbau der Secure-Source. Die im folgenden genannten Schritte, entstammen dieser Abbildung.

Die Secure-Source schaltet die von zuordneten Zugriffsrichtlinien definierten Filter zwischen die Datenausgänge der eingebetteten Quelle und den Datenausgängen vorgelagerten **Encodern**. So ist sichergestellt, dass alle definierten Filter zur Anwendung kommen. Sind Datenausgänge explizit für den unkontrollierten Teil von NexusDS freigegeben, dann wird keine Verschlüsselung der Datenelemente und keine Interpunktion vorgenommen und sie können an ursprüngliche Operatoren angebunden werden. Reihenfolge und Auswahl der Filter werden noch im Abschnitt zur Anfrageplanung erläutert und hier nicht weiter ausgeführt. Die direkt vor den Datenausgängen arbeitenden Encoder, interpunktieren den zugeordneten Datenstrom mit Zugriffsrichtlinien und verschlüsseln alle ausgehenden Datenelemente, inklusive der Zugriffsrichtlinien. Die Verschlüsselung erfolgt auf Basis eines symmetrischen Schlüssels, der exklusiv für jeweils eine Anfrage gilt und während der Anfrageplanung erzeugt wird.

Der **Local Policy Administration Point (L-PAP)** steuert die Ausführung der Secure-Source. Zum Start der Anfrage erhielt der L-PAP von dem verantwortlichen **Secure Execution Manager (SEM)** alle Zugriffsrichtlinien, die die Ausführung der Quelle erlauben. Die Menge wird als lokale Zugriffsrichtlinienmenge bezeichnet. Zur Laufzeit übergibt der SEM nur neue oder geänderte Zugriffsrichtlinien an die Secure-Source, wenn diese betroffen ist **(1)**. Das ist dann der Fall, wenn das Attribut `operator` die Identität der von der Secure-Source eingebetteten Quelle enthält.

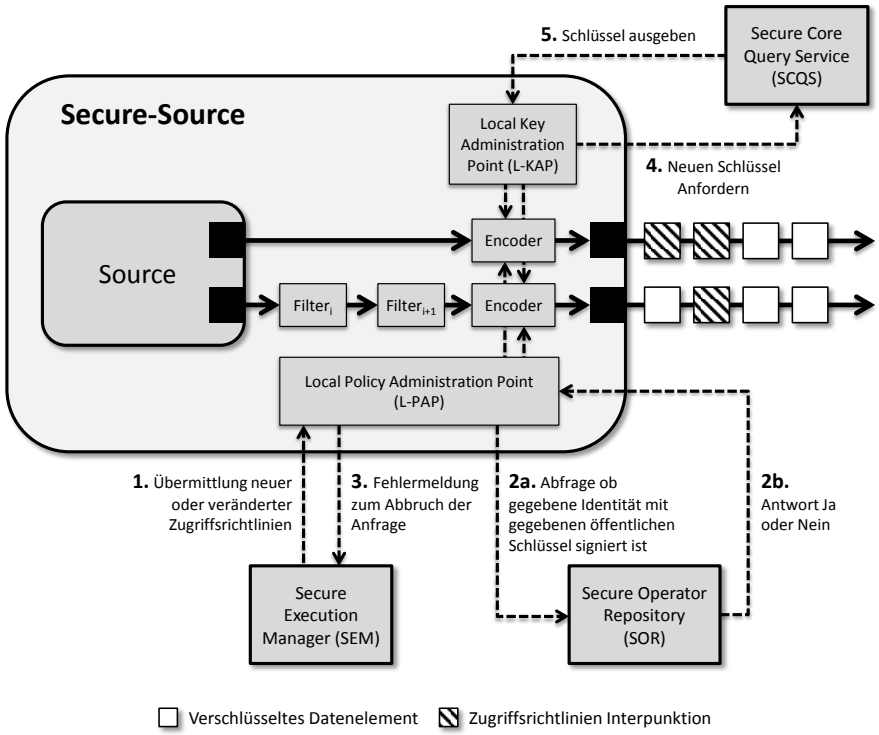

Abbildung 7.6: Illustration der Architektur der Secure-Source. Die gestrichelten Pfeile stellen die Interaktion der Komponenten dar, als durchgezogener Pfeil der von der Quelle erzeuge Datenstrom aus Datenelementen.

Das Attribut `nodes` muss nicht überprüft werden, wäre die Secure-Source nicht auf einem Rechenknoten, dass das Attribut definiert, wäre die Zugriffsrichtlinie nicht übertragen worden. Erhält die Secure-Source eine Zugriffsrichtlinie vom SEM, wird sie sofort über die Encoder in die ausgehenden Datenströme interpunktiert. Vorher berechnet der Encoder noch einen eindeutigen Hashwert der Zugriffsrichtlinie, mit einer global definierten Funktion und verschlüsselt das Ergebnis mit dem geheimen Schlüssel der Anfrage. Das Ergebnis wird dann dem Attribut *policyS* zugewiesen und die so modifizierte Zugriffsrichtlinie interpunktiert. Trifft eine Zugriffsrichtlinie ein, die im Attribut `use` mit ''+'' markiert ist, prüft der L-PAP, ob sie von der Quelle erfüllt wird. Das gilt dann, wenn der Quelle alle in der Zugriffsrichtlinie definierten Filter, an den definierten Datenausgängen vorgelagert sind. Falls von der Zugriffsrichtlinie digitale Signaturen definiert werden, prüft der L-PAP zuerst, ob diese bereits zu Ausführungsbeginn von SEM übergeben wurde. Ist das nicht der Fall, kann der L-PAP über das zentrale **Secure Operator Repository (SOR)** prüfen, ob diese mittlerweile für den Operator verfügbar ist. Dazu stellt der L-PAP eine Anfrage an das SOR. Die Anfrage besteht aus den öffentlichen Schlüsseln der Zugriffsrichtlinie und der Identität der eingebetteten Quelle, die überprüft werden soll **(2)**. Sind die korrekten Filter vorgelagert und die digitale Signatur stimmt überein **(2b)**, wird die Zugriffsrichtlinie zur lokalen Zugriffsrichtlinienmenge hinzugefügt, falls nicht verworfen. An dieser Stellte tritt die Überlegung in den Vordergrund, auch vom Rang höhere Zugriffsrichtlinien der lokalen Zugriffsrichtlinienmenge zuzuordnen, die nur eine Teilmenge der eingesetzten Filter definieren

und damit den Informationsdetailgrad weniger einschränkend. Jedoch widerspricht dies dem Ansatz, die Verarbeitung auf dem höchsten möglichen Detailgrad durchzuführen. Die lokale Menge an Zugriffsrichtlinien dient dazu, dass wenn eine der Zugriffsrichtlinien zurückgenommen werden sollte, eventuell noch weitere vorhanden sind um die Ausführung gewähren zu lassen. Erhält die Secure-Source eine Zugriffsrichtlinie, die im Attribut `use` mit ''-'' markiert ist, wird geprüft, ob diese in der Menge der lokalen Zugriffsrichtlinien vorhanden ist. Wenn ja, wird sie entfernt. Wurde die letzte Zugriffsrichtlinie gelöscht, die den Zugriffstyp ''`execute`'' erlaubt, ist die Ausführung nicht weiter gestattet. Dann sperrt die Secure-Source sofort alle Datenausgänge und sendet eine Fehlermeldung an den **Secure Execution Manager (SEM)**, der die Anfrage abbricht **(3)**. Der Zugriffstyp ''`read`'' ist für eine Quelle nicht relevant, da diese nicht ihre selbst produzierten Datenelemente liest.

Zur Sicherstellung, dass die Secure-Source zu jeder Zeit über den gültigen geheimen Schlüssel für die Anfrage verfügt, ist der **Local Key Administration Point (L-KAP)** vorgesehen. Er überwacht zur Ausführungszeit, ob die Gültigkeit des geheimen Schlüssels, der der Secure-Source bei der kontrollierten Anfrageplanung zugewiesen wurde, noch gültig ist. Sollte die Gültigkeit nur noch weniger als 10 Minuten betragen, was als ausreichendes Zeitpolster betrachtet werden kann, fordert der L-KAP einen neuen Schlüssel vom für die Anfrage verantwortlichen **Secure Core Query Service (SCQS)** an **(4)**. Daraufhin erhält dieser einen neuen Schlüssel, der ab Ablauf des vorherigen Schlüssels für die gesamte Anfrage gilt **(4)**. Läuft der geheime Schlüssel ab, wird er sofort durch den neuen Schlüssel ersetzt, indem der S-KAP den Schlüssel an die Encoder übergibt.

7.6 Secure-Box, Architektur und Ausführungsmodell

Sollen die verschlüsselten Datenströme aus einer Secure-Source von Operator-Boxen verarbeitet werden, dann muss jede ursprüngliche Operator-Box, bestehend aus Warteschlangen, Input-Manger und Operator in eine Secure-Box eingebettet werden. Das ist auch dann der Fall, wenn eine Operator-Box die Datenströme einer Secure-Box verarbeiten will. Die Secure-Box entschlüsselt eintreffende Datenströme, transformiert die Datenelemente mit Filtern gemäß der in Zugriffsrichtlinien definierten Filter und reicht sie an die Operator-Box weiter. Nach der Verarbeitung der Datenelemente durch die eingebettete Operator-Box werden Zugriffsrichtlinien wiederinterpunktiert und mit den ausgehenden Datenströmen verschlüsselt. Die Secure-Box verhält sich analog zur bereits vorgestellten Secure-Source. Die folgende Erläuterung geht nur noch auf die Unterschiede und Erweiterungen gegenüber der Secure-Source ein. Abbildung 7.7 illustriert den Aufbau einer Secure-Box, die angegebenen Schrittzahlen beziehen sich auf Erläuterungen im Text. Einige Interaktionen sind vereinfacht dargestellt und verhalten sich wie in Abbildung 7.6, die die Secure-Source illustriert.

Im Gegensatz zu einer Secure-Source können in einer Secure-Box Zugriffsrichtlinien nicht nur über den **Secure Execution Manager (SEM)** eintreffen, sondern auch als Interpunktionen über die Dateneingänge **(1)**. Die Auswertung der interpunktierten Zugriffsrichtlinien unterscheidet sich in der Auswertung gegenüber denen, die über den SEM eintreffen. Denn diese Zugriffsrichtlinien

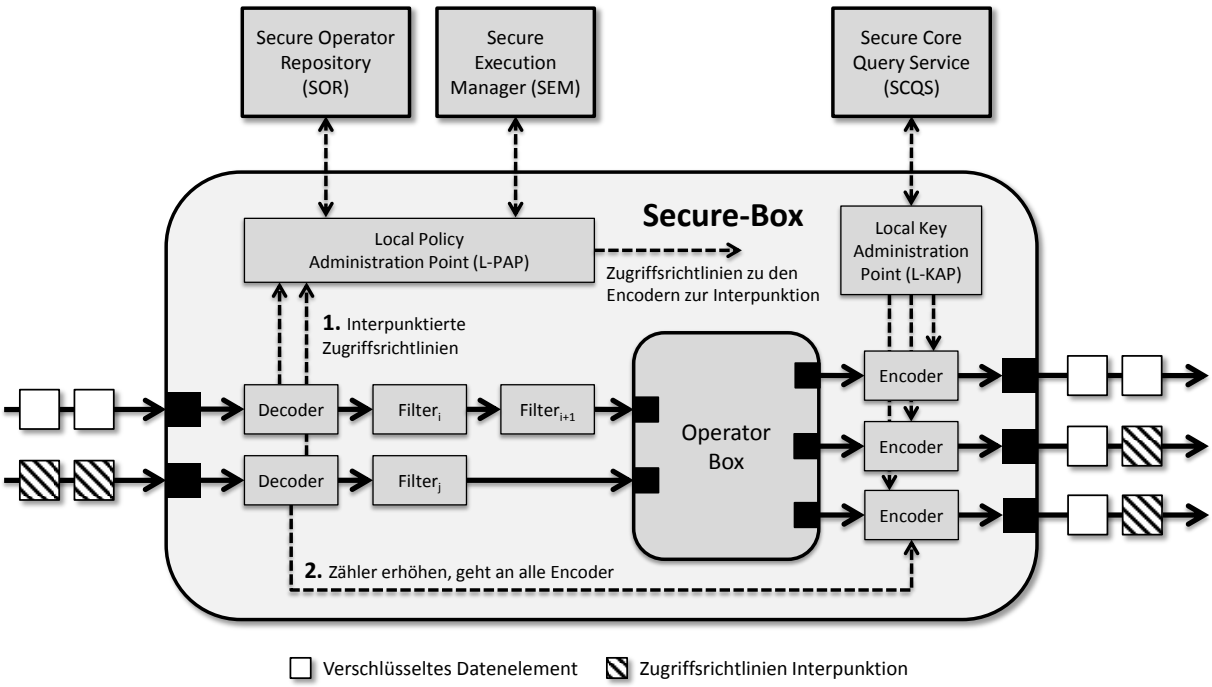

Abbildung 7.7: Architektur der Secure-Box. Die gestrichelten Pfeile stellen vereinfacht Teile der Interaktion zwischen den Komponenten dar, als durchgezogener Pfeil der von der Quelle erzeugte Datenstrom aus Datenelementen.

definieren im Attribut `operator` ein in der Ausführung befindliches Objekt, dass der Secure-Box vorgelagert ist und dessen Datenelemente von der eingebetteten Operator-Box gelesen werden sollen. In diesem Fall müssen die Attribute `with`, `on` und `signatureK` ausgewertet werden. Ist die Identität eines Operators der eingebetteten Operator-Box im Attribut `with` enthalten und der die Operator-Box ausführende Rechenknoten in `on`, dann gilt die Zugriffsrichtlinie für den oder die Operator-Identitäten der eingebetteten Box. Die weitere Prüfung, ob die Zugriffsrichtlinie gilt, verläuft analog zu dem Vorgang der Secure-Source. Je nach Markierung im Attribut `use` mit ''+'' oder ''-'' wird die Zugriffsrichtlinie dann, wenn sie zutreffend, der lokalen Zugriffsrichtlinienmenge im **Local Policy Administration Point (L-PAP)** hinzugefügt oder entfernt. Es gilt wieder, analog zur Secure-Source, dass für jeden Operator der Operator-Box mindestens eine Zugriffsrichtlinie existieren muss, die den Zugriff ''`execute`'' erlaubt. Für den Zugriff ''`read`'' muss nicht nur für jeden Operator der Operator-Box eine Zugriffsrichtlinie existieren, sondern auch für alle vorgelagerten Operatoren. Um diesen Abgleich durchzuführen, hat der L-PAP von der Anfrageplanung die Menge der vorgelagerten Operatoren mitgeteilt bekommen. Verliert die lokale Zugriffsrichtlinienmenge alle Zugriffsrichtlinien, die das Lesen einer der vorgelagerten Operatoren erlauben, muss die Anfrage abgebrochen werden.

Weitere Unterschiede sind das Filter in der Secure-Box den Dateneingängen vorgelagert sind und Decoder die eingehenden Datenströme entschlüsseln. Erkennen die Encoder, dass eine Zugriffsrichtlinie entschlüsselt wurde, wird zuerst die Signatur aus Attribut *policyS* überprüft.

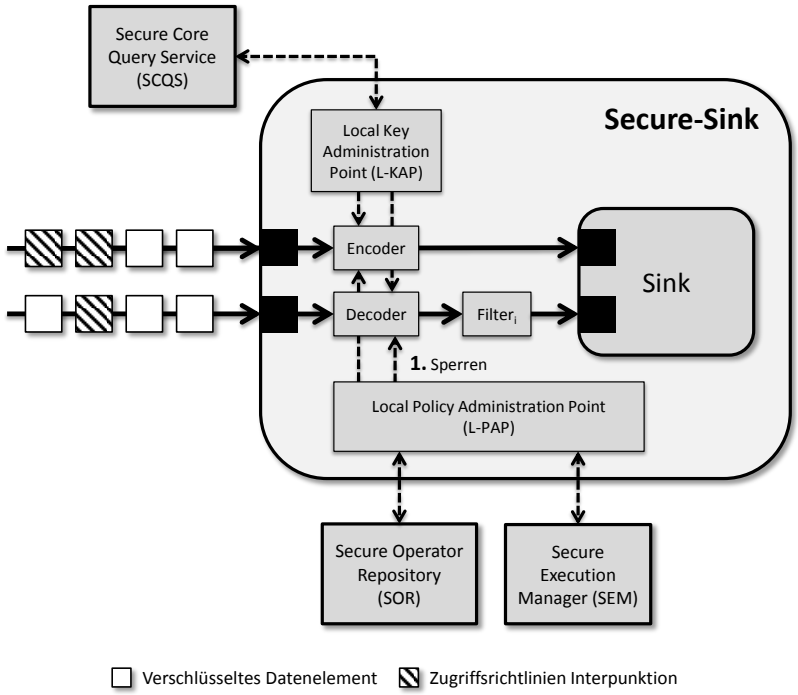

Abbildung 7.8: Architektur der Secure-Sink. Die gestrichelten Pfeile stellen vereinfacht Teile der Interaktion der Komponenten dar, als durchgezogener Pfeil der von der Quelle erzeugte Datenstrom aus Datenelementen.

Dazu wird wieder mit der global bekannten und eindeutigen Hashfunktion die digitale Signatur der Zugriffsrichtlinie, ohne das Attribut *policyS*, berechnet und mit dem geheimen Schlüssel der Anfrage verschlüsselt. Stimmt das Ergebnis mit der Signatur in Attribut *policyS* überein, ist die Zugriffsrichtlinie authentisch und wird an den L-PAP weitergeleitet **(1)**. Es folgt die bereits oben erwähnte Auswertung. Neben der Auswertung trifft der L-PAP auf Basis der in Abschnitt 7.3 definierten Vorganges die Entscheidung, in welche Datenausgänge eintreffende Zugriffsrichtlinien wiederinterpunktiert werden müssen. Zur Bestimmung des Zeitpunktes der Wiedereinflechtung müssen Datenelemente abgezählt werden, was ebenso im genannten Abschnitt erläutert wurde. Deshalb stellt jeder Decoder eine Verbindung zu jedem Encoder her und für jedes eingehende Datenelement wird eine kurze Nachricht abgesetzt, sodass der Encoder einen Zähler für den Dateneingang inkrementieren kann **(2)**. In der Abbildung ist dies nur für einen Encoder angedeutet, der Kanal besteht jedoch zu allen Encodern. Anhand einer Zählerdifferenz von eingegangenen Datenelementen und ausgehenden Datenelementen ist dem Encoder immer bekannt, wie viele Datenelemente sich in der Verarbeitung befinden. Auf Basis von zusätzlichen Meta-Daten der eingebetteten Operation und der Zähler kann der Encoder die Zugriffsrichtlinie zum richtigen Zeitpunkt interpunktieren. Die Meta-Daten erhalten die Encoder vom SEM zu Ausführungsbeginn. Die Laufzeit der Nachricht zum Inkrementieren des Zählers kann vernachlässigt werden, da die direkte Verbindung zwischen Decoder und Encoder eine kürzere Laufzeit hat als die eines Datenelementes bis zum Ausgang der Operation.

7.7 Secure-Sink, Architektur und Ausführungsmodell

Die Aufgabe der Secure-Sink ist eingehende verschlüsselte Datenströme zu entschlüsseln, Zugriffsrichtlinieninterpunktion zu berücksichtigen und gegebenenfalls Filter anzuwenden. Abbildung 7.8 zeigt die Architektur der Secure-Source. Nicht explizit erwähnte Interaktionen beziehen sich auf die Abbildung der Secure-Source 7.6 und der Secure-Box 7.7.

Das Verhalten ist analog zur Secure-Box mit dem Unterschied, dass Zugriffsrichtlinien nicht weiter in Datenströme eingeflochten werden müssen. Die Überprüfung auf Änderungen der Zugriffsrichtlinien ist ebenfalls analog zur Secure-Box. Sollten sich Änderungen ergeben, sodass die weitere Ausführung der Secure-Sink nicht mehr gestattet ist, werden die Decoder gesperrt **(1)** und der **Secure Execution Manager (SEM)** über den Fehler benachrichtigt.

7.8 Services der Sicherheitsarchitektur

Die Sicherheitsarchitektur besteht aus mehreren Services, die unterschiedliche Funktionen realisieren. Jeder der Services besitzt ein digitales Zertifikat, ausgestellt vom zentralen **Certificate Authority Point (CAP)**. Immer dann, wenn zwei Services in Interaktion treten, überprüft jede Seite die Gültigkeit des Zertifikates der Gegenseite über den CAP. Nur falls die Überprüfung erfolgreich war, treten die Services in Interaktion. Die Sicherheitsarchitektur ist deshalb in verschiedene Services zerlegt, um die Erweiterbarkeit zu erleichtern und die Services mehrfach, beziehungsweise verteilt auszuführen. Dadurch wird es möglich, Services die einer hohen Last unterliegen, gezielt auf Rechenknoten mit einer hohen Anzahl an Anfragen auszuführen und so die Latenz der Antworten zu reduzieren. Weiter eröffnet sich die Möglichkeit, verschiedene Datenhaltungen, zum Beispiel im Anwendungsszenario der intelligenten Fabriken, siehe Abschnitt 5.1.3, als verteilte Datenhaltungen zu realisieren [22]. Fragestellungen, die sich aus der Verteilung der Services ergeben, wie konsistente Datenhaltung und Verteilung der Daten, wird von der Diplomarbeit nicht weiter erläutert. Es wird mit der Architektur lediglich eine diesbezügliche Erweiterung grundsätzlich unterstützt.

In Abschnitt 7.8.1 wird zuerst der zentrale Service **Identity Administration Point (IAP)** zur Registratur und Verwaltung von Subjekt-Identitäten vorgestellt. Jedes Subjekt, das mit der Sicherheitsarchitektur interagieren will, muss von diesem Service authentifiziert werden können. Zur Organisation von Subjekten existiert der **Role Administration Point (RAP)**, der ein Rollenmodell verwaltet. In Zugriffsrichtlinien werden Rollen definiert, anstatt die Identitäten von Objekten direkt zu referenzieren. Die genaue Funktionsweise erläutert Abschnitt 7.8.2. Zur zentralen Verwaltung und Vorhaltung von Zugriffsrichtlinien beschreibt Abschnitt 7.8.3 den **Policy Administration Point (PAP)**. Alle geschützten Operatoren sind nur im zentralen **Secure Operator Repository (SOR)** verfügbar, der detailliert in Abschnitt 7.8.4 vorgestellt wird. Der bereits erwähnte Service zur Vergabe und Prüfung von Zertifikaten ist der **Certificate Authority Point (CAP)**, der in Abschnitt 7.8.5 erläutert wird. Die Entscheidung, ob Zugriffsrichtlinien die Ausführung oder das Lesen von Datenelementen bestimmter Operatoren erlauben, wird über **Policy Decision Point (PDP)** entschieden. Details zum PDP erläutert Abschnitt 7.8.6.

7.8.1 Identity Administration Point (IAP)

Unentbehrlich für den Aufbau der Sicherheitsarchitektur ist, dass Subjekte zweifelsfrei und zuverlässig authentifiziert werden können. Das setzt voraus, dass jedes Subjekt über eine eindeutige Identität identifiziert werden kann. Weiter muss ein Mechanismus existieren, mit dem Subjekte ihre Identität nachweisen können, sodass die von der Sicherheitsarchitektur angenommenen Identitäten der tatsächlichen Identitäten der Subjekte entsprechen. Diese Aufgabe wird von dem zentralen **Identity Administration Point (IAP)** übernommen.

Der Charakter von digitalen Identitäten und Mechanismen zum Authentifizieren von behaupteten Identitäten kann in unterschiedlichster Art und Weiße gestaltet werden. In aktuellen Entwicklungen werden bereits Personalausweise mit digitalen Identitäten versehen, über die sich eine Person zum Beispiel im Internet ausweisen kann, um rechtsverbindliche Verträge digital abzuschließen. Auch Mechanismen zur Authentifizierung in verteilten und offenen Systemen, wie NexusDS, existieren bereits, bekannt ist zum Beispiel *Kerberos*[1]. In der vorliegenden Diplomarbeit steht aber nicht im Vordergrund, eines der besten oder neuesten Verfahren zur Abbildung digitalen Identitäten und deren Authentifikation zum Einsatz zu bringen. Es soll lediglich ein solides und zuverlässiges Verfahren zum Aufbau und Darlegung des Sicherheitskonzeptes eingesetzt werden. Deswegen wird für Subjekte das hinlänglich bekannte System eines eindeutigen Benutzernamens und eines Passworts verwendet. Zu beachten ist auch, dass Rechenknoten Subjekte sind, die Operatoren ausführen. Deshalb muss auch für Rechenknoten eine eindeutige Identifikation möglich sein, um in Zugriffsrichtlinien eindeutig auf Rechenknoten verweisen zu können. Nur so kann sichergestellt werden, dass nur eine beschränkte Gruppe von NexusDS Teilnehmern die Ausführung von Operatoren auf geschützten Rechenknoten erlaubt ist. Zur Identifikation von Rechenknoten können zum Beispiel *Trusted Platform Module*[2] eingesetzt werden, die einen Computer mit einer eindeutigen Identität versehen. Zur Vereinfachung verwendet die Diplomarbeit zur Identifikation eines Rechenknoten, die *MAC-Adresse* des Netzwerkadapters, der von dem Rechenknoten zum Datenaustausch mit weiteren Rechenknoten verwendet wird. Bei der Adresse handelt es ich um einen eindeutigen Wert, der jedem Netzwerkadapter zugewiesen ist.

Kern des IAP ist eine Datenhaltung, in der die Authentifizierungsinformationen zu Subjekten abgelegt werden. **Subjekte** registrieren sich unter Angabe von **Benutzername** und **Passwort**. Der Service nimmt die Informationen über eine verschlüsselte Verbindung entgegen und prüft, ob der Benutzername schon vorhanden ist. Sollte das der Fall sein, wird das Subjekt aufgefordert, einen anderen Benutzernamen zu wählen. Ist der Benutzername eindeutig, wird das Passwort mit einem kryptografischen Verfahren verschlüsselt, sodass es zum Beispiel vor dem Einblick der Administratoren, die direkten Zugriff auf die Datenhaltung besitzen, geschützt ist. Das Verfahren setzt eine kryptografische Hashfunktion ein, die einen eindeutigen Wert liefert. Danach wird der Benutzername zusammen mit dem Hashwert in die Datenhaltung des Service eingelagert. Gelöscht werden kann eine Identität nur unter Angabe von Benutzernahme und Passwort.

[1] Definition der aktuellen Version 5 unter http://tools.ietf.org/html/rfc4120
[2] Weiterführende Informationen von Infineon auf http://www.infineon.com/tpm

Rechenknoten werden ebenfalls über den IAP registriert und mit der entsprechenden **MAC-Adresse** als Identität abgelegt. Zu jeder Rechenknotenidentität kann eine Menge von Rollen definiert werden, die Zugriffsrichtlinien für die Rechenknoten vergeben dürfen. Weiter werden einzelne Subjektidentitäten einem Rechenknoten hinzugefügt, die als Administration die Rollenzuweisungen verweisen. Auf weitere Details wird in der Diplomarbeit aus Gründen des Umfanges nicht eingegangen.

Will ein Service der Sicherheitsarchitektur überprüfen, ob die Identität eines Subjektes authentisch ist, überträgt der Service den vom Subjekt angegebenen **Benutzernamen** und das **Password** an den Service. Der Service prüft, ob die gegebenen Einträge in der Identitätsdatenhaltung vorhanden sind. Ist für den Benutzernamen das gegebene Passwort hinterlegt, wird der Benutzername als Identität bestätigt, andernfalls erfolgt die Ablehnung mit einer Fehlermeldung.

Das besprochene Modell birgt einige Schwachstellen. Darunter fällt die Auswahl schwacher Passwörter, wie zum Beispiel 12345. Dies kann abgemildert werden, wenn die Subjekte gezwungen sind, sicherere Passwörter zu wählen, indem diese algorithmisch auf ihre Qualität geprüft werden. Aber auch nach der Wahl eines sicheren Passwortes muss es geheim gehalten werden. Die Subjekte dürfen es nicht weitergeben, die Verbindungen über die Passwörter im Klartext übertragen werden, müssen abhörsicher sein und die Ziele für die Passwörter müssen vor Übertragung zweifelsfrei als berechtigte Empfänger verifiziert werden. Die Sicherheit von Mechanismen zur Authentifizierung ist kein direktes Thema der Diplomarbeit, eine detailliertere Abhandlung findet sich in [16] oder einschlägigen wissenschaftlichen Abhandlungen.

7.8.2 Role Administration Point (RAP)

Bereits in den Anforderungen der Anwendungsszenarien in Abschnitt 5.1 wurde angedeutet, dass die Sicherheitsarchitektur unter Umständen eine große Zahl an Subjekten verwalten muss. Zur Abbildung wird das **Role Based Access Control Modell (RBAC)** [20] verwendet, dass neben den Modellen MAC und DAC bereits in Abschnitt 2.3 mit Vor- und Nachteilen erläutert wurde. RBAC wird ausgewählt, da es gegenüber den anderen vorgestellten Modellen Vorteile bei der Verwaltung großer Anzahlen von Subjekten bietet und wegen seines hohen Verbreitungsgrades allgemein gut verstanden wird. Ein gutes Verständnis wirkt sich positiv auf die Sicherheit aus, da weniger Fehler bei der Verwendung zu weniger Lücken in Sicherheitseinstellungen führen.

Abbildung 7.9 zeigt, wie Zugriffsrichtlinien **indirekt über Rollen Subjekten zugeordnet werden**. Mit der Zuweisung über Rollen lassen sich leicht Mengen von Zugriffsrichtlinien mit Mengen von Subjekten verbinden, ohne dass jede Zugriffsrichtlinie jedem Subjekt Einzel zugewiesen werden muss. Gleichzeitig ergibt sich mit der Anordnung von Rollen eine Hierarchie, in der Abbildung als gerichtete Kanten visualisiert, in der **Zugriffsrichtlinien vererbt werden können**. Das ist insbesondere dann von Vorteil, wenn Organisationsstrukturen, wie im Beispiel des Anwendungsszenarios der intelligenten Fabriken aus Abschnitt 5.1.3, abgebildet werden sollen. Beispielsweise könnten in einer Abteilungsleiterrolle alle Rollen der mit der Abteilung verbunden Subjekte untergeordnet werden. Im einfachsten Vererbungsfall, dass alle Zugriffsrichtlinien vererbt werden, erhalten die Subjekte mit zugeordneter Abteilungsleiter-Rolle alle

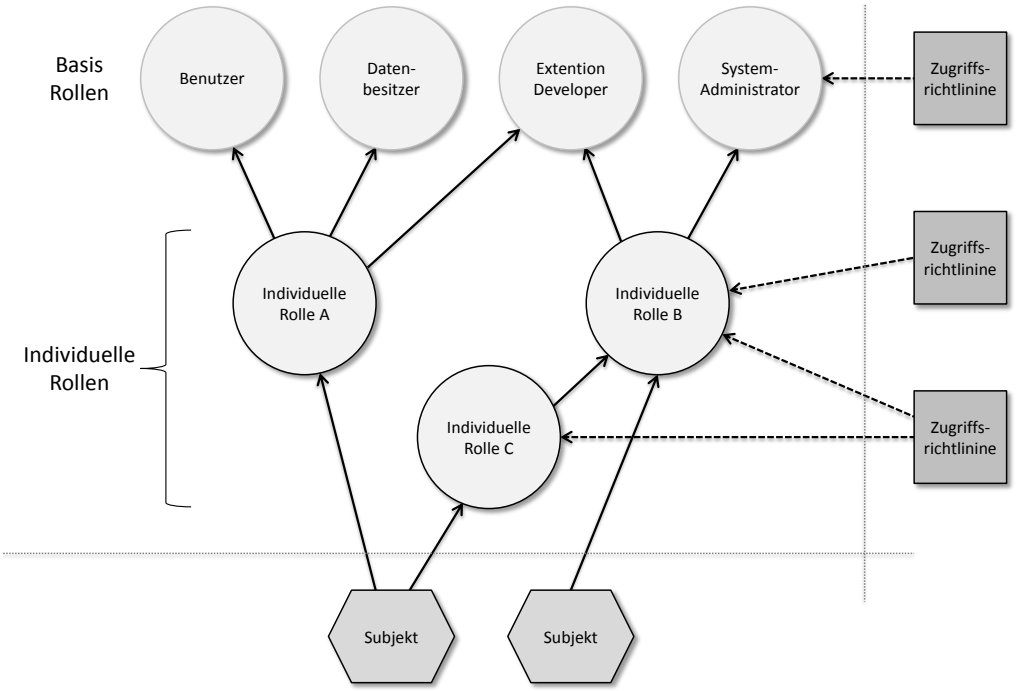

Abbildung 7.9: RBAC Modell das Subjekten Rollen zuweist und Zugriffsrichtlinien an Rollen bindet [20].

Zugriffsrichtlinien der Mitarbeiter-Rolle. Umgekehrt beinhaltet die Mitarbeiter-Rolle keine Zugriffsrichtlinien der Abteilungsleiter-Rolle. Grundsätzlich gilt, ist eine Rolle über eine gerichtete Kante zugeordnet, werden alle Zugriffsrichtlinien an die Ziel-Rolle vererbt, sofern die Definition der Zugriffsrichtlinie eine Vererbung nicht verbietet. Konflikte in der Vererbungskette können nicht auftreten, da Zugriffsrichtlinien nur Zugriffe erlauben und nicht verbieten. Abbildung 7.9 zeigt ein Beispiel für eine mögliche Mehrfachvererbung, in der die *Individuelle Rolle C* alle nicht explizit ausgeschlossenen Zugriffsrichtlinien von *Individuelle Rolle C* und den Basis-Rollen *Extention Developer* und *System-Administrator* erbt. Auch bei Veränderungen der Relationen von Subjekten und Zugriffsrichtlinien spielt RBAC Vorteile aus. Veränderungen, die mehrere Subjekte betreffen, müssen nicht für jedes einzelne Subjekt durchgeführt werden, was möglicherweise zu einem Übersehen einzelner Subjekte führen kann, sondern es wird nur die Zuordnung zur betroffenen Rolle angepasst. Die Anpassung wirkt sich auf alle mit der Rolle verbundenen Subjekte aus und pflanzt sich automatisch in der gegebenen Vererbungshierarchie fort. Um jede Rolle einem gewissen Typus in NexusDS zuordnen zu können, endet die Vererbungskette von individuellen Rolle an einer Basis-Rolle. Beispielsweise können nur Subjekte einem Rechenknoten als Administrator zugeordnet werden, die über die Vererbungskette der Basis-Rolle Administrator zugeordnet sind.

Rollen und Subjekten werden verbunden, indem für jede Rolle die Benutzernamen aller zugeordneten Subjekte abgelegt werden. Der Benutzername entspricht der Identität der Subjekte und definiert so eine eindeutige Zuordnung. Die Zuordnungen werden im zentralen **Role Administration Point (RAP)** Service administriert. Der RAP ist dafür zuständig, neue Rollen aufzunehmen,

Rollen zu löschen und die Zuordnungen von Subjekten zu Rollen abzubilden. Jeder Rolle können Besitzer zugeordnet werden, Besitzer dürfen Subjekte löschen, hinzufügen und neue Rollen erzeugen, die den Rollen untergeordnet werden dürfen, von denen sie selbst Besitzer sind. Sie können ebenfalls Besitzer hinzufügen und löschen, es muss aber mindestens immer ein Besitzer existieren. Wird eine Rolle neu erstellt, ist das erstellende Subjekt der Besitzer.

7.8.3 Policy Administration Point (PAP)

Im Wesentlichen erfüllt der Service die Aufgaben einer zentralen Administration und Datenhaltung für Zugriffsrichtlinien. Sollen aktuelle Zugriffsrichtlinien abgefragt werden, ist der **Policy Administration Point (PAP)** der zentrale Service für Anfragen. Werden neue Zugriffsrichtlinien erstellt oder bestehende verändert, überträgt der PDP diese an jeden sich in der Ausführung befindlichen **Secure Core Query Services (SCQS)**. Der wiederum die Aufgabe hat, Zugriffsrichtlinien auf sich in der Ausführung befindliche Operatoren weiterzuleiten. Sollte eine Zugriffsrichtlinie verändert oder gelöscht werden, besteht die Übertragung an den SCQS aus zwei Zugriffsrichtlinien. Einmal die Ursprüngliche, im Attribut `use` mit ''-'' versehen und einmal der veränderten Zugriffsrichtlinie mit ''+''. Erstere sorgt als Interpunktion dafür, dass alle betroffenen Einbettungen für Operatoren darüber informiert werden, welche Zugriffsrichtlinie nicht mehr gilt. Umgekehrt zur Markierung ''-'' weist die Markierung ''+'' Zugriffsrichtlinien als gültig aus. Im Fall, dass eine Zugriffsrichtlinie neu erstellt wurde, besteht die Übertragung an den SCQS nur aus einer Zugriffsrichtlinie, die mit ''+'' markiert ist. Das Markieren der Zugriffsrichtlinien wird vom PAP automatisch durchgeführt.

Neue Zugriffsrichtlinien dürfen von jedem Subjekt erstellt werden, das mit einer eindeutigen Identität am zentralen **Identity Adminstration Point (IAP)** authentifiziert werden kann. Die Belegung der Zugriffsrichtlinien-Attribute unterliegt jedoch Bedingungen. Ein Subjekt kann nur dann für einen Operator Zugriffsrichtlinien vergeben, wenn es Mitglied einer Rolle ist, die im zentralen **Secure Operator Repository (SOR)** für den Operator zur Zugriffsrichtlinienvergabe eingetragen ist. Gleiches gilt für Angabe auf welchen Rechenknoten ein Operator ausgeführt werden darf. Es können nur die Identitäten von Rechenknoten für das Attribut `nodes` eingetragen werden, für die das Subjekt als Besitzer im IAP vermerkt ist. Alle anderen Attribute können beliebig belegt werden und unterliegen keinen Restriktionen. Jedoch stellt der PAP sicher, dass das Subjekt einen Bezeichner für die Zugriffsrichtlinie wählt, der eindeutig ist. Die Belegung der Attribute bezüglich Evaluatoren und Filter ist nicht eingeschränkt. Das ermöglicht in dieser Version der Sicherheitsarchitektur deren freie Verwendung. Die Vereinfachung wurde gewählt, um den Rahmen der Diplomarbeit nicht zu sprengen. Denn eine diesbezügliche Zugriffsbeschränkung, erhöht die Komplexität der Anfrageplanung erheblich.

Wollen andere Services oder Werkzeuge, zum Beispiel grafische Werkzeuge zum Erstellen von Zugriffsrichtlinien, Informationen vom PAP abrufen, müssen sie über ein gültiges Zertifikat der Sicherheitsarchitektur verfügen, das vom zentralen **Certifiate Authority Point (CAP)** ausgestellt wurde.

7.8.4 Secure Operator Repository (SOR)

Operatoren werden in der ursprünglichen Version von NexusDS im *Operator Repository (OR)* vorgehalten. Für die Ablage von geschützten Komponenten ist das Repository jedoch nicht geeignet, da den Operatoren digitale Signaturen zugeordnet werden sollen. Ebenfalls muss der Zugriff auf die eingelagerten Operatoren beschränkt werden, sodass nur die Anfrage eingelagerte Operatoren laden können, die von Zugriffsrichtlinien erfüllt werden. Für diese Aufgabe wird das ursprüngliche Repository erweitert und durch das **Secure Operator Repository (SOR)** ergänzt. Abgerufen werden können die Implementierungen der eingelagerten Operatoren nur von den Subjekten, die als Besitzer eingetragen sind oder von Services, die über ein bestätigtes Zertifikat des zentralen **Certificate Authority Point (CAP)** verfügen. Weitere Detaillierungen sind nicht vorgesehen, um die Komplexität zu begrenzen.

Neue Operatoren können von jedem Subjekt eingespielt werden, das mit einer eindeutigen Identität authentifiziert werden kann. Es wird dann gleichzeitig als Besitzer des Operators hinterlegt. Besitzer können weitere Subjekte als Besitzer hinzufügen oder löschen, es muss aber mindestens immer ein Besitzer vorhanden sein. Sollte ein Besitzer den Operator löschen, wird dieser aus der Datenhaltung mit den Zuweisungen von Besitzern gelöscht. Jeder Operator wird unter einer eindeutigen Identität hinterlegt, die vom Subjekt bei Erstellen des Eintrages gewählt wird. Die Identität ist analog zu dem Bezeichner, der im ursprünglichen OR gewählt ist, um Operatoren eindeutig in Anfragen zu referenzieren. Weitere Angabe ist eine Zuordnung von Rollen, die definiert, welche Subjekte Zugriffsrichtlinien für den Operator vergeben dürfen.

Will ein Subjekt, das ein Prüfer für Operatoren ist, eine digitale Signatur zu Operator ablegen, muss das Subjekt sich mit einer eindeutigen Identität authentifizieren. So wird sichergestellt, dass ein Prüfer nur mit seiner wahren Identität Operatoren zertifizieren kann. Der Prüfer überträgt die Identität des geprüften Operators und die berechnete digitale Signatur. Die Signatur wurde vom Prüfer über die Implementierung des Operators mit einer global bekannten Hashfunktion berechnet und verschlüsselt. Der Schlüssel zur Verschlüsselung ist der private Schlüssel aus dem Zertifikat, dass dem Prüfer am CAP ausweist. Dadurch, dass der öffentliche Schlüssel jedem Subjekt bekannt ist, kann der öffentliche Schlüssel in Zugriffsrichtlinien vergeben werden. Stellt ein Service eine Anfrage, ob ein Operator von einem Prüfer P zertifiziert wurde, muss nur die Identität des Operators und der öffentliche Schlüssel von P übertragen werden. Das SOR berechnet daraufhin den Hashwert des Operators analog zum Prüfer und entschlüsselt die verschlüsselte Signatur mit dem in der Anfrage gelieferten öffentlichen Schlüssel. Stimmen beide Werte überein, wurde der Operator von Prüfer P zertifiziert. Dann kann die Anfrage bejaht werden, sonst verneint.

7.8.5 Certificate Authority Point (CAP)

Der zentrale Certificate Authority Point (CAP) ist die Vertrauensinstanz der Sicherheitsarchitektur. Sie stellt digitale Zertifikate gemäß einer *Public-Key Infrastruktur (PKI)* aus, die bereits in Abschnitt 2.2.1 erläutert wurde. Die Hintergründe zu PKI sind nicht Thema der Diplomarbeit, es werden

deshalb lediglich die Aufgaben und die Interaktion des CAP in der Sicherheitsarchitektur erläutert. Für Details zu PKI und digitalen Zertifikaten wird in [16] verwiesen.

Der Service arbeitet als **Registrierungsstelle**, die zur Ausfertigung von digitalen Zertifikaten für Services der Sicherheitsarchitektur und Prüfer, die Operatoren digital signieren, dient. Will ein weiterer Service mit einem der Services der Sicherheitsarchitektur kommunizieren, muss er zuerst über den CAP registriert werden. Das dann ausgestellte Zertifikat ermöglicht dem Service, mit den Services der Sicherheitsarchitektur zu kommunizieren. Das gilt auch, wenn Werkzeuge, zum Beispiel zur Erstellung von Zugriffsrichtlinien Informationen an der Sicherheitsarchitektur abfragen wollen.

Weitere Aufgabe ist für die Sicherheitsarchitektur als **Validierungsdienst** zu arbeiten. Über den CAP werden dann im Fall vom Aufbau von Verbindungen die Details zu Zertifikaten abgefragt, sodass die Services der Sicherheitsarchitektur einen unbekannten Kommunikationspartner sicher authentifizieren können.

Neben dem Ausstellen von digitalen Zertifikaten für Prüfer, die Operatoren mit digitalen Signaturen versehen, dient der Service auch als **Verzeichnisdienst**. Über ihn können die den Zertifikaten zugeordneten öffentlichen Schlüssel abgerufen werden, die eine Prüfung durch einen mit dem Zertifikat verbundenen Prüfer implizieren. Das ist notwendig um in Zugriffsrichtlinien den öffentlichen Schlüssel eines gewünschten Prüfers einzutragen.

7.8.6 Policy Decision Point (PDP)

Der Policy Decision Point (PDP) entscheidet, ob ein Zugriff auf einen Operator gestattet ist oder nicht. Dazu kann jeder Service, eine Verbindung zum PDP aufbauen, der die Verbindung akzeptiert, sofern der Service über ein gültiges Zertifikat des **Certificate Authority Point** (CAP) verfügt. Das können aber auch zum Beispiel Werkzeuge sein, die die Planung von Anfragen unterstützen, unter der Voraussetzung, sie verfügen über ein digitales Zertifikat des CAP. Eine Anfrage an den PDP besteht immer aus einem Zugriff, aus der Menge {”read”,”execute”}, die die Art der Prüfung und die Struktur der Anfrage bestimmt. Zu jeder Anfrage kann ein boolscher Parameter `short` übergeben werden, der bestimmt wie weit die Auswertung von Zugriffsrichtlinien ausgeführt wird. Ist der Parameter mit wahr belegt, stoppt die Prüfung, sobald eine Zugriffsrichtlinie gefunden wurde, die die Anfrage erfüllt. Sonst werden alle Zugriffsrichtlinien ausgewertet und zurückgegeben, das ist zum Beispiel bei der kontrollierten Anfrageplanung notwendig. Dieser Parameter ist dann sehr hilfreich, wenn nur eine Antwort benötigt wird, ob ein Zugriff gestattet ist. So lassen sich umfangreiche Auswertungen bei einer hohen Anzahl von Zugriffsrichtlinien begrenzen.

Im Fall einer Anfrage mit dem Zugriff ’’`execute`’’, ist die Anfrage wie folgt aufgebaut. Identität des Subjektes, das die Anfrage ausführt wird als *S* bezeichnet. Die Identität des Operators, der ausgeführt werden soll, wird als *O* bezeichnet.

Zur Auswertung bezieht der PDP zuerst alle mit *S* verbundenen Rollen aus dem **Role Administration Point (RAP)**. Zurückgeliefert werden alle die Rollen, denen das Subjekt zugeordnet ist.

Falls das Subjekt nur direkt einer Rolle zugewiesen ist und nicht durch Vererbung Mitglied ist, enthält die Rolle hierzu eine Markierung. Dann alle Zugriffsrichtlinien aus dem **Policy Administration Point (PAP)**, in denen eine der gefundenen Rollen unter dem Attribut `role` eingetragen ist. Sollte in der Zugriffsrichtlinie das Attribut `roleInherit` auf falsch gesetzt sein, dann gelten nur die gefundenen Rollen, die markiert sind. Denn in diesem Fall erlaubt die Zugriffsrichtlinie keine Vererbung. Weiter muss das Attribut `access` mit ''`execute`'' belegt sein und in Attribut `operator` die Identität von O vorkommen. Ist die Abfrage am PAP leer, ist der Zugriff nicht gestattet und es wird Zugriff verboten geantwortet. Wird eine Menge von Zugriffsrichtlinien gefunden, die keinen Evaluator definieren, antwortet der PDP mit Zugriff erlaubt und gibt die gefundenen Zugriffsrichtlinien ohne Evaluator zurück. Finden sich nur Zugriffsrichtlinien mit Evaluator, werden diese in aufsteigender Reihenfolge nach dem Attribut `timestamp` der Zugriffsrichtlinien ausgewertet. Zur Auswertung wird der für die Zugriffsrichtlinie definierte Evaluator aus dem **Secure Operator Repository (SOR)** geladen und von dem PDP zur Ausführung gebracht. Die Beschreibung des Verhaltens eines Evaluators wurde in Abschnitt 7.2.4 bereits besprochen. Sollte eine der Zugriffsrichtlinien digitale Signaturen anfordern, überprüft der PAP über das SOR, ob für den gegebenen öffentlichen Schlüssel in der Zugriffsrichtlinie eine digitale Signatur vorhanden ist. Sobald eine Zugriffsrichtlinie erfolgreich ausgewertet wurde und `short` mit wahr belegt ist, antwortet der PDP Zugriff erlaubt und gibt die positiv ausgewerteten Zugriffsrichtlinien zurück. Sonst wird die Auswertung für alle Zugriffsrichtlinien durchgeführt, die die gegebene Rolle und Operator betreffen und alle erfüllenden Zugriffsrichtlinien zurückgegeben.

Im Fall einer Anfrage mit dem Zugriff ''`read`'', ist die Anfrage wie folgt aufgebaut. S und O sind analog definiert. Zudem ein Teilgraph der Anfrage, der alle Operatoren V und Verbindungen enthält, die vor O liegen. Zu jedem Operator der vor O liegt, bezeichnet als v_i, wird eine Menge an Rechenknoten mitgeliefert, R_i, die definiert, auf welchen Rechenknoten v_i ausgeführt werden darf.

Nun wird für jedes v_i überprüft, ob eine Zugriffsrichtlinie gefunden werden kann, die die folgende Bedingungen erfüllt. Eine der S zugeordneten Rollen findet sich in dem Attribut `role`, abhängig der bereits oben erwähnten Vererbung, das Attribut `access` ist mit ''`read`'' belegt und für jeden Datenausgang von v_i, findet sich im Attribut `with` ein Eintrag für O und das Attribut ''`on`'' enthält einen der Rechenknoten R_i. Kann für jedes v_i eine Zugriffsrichtlinie gefunden werden, antwortet der PDP Zugriff erlaubt und es wird für jedes v_i, alle gefundenen Zugriffsrichtlinien zurückgegeben. Die Auswertung mit Evaluatoren ist analog, nur dass im Erfolgsfall nur maximal eine Zugriffsrichtlinie zurückgeliefert wird. So wird verhindert, das unter Umständen sehr viele Evaluatoren zur Ausführung gebracht werden müssen, obwohl bereits der Zugriff erlaubt ist. Falls mehrere Zugriffsrichtlinien vorhanden sind, die einen Evaluator bestimmen, werden zuerst die ausgewertet, die am wenigsten Einschränkungen definieren. Die hierzu definierte Rangfolge wurde bereits bei der kontrollieren Anfrageplanung erörtert.

Kapitel 8

Implementierung

In den vorhergehenden Kapiteln wurde ausführlich der Aufbau einer Sicherheitsarchitektur für die verteilte Datenstromverarbeitung in Kontext von NexusDS erläutert. Nach dem Vorgehensmodell der Diplomarbeit erläutert das vorliegende Kapitel die prototypische Einbringung der Sicherheitsarchitektur in die aktuelle Implementierung von NexusDS. Es geht nur auf die wichtigsten Implementierungsdetails ein. Erörtert werden Designentscheidungen, die grundsätzliche Vorgehensweise und die wichtigsten Veränderungen an NexusDS, die vorgenommen werden mussten.

Die Implementierung des Prototyps wird wie NexusDS selbst in Java realisiert, um Schnittstellenprobleme mit der vorhandenen Implementierung zu vermeiden. Besonderer Wert wird auf eine homogene Einbringung der Sicherheitsarchitektur gelegt, sodass die Fähigkeit, Anfragen kontrolliert auszuführen sich nicht mehr als nötig auf das ursprüngliche NexusDS auswirkt. Diesem Ziel wurde ein "sicheres" Programmieren, das in Kapitel 2 erläutert wurde, übergeordnet. Zur Vertiefung der Entwicklung sicherer Software mit Java, sei der Leser in [17] verwiesen.

Der erste Abschnitt 8.1 stellt die Struktur der Services der Sicherheitsarchitektur vor. Im Vordergrund steht das Konzept der Implementierung und die Realisierung der Kommunikation. Der Abschnitt geht nicht auf die individuellen Interaktionen der Services ein, diese wurde bereits in den vorherigen Kapiteln ausführlich erläutert.

Die Abbildung von Zugriffsrichtlinien, deren optionale Auswertung mit Evaluatoren und die Implementierung von Filtern beschreibt Abschnitt 8.2.1. Weiter wird erläutert, wie Zugriffsrichtlinien ausgehen vom *Policy Decicion Point (PDP)* durch an sich in der Ausführung befindliche Operatoren propagiert wird. Der Abschnitt enthält ebenfalls eine kurze Einführung in die Implementierung der AWML.

Abschnitt 8.3 erörtert die Vorgehensweise zur Implementierung der kontrollierten Anfrageplanung. Diesbezüglich sind einige Randbedingungen zu beachten, die sich aus der noch prototypischen Implementierung von NexusDS ergeben. Die Umsetzung der Einbettungen, Ausführung von Filtern und die Realisierung von Interpunktionen erläutert Abschnitt 8.4.

8.1 Implementierung der Services

Die Sicherheitsarchitektur besteht aus mehreren Services, die unterschiedliche Aufgaben wahrnehmen. Darunter gehören zum Beispiel der *Role Administration Point (RAP)*, der Subjekte eine oder mehrere Rollen zuordnet. Im Allgemeinen besitzen die Services der Sicherheitsarchitektur einen gemeinsamen Aufbau, der von diesem Abschnitt erläutert wird. In Abschnitt 8.1.1 wird auf die Datenhaltung von Services eingegangen und in Abschnitt 8.1.2 die Implementierung der Kommunikation von Subjekt zu Service und von Service zu Service erläutert.

8.1.1 AWML Datenhaltung für Services

Häufig werden Datenhaltungen in Form von relationalen Datenbanken realisiert. Ein bekanntes Beispiel ist Hibernate[1], ein für Java und .NET verfügbares Open-Source Persistenz Framework, dass über Objekt-Relationale Zuordnungen die Beziehungen zwischen Objekten, deren Attributen und Methoden in relationalen Datenbanken ablegt. Punkte, die für eine Verwendung von Hibernate für die Datenhaltung der Sicherheitsarchitektur sprechen, sind die umfangreichen Funktionen und eine große Entwicklergemeinde, die für eine ständige Weiterentwicklung und Werkzeugunterstützung sorgt. Gegen die Verwendung spricht, dass Hibernate mit seinem großen Funktionsumfang eine nicht unerhebliche Komplexität in die Sicherheitsarchitektur einbringt. Die Komplexität und die Einbindung eines externen Frameworks birgt zusätzliche Fehlerquellen, die unter Umständen die Sicherheit der Datenhaltung negativ beeinflussen kann. Ein weiterer negativer Aspekt ist, dass sich Hibernate nicht homogen in die von Nexus genutzte Struktur für Kontextmodelle einfügt, weswegen zusätzliche Erweiterungen und Typtransformationen eingeführt werden müssten. Diese Überlegung führt direkt zu dem Ansatz, die **Argumented World Language (AWML)** als Modell zur Datenhaltung einzusetzen. Mit der Nutzung der AWML lassen sich die Informationen, die für die Sicherheitsarchitektur erzeugt werden, wiederum in Kontextmodellen verwenden. Zudem können bereits vorhandene Werkzeuge von Nexus weiter eingesetzt werden.

Um die AWML für die Datenhaltungen der Services verwenden zu können, müssen die Schemas, die die XML basierende Datenstruktur der AWML definieren, erweitert werden. Alle Erweiterungen betreffend der Datenhaltungen sind über das *Extended Attribute Schema* `AccessControlDataStoreExtendedAttributeSchema` und das *Extended Class Schema* `AccessControlDataStoreExtendedClassSchema` definiert[2]. Listing 8.1 zeigt ein Beispiel, wie der XML-Dialekt der AWML mit einem Typ erweitert wird. Der gezeigte Ausschnitt definiert den Typ `NexusAccessControlIdentitySubjectAttributeType`, das die Identität eines Subjektes enthält und in der Datenhaltung des *Identity Administration Point (IAP)* zum Einsatz kommt.

Die Basisfunktionalität zur Bereitstellung einer AWML-Datenhaltung für die Services ist in der abstrakten Klasse `AbstractAWMLDataStore` gekapselt. Sie enthält ein `ResultSet` in dem alle

[1] Hibernate ist verfügbar unter http://www.hibernate.org

[2] Alle aufgeführten Schemas finden sich gesammelt im dem Nexus Projekt nexus–federation–streamFederation–services–core—accessControl.

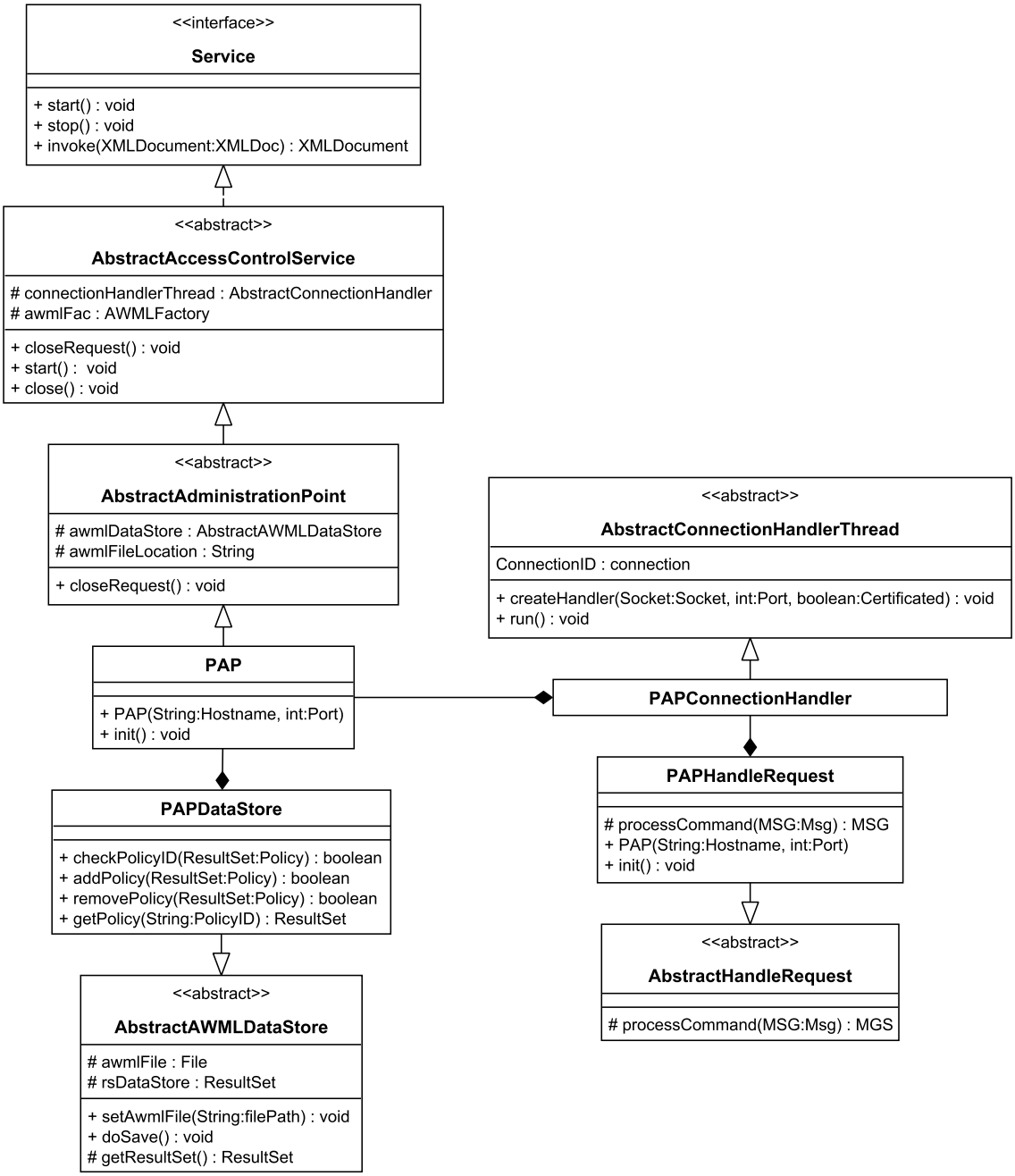

Abbildung 8.1: Klassenstruktur zur Implementierung eines Services der Sicherheitsarchitektur. Die Klassen sind vereinfacht dargestellt und enthalten nicht alle Attribute und Funktionen. Zur Darstellung wurde beispielhaft die Implementierung des *Policy Administration Point (PDP)* gewählt.

```
<complexType name="NexusAccessControlIdentitySubjectAttributeType">
  <sequence>
    <element name="value">
      <complexType>
        <sequence>
          <element name="username" type="nsat:NexusStringType" minOccurs="1" maxOccurs="1"/>
          <element name="password" type="nsat:NexusStringType" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
    </element>
    <element ref="nsas:meta" minOccurs="0"/>
  </sequence>
</complexType>
```

Listing 8.1: Gekürztes Beispiel für einen komplexen Typ zur Definition einer Subjekt Identität.

Daten des Services abgelegt werden und eine Referenz auf eine AWML-Datei zur persistenten Speicherung des `ResultSet`. Die notwendigen Vorgänge zum Anlegen einer neuen Datei, das Laden und Speichern wird von der abstrakten Klasse übernommen. Der Umgang mit einer an die Klasse übergebenen Datei hängt davon, ob die Datei leer ist. Im Falle einer nicht leeren Datei wird versucht, die Datei entsprechend als `ResultSet` zu laden, ansonsten wird ein neues `ResultSet` angelegt. Sollte zur Laufzeit über `setAwmlFile(String:awmlFile)` eine neue Dateireferenz angegeben werden, wird der aktuelle Zustand der Datenhaltung der vorherigen Datei abgelegt und die neue Dateireferenz zur Persistenz vorgesehen. Die Funktion dient dazu, dass die Datenhaltungen von Services zur Laufzeit umgestellt werden können. Beispielsweise falls ein Service mehrfach ausgeführt wird und als Cache arbeitet, dann kann über die Funktion die Datenhaltung regelmäßig aktualisiert werden, ohne den Service neu starten zu müssen.

Jeder Service implementiert eine eigene Spezialisierung zur Datenhaltung als Erweiterung der abstrakten `AbstractAWMLDataStore` Klasse. Beispielsweise im Fall des *Policy Administration Point (PAP)* die Klasse `PAPDataStore`. Die Klasse implementiert die Funktionen, die von dem zugeordneten Service benötigt werden um Zugriffsrichtlinien in das `ResultSet` der Datenhaltung abzubilden. Für die Klasse `PAPDataStore` wäre hierzu ein Beispiel die Funktion `checkPolicyID(ResultSet:rsPolicy)` mit einem booleschen Rückgabetyp, die zu einer gegebenen Zugriffsrichtlinie ermittelt, ob deren Identifikator bereits in der Datenhaltung vorhanden ist. So werden die speziellen Funktionen des jeweiligen Services klar getrennt und gleichbleibende Aufgaben wie das Laden und Speichern der Datenhaltung müssen nicht mehrfach implementiert werden.

8.1.2 Kommunikation mit den Services

Die Kommunikation von Service zu Service und von Subjekt zu Service läuft grundsätzlich verschlüsselt ab. Dazu werden alle Datenkanäle über das **Secure Sockets Layer (SSL)** Protokoll aufgebaut, welches von der Java-Erweiterung *Secure Socket Extension (JSSE)* hinreichend unterstützt wird. Zur Behandlung von Anfragen erwartet der Service an einem SSL-Socket eingehende Verbindungen. Die Verbindungen können entweder dazu aufgebaut werden, um dem Service

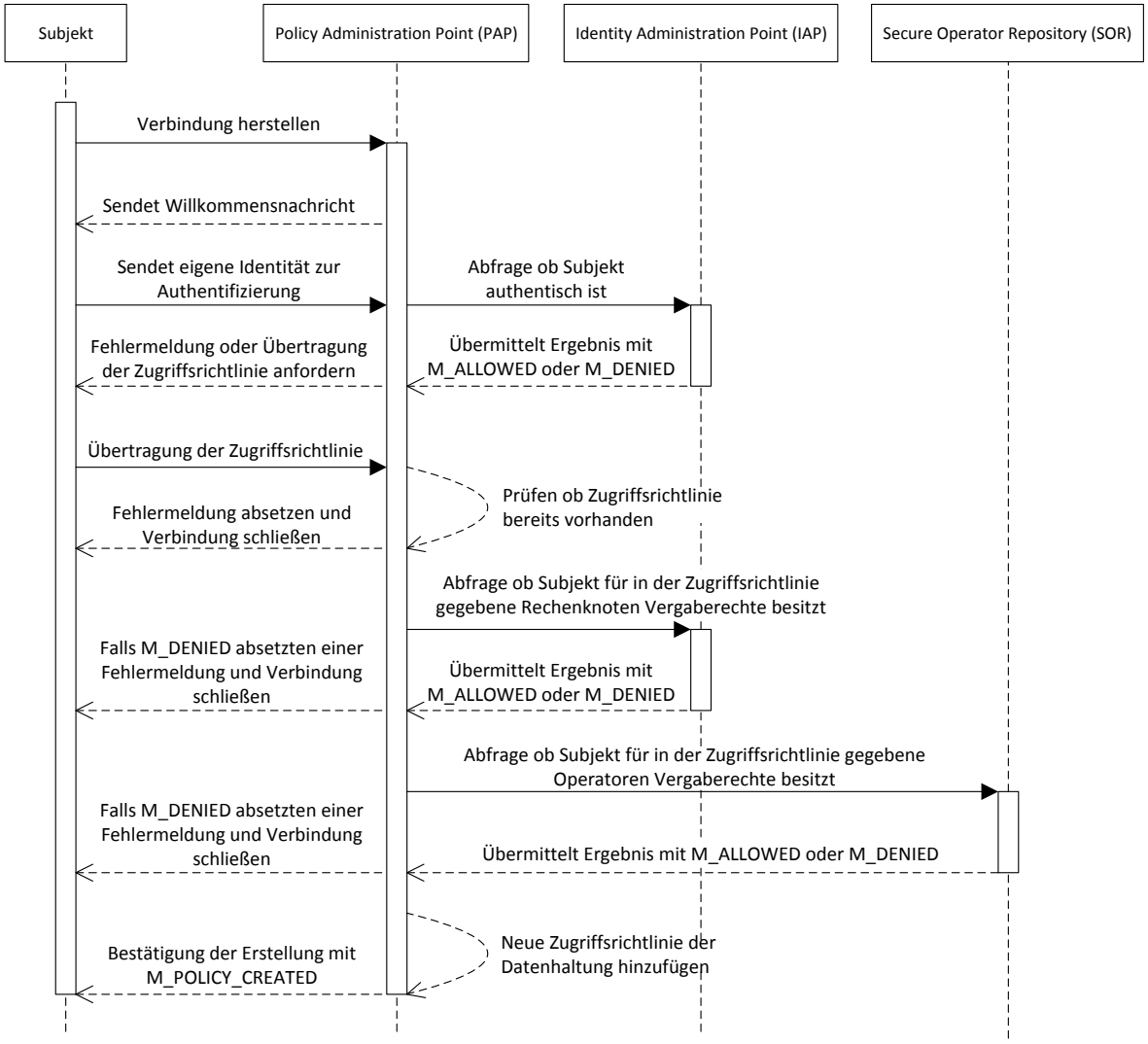

Abbildung 8.2: Sequenzdiagramm für die Erstellung einer neuer Zugriffsrichtlinie am *Policy Administration Point (PAP)*.

neue Daten zuzuspielen, wie zum Beispiel Zugriffsrichtlinien im Fall des *Policy Administration Point (PAP)*, oder zum Abrufen von Information. Information kann entweder von weiteren Services abgerufen werden, Subjekten oder auch von Werkzeugen, die an den Service angebunden werden. Für den PAP könnte dies zum Beispiel ein grafisches Werkzeug zur Erstellung von Zugriffsrichtlinien sein. Aus diesen Gründen eignet sich die Realisierung als öffentlicher Socket, sodass beliebige Erweiterungen und Anwendung lediglich unter Angabe von Hostname und Port mit dem Service in Kontakt treten können.

Zur Abwicklung der eingehenden Verbindungen implementiert jeder Service die abstrakte Klasse `AbstractConnectionHandlerThread`. Sie enthält die grundlegenden Funktionen, um an den Service gerichtete Verbindung anzunehmen. Eingehende Verbindungen werden nur über das

SSL-Protokoll akzeptiert. Falls der Klient der eingehenden Verbindung über ein digitales Zertifikat verfügt, wird es über den zentralen *Certificate Authority Point (CAP)* überprüft. Solle die Prüfung erfolgreich sein, erhält der Klient Zugriffsrechte auf alle Daten des Services. Die vorliegende Implementierung unterscheidet in diesem Fall keine detaillierteren Zugriffsbeschränkungen. Sollte der Klient über kein Zertifikat verfügen, da es sich zum Beispiel um ein Subjekt handelt, das neue Zugriffsrichtlinien eintragen will, muss die Implementierung des Service die Prüfung auf ausreichende Rechte vornehmen. Das kann zum Beispiel durch eine Kontaktaufnahme zum *Identity Administration Point (IAP)* geschehen, der eine gegebene Identität bestätigt oder ablehnt. Jede eingehende Verbindung wird an eine Implementierung der abstrakten Klasse `AbstractHandleRequest` übergeben. Diese läuft in einem eigenen Thread, sodass Anfragen behandelt werden können, ohne den Service für weitere Anfragen zu blockieren.

Zur Bearbeitung von Anfragen wird die abstrakte Klasse `AbstractHandleRequest` mit der Funktion `processCommand(MSG:msg)` implementiert. Die Funktion erhält eine Instanz der Klasse `MSG`, die die eingehende Anfrage darstellt. Eine Anfrage besteht aus einem `String`, der mit einem Befehl oder einer Nachricht belegt werden kann und einem `ResultSet`, für Informationen auf Basis der AWML. In der Funktion behandelt die individuelle Implementierung die Anfrage und liefert als Antwort an den Klienten wiederum eine Instanz der Klasse `MSG`. Als Beispiel wird eine Anfrage an den Service *Role Administration Point (RAP)* angeführt, die den Zweck hat eine Menge von Rollen für eine Subjektidentität abzufragen. Als `String` für den Befehl wird eine Konstante eingetragen, die den Abruf von Rollen zu einem Subjekt definiert. Der RAP liefert dann als Antwort eine Nachricht über Erfolg oder Misserfolg und bei Erfolg zusätzlich ein `ResultSet` mit den gefundenen Rollen. Zur Auswertung von Anfragen greift der Service in der Regel auf die angeschlossene AWML-Datenhaltung zurück, indem Information abgefragt oder verändert wird. Abbildung 8.2 zeigt in Form eines Sequenzdiagrammes[3] beispielhaft, wie die Interaktion zwischen den Services der Sicherheitsarchitektur abläuft, wenn ein Subjekt am *Policy Administration Point (PAP)* eine neue Zugriffsrichtlinie eintragen will.

8.2 Zugriffsrichtlinien

Der Abschnitt 8.2.1 gibt einen kurzen Einblick in die Implementierung der Argumented World Language (AWML) und beschreibt, wie die Attribute der Zugriffsrichtlinien in die AWML abgebildet werden. Zur Auswertung von Zugriffsrichtlinien werden die bereits im vorherigen Kapitel erläuterten Vergleiche durchgeführt. Die Vergleiche basieren dabei immer auf der Suche nach Zugriffsrichtlinien, die einen geforderten Zugriff abdecken. Um diese einfache Art der Vergleiche zu erweitern, wurden Evaluatoren vorgestellt, deren Implementierung in Abschnitt 8.2.2 beschrieben wird. Dass Individuelle transformieren von Datenelementen, um Information vor einer Verarbeitung zu löschen oder zu verschleiern, wird mit Filtern realisiert. Wie Filter individuell implementiert werden können, beschreibt Abschnitt 8.2.3.

[3]Ergänzende Sequenzidiagramme für die weiteren Services befinden sich im Nexus Projekt nexus–federation–streamFederation–services–core—accessControl.

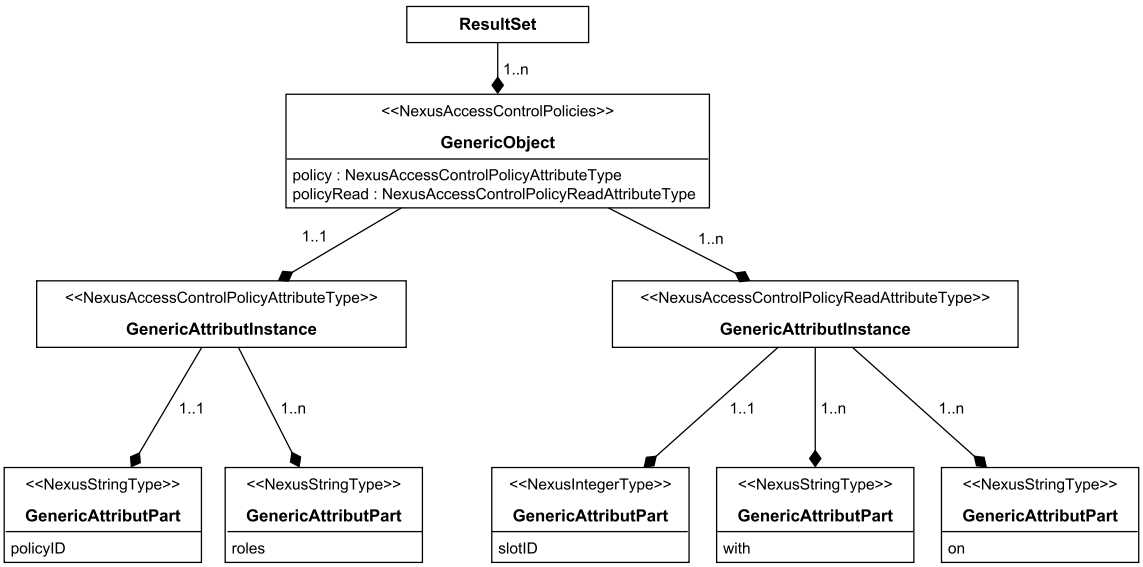

Abbildung 8.3: Unvollständiger Ausschnitt der Abbildung von Zugriffsrichtlinien in die AWML, die vollständige Menge an Attribute findet sich in den erwähnten Schemas in Abschnitt 8.2.1.

8.2.1 Abbildung der Zugriffsrichtlinien

Die Abbildung von Zugriffsrichtlinien erfolgt in der AWML mit den Schemas `AccessControlPolicyExtendedAttributeSchema` zur Definition der Typen und `AccessControlPolicyExtendedClassSchema` zur Definition der Klassen. Die Attribute wurden bereits in Abschnitt 7.2 umfassend erläutert und werden direkt in die AWML abgebildet. Unterschiede finden sich lediglich in der Gruppierung, da in der AWML Attribute nicht beliebig tief verschachtelt werden können. Abbildung 8.3 zeigt einen unvollständigen Ausschnitt, wie die Attribute abgebildet werden. Zum besseren Verständnis der nachfolgenden Erläuterungen folgt eine kurze Einführung in das Modell der AWML. Basiselement ist ein `ResultSet`, dass eine beliebige Anzahl von `GenericObject` Instanzen enthält. Jedes `GenericObject` verfügt wiederum über eine beliebige Anzahl von `GenericAttributeInstance`. Sie bilden den Container für eine Gruppe von `GenericAttributPart` Instanzen, die die Werte für die Attribute der Zugriffsrichtlinien abbilden. In diesen drei Ebenen werden die Zugriffsrichtlinien abgebildet, indem jedes Attribut einer bestimmten Gruppe zugeordnet wird. Jede Zugriffsrichtlinie ist ein `GenericObject` und enthält für die erste Attributgruppe eine `GenericAttributeInstance` vom Typ `policyDescription`. Sie erfasst die wichtigsten Attribute zur Beschreibung einer Zugriffsrichtlinie wie zum Beispiel `policyID` und `access`. Mit der Attributgruppe wird die Zugriffsrichtlinie identifiziert und die elementaren Eigenschaften beschreiben, weshalb sie genau einmal vorkommen muss. Im Gegensatz dazu darf die Gruppe `policyAssignment` beliebig häufig vorkommen, mindestens aber einmal. Sie fasst die Attribute zusammen, an welche

Rolle `role`, Operator `operator` und Rechenknoten `node` die Zugriffsrichtlinie gebunden ist. Die Gruppe `policyRead` fasst alle Attribute zur Beschreibung des Lesezugriffs zusammen. Das heißt, eine `slotID` auf die sich der lesende Zugriff bezieht, das Attribut `with` zur Referenzierung von Operatoren und das Attribut `on`, zur Definition auf welchen Rechenknoten das Lesen der ausgehenden Datenelemente des Operators erlaubt ist. Da diese Gruppe nur im Falle des Zugriffstyp ''`read`'' notwendig ist, muss sie nicht zwangsläufig vorkommen. Sollte der Zugriffstyp jedoch ''`read`'' sein, muss sichergestellt werden, dass die Gruppe mindestens einmal vorkommt. Derartige Restriktionen sind allerdings mit der aktuellen Version der AWML nicht möglich, weshalb die Implementierung der *Policy Administration Point (PAP)* die Einhaltung überwacht. Die Definition der Attributgruppen `policyEvaluator` zur Definition von einem Evaluator, die maximal einmal vorkommen darf, und die Gruppe `policyFilter` zur Definition von Filtern verhalten sich analog.

8.2.2 Implementierung von Evaluatoren

Evaluatoren können optional anstelle der Sicherheitsarchitektur die Entscheidungen treffen, ob ein Zugriff gestattet ist oder nicht. Dazu werden Zugriffsrichtlinien zur Auswertung an Evaluatoren übergeben. Welcher Evaluator das ist, wird von der auszuwertenden Zugriffsrichtlinie bestimmt. Implementiert werden Evaluatoren mit dem Interface `Evaluator` und unter der Verwendung der abstrakten Klasse `AbstractEvaluator`. Die abstrakte Klasse realisiert bereits die grundlegenden Funktionen, auf denen individuelle Evaluatoren aufgebaut werden können. Zur Ausführung gebracht werden die Operatoren im *Policy Decision Point (PDP)*, der nach Bedarf die Evaluatoren aus dem *Secure Operator Repository (SOR)* lädt, instanziiert und mit den notwendigen in Abschnitt 7.2.4 beschriebenen Parametern versorgt. Danach wird die Funktion `evaluate()` ausführt, die im Erfolgsfall ein `ResultSet` zurückliefert. Im erfolgreichen Fall enthält das `ResultSet` eine Zugriffsrichtlinie nach dem bereits definierten Schema, das dann die ursprüngliche Zugriffsrichtlinie für die Auswertung ersetzt. Zu beachten ist, die Ersetzung ist nur temporär für die Weiterverarbeitung der Zugriffsrichtlinie und nicht von Dauer. Das erleichtert die Weiterverarbeitung im Falle der Anfrageplanung, weil die Rückgabe des Evaluators wie eine reguläre Zugriffsrichtlinie verarbeitet werden kann. Sollte der Zugriff nicht gestattet worden sein, wird anstatt einem `ResultSet` nur `null` zurück geliefert.

8.2.3 Implementierung von Filter

Filter ermöglichen, wie in Abschnitt 7.2.5 beschrieben, beliebige Transformationen auf Datenelemente durchzuführen. Sie können grundsätzlich im Rahmen ihrer Spezifikation frei implementiert werden. Die Spezifikation wird vom Interface `Filter<T>` vorgegeben, dass das Minimum an Funktionen definiert, die ein Filter implementieren muss. Dazu gehört die Funktion `process(T:element)`, die die Transformation des Filters implementiert und deren Rückgabetyp ebenfalls von Typ `T` ist. Implementierungen von Filter müssen auf die abstrakte Klasse `AbstractFilter<T>` aufsetzen, die die grundlegenden Funktionen für einen Filter implementiert. Darunter fallen zum Beispiel setter Methoden, die den Filter mit Parameter versorgen, wie

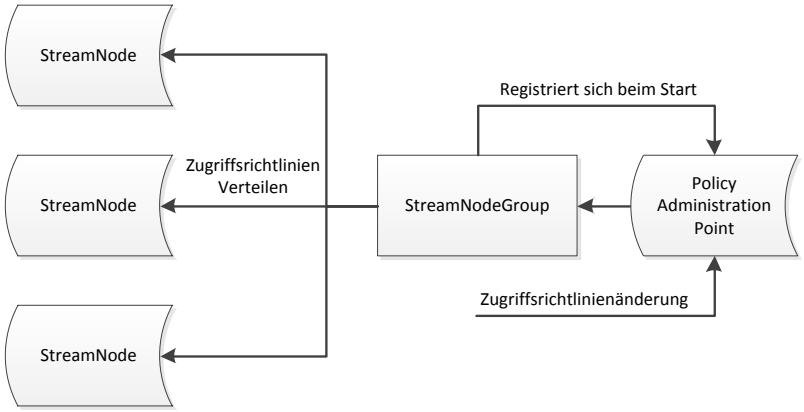

Abbildung 8.4: Schema wie neue Zugriffsrichtlinien von der Sicherheitsarchitektur an sich in der Ausführung befindliche Operatoren weitergeleitet werden.

zum Beispiel welches Subjekt die Anfrage ausführt. Zur Beschreibung des Datentyps, der vom Filter verarbeitet wird, greift die Implementierung auf das Meta-Daten-Modell der Operatoren zurück. Die direkte Übernahme bietet sich deshalb an, da die Filter direkt an die Operatoren die Datenausgänge und Dateneingänge angeheftet werden, deren Datentyp kompatible sein muss. Für den Prototyp wird nur die Definition des Typs `NexusSlotAttributeType` aus dem `OperatorExtendedAttributeSchema` beachtet, die direkt mit den Angaben zur jeweilig zugeordneten `slotID` der Operatoren abgeglichen werden kann. Die Meta-Daten werden jeweils im Paket des Filters, in einer Datei namens `Descriptor.awml` abgelegt.

Der Prototyp der Sicherheitsarchitektur implementiert zur Illustration des Konzeptes den Filter `ResultSetFilter`. Dieser Filter erhält über das Zugriffsrichtlinien Attribut `rule` ein `ResultSet`, dass über die Kette von `GenericObject`, `GenericAttributInstance` und `GenericAttributPart` definiert, welche Instanzen aus eingehenden `ResultSets` zu löschen sind. Wie in Abbildung 8.3 kann ein `ResultSet` als Baum dargestellt werden. Der Filter löscht genau dann jeweils die im `ResultSet` bestehenden Blätter aus den eingehenden `ResultSets`. Zum Beispiel falls ein `GenericObject` keine Kindobjekte besitzt, dann werden alle eingehenden `GenericObject` dieses Typs gelöscht. Besitzt das `GenericObject` mehrere `GenericAttributInstance` Kinder, die keine Kindobjekte besitzt, werden alle `GenericAttributInstance` vom gleichen Typ gelöscht. Mit dieser Konvention müssen keine zusätzlichen Attribute eingeführt werden, die explizit ein Löschen von einzelnen Teilen definieren, sondern es werden immer die Blätter des Baumes der AWML-Struktur aus eingehenden `ResultSets` geschnitten.

8.2.4 Propagierung von neuen Zugriffsrichtlinien

Werden neue Zugriffsrichtlinien am *Policy Administration Point (PAP)* erstellt und betreffen diese sich in der Ausführung befindliche Operatoren, müssen diese über die Änderung benachrichtigt werden. Hierzu muss der PAP in der Lage sein, die entsprechenden Operatoren zu erreichen, um die Zugriffsrichtlinien zu übermitteln. Handelt es sich um eine Zugriffsrichtlinie, die im Attribut

`immediate` mit falsch belegt ist, dann müssen nur Quellen benachrichtigt werden, die ihrerseits die Zugriffsrichtlinie in die Datenströme interpunktieren und so die Änderung der Operatoren in der Anfrage bekannt werden. Ist das Attribut `immediate` mit wahr belegt, muss der PAP alle Operatoren erreichen, denn die Zugriffsrichtlinie soll umgehend durchgesetzt werden und nicht die Verbreitung über die Interpunktion abgewartet werden. Im vorliegenden prototypischen NexusDS, lässt sich die im Konzept definierte Zuteilung der Zugriffsrichtlinien nicht direkt umsetzten. Deswegen wurde für den Prototyp der Sicherheitsarchitektur ein Weg gewählt, indem der PAP darüber informiert wird, an welche Stellen Zugriffsrichtlinien zu leiten sind.

Abbildung 8.4 zeigt, wie die Weiterleitung realisiert wird. Innerhalb eines *StreamNodes* befinden sich die *ProcessLines*, die die Operatoren eines Anfragefragmentes ausführen. Jeder *StreamNode* ist wiederum Mitglied in einer *StreamNodeGroup*, die sich zu Beginn der Ausführung an dem zentralen PAP anmeldet. Kommt es zu einer Veränderung von Zugriffsrichtlinien, indem vorhandene angepasst, gelöscht oder neue Zugriffsrichtlinien erstellt werden, leitet der PAP die Zugriffsrichtlinien an alle registrierten *StreamNodeGroups* weiter. Die *StreamNodes* erhalten nur die Zugriffsrichtlinien, für die sie als Rechenknoten eingetragen sind und leiten dann die Zugriffsrichtlinien an die betroffenen, sich in der Ausführung befindlichen Operatoren, weiter. Gleiches Konzept kommt dann zum Einsatz, wenn eine Zugriffsrichtlinie das Attribut `immediate` mit wahr belegt. Dann wird die Zugriffsrichtlinie an alle Operatoren der Anfragen weitergeleitet, in denen sich eine Instanz des von der Zugriffsrichtlinie betroffenen in der Ausführung befindet. Auf diese Weiße wird die Zugriffsrichtlinie sofort durchgesetzt und es entsteht keine Wartezeit durch die Übertragung als Interpunktion.

8.3 Planung von Anfragen

Wesentlicher Teil zur kontrollierten Ausführung von Anfragen ist eine Anfrageplanung, die nur die Anfrage zur Ausführung bringt, die durch Zugriffsrichtlinien gedeckt sind. Kern ist, bevor eine Anfrage zur Ausführung gebracht wird zu prüfen, ob alle Operatoren von dem Absender der Anfrage ausgeführt werden dürfen und ob diese über ausreichend Leserechte an den Datenelementen ihrer vorgeschalteten Operatoren verfügen. In Abschnitt 7.4 wurde der Ablauf bereits ausführlich beschrieben. Zur Umsetzung geht die Implementierung der Anfrageplanung nach dem in Abbildung 8.5 gezeigten Flussdiagramms durch die Menge der Operatoren einer Anfrage. Der erste Schritt ist die Prüfung der Ausführungsrechte, Abschnitt 8.3.1, der zweite Schritt die Kontrolle der Leserechte, Abschnitt 8.3.2 und abschließend die Einbringung von zusätzlichen Meta-Daten in die Anfrage, Abschnitt 8.3.2.

In der bestehenden prototypischen Implementierung von NexusDS ist der *Core Query Service (CQS)* noch nicht vollständig realisiert. Deshalb wird die prototypische Implementierung des *Secure Core Query Services (SCQS)* auf den wesentlichen Teil, den *Secure Query Planer (SQP)* reduziert. Die Implementierung realisiert die Klasse `SecureQueryPlaner` und ist so gestaltet, dass sie in die zukünftige Implementierung des SCQS direkt eingebunden werden kann.

8.3.1 Überprüfung der Ausführbarkeit von Operatoren

Zu Beginn der Überprüfung erwartet `SecureQueryPlaner` über die Funktion `checkQuery(ResultSet:Query, String:Subject)` sowohl die Anfrage, als auch die Identität des Subjektes, dass die Anfrage zur Ausführung bringen will. Die Ausführbarkeit wird getestet, indem für jede Instanz eines `GenericAttributInstance` vom Typ `NexusBlockAttributeType` überprüft wird, ob mindestens eine Zugriffsrichtlinie dem Subjekt die Ausführung gestattet. Hierzu öffnet der SQP eine SSL gesicherte Verbindung zum PDP und überträgt die Identität des Subjektes und ein `ResultSet` mit dem zu prüfenden Operator. Als Identität für Operatoren wird die `classURI` oder eine `NOL` verwendet, die in der vorliegenden Implementierung von NexusDS bereits zur eindeutigen Identifikation von Operatoren verwendet wird. Unter welchem Hostnamen und Port der verantwortliche PDP zu erreichen ist, wurde dem *Secure Query Planer (SQP)* zur Instanziierung mitgeteilt. Konnte ein Operator erfolgreich überprüft werden, erhält der SQP ein `ResultSet` zurück, dass mindestens eine Zugriffsrichtlinie enthält. Jede gefundene Zugriffsrichtlinie wird dann dem überprüften Operator zugeordnet. Die gleiche Vorgehensweise gilt für Einträge des Typs `NexusInputManagerAttributeType` für Input-Manager und `NexusQueueAttributeType` für Warteschlangen. Konnte der Schritt für jeden Eintrag der Anfrage erfolgreich durchführt werden, geht die Planung in den zweiten Schritt über, sonst wird die Planung abgebrochen. Der Implementierte Algorithmus geht iterativ durch die Menge der Operatoren, was zu einer Laufzeit von $\Theta(n)$ führt, wobei n als Anzahl der geschützten Operatoren definiert ist. Für nicht geschützte Operatoren ist die Überprüfung nicht notwendig.

8.3.2 Überprüfung vorgelagerter Operatoren

Der zweite Schritt überprüft, dass jeder Operator Leserechte an seinen vorgelagerten Operatoren besitzt. Dazu durchläuft die Implementierung ausgehend von den ersten Operatoren, die geschützten Quellen nachgelagert sind, den Anfragegraph. Für jeden Operator wird der entsprechende vorgelagerte Teilgraph abgetrennt und zur Auswertung an den PDP übertragen. Der PDP verfolgt dann rekursiv alle in den zu prüfenden Operator eingehenden Kanten und prüft, ob für den adjazenten Operator eine Zugriffsrichtlinie gefunden werden kann. Von da aus geht wiederum die Verfolgung der Kanten rekursiv weiter, für alle die Kanten, die von der eingehenden Kante abhängen, über die der Rekursionsschritt eingetroffen ist. Die Abhängigkeit von Dateneingängen und Datenausgängen lassen sich entweder über die Meta-Daten des Operators oder über eine eingesetzte Heuristik ermitteln, die in Abschnitt 7.3.1 erläutert wurde. Sollte die Überprüfung zu einem erfolgreichen Abschluss kommen, erhält der SQP eine Menge von Zugriffsrichtlinien, die den Lesezugriff für jeden vorgelagerten Operator enthält. Schlug die Prüfung fehl, ist das `ResultSet` leer, wird die Planung mit einer Fehlermeldung abgebrochen. Sei n wieder die Anzahl der geschützten Operatoren (Knoten) und m die Anzahl der Verbindungen (Kanten). Dann benötigt die Prüfung $\mathcal{O}(n * (n + m))$ Schritte, wobei abhängig von der Anzahl der Kanten entweder n oder m dominiert. Grundsätzlich ist es nicht möglich, den Algorithmus durch eine geschickte Wiederverwendung von vorherigen Schritten asymptotisch zu optimieren. Das liegt daran, dass für die Überprüfung eines Operators die Ergebnismengen eines vorhergehenden Schrittes zwar weiterverwendet werden können, aber dennoch jedes Mal eine

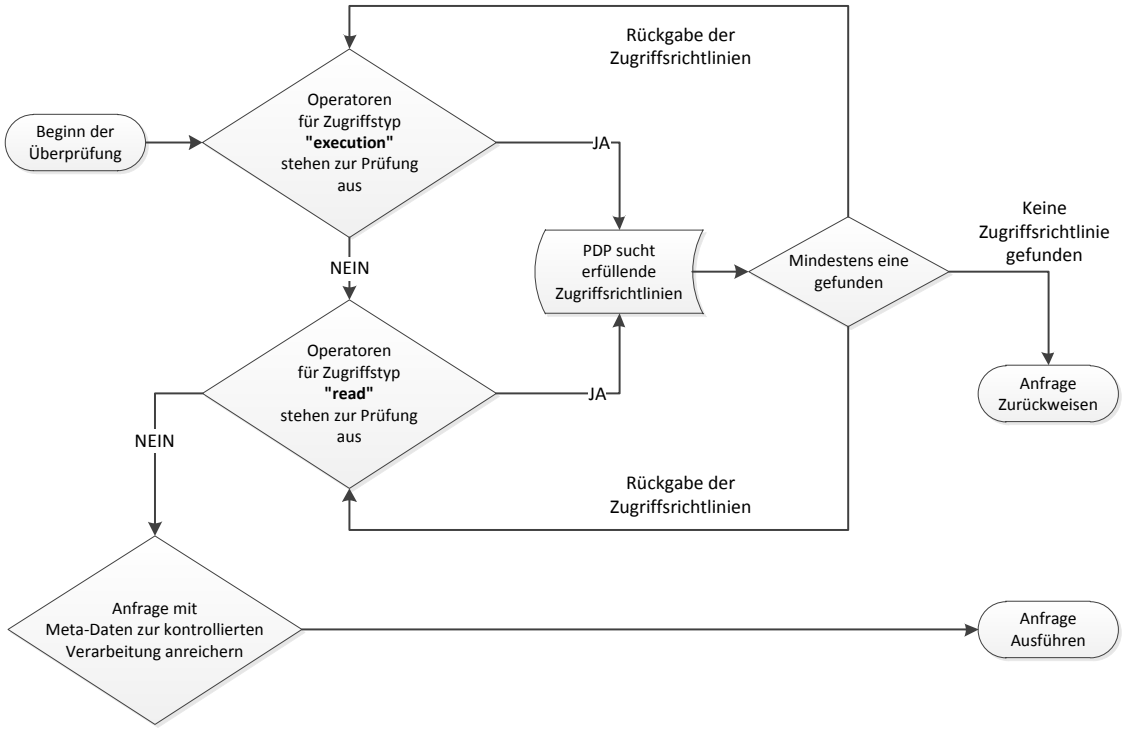

Abbildung 8.5: Schema des Ablaufes der Überprüfung einer Anfrage durch den *Secure Query Planer (SQP)*.

Suche nach Zugriffsrichtlinien für den zu überprüfenden Operator notwendig ist. Die weitere Berechnung der Schnittmengen an Rechenknoten, was schon in Abschnitt 7.4.2 ausführlich erläutert wurde, ist NP-Vollständig und lässt sich auf das Problem *Bin-Packing* reduzieren. Es handelt sich dabei um eine sehr ähnliche Problemstellung wie die Berechnung einer optimalen Verteilung von Operatoren auf verschiedene Rechenknoten, die sich auch als NP-Vollständig erweist. Deswegen ist dieser Schritt der Anfrageplanung sehr aufwendig und die Planung enthält einen Zwischenspeicher, der bei einem erneuten Aufruf mit gleichen Parametern auf ein vorberechnetes Ergebnis zurückgreift. Vor Rückgriff wird jedoch geprüft, ob sich Änderungen in betroffenen Zugriffsrichtlinien, Rollen oder Rechenknoten Zuordnungen ergaben. Dadurch lässt sich die aufwendige Berechnung der Vorgänger und einer Lösung zur Rechenknotenverteilung unter Umständen vermeiden.

8.3.3 Anpassung von Anfragen

Ist die Prüfung für alle Operatoren erfolgreich abgeschlossen, wird die Anfrage mit zusätzlichen Informationen versehen. Hierzu wurden die Schemas `SNSetupExtendedAttributeSchema` und `SNSetupExtendedClassSchema` erweitert, sodass zusätzliche Informationen vermerkt werden können. Zum einen ist das die Information, welche Filter, in welcher Reihenfolge, an welchen Dateneingängen und Datenausgängen an Operatoren angebunden werden müssen. Zweites eine

Auswahl an Rechenknoten, auf denen die Ausführung gestattet ist. Die Information wird dann wiederum beim Start der Ausführung einer Anfrage ausgelesen, was im folgenden Abschnitt erörtert wird. Wie im Folgenden noch erörtert wird, werden Filter im Fall von Operator-Boxen und Senken in den vorgelagerten Warteschlangen angewendet. Deshalb müssen die Warteschlangen, die an geschützte Operatoren angebracht werden, das Interface `SecureQueue<E>` implementieren. Für den Prototyp werden nicht alle bereits existierenden Warteschlangen erweitert, sondern beispielhaft die Warteschlange `SecureCountBasedQueue<E>` implementiert. Die Implementierung der erweiterten Anfrageplanung ersetzt dann all Warteschlangen vom Typ `CountBasedQueue<E>` mit der genannten, erweiterten Warteschlange. Sollen Operatoren geschützt werden, die andere Warteschlangen fordern, müssen diese noch entsprechend erweitert werden.

8.4 Kontrollierte Ausführung von Operatoren

Das Konzept der Sicherheitsarchitektur bettet die Operatoren in eine spezifische Einbettung ein, die Secure-Source für Quellen, die Secure-Box für Operator-Boxen und die Secure-Sink für Senken. Hauptaufgaben der Einbettungen sind der Einsatz von Filtern, um Datenströme zu transformieren und die Reaktion auf Zugriffsrichtlinien zur Ausführungszeit. Im Architekturkapitel wurde bereits angemerkt, dass die Einbettungen von logischer Natur sind und nicht direkt als physische Komponenten vorgesehen sind. Für die Implementierung ist nach Möglichkeit ein Ansatz zu wählen, der die bestehende Implementierung von NexusDS so wenig wie möglich beeinflusst, um die Funktionalität von NexusDS nicht unnötig zu beschränken.

Aus dieser Überlegung lassen sich zwei Hauptansätze entwickeln. Der Erste fügt neue Komponenten zu NexusDS hinzu, die die Operatoren umschließen und Datenelemente mit Filter vorverarbeiten oder im Fall von Quellen nachverarbeiten. Dies käme einer verhältnismäßig direkten Abbildung der Einbettungen gleich, die sich zum Beispiel durch den Einsatz von speziell für die Sicherheitsarchitektur entwickelten Operatoren umsetzten lassen würde. Nachteilig wirkt sich der entstehende Zusatzaufwand aus, der die Verwaltung von einer deutlich höheren Anzahl an Operatoren nach sich zieht. Das ist zum Beispiel dann der Fall, wenn mehrere Filter zum Einsatz kommen und jeder Filter als einzelner Operator ausgeführt wird. Daraus folgt der Ansatz, eine stärkere Integration anzustreben und die Einbettungen in die bestehenden Operatoren zu integrieren. Dadurch wird zusätzlicher Verwaltungsaufwand vermieden und die notwendigen Prozesse der Einbettungen direkt in den betroffenen Operatoren abgewickelt. Weshalb der Ansatz der direkten Integration für die prototypische Implementierung gewählt wurde.

Die Verschlüsselung der einzelnen Datenelemente wird für die Realisierung des Prototyps vereinfacht. Innerhalb einer Ausführungsumgebung, der sogenannten *ProcessLine*, wird keine Verschlüsselung durchgeführt. Im bestehenden Prototyp von NexusDS wird je Anfragefragment eine *ProcessLine* vorgesehen, weshalb der Austausch von Informationen zwischen verschiedenen Fragmenten nicht ohne weiteres möglich ist. Zwar besteht durch gezielte Manipulation an NexusDS während der Ausführung die Gefahr, dass Informationen abgegriffen werden, jedoch ist die Abwehr nicht Thema der vorliegenden Diplomarbeit. Ferner würde eine Verschlüsselung innerhalb einer *ProcessLine* größere Änderungen an NexusDS erfordern, denn jedes Datenelement

würde verschlüsselt nicht mehr seinem ursprünglichen Datentyp entsprechen. Wichtig ist die Verschlüsselung jedoch bei einem Datentransfer über *ProcessLine* hinweg. Das ist dann der Fall, wenn Plattform-Senken und Plattform-Quellen Datenkanäle über die Grenzen von einzelnen *ProcessLines* öffnen. Weshalb in diesem Fall die Verschlüsselung zum Einsatz kommt.

Der integrative Ansatz besteht aus drei wesentlichen Teilen. Erstens erfordert die Verwendung von Interpunktionen eine Erweiterung, sodass Datenelemente von Zugriffsrichtlinien getrennt werden können. Abschnitt 8.4.2 erörtert die hierzu durchgeführten Erweiterungen der Implementierung von NexusDS. Zweitens die Reaktion auf Zugriffsrichtlinien und die Wiedereinflechtung von Zugriffsrichtlinien, was von Abschnitt 8.4.3 erläutert wird. Drittens in Abschnitt 8.4.4, die Anwendung von Filtern, bevor Datenelemente eine Quelle verlassen, von Operatoren verarbeitet oder von Senken an ihr Ziel weitergeleitet werden.

8.4.1 Anpassungen der ProcessLine

Im vorherigen Abschnitt wurde bereits besprochen, dass Anfragen um zusätzliche Attribute erweitert werden, um die zusätzlichen Informationen zu Filtern abzubilden. Die Instanziierung der Filter wird parallel mit der Instanziierung der Operatoren beim Starten von Anfragen durchgeführt. Hierzu wird die `ProcessLine` angepasst, die aus eingehenden Anfragen die zusätzlichen Informationen bezüglich der Filter aus Anfragen extrahiert. Die extrahierten Daten werden dann während direkt and die betroffenen Operatoren weitergereicht, im Fall von Quellen an die Quelle selbst und für Operator-Boxen und Senken wie bereits erläutert an die jeweilige Warteschlange.

8.4.2 Transport von Zugriffsrichtlinien

Das bisherige Konzept Daten zwischen Quellen, Operator-Boxen und Senken auszutauschen, basiert auf der Implementierung des Interface `Receivern<T>`. Über die Klassen, die das Interface implementieren, wird eine Koppelung zwischen den Operatoren innerhalb einer *ProcessLine* hergestellt. Zur Ausführungszeit wird dann die implementierte Funktion `receive(int:InputID, T:Data)` aufgerufen, die ein Datenelement an den mit der `InputID` gekoppelten Dateneingang eines Operators weiterreicht[4]. Der Datentyp des zu übertragenden Datenelementes ist dabei mit dem Typ `T` festgelegt, was eine Übermittlung der Zugriffsrichtlinien, die vom Typ `ResultSet` sind, über den gleichen Weg ausschließt. Um dieses als *Templates* bezeichnete Konzept zu erhalten, mit dem jeder Operator einen eigenen Datentyp definieren kann, werden die Receiver so erweitert, dass parallel Zugriffsrichtlinien übertragen werden können. Das führt zu der Einführung des `SecureReceiver<T>`, der das Interface `Receivern<T>` um die Funktion `receivePolicy(int:InputID, ResultSet:Policy)` erweitert. Abbildung 8.6 zeigt die Erweiterung im oberen Teil **a)**. Auf diesem Weg lassen sich Zugriffsrichtlinien parallel zu den Datenelementen zu den angebundenen Operatoren transportieren, ohne das bereits bestehende

[4]Weitere Details dazu finden sich in der Masterarbeit von Daniel García Sardina [35].

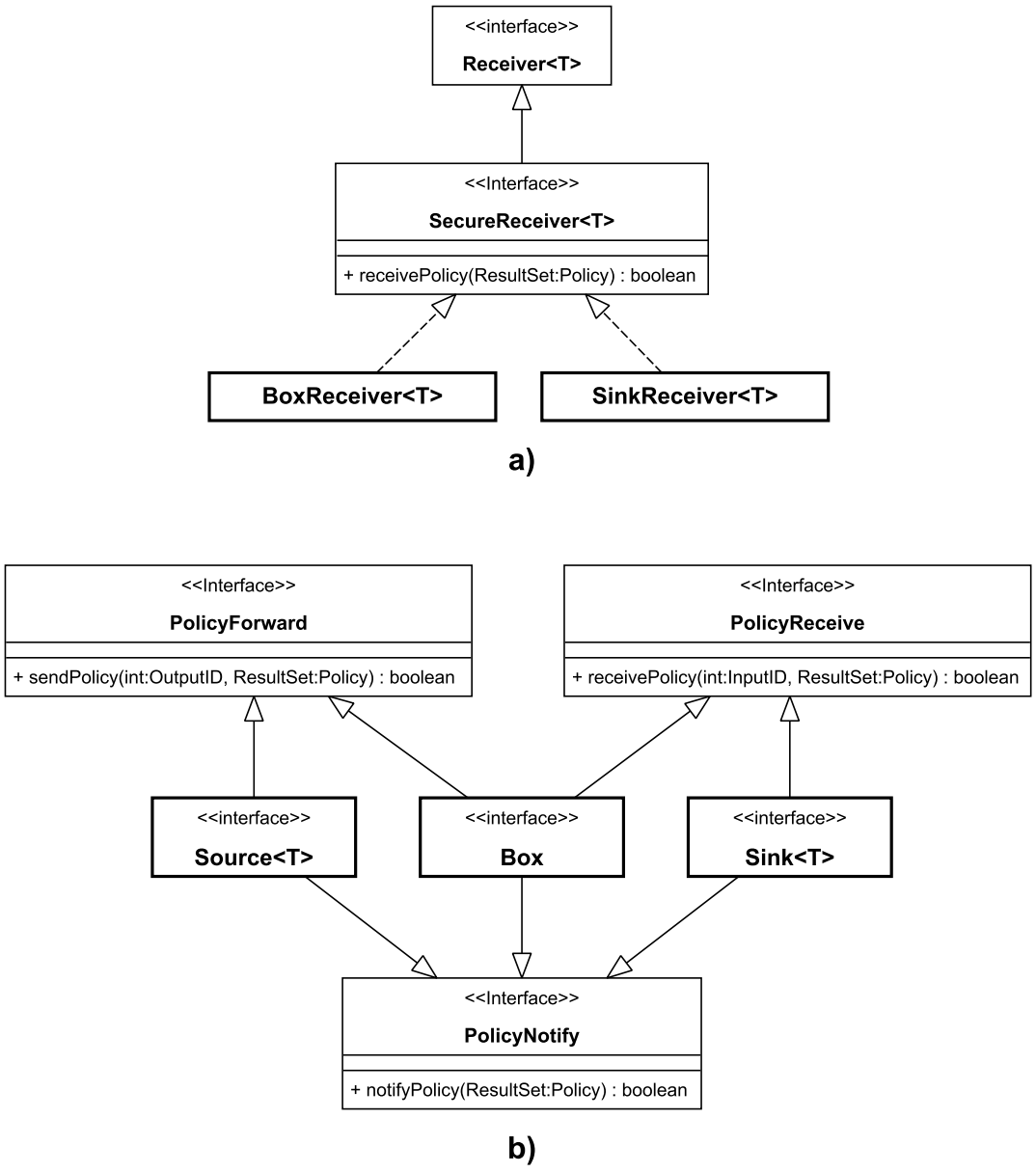

Abbildung 8.6: Erweiterung der Receiver Implementierung zum Transport von Zugriffsrichtlinien und zur Benachrichtigung von Operatoren über neue Zugriffsrichtlinien.

Konzept zu beeinflussen. Der Prototyp der Sicherheitsarchitektur implementiert die Erweiterung für `BoxReceiver<T>` und `SinkReceiver<T>`, sodass die vom NexusDS Prototyp implementierten Standardoperatoren unterstützt werden. Für die Quelle ist das Interface nicht zu implementieren, da diese nur direkt von der Anfrageplanung über neue Zugriffsrichtlinien informiert werden kann und selbst nicht über Dateneingänge verfügt, über die die Quelle Zugriffsrichtlinien erhalten könnte.

Um die von den Receivern realisierten Transport zu unterstützen, muss auch die Implementierung der Operatoren angepasst werden, dazu sind die in Abbildung 8.6 im unteren Teil **b)** gezeigten Interfaces `PolicyForward` und `PolicyReceive` vorgesehen. Erstes Interface um Zugriffsrichtlinien an nachfolgende Operatoren weitergeben zu können und zweites Interface um Zugriffsrichtlinien empfangen zu können. Eine Quelle leitet offenbar nur Zugriffsrichtlinien weiter, weshalb sie nur das Interface `PolicyForward` implementieren muss. Eine Operator-Box, repräsentiert mit dem Interface `Box`, jedoch beide Interfaces, da diese Zugriffsrichtlinien sowohl empfängt als auch weiterleitet. Die Quelle empfängt lediglich Zugriffsrichtlinien und implementiert nur das Interface `PolicyReceive`.

Wie bereits in der Einleitung für den Abschnitt erläutert, ist für die Kombination aus Plattform-Senke und Plattform-Quelle eine verschlüsselte Verbindung notwendig. Bisher wird im Prototyp eine Verbindung zwischen Plattform-Senke und Plattform-Quelle über einen Socket hergestellt und ein serialisiertes Datenelement im Klartext übertragen. Für die Deserialisierung ist aufgrund des bereits erwähnten Template Konzeptes klar, um welchen Datentyp es sich handelt. Soll auf gleichem Wege Zugriffsrichtlinien in der Form von `ResultSets` übertragen werden, muss eine Unterscheidung zur Deserialisierung eines verschlüsselten Objektes getroffen werden, ob es sich um eine Zugriffsrichtlinie oder um ein Datenelement handelt. Das Problem wird gelöst, indem eine serialisierbare Klasse zwei Byte Arrays beinhaltet, wobei Erstere für ein Datenelement und die zweite für eine Zugriffsrichtlinie vorgesehen ist. Je nachdem, ob eine Zugriffsrichtlinie oder ein Datenelement übertragen wird, ist eine der Variablen leer. Dadurch kommt es nur zu einem sehr geringen Überhang und der Aufbau der Plattform-Senken und Plattform-Quellen kann weitgehend beibehalten werden. Denn je nachdem, welche Variable belegt ist, wird der ursprüngliche Mechanismus einer Plattform-Quelle aufgerufen oder die Zugriffsrichtlinie direkt deserialisiert und an die Plattform-Quelle angebunden Operatoren weitergeleitet. Vor Übertragung wird die serialiserte Instanz verschlüsselt, sodass die übertragenen Datenelemente bei der Übertragung durch unbekannte Netze geschützt sind. Die Plattform-Quelle verfügt von der Anfrageplanung über den gleichen Schlüssel wie zu zugehörige Plattform-Senke und kann die Daten wieder entschlüsseln.

8.4.3 Auswertung und Wiederinterpunktion von Zugriffsrichtlinien

Neben dem Empfang von Zugriffsrichtlinien über Dateneingänge werden Zugriffsrichtlinien zur Ausführungszeit auch direkt an Operatoren übermittelt. Das ist dann notwendig, wenn eine neue Zugriffsrichtlinie erstellt wird und ein in der Ausführung befindlicher Operator betroffen ist. Für diesen Zweck implementiert jeder Operator das Interface `PolicyNotify`, das die Funktion `notifyPolicy(ResultSet:Policy)` definiert. Der Aufruf der Funktion bewirkt, dass der Operator

eine Überprüfung bezüglich des Attributes `use` der gelieferten Zugriffsrichtlinien vornimmt, um zu überprüfen, ob der Operator betroffen ist. Die jeweilige Vorgehensweise wurde schon ausführlich in den Erläuterungen der Einbettungen, in Abschnitt 7.5 bis Abschnitt 7.7 beleuchtet. Im Prototypen der Sicherheitsarchitektur wird die Auswertung wie beschrieben durchführt und im Fall, dass die Anfrage abgebrochen werden muss eine `NoPolicyException` ausgelöst, die zum Abbruch der Anfrage führt. Neben der Auswertung wird bei einem Aufruf der Funktion auch die Wiederinterpunktion für Quellen und Operator-Boxen behandelt. Im Fall von Quellen folgt eine unmittelbare Interpunktierung der Zugriffsrichtlinie. Bei einer Senke wird eine Zugriffsrichtlinie nicht weitergeleitet, die Senke wertet die Zugriffsrichtlinie lediglich aus. Für Operator-Boxen müssen wie bei der Wiederinterpunktion noch weitere Bedingungen beachtet werden, die im Folgenden beschrieben werden.

Treffen Interpunktionen über Datenströme an einer Operator-Box ein, die im NexusDS Prototyp mit der Klasse `QueuedBox` implementiert wird, ist einer Wiederinterpunktion der Zugriffsrichtlinien notwendig. Um die Interpunktion nur an den Datenausgängen zu Interpunktieren, die von dem Dateneingang abhängig sind über den die Zugriffsrichtlinie einging, wird das Meta-Daten Schema der Operatoren erweitert. Die Schemas `OperatorExtendedAttributeSchema` und `OperatorExtendedClassSchema` erhalten zusätzliche Attribute, die eine genaue Zuordnung von jedem Dateneingang auf die Datenausgänge ermöglicht. Bei der Initialisierung der `QueuedBox` werden die Meta-Daten gelesen, die definieren, in welche Datenausgänge eintreffende Zugriffsrichtlinien wiederinterpunktiert werden. Falls keine Angaben vorhanden waren, ruft die `QueuedBox` die `sendPolicy(int:OutputID, ResultSet:Policy)` für alle Datenausgänge auf.

Der Zeitpunkt des Aufrufs ist abhängig von Meta-Daten, die dem Operator zugeordnet werden können. Abschnitt 7.3.2 erläuterte die Details zur Bestimmung des Zeitpunktes der Wiederinterpunktion. Dazu notwendig sind Zähler, die sowohl an Dateneingängen und Datenausgängen angebracht werden. Anhand der Zählerdifferenz kann bestimmt werden, wie viele Datenelemente sich in der Verarbeitung, beziehungsweise in den Warteschlangen befinden. Die Meta-Daten definieren, an welchen Dateneingängen, wie viele Datenelemente eintreffen müssen, dass eine bestimmte Anzahl von Datenelementen an einem definierten Datenausgang ausgegeben werden. Triff eine Zugriffsrichtlinie ein, werden die Werte der Zähler festgehalten. Für jeden Datenausgang, an dem die Zugriffsrichtlinie interpunktiert werden soll, wird berechnet, wie viele Datenelemente sich in der Verarbeitung befinden. Das ergibt sich aus der Differenz der Zähler der abhängigen Dateneingänge und des Datenausganges und der Proportion, die die Mata-Daten des Operators definieren. Die Implementierung wartet dann die berechnete Anzahl an Datenelementen für den Datenausgang ab und interpunktiert die Zugriffsrichtlinie über `sendPolicy(int:OutputID, ResultSet:Policy)` genau dann, wenn alle noch in der Verarbeitung gewesenen Elemente ausgegeben wurden. Sollten keine Meta-Daten vorhanden sein, die die Zählung ermöglichen, werden die Zugriffsrichtlinien nach der in Abschnitt 7.3.2 besprochenen Heuristik sofort interpunktiert.

8.4.4 Ausführung von Filter

Filter werden auf zwei unterschiedlichen Wegen angewendet, entweder in Warteschlangen oder direkt im Operator. Handelt es sich um eine Operator-Box oder Senke, ist jedem Dateneingang eine Warteschlange zugeordnet. Die Warteschlangen laden zur Initialisierung die in der Anfrage definierten Filter und binden sie so ein, dass wenn ein Datenelement in die Warteschlange gegeben wird, die Transformationen der Filter angewendet werden und anschließend das Datenelement in der Warteschlange abgelegt wird. Im Fall von Quellen sind keine Warteschlangen an die Datenausgänge angebunden, deshalb müssen die Filter direkt in der Quelle, vor der Auslieferung des Datenelementes, angewendet. Die Anwendung findet im Fall der Quelle direkt in der Funktion `send(E:Data, int:OutputID)` statt, bevor das Element an die Empfänger weitergereicht wird. Der Weg ist weniger vorteilhaft, da die Implementierung der Quelle direkt abgeändert werden muss, um die Filter auszuführen. Dagegen führt die Anwendung in Warteschlangen zu einer getrennten Ausführung der Filter je Dateneingang, was keine zusätzlichen Fallunterscheidungen notwendig macht, welche Filter für welchen Dateneingang oder Datenausgang angewendet werden müssen. Zudem lässt sich die Implementierung so für alle Operatoren wiederverwenden, indem die entsprechende Warteschlange angebracht wird. Der Prototyp der Sicherheitsarchitektur implementiert dazu beispielhaft die `SecureCountBasedQueue`, die die vorhandene `CountBasedQueue` um die Funktionen zur Filterausführung erweitert.

Kapitel 9

Zusammenfassung und Ausblick

Das Kapitel fasst im ersten Abschnitt 9.1 zusammen wie die Sicherheitsarchitektur jedes der zu Anfang der Diplomarbeit definierten Schutzziele erfüllt. Es wird damit gezeigt, dass die Sicherheitsarchitektur den gewünschten kontrollierten Zugriff für NexusDS bezüglich des aufgespannten Rahmens an Anforderungen abdeckt.

Der begrenzte Umfang der Diplomarbeit erforderte die Einschränkung verschiedener Zugriffskontrollen. Abschnitt 9.2 gibt einen Ausblick auf mögliche Erweiterungen der Sicherheitsarchitektur, die von der Diplomarbeit nicht mehr berücksichtigt werden konnten.

9.1 Abdeckung der Schutzziele

Der vorliegende Abschnitt fasst die Fähigkeiten des Sicherheitskonzeptes anhand der Schutzziele zusammen. Tabelle 9.1 gibt eine Übersicht, in welchem Umfang die Schutzziele je Sicherheitszone für das individuelle Operatoren-Modell durchgesetzt werden. Es ist zu beachten, dass die im Folgenden vorgestellten Fähigkeitn in der *Sicherheitszone-Mittel* und *Sicherheitszone-Hoch* gelten. *Sicherheitszone-Null* ist ohne Zugriffskontrolle und entspricht NexusDS vor der Einbringung einer Sicherheitsarchitektur. Die Zone wird in der Zusammenfassung nicht explizit betrachtet.

Authentizität: Wird sichergestellt, indem Subjekte mit einer eindeutigen Identität registriert werden müssen. Bevor die Identität eines Subjektes anerkannt wird, wird dessen behauptete Identität überprüft. Subjekte, die der Prüfung nicht standhalten, werden von der Teilnahme am gesicherten Teil von NexusDS ausgeschlossen. Zur Bestätigung der Authentizität von Operatoren können in Zugriffsrichtlinien digitale Signaturen gefordert werden. Die Ablage von durch den Prüfer verschlüsselten Signaturen im **Secure Operator Repository (SOR)** stellt sicher, dass eine eindeutige Beziehung hergestellt wird. Der private Schlüssel zur Verschlüsselung der digitalen Signaturen steht nur dem Prüfer zur Verfügung, damit ist das Einbringen von gefälschten Signaturen nicht möglich. Solange der Prüfer den Schlüssel geheim hält.

Datenintegrität: Ein Operator ist geschützt, wenn dieser nur im zentralen **Secure Operator Repository (SOR)** verfügbar gemacht wird. Danach kann der Operator nur von authentifizierten und berechtigten Subjekten gelöscht oder ersetzt werden. Gleiches gilt für die Vergabe von Zugriffsrichtlinien, es ist nur den Subjekten möglich, die explizit dafür berechtigt wurden. Die Bearbeitung von Rollen im **Role Administration Point (RAP)** ist in gleicherweise geschützt. Für die Erstellung und Veränderung von Zugriffsrichtlinien, werden die gegebenen Berechtigungen zur Vergabe an die referenzierten Operatoren und Rechenknoten gleichfalls überprüft. Somit ist die Datenintegrität der Zugriffsdefinitionen in der Sicherheitsarchitektur gewährleistet.

Die Datenströme, die zur Ausführungszeit zwischen geschützten Operatoren übertragen werden, sind mit einem einmaligen und geheimen Schlüssel je Anfrage codiert. Der Schlüssel ist nur der kontrollierten Anfrageplanung **Secure Core Query Service (SCQS)** und den Einbettungen in der Anfrage bekannt. Deshalb ist es nicht möglich, dass unberechtigte Subjekte die Datenintegrität der Datenströme beeinflussen.

Informationsvertraulichkeit: Mit der kontrollierten Anfrageplanung **Secure Core Query Service (SCQS)** ist sichergestellt, dass nur die Operatoren zur Ausführung kommen, für die Zugriffsrichtlinien explizit dem Subjekt, das eine Anfrage ausführen möchte, die Ausführung gestatten. Die ursprüngliche Anfrageplanung von NexusDS hat keinen Zugriff auf das **Secure Operator Repository (SOR)**, in dem die geschützten Operatoren vorgehalten werden. Der SCQS trägt ebenfalls Sorge, dass Operatoren nur die Datenelemente von vorgelagerten Operatoren verarbeiten, für die Zugriffsrichtlinien den Zugriff decken. Das Vergeben von Zugriffsrichtlinien am **Policy Administration Point (PAP)** ist nur den Subjekten möglich, die über die Erlaubnis verfügen, in Zugriffsrichtlinien auf die Operatoren zu referenzieren. Die Mengen von Subjekten, die für die Zuweisung von Zugriffsrichtlinien vorgesehen sind, können ebenfalls nur unter der Kontrolle der Sicherheitsarchitektur verändert werden.

Analog zur Datenintegrität ist der Lesezugriff gegenüber nicht autorisierten Subjekten durch die Verschlüsselung der Datenströme gesichert. Tritt eine Veränderung der Zugriffsrichtlinien zur Ausführungszeit ein, sorgt die Interpunktion mit Zugriffsrichtlinien an den Einbettungen zu einer direkten Umsetzung der Bedingungen. Damit wird zu jeder Zeit sichergestellt, dass aktuelle Veränderungen bezüglich der Informationsvertraulichkeit umgesetzt werden. Den Missbrauch von Datenelementen, auf die Operatoren und Senken Zugriff erhalten, zum Beispiel durch Weiterleitung an unberechtigte Dritte, kann in *Sicherheitszone-Mittel* nicht ausgeschlossen werden. Zur vollständigen Kontrolle der ausgeführten Operatoren und Senken muss die in *Sicherheitszone-Hoch* mögliche Zertifizierung angewendet werden. Sonst kann die Informationsvertraulichkeit nur eingeschränkt gewährleistet werden.

Der Informationsaustausch zwischen Services und Subjekten ist reglementiert. Services benötigen digitale Zertifikate, die vom zentralen **Certificate Authority Point (CAP)** vergeben und validiert werden. Damit können Services und Subjekte überprüfen, ob ein Service authentisch ist und die Informationsvertraulichkeit wahrt. Umgekehrt stellen Services sicher, dass Subjekte korrekt über den **Identity Administration Point (IAP)** authentifiziert sind.

Sicherheitsstufe	Authentizität	Datenintegrität	Informationsvertraulichkeit
Hoch	Ja	Ja	Ja
Mittel	Ja	Ja	Teilweise
Null	Nein	Nein	Nein

Sicherheitsstufe	Verfügbarkeit	Verbindlichkeit	Anonymisierung
Hoch	Ja	Ja	Ja
Mittel	Ja	Teilweise	Ja
Null	Nein	Nein	Nein

Tabelle 9.1: Tabelle über die Erfüllung der Schutzziele in den Sicherheitszonen, der Eintrag teilweise steht für eine nur partielle Erfüllung.

Verfügbarkeit: Zugriffsrichtlinien können das Ausführen von Quellen, Operatoren und Senken auf eine nur begrenzte Auswahl von Rechenknoten erlauben. Für die erweiterte Anfrageplanung können Einschränkungen vorgesehen werden, dass nur geschützte Anfragefragmente auf die ausgewählten Rechenknoten ausgebracht werden. Dann können nur noch die Anfragen auf den ausgewählten Rechenknoten ausgeführt werden, die alle relevanten Zugriffsrichtlinien erfüllen. Somit können unberechtigte Subjekte auf den Rechenknoten keine Anfragen zur Ausführung bringen, die deren Leistungsfähigkeit unberechtigt beeinträchtigt.

Verbindlichkeit: Das Sicherheitskonzept sieht Protokolle vor, die Veränderungen an den Datenhaltungen der Sicherheitsarchitektur, wie zum Beispiel für Zugriffsrichtlinien, festhalten. Ebenfalls wird festgehalten, welche Anfragen welche Datenströme verarbeiten. Protokolliert werden Zeiten, beteiligte Subjekte und Objekte. Anhand der Protokolle, dass manipulationssicher vorgehalten wird, kann eine verbindliche Zuordnung von Zugriffen in NexusDS vorgenommen werden.

Anonymisierung: Die Secure-Source kann Filter auf die Datenelemente der eingebetteten Quelle anwenden. Filter können frei implementiert werden und so eine feingranulare Anonymisierung der Informationen vornehmen. Besondere Fähigkeit der feingranularen Anonymisierung ist die Möglichkeit neben verschlüsselten Datenströmen, unverschlüsselte Datenströme mit verschleierter Information in das ungeschützte NexusDS zurückzuführen. Der unverschlüsselte Datenstrom kann dann in *Sicherheitszone-Null* frei verwendet werden. Ein öffentlicher Datenstrom wird nur dann zugelassen, wenn die Zugriffsrichtlinien diesen explizit definieren.

9.2 Ausblick

Die Verwaltung des Rollenmodells und die Zugriffskontrolle welche Subjekte Zugriffsrichtlinien für Operatoren und Rechenknoten vergeben dürfen ist vereinfacht realisiert worden. Mögliche

Erweiterungen könnten die Vergabe gezielter Steuern, um den Verwaltungsaufwand zu reduzieren. Filter und Evaluatoren sind bezüglich der Zugriffskontrolle vollständig ausgeschlossen worden. Eine Berücksichtigung erfordert eine Verfeinerung der kontrollierten Anfrageplanung, sodass auch diesbezüglich eine Zugriffskontrolle eingefordert wird. Zu beachten ist, dass eine Zugriffskontrolle, ob Subjekte in Zugriffsrichtlinien definierte Evaluatoren und Filter einsetzen dürfen, die Auswertung erheblich aufwendiger ausfallen lässt. Das impliziert eine effizientere Auswertung der Zugriffsrichtlinien gegenüber dem vorgestellten Modell der kontrollierten Anfrageplanung. Die Authentifizierung zwischen Services findet bisher nur über die Prüfung von digitalen Zertifikaten auf deren Gültigkeit statt. Eine Einführung einer feineren Zugriffskontrolle würde die Anbindung externer Werkzeuge erleichtern. So ist es bisher nur möglich, einem Werkzeug den vollen Zugriff auf die Sicherheitsarchitektur zu gewähren. Eine Verfeinerung zur gezielten Steuerung des Zugriffes würde zu einer Erhöhung der Sicherheit führen, da das Werkzeug nicht mehr potentiell auf den gesamten Datenbestand zugreifen könnte, sondern nur noch auf ausgewählte Teile. Daraus folgt gleichzeitig ein breites Spektrum an Werkzeugen, da je nach Zuverlässigkeit des Werkzeugs die Menge der verfügbaren Daten beschränkt werden könnte. Gleiches gilt für Evaluatoren und Filter, die ebenfalls nur über digitale Zertifikate angebunden werden könne.

Eine zukünftige Erweiterung, die die Flexibilität der Sicherheitsarchitektur maßgeblich erhöhen würde, ist die dynamische Umgestaltung von Anfragen zur Ausführungszeit. Jedoch ist bisher die Plattform NexusDS nicht in der Lage, Anfragen zur Laufzeit unterbrechungsfrei zu verändern. Ist dies in Zukunft möglich, könnten zur Laufzeit im Fall von veränderten Zugriffsrichtlinien Filter dynamisch aus den Datenströmen entfernt und eingefügt werden. Gleiches gilt für Zugriffsrichtlinien, die die Ausführung von Operatoren auf Rechenknoten betreffen oder deren Zertifizierung. Auch hierbei könnte eine dynamische Umplanung zur Laufzeit die Möglichkeit eröffnen, Anfrage unterbrechungsfrei an veränderte Zugriffsrichtlinien anzupassen.

Eine Abfolge von Filtern, die sich aus den für einen Operator relevanten Zugriffsrichtlinien ergibt, orientiert sich nur an einer einfachen Rangfolge. Zukünftige Erweiterungen könnten eine Beschreibung der Semantik der jeweiligen Filtertransformation vorsehen und eine durch Werkzeuge und algorithmisch gestützte Optimierung von Filterketten vorsehen. Damit würde die Spezifikation von Zugriffsrichtlinien bezüglich der notwendigen Filter erleichtert und gegebenenfalls Optimierungen möglich, sodass nicht notwendige Filteroperationen eingespart werden.

Die Sicherheitsarchitektur berücksichtigt nicht, wie die Verarbeitung von Datenelementen nach den Senken einer Anfrage weitergeht. Zwar wird eine Verschlüsselung innerhalb der Anfragen, über Plattform-Senken zu Plattform-Quellen aufrechterhalten, es besteht aber keine Kontrolle über den Zugriff, nachdem die Datenelemente einer Senke zugestellt wurden. Hierzu könnten zukünftige Arbeiten zum Beispiel Markierungen definieren, die nachfolgende Anfragen darüber in Kenntnis setzen, von welchen Quellen ein Datenelement abstammt und von welchen Operationen und Filtern Transformationen durchgeführt wurden. Einen Einstieg findet der Interessierte Leser in [37].

Literaturverzeichnis

[1] Abadi, Daniel J. ; Carney, Don ; Çetintemel, Ugur ; Cherniack, Mitch ; Convey, Christian ; Lee, Sangdon ; Stonebraker, Michael ; Tatbul, Nesime ; Zdonik, Stan: Aurora: a new model and architecture for data stream management. In: *The VLDB Journal* 12 (2003), August, S. 120–139. – ISSN 1066–8888 (Zitiert auf den Seiten 9, 25, 32 und 35)

[2] Anderson, Ross J.: *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2. Wiley Publishing, 2008. – ISBN 9780470068526 (Zitiert auf den Seiten 14, 15 und 16)

[3] Arasu, A. ; Babcock, B. ; Babu, S. ; Cieslewicz, J. ; Datar, M. ; Ito, K. ; Motwani, R. ; Srivastava, U. ; Widom, J.: Stream: The stanford data stream management systems. In: *a book on data stream management edited by Garofalakis, Gehrke, and Rastogi* (2004) (Zitiert auf Seite 32)

[4] Ayewah, N. ; Hovemeyer, D. ; Morgenthaler, J.D. ; Penix, J. ; Pugh, W.: Using Static Analysis to Find Bugs. In: *Software, IEEE* 25 (2008), Nr. 5, S. 22 –29. http://dx.doi.org/10.1109/MS.2008.130. – DOI 10.1109/MS.2008.130. – ISSN 0740–7459 (Zitiert auf Seite 18)

[5] Bauer, Martin ; Dürr, Frank ; Geiger, Jan ; Grossmann, Matthias ; Hönle, Nicola ; Joswig, Jean ; Nicklas, Daniela ; Schwarz, Thomas: Information Management and Exchange in the Nexus Platform / Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Germany. Version: Juli 2004. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2004-04&engl=0. Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Verteilte Systeme; Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Anwendersoftware, Juli 2004 (2004/04). – Technischer Bericht Informatik. – 58 S. (Zitiert auf den Seiten 9 und 24)

[6] Bedner, Mark ; Ackermann, Tobias: Schutzziele der IT-Sicherheit. In: *Datenschutz und Datensicherheit (DuD)* 33 (2010), Mai, Nr. 5, S. 323–328 (Zitiert auf Seite 15)

[7] Brandeis University, Brown U. ; MIT: *Borealis - Distributed Stream Processing Engine*. http://www.cs.brown.edu/research/borealis/public/ (Zitiert auf den Seiten 25 und 32)

[8] Bundesamt für Sicherheit in der Informationstechnik: *BSI-Standard 100-2 IT-Grundschutz-Vorgehensweise*. Bd. 2.0. Bundesamt für Sicherheit in der Informationstechnik, 2008 https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/ITGrundschutzstandards/standard_1002.pdf (Zitiert auf den Seiten 18 und 21)

[9] Bundesamt für Sicherheit in der Informationstechnik: *IT-Grundschutz-Kataloge*. Bd. 11. Bundesamt für Sicherheit in der Informationstechnik, 2009 http://www.bsi.bund.de/grundschutz (Zitiert auf Seite 16)

[10] Cao, Jianneng ; Carminati, B. ; Ferrari, E. ; Tan, Kian-Lee: ACStream: Enforcing Access Control over Data Streams. In: *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*, 2009. – ISSN 1084–4627, S. 1495 –1498 (Zitiert auf den Seiten 35 und 37)

[11] Carminati, Barbara ; Ferrari, Elena ; Tan, Kian: Specifying Access Control Policies on Data Streams. In: Kotagiri, Ramamohanarao (Hrsg.) ; Krishna, P. (Hrsg.) ; Mohania, Mukesh (Hrsg.) ; Nantajeewarawat, Ekawit (Hrsg.): *Advances in Databases: Concepts, Systems and Applications* Bd. 4443. Springer Berlin / Heidelberg, 2007, S. 410–421 (Zitiert auf den Seiten 10, 35 und 39)

[12] Carminati, Barbara ; Ferrari, Elena ; Tan, Kian L.: Enforcing access control over data streams. In: *Proceedings of the 12th ACM symposium on Access control models and technologies*. New York, NY, USA : ACM, 2007 (SACMAT '07). – ISBN 978–1–59593–745–2, 21–30 (Zitiert auf den Seiten 35 und 36)

[13] Cipriani, Nazario ; Eissele, Mike ; Brodt, Andreas ; Grossmann, Matthias ; Mitschang, Bernhard: NexusDS: a flexible and extensible middleware for distributed stream processing. In: *Proceedings of the 2009 International Database Engineering; Applications Symposium*. New York, NY, USA : ACM, 2009 (IDEAS '09). – ISBN 978–1–60558–402–7, 152–161 (Zitiert auf den Seiten 9 und 26)

[14] Cipriani, Nazario ; Nicklas, Daniela ; Grossmann, Matthias ; Hönle, Nicola ; Lübbe, Carlos ; Mitschang, Bernhard: Verteilte Datenstromverarbeitung von Sensordaten. In: *Datenbank-Spektrum* 9 (2009), Februar, Nr. 28, 37–43. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2009-08&engl= (Zitiert auf Seite 9)

[15] Department of Defence Standard: *Trusted Computer System Evaluation Criteria, DoD 5200.28-STD*. http://csrc.nist.gov/publications/history/dod85.pdf. Version: August 1982 (Zitiert auf den Seiten 18, 19 und 20)

[16] Eckert, Claudia: *IT-Sicherheit*. Bd. 6. Auflage. München [u.a.] : Oldenbourg, 2009. – ISBN 978–3–486–58999–3 (Zitiert auf den Seiten 14, 15, 16, 41, 56, 60, 85 und 89)

[17] Englbrecht, Michael: *Entwicklung sicherer Software*. Heidelberg [u.a.] : Spektrum Akad. Verl, 2004. – ISBN 3–8274–1432–6 (Zitiert auf den Seiten 14 und 91)

[18] Eschweiler, Jörg ; Psille, Daniel E. A.: *Security@Work: Pragmatische Konzeption und Implementierung von IT-Sicherheit mit Lösungsbeispielen auf Open-Source-Basis (X.systems.press)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 3540220283 (Zitiert auf Seite 15)

[19] Europäischen Kommission: *Information Technology Security Evaluation Criteria*. https://www.bsi.bund.de/cae/servlet/contentblob/471346/publicationFile/30220/itsec-en_pdf.pdf. Version: 1991 (Zitiert auf Seite 18)

[20] Ferraiolo, David ; Kuhn, Richard: Role-Based Access Control. In: *In 15th NIST-NCSC National Computer Security Conference*, 1992, S. 554–563 (Zitiert auf den Seiten 20, 85 und 86)

[21] Intel: *Intel Parallel Studio 2011*. http://software.intel.com/sites/products/collateral/studio/Intel_Parallel_Studio_Brief_081610_HighRes.pdf (Zitiert auf Seite 18)

[22] Kemper, Alfons ; Eickler, André: *Datenbanksysteme - Eine Einführung, 6. Auflage*. Oldenbourg, 2006. – ISBN 3–486–57690–9 (Zitiert auf den Seiten 33, 34 und 83)

[23] Lindner, Wolfgang ; Meier, Jorg: Securing the Borealis Data Stream Engine. In: *IDEAS '06: Proceedings of the 10th International Database Engineering and Applications Symposium*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2577–6, S. 137–147 (Zitiert auf den Seiten 10 und 34)

[24] Lindner, Wolfgang ; Meier, Jörg: Towards a secure data stream management system. In: *in TEAA 2005*, 2005, S. 114–128 (Zitiert auf den Seiten 34 und 39)

[25] Lucke, Dominik ; Constantinescu, Carmen ; Westkämper, Engelbert: Smart Factory - A Step towards the Next Generation of Manufacturing. Version: 2008. http://dx.doi.org/10.1007/978-1-84800-267-8_23. In: Mitsuishi, Mamoru (Hrsg.) ; Ueda, Kanji (Hrsg.) ; Kimura, Fumihiko (Hrsg.): *Manufacturing Systems and Technologies for the New Frontier*. Springer London, 2008. – ISBN 978–1–84800–267–8, 115-118 (Zitiert auf Seite 44)

[26] Ludewig, Jochen ; Lichter, Horst: *Software Engineering*. 1. Aufl. Heidelberg : dpunkt, 2007 http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+381003574&sourceid=fbw_bibsonomy. – ISBN 978–3–89864–268–2 (Zitiert auf Seite 20)

[27] Mahlmann, Peter ; Schindelhauer, Christian: *Peer-to-Peer-Netzwerke*. Springer Berlin, 2007 (Zitiert auf Seite 26)

[28] Microsoft: *Access Control Lists*. http://msdn.microsoft.com/en-us/library/aa374872(v=VS.85).aspx (Zitiert auf Seite 19)

[29] Microsoft: *Security Development Lifecycle*. http://www.microsoft.com/security/sdl. Version: November 2010 (Zitiert auf Seite 18)

[30] Nazario Cipriani, Carlos L.: *Ausnutzung von Restriktionen zur Verbesserung des Deployment-Vorgangs des Verteilten Datenstromverarbeitungssystems NexusDS*. 2007 (Zitiert auf Seite 9)

[31] Nehme, Rimma V. ; Lim, Hyo-Sang ; Bertino, Elisa ; Rundensteiner, Elke A.: StreamShield: a stream-centric approach towards security and privacy in data stream environments. In: *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 2009. – ISBN 978–1–60558–551–2, S. 1027–1030 (Zitiert auf Seite 10)

[32] Nehme, R.V. ; Lim, Hyo-Sang ; Bertino, E.: FENCE: Continuous access control enforcement in dynamic data stream environments. In: *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010, S. 940 –943 (Zitiert auf den Seiten 37, 38 und 39)

[33] Nehme, R.V. ; Rundensteiner, E.A. ; Bertino, E.: A Security Punctuation Framework for Enforcing Access Control on Streaming Data. In: *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, 2008, S. 406 –415 (Zitiert auf den Seiten 37, 38 und 39)

[34] Saltzer, J.H. ; Schroeder, M.D.: The protection of information in computer systems. In: *Proceedings of the IEEE* 63 (1975), Nr. 9, S. 1278 – 1308. http://dx.doi.org/10.1109/PROC.1975.9939. – DOI 10.1109/PROC.1975.9939. – ISSN 0018–9219 (Zitiert auf den Seiten 16 und 18)

[35] Sardina, Daniel G.: *Framework for Distributed Data Processing*, Universität Stuttgart, Diplomarbeit, 2008 (Zitiert auf Seite 104)

[36] Scherbaum, Andreas: *PostgreSQL - Datenbankpraxis für Anwender, Administratoren und Entwickler*. Open Source Press, 2009. – 518 S. (Zitiert auf Seite 34)

[37] Simmhan, Yogesh L. ; Plale, Beth ; Gannon, Dennis: A survey of data provenance in e-science. In: *SIGMOD Rec.* 34 (2005), September, S. 31–36. – ISSN 0163–5808 (Zitiert auf Seite 112)

[38] Stegmaier, Bernhard ; Kuntschke, Richard ; Kemper, Alfons: StreamGlobe: Adaptive query processing and optimization in streaming P2P environments. In: *In Proc. of the Intl. Workshop on Data Management for Sensor Networks*, 2004, S. 88–97 (Zitiert auf Seite 32)

[39] StreamBase Systems, Inc.: *StreamBase Event Processing Platform*. http://www.streambase.com (Zitiert auf Seite 35)

[40] TomTom: *TomTom Hintergrund - HDTraffic Manifest*. http://www.tomtom.com/landing_pages/trafficmanifesto/index-project.php?Lid=3. Version: Oktober 2010 (Zitiert auf Seite 24)

[41] Universtität Stuttgart: *Nexus Projekt Webseite*. http://www.nexus.uni-stuttgart.de (Zitiert auf Seite 9)

[42] Universtität Stuttgart: *XML-Schema Definitionen der Nexus Plattform*. http://www.nexus.uni-stuttgart.de/de/forschung/dokumente/ (Zitiert auf Seite 25)

[43] W3C: *Extensible Markup Language*. http://www.w3.org/XML/ (Zitiert auf Seite 25)

Alle URLs wurden zuletzt am 28.04.2011 geprüft.

Erklärung

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

(Oliver Dörler)