INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen

Thomas Binder

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Nils gentschen Felde
Vitalian Danciu

Abgabetermin: 18. Dezember 2007

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen

Thomas Binder

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Nils gentschen Felde
Vitalian Danciu

Abgabetermin: 18. Dezember 2007

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 18. Dezember 2007

. . . . . .

(Unterschrift des Kandidaten)

Grid Infrastrukturen werden in zunehmenenden Maße nicht nur in der Forschung, sondern auch in der Industrie eingesetzt. Sie dienen der verteilten Nutzung von unterschiedlichsten Ressourcen. Unternehmen, die zur Entwicklung und Produktion einzelner Güter zusammenarbeiten, gründen virtuelle Organisationen und verwenden Grid Technologien zur Koordination der geteilten Ressourcen. Daraus erwächst das Bedürfnis nicht nur die eigenen Komponenten zu schützen, sondern auch über die Sicherheit von Ressourcen der anderen beteiligten Unternehmen Bescheid zu wissen.

In dieser Arbeit wird die Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen untersucht und ein System konzipiert, das es einem Beauftragten einer Virtuellen Organisation ermöglichen soll die Sicherheit einzuschätzen. Dazu werden Analyseregeln zur Überwachung der Grid Middleware Globus Toolkit entwickelt. Darüberhinaus wird die Service Monitoring Architecture (SMONA) zur Konzentration sicherheitsrelevanter Nachrichten und deren Weiterverarbeitung erweitert. Dazu wird ein Adapter entwickelt der Protokolldateien von Intrusion Detection Systemen auslesen kann und deren Einträge er nach konfigurierbaren Datenschutzrichtlinien verändert, bevor er sie in der SMONA propagiert. Ein weiterer Adapter soll Logeinträge an Hand ihrer Zeitstempel analysieren und sich wiederholende Einträge, wie sie bei Brute Force oder DoS Angriffen auftreten, melden. Die Nachrichten der Adapter werden im Rich Event Composer gesammelt und sollen mit verschiedenen Funktionen weiterverarbeitet werden können und beim Eintreten bestimmter Bedingungen gemeldet werden. Zur Konfiguration der SMONA wird die Service Information Specification Language verwendet.

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Aufgabenstellung . . . 2
- 1.2 Abgrenzung von bestehenden Ansätzen . . . 2
- 1.3 Vorgehensweise . . . 3

2 Szenario und Anforderungsanalyse **5**
- 2.1 Das Meteorologen Grid . . . 5
- 2.2 Aspekte eines Systems zur Föderation von Intrusion Detection Systemen . . . 6
  - 2.2.1 Anforderungen an ein Intrusion Detection System in Grids . . . 6
  - 2.2.2 Die Zusammensetzung der IDS Architektur . . . 7
  - 2.2.3 Richtlinien zur Überwachung der Grid Middleware . . . 8
  - 2.2.4 Anforderungen an die Komponenten der Service Monitoring Architecture (SMONA) . 8

3 Grundlagen **12**
- 3.1 Grid Computing . . . 12
  - 3.1.1 Definitionen . . . 12
  - 3.1.2 Überblick über das Grid Computing . . . 13
  - 3.1.3 Die Open Grid Service Architecture . . . 13
  - 3.1.4 Sicherheitsmechanismen im Grid Computing . . . 15
  - 3.1.5 Monitoring im Grid Computing . . . 17
  - 3.1.6 Das Globus Toolkit als Grid Middleware . . . 18
- 3.2 Die Service Monitoring Architecture (SMONA) . . . 20
  - 3.2.1 Service Information Specification Language (SISL) . . . 21
- 3.3 Intrusion Detection Systeme . . . 22
  - 3.3.1 Definitionen . . . 22
  - 3.3.2 Komponenten eines IDS . . . 23
  - 3.3.3 Arten der Intrusion Detection Systeme . . . 24
  - 3.3.4 Einbruchserkennungsverfahren . . . 24
  - 3.3.5 Die Kommunikationsstandards IDMEF und IDXP . . . 26
  - 3.3.6 Beispiel Prelude . . . 27
- 3.4 Der CORBA Kommunikationsstandard . . . 29

4 Konzeption und Entwurf des Sicherheitssystems **31**
- 4.1 Konzeption des SMONA Frameworks . . . 32
  - 4.1.1 Kommunikation . . . 32
  - 4.1.2 Die SMONA Komponenten der höheren Schichten . . . 34
- 4.2 Konzeption der Adapter . . . 40
  - 4.2.1 Das SMONA Adapter Framework . . . 40
  - 4.2.2 Anpassungen der bisherigen Push und Pull Adapter . . . 41
  - 4.2.3 Der Message Manipulator Adapter . . . 42
  - 4.2.4 Der Log Adapter . . . 44
- 4.3 Föderation der Intrusion Detection Systeme zur Überwachung des Globus Toolkits . . . 46
  - 4.3.1 Sicherheitsanalyse des Globus Toolkits . . . 46
  - 4.3.2 Voraussetzungen der Prelude Architektur . . . 47
  - 4.3.3 Konzeption von Regeln zur Protokollanalyse . . . 47
  - 4.3.4 Auswahl von Dateien zur Integritätsüberwachung . . . 48
  - 4.3.5 Überwachung des Globus Toolkit Netzverkehrs . . . 49

5 Prototypische Implementierung **50**
5.1 Konfiguration der Umgebung . . . 50
5.1.1 Einstellungen zur verschlüsselten Kommunikation . . . 50
5.1.2 Der Namensdienst . . . 52
5.2 Die SMONA Komponenten . . . 52
5.2.1 Das SISL XML Schema und die SISL Objekte . . . 52
5.2.2 Die Management Anwendung . . . 54
5.2.3 Der Adapter Configurator . . . 55
5.2.4 Der Rich Event Composer . . . 58
5.2.5 Implementierung der Adapter . . . 61
5.3 Überwachung des Globus Toolkits mit drei exemplarischen IDS . . . 67
5.3.1 Protokollanalyse Regeln für den Prelude-LML . . . 67
5.3.2 Integritätsanalyse des Globus Toolkits mit Samhain . . . 70
5.3.3 Analyse des Globus Toolkit Netzverkehrs mit Snort . . . 71

6 Evaluation und Test des Sicherheitssystems **73**
6.1 Testaufbau . . . 73
6.1.1 Konfiguration des Globus Toolkits . . . 74
6.1.2 Konfiguration der Prelude Architektur . . . 75
6.1.3 Konfiguration der SMONA . . . 76
6.2 Test des Sicherheitssystems . . . 77
6.2.1 Einschleusen von Fehlmeldungen . . . 77
6.2.2 DoS Angriffe auf die SMONA Architektur . . . 78
6.2.3 Beenden eines Adapters als Betriebsstörung . . . 79
6.2.4 Anwendungsfall: Erkennung von SSH Wurm Angriffen . . . 79
6.3 Bewertung der Konzeption und ihrer Umsetzung . . . 80
6.3.1 Bemerkungen . . . 80
6.3.2 Überprüfung der Anforderungen . . . 82
6.3.3 Abschließende Bewertung . . . 82

7 Zusammenfassung und Ausblick **84**
7.1 Zusammenfassung . . . 84
7.1.1 Ergebnisse . . . 85
7.2 Ausblick . . . 86

A Installation und Konfiguration der Komponenten **87**
A.1 Das SMONA Framework . . . 87
A.2 Die Adapter . . . 87

B XML Schema der Service Information Specification Language **88**

C Die Schnittstellen der SMONA Komponenten **91**
C.1 Die Schnittstelle der Management Anwendung . . . 91
C.2 Die Schnittstelle des Rich Event Composers . . . 91
C.3 Die Schnittstelle des Adapter Configurators . . . 92

D Exemplarische Aggregation zur Erkennung von SSH Würmern und SSH Angriffen **93**

Abbildungsverzeichnis

1.1 Vorgehensweise in der Diplomarbeit 4

2.1 Grid Netz bestehend aus zwei Domänen 5

3.1 Überblick über die geschichtete Grid Infrastruktur nach [FKS+ 06] 14
3.2 Komponenten des Grid Sicherheitsmodels nach [FKS+ 06] S.48 16
3.3 R-GMA Modell nach [RGMA5 07] 17
3.4 Komponenten des Globus Toolkit 4.0 aus [GT4COMP 07] 18
3.5 Die Service Monitoring Architecture aus [DgFS 07] 21
3.6 IDMEF Datenmodell nach [DCF 07] 26
3.7 Prelude Architektur nach [PREARCH 07] 28

4.1 Föderation von IDS zur Erkennung von Angriffen 31
4.2 Die Kommunikation im Sicherheitssystem 33
4.3 Verkürztes Kollaborationsdiagramm für das Hinzufügen einer neuen Aggregation 34
4.4 Verkürztes Kollaborationsdiagramm über die Datenverarbeitung 36
4.5 Vereinfachtes Kollaborationsdiagramm des Message Manipulator Adapters 43
4.6 Algorithmus des Log Adapters 45

5.1 Klassendiagramm der Kommunikationsobjekte 53
5.2 Klassendiagramm der Management Anwendung 55
5.3 Die graphische Oberfläche der Management Anwendung 56
5.4 Klassendiagramm des Adapter Configurator 57
5.5 Klassendiagramm des Rich Event Composers 59
5.6 Verkürztes Klassendiagramm des Message Manipulator Adapters 64
5.7 Verkürztes Klassendiagramm des Log Adapters 66

6.1 Testaufbau des Globus Toolkits und der SMONA mit mehreren IDS 73
6.2 Die installierten Globus Toolkit Dienste 75
6.3 Verteilung und Kommunikation der IDS bis zum Rich Event Composer 76
6.4 Verteilung und Kommunikation der SMONA Komponenten 77

Tabellenverzeichnis

4.1 Methoden der Rich Event Composer Funktionsbibliothek 38

5.1 Veränderungen an den Klassen des SMONA Adapter Frameworks 62
5.2 Die wichtigsten Konstanten des Message Manipulator Adapters aus der Klasse Constants . . . 65
5.3 Die wichtigsten Konstanten des Log Adapters aus der Klasse Constants 67

6.1 Rechner des Testaufbaus, ihre Aufgaben und Anwendungen 74

1 Einführung

Im April 2007 meldete das Marktfoschungsunternehmen Experton, dass für das Jahr 2007 das Marktvolumen von Netzsicherheitskomponenten in Deutschland um 12,5 Prozent auf 3,94 Milliarden Euro steigen wird (siehe [EXPE 07]). 54,7 Prozent der befragten Unternehmen setzen bereits Intrusion Detection Systeme (IDS) oder Intrusion Prevention Systeme (IPS) ein.

Zu Beginn der Entwicklung von Einbruchserkennungssystemen standen Komponenten, die Rechner separat oder den Netzverkehr überwacht haben. Die beschränkte Funktionalität der IDS machte die Verwendung mehrerer Systeme notwendig und erschwerte die Beurteilung der Sicherheitslage eines Unternehmensnetzes. Auf Grund dieser Problematik gründete die Internet Engineering Task Force (IETF, siehe [IETF 07]) die Intrusion Detection Working Group (IDWG, siehe [IDWG 07]), in der Standards erarbeitet werden, die eine Zusammenarbeit unterschiedlicher Systeme gewährleisten sollen. Ein Entwurf der IDWG ist das Intrusion Detection Message Exchange Format (IDMEF, siehe [DCF 07]) welches ein einheitliches Alarmierungsformat für Einbruchserkennungssysteme beschreibt und Anwendungen wie Prelude (siehe [PREL 07]) die Möglichkeit bietet, die Meldungen unterschiedlicher IDS zu sammeln.

Nicht nur die Sicherheitstechnik in Netzen, sondern auch die Anwendungen die sie verbinden, wurden weiterentwickelt. Es entstand ein neues Konzept zur Computer- und Netznutzung, das Grid Computing, das im Jahr 2004 von IBM folgendermaßen beschrieben wurde:

„Ein Grid ist eine auf Standards basierende Anwendung, beziehungsweise eine Ressourcen verteilende Architektur, die es heterogenen Systemen und Anwendungen ermöglicht Rechen- und Speicherressourcen gemeinsam und transparent zu benutzen" (siehe [JBFT 05]).

Diese Idee, Ressourcen zu Bündeln und bereitzustellen, führt zu einer verletzlichen Infrastruktur, die für Angreifer eine Spielwiese und für Konkurrenten einen Angriffspunkt bietet. Die enorme Rechenleistung, die durch die Kooperation erreicht wird, verführt dazu in dieses System einzubrechen, es unautorisiert zu nutzen oder Daten zu stehlen. Weiterhin bestehen in Grids zentrale Einheiten, die von Konkurrenten mit DoS Angriffen ausgeschaltet werden können. Daraus ergibt sich die Notwendigkeit Grids bezüglich sicherheitsrelevanter Ereignisse zu überwachen.

Zwar beeinträchtigen verteilte und heterogenen Strukturen, wie sie im Grid Computing gegeben sind, moderne Einbruchserkennungs Systeme in ihrer Funktionsweise nur geringfügig. Eine große Problematik besteht aber in der bisherigen Nichtbeachtung der IDS gegenüber der Grid Middleware, so dass es bis heute kaum Sensoren und Signaturen gibt, die an die Überwachung der Grid Komponenten und ihrer Kommunikation angepasst sind.

Ein zweites Problemfeld für Intrusion Detection Systeme bereitet die Tatsache, dass Grids aus mehreren administrativen Domänen bestehen. Die Alarmmeldungen der IDS können persönliche Daten und Informationen über die Infrastruktur einer Domäne enthalten, die eine am Grid beteiligte Organisation nicht preisgeben will oder aus rechtlichen Gründen nicht weitergeben darf. Moderne IDS versenden ihre Alarmmeldungen ohne eine Funktion vorzusehen diese Nachrichten zu verändern. Bei organisationsübergreifenden Grids besteht aber die Notwendigkeit bestimmte Informationen der Mitteilungen ausblenden zu können.

Im Rahmen dieser Diplomarbeit wird ein System entwickelt, das die Grid Middleware Globus Toolkit mit unterschiedlichen Intrusion Detection Systemen auf Angriffe überwacht. Die Alarmnachrichten der IDS werden gesammelt und ihr Inhalt dabei verändert. Die Manipulationen sind durch die Verantwortlichen der Domänen selbst einzustellen, so dass lokale Datenschutz- und Sicherheitsrichtlinien berücksichtigt werden können. Weiterhin ermöglicht das System die sicherheitsrelevanten Meldungen in einfachen, konfigurierbaren Funktionen nachzubearbeiten und unter bestimmten Umständen auf einer graphischen Oberfläche auszugegeben.

1.1 Aufgabenstellung

Heutige Intrusion Detection Systeme wie Prelude sind nicht geeignet domänenübergreifende Grids zu überwachen. Sie können die Alarmmeldungen mehrerer IDS aus einem Netz registrieren, auf dem Bildschirm ausgeben und an andere IDS weiter versenden. Grids und andere verteilte Systeme werden bisher nur eingeschränkt unterstützt.

Ziel dieser Diplomarbeit ist die Konzeption und prototypische Implementierung eines Systems zur Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen. Das System muss sich in die bereits bestehende SMONA Architektur (siehe Kapitel 3.2) der Lehr- und Forschungseinheit für Kommunikationssysteme und Systemprogrammierung der Ludwig-Maximilians-Universität eingliedern und die Funktionalität heutiger IDS um die Unterstützung domänenübergreifender Grids erweitern. Zusätzlich soll das SMONA Adapter Framework verwendet und etwaig angepasst werden. Daraus ergeben sich mehrere Teilaufgaben:

Zum einen müssen IDS an die Überwachung von Grid Middleware angepasst werden. Exemplarisch für diese Aufgabe werden drei Systeme ausgewählt, die das Globus Toolkit überwachen sollen. Es müssen Regeln für die Protokollanalyseeinheit Prelude-LML entwickelt werden, die die Logdateien des Toolkits auf Angriffe hin untersuchen und sicherheitsrelevante Meldungen identifizieren. Weiterhin müssen wichtige Dateien der Middleware dokumentiert und durch das Host IDS Samhain auf Veränderungen überprüft werden. Zur Überwachung des Globus Netzverkehrs werden schließlich für das Netz IDS Snort Analyseregeln entworfen und realisiert.

Die zweite Aufgabe besteht in der Anpassung der Service Monitoring Architecture (SMONA) zur Überwachung sicherheitsrelevanter Ereignisse. Dazu müssen Adapter gefunden und gegebenenfalls konzipiert werden, die für diese Aufgabe von Bedeutung sind und Meldungen über Angriffe oder deren Auswirkungen generieren oder sammeln und in der Architektur publizieren. Diese Adapter sollen nicht die verwendeten Intrusion Detection Systeme ersetzen, sondern ihre Funktionalität erweitern. Besonderes Augenmerk ist auf einen Adapter zu legen, der den Inhalt von Alarmmeldungen zum Zweck des Datenschutzes verändern kann. Zusätzlich müssen Komponenten der SMONA identifiziert und realisiert werden, die Adapter konfigurieren und Nachrichten entgegennehmen, verarbeiten und ausgeben. Außerdem müssen neue Funktionen entworfen werden, die der Überwachung von Nachrichten auf sicherheitsbezogene Ereignisse dienen. Gegebenenfalls muss der Funktionsumfang des Rich Event Composers, einer Komponente der SMONA Architektur, der bisher aus hauptsächlich mathematischen Funktionen besteht um neue Methoden erweitert werden.

Eine prototypische Implementierung des gesamten Systems zu Testzwecken stellt die letzte Hauptaufgabe dar. Dazu müssen die ausgewählten Komponenten der SMONA Architektur realisiert werden. In einer Analyse des Systems soll die Konzeption und seine Umsetzung in mehreren Tests und einem speziellen Anwendungsfall evaluiert werden. Zur Prüfung der Architektur soll eine Testumgebung mit den oben genannten IDS, dem Prelude System, den SMONA Komponenten und den konzipierten Adaptern in einem Globus Toolkit Grid installiert und getestet werden.

1.2 Abgrenzung von bestehenden Ansätzen

Das Thema dieser Arbeit ist die Entwicklung eines Systems zur Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen. Es gibt bereits zwei Ansätze, die aber beide für diese Aufgabenstellung unzulänglich sind.

Das Grid-based Intrusion Detection System (siehe [ChSa 03]) von Ong Tian Choon und Azman Samsudin ist ein IDS Framework, das einen Grid Dienst realisiert, der Daten von Agenten auf Sicherheitsangriffe untersucht. Das System besteht aus vier Komponenten: den GIDS Agenten, dem GIDS Server, dem GIDS Manager und dem GIDS Communicator. Die Agenten laufen auf den zu überwachenden Rechner als Hintergrunddienste und senden ihre gesammelten, komprimierten und formatierten Informationen über Systemressourcen, Log-, Globus- und Systemdateien zur Analyse mittels eines Moduls des Secure Communicators an einen zentralen GIDS Server. Darüberhinaus enthält jeder Agent eine Response Engine, die Funktionen zur Abwehr von Angriffen enthält. Der GIDS Server analysiert die empfangenen Daten auf Einbruchsversuche und speichert sie

in einer Datenbank. Er bietet den GIDS Dienst für das Grid an. Der GIDS Manager verwaltet globale und VO bezogene Richtlinien des GIDS Dienstes. Mit ihm werden alle GIDS Server über eine graphische Oberfläche gesteuert. Ein Problem des Systems besteht in der fehlenden Überwachung des Grid Netzverkehrs. Außerdem wird die Konfiguration der Agenten und ob sie ihre Informationen manipulieren oder filtern können, nicht beschrieben.

Das Cooperative Anomaly and Intrusion Detection System (CAIDS, siehe [HKS+ 06]) ist die Kombination eines Intrusion Detection Systems mit einem Anomaly Detection System (ADS). Der Ansatz von Hwang et al. verwendet einen Paketanalysator (Snort) der zwei Regelsätze zur Analyse von Netzdaten benutzt. Ein Regelsatz enthält Signaturen für bereits bekannte Angriffe. Der zweite beinhaltet vom ADS neu erstellte Signaturen. Entdeckte Angriffe werden von Snort direkt an ein Intrusion Response System weitergeleitet. Außerdem generiert er Alarmnachrichten für den restlichen Netzverkehr, die anschließend von einem Anomaly Detection System auf Unregelmäßigkeiten überprüft werden. Das ADS erzeugt daraus Berichte über gefundene Einbrüche und erstellt zusätzlich neue Snort Signaturen. In der CAIDS Architektur sind Snort IDS im Grid verteilt und lösen viele kleine Alarme aus, die im Anomaly Detection System zuerst lokal zusammengefasst, formatiert und klassifiziert werden, bevor sie von einem Distributed Hash Table (DHT) Modul zur weiteren Analyse an eine zentrale Stelle verschickt werden. In einem zentralen Korrelationsmodul werden sie schließlich zu Angriffsberichten zusammengefasst. Die Analyse von Logdateien und der Integrität von Systemdateien wird nicht durchgeführt. Außerdem ist auch hier nicht bekannt, ob der Datenschutz im Entwurf berücksichtigt wurde.

1.3 Vorgehensweise

Das Vorgehen dieser Arbeit gliedert sich analog zu Abbildung 1.1 und kann in drei Phasen beschrieben werden:

In der Konzeptionsphase wird ein Grid Szenario in Anlehnung an real existierende Implementierungen skizziert und daran rudimentäre Bedrohungen von Grids erarbeitet. Aus den Bedrohungen und Grid spezifischen Eigenschaften entstehen Anforderungen an eine IDS Architektur und die verwendete SMONA. Die Intrusion Detection Systeme dienen der Überwachung des Grids auf sicherheitsrelevante Ereignisse während die SMONA Architektur die Föderation und Nachbearbeitung der Nachrichten zur Aufgabe hat. Es werden weiterhin spezielle Anforderungen an die Adapter zur Informationsbeschaffung, an die SMONA Komponenten und ihre Funktionalität und an die Globus Toolkit Analyseregeln entwickelt.

Zur Bewältigung der ersten Hauptaufgabe müssen Eigenschaften der Adapter definiert werden, die für die Föderation von Intrusion Detection Systemen in domänenübergreifenden Grids notwendig sind. An Hand dieser Anforderungen wird das bestehende SMONA Adapter Framework untersucht und der fehlende Funktionsumfang konzipiert.

Die SMONA Architektur ist an das Überwachen von Systemen und ihren Dienstattributen angepasst und umfasst eine größere Funktionalität, als sie in dieser Arbeit benötigt wird. Eine Analyse identifiziert Funktionalitäten und Komponenten, die zur Föderation von Intrusion Detection Systemen wesentlich sind. Auf die Auswahl der Komponenten, die die geforderten Funktionen bieten, folgt deren Entwurf. Außerdem werden neue Methoden entwickelt, die der Verarbeitung sicherheitsrelevanter Nachrichten dienen.

Zur Erfüllung der letzten Aufgabe werden drei IDS (Prelude-LML, Samhain und Snort) ausgewählt und Regeln zur umfangreichen Überwachung des Globus Toolkits entwickelt. Zuerst werden allgemeine Anforderungen zur Analyse des Toolkits erstellt. Anschließend werden Eigenschaften zur speziellen Analyse der Grid Middleware konzipiert.

Die Implementierungsphase beinhaltet die prototypische Realisierung der Teilaufgaben und deren Komposition zu einem föderativen Gesamtsystem. Die SMONA Komponenten und die Adapter werden in der Programmiersprache Java implementiert und für deren Konfiguration das bestehende SISL XML Schema erweitert. Die Regelsätze zur Analyse der Logeinträge, des Netzverkehrs und der Systemdateien sind reguläre Ausdrücke in der Syntax der ausgewählten Intrusion Detection Systeme.

In der letzten Phase wird das System mit fingierten Angriffen auf seine Stabilität und Performanz getestet. Außerdem wird die Architektur an Hand eines speziellen Anwendungsfalls konfiguriert und anschließend überprüft. In einer abschließenden Evaluation wird die Konzeption und Umsetzung bewertet.

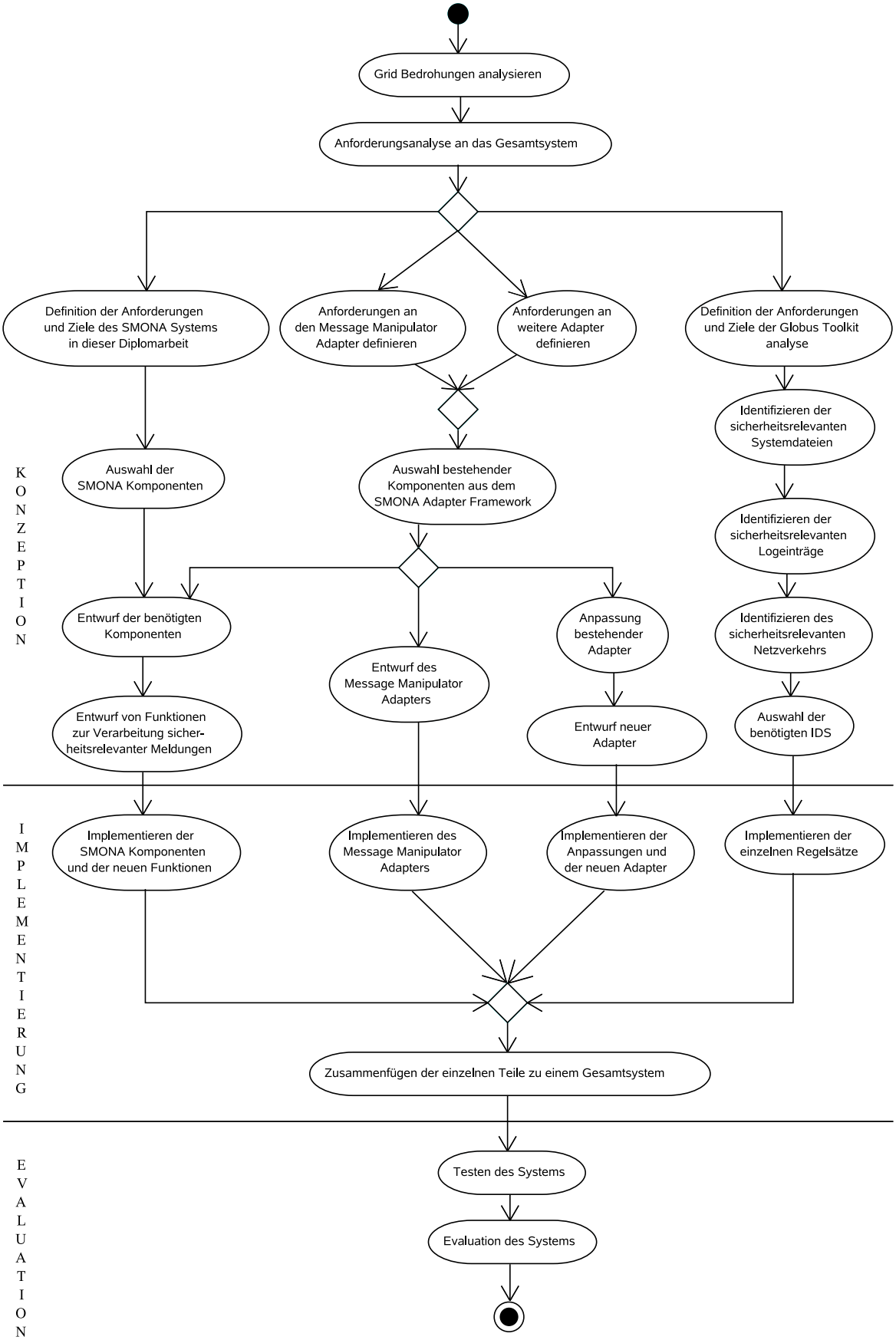

Abbildung 1.1: Vorgehensweise in der Diplomarbeit

2 Szenario und Anforderungsanalyse

Dieses Kapitel beschreibt ein einfaches Grid Szenario, das für diese Arbeit als Veranschaulichung dient. Anhand dieses Beispiels werden Anforderungen an das zu konzipierende Sicherheitssystem und die Föderation von Intrusion Detection Systemen in Grids erarbeitet. In Abschnitt 2.2 werden Anforderungen an die Teilkomponenten des Systems dargelegt. Die geforderten Eigenschaften dienen der Auswahl bestehender IDS und fließen in die Konzeption und Implementierung des Sicherheitssystems ein.

2.1 Das Meteorologen Grid

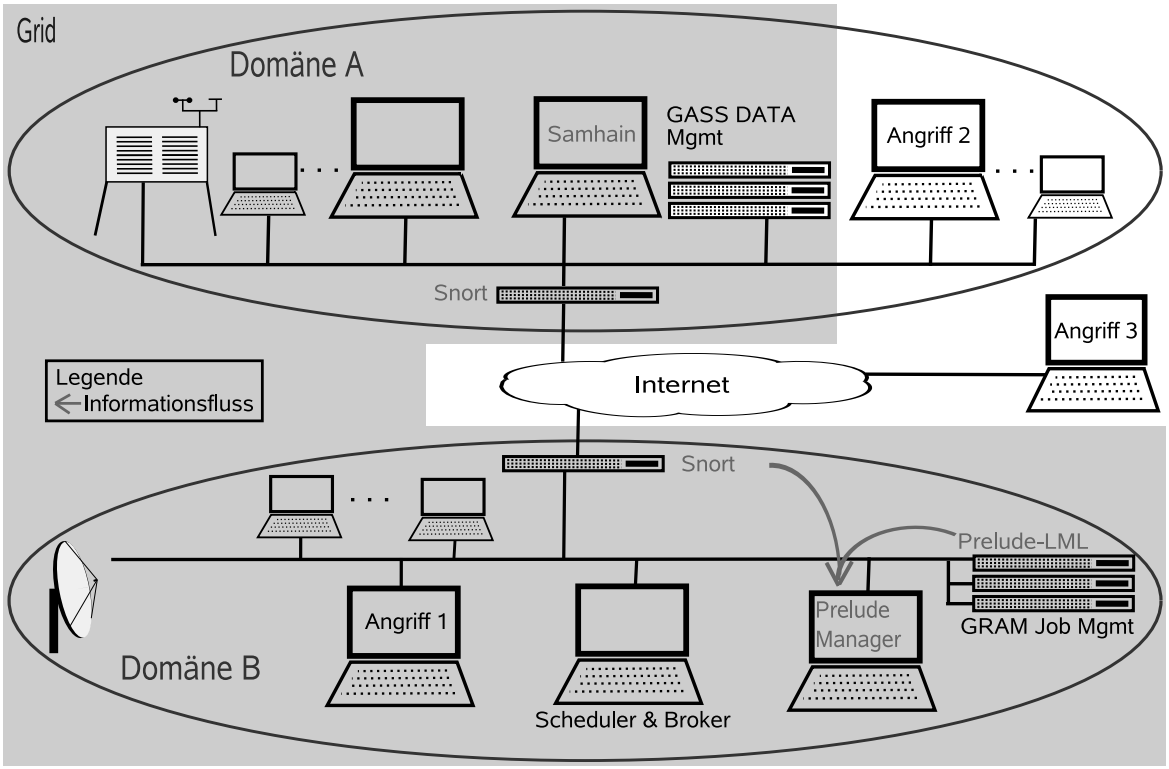

Abbildung 2.1: Grid Netz bestehend aus zwei Domänen

In Abbildung 2.1 ist ein exemplarisches Grid dargestellt, das die Wetterbeobachtung und -vorhersage zur Aufgabe hat. Seine Struktur ist in Anlehnung an real existierende Grids aufgebaut. Es besteht aus zwei administrativen Domänen, A und B. In der Domäne A sind mehrere Rechner (symbolisiert durch drei Punkte zwischen den Rechnern), eine Wetterstation, und ein Rechencluster über ein Firewall Gateway mit dem Internet verbunden. In der zweiten Domäne befinden sich mehrere Computer, mehrere Dateiserver und eine Parabolantenne, die mittels eines Firewall Gateway ebenfalls Zugang zum Internet haben. In diesem Szenario nehmen alle Rechner am Grid teil, die sich innerhalb der grauen Fläche befinden. Am rechten Rand befindet sich ein Angreifer (Angriff 3), der Zugriff auf das Internet hat.

Benutzer dieses Grids können mittels der Rechner auf die Daten der Sensoren zugreifen, rechenintensive Prozesse im Rechencluster ausführen und Daten bei den Dateiservern abspeichern. Eine Grid Middleware

bietet Dienste an, koordiniert die Ressourcen und sichert die Kommunikation zwischen den Komponenten.

Bedrohungen für das exemplarische Grid gehen von drei verschiedenen Orten aus. Ein böswilliger Mitarbeiter einer beteiligten Organisation kann das Grid als berechtigter Nutzer missbrauchen (Angriff 1, Domäne B). Eine weitere Angriffsmöglichkeit besteht von Rechnern, die sich in der selben Domäne befinden, wie die Grid Komponenten (Angriff 2, Domäne A), aber keine Zugriffsrechte für das Grid besitzen. Nachdem organisationsübergreifende Grids normalerweise über das Internet kommunizieren, sind sie auch von außen angreifbar (Angriff 3). Von allen drei Orten aus kann versucht werden mittels Denial of Service (DoS) Attacken die Grid Dienste zu stören. Mit Hilfe von Würmern oder Trojanern kann ein Angreifer versuchen, Zugriff auf die Daten und Ressourcen des Grids zu erlangen und diese anschließend auszunutzen, beziehungsweise zu stehlen. Eine genauere Bedrohungsanalyse ist im Abschnitt 4.3.1 beschrieben.

In diesem Szenario sind Verantwortliche einer Domäne für die Sicherheit in ihrem Netz und für die darin enthaltenen Komponenten, die am Grid beteiligten sind, selbst zuständig. In der Domäne A werden zwei Intrusion Detection Systeme (Samhain und Snort) verwendet, die nicht miteinander kommunizieren. Domäne B wird ebenfalls von zwei IDS (Snort und Prelude-LML) überwacht, die ihre Alarmmeldungen an eine Managementstation (Prelude Manager) weiterschicken. Durch diese Struktur ist die Sicherheitslage des Grids nur bedingt einsehbar. Die zuständigen Administratoren müssen sich gegenseitig über Angriffe auf die Grid Komponenten in ihrer Domäne informieren. Eine zentrale Überwachung des Grids auf sicherheitsrelevante Ereignisse, ähnlich einer Grid Managementsoftware, ist derzeit noch nicht implementiert. Sie könnte die Verantwortlichen des Grids zeitnah informieren und Grundlagen für schnellere und koordinierte Gegenmaßnahmen schaffen. Derzeit können Intrusion Detection Systeme Alarmmeldungen filtern und weiterleiten. Die Implementierung eines solchen Systems für ein gesamtes Grid kann zu Datenschutzverletzungen führen und die Souveränität einer administrativen Domäne beeinträchtigen. Deshalb muss die Filterfunktion der IDS um die Möglichkeit der Datenmanipulation erweitert werden.

Der nächste Abschnitt thematisiert Anforderung an ein System zur Erkennung von Angriffen auf Grid Infrastrukturen.

2.2 Aspekte eines Systems zur Föderation von Intrusion Detection Systemen

Das zu entwerfende System besteht aus diversen Teilkomponenten, die zur Bewältigung der Aufgabenstellung bestimmten Richtlinien folgen müssen. Dieser Abschnitt erörtert die notwendigen Eigenschaften eines Systems zur Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen. Die einzelnen Unterpunkte gehen auf die verschiedenen Komponenten des Systems ein. Zuerst werden Ansprüche an die zu verwendenden IDS gestellt. Anschließend legt Abschnitt 2.2.2 allgemeine Anforderungen an die Infrastruktur und Kooperation der Systeme dar und es werden im Abschnitt 2.2.3 benötigte Eigenschaften der Analyseregeln der Intrusion Detection Systeme beschrieben. Schließlich werden Ansprüche an die zu entwickelnden Adapter und die Komponenten der höheren Schichten der SMONA Architektur erläutert (siehe Abschnitt 2.2.4).

2.2.1 Anforderungen an ein Intrusion Detection System in Grids

Dieser Abschnitt listet zuerst allgemeine Ansprüche an Intrusion Detection Systeme auf. Anschließend erfolgt eine Erläuterung von Grid spezifischen Anforderungen.

In [BSI 02] und [Spen 05] werden konkrete Ansprüche an ein Intrusion Detection System aufgelistet und erläutert. Eine Zusammenfassung der wichtigsten Eigenschaften bietet die folgende Auflistung:

- leichte Integrierbarkeit in IT-Systeme
- einfache Konfigurierbarkeit und Wartung
- autonome und fehlertolerante Arbeitsweise
- geringe Inanspruchnahme von Systemressourcen

- Erkennung von Abweichungen bezüglich des normalen Systemverhaltens bei Minimierung von Fehlalarmen und nicht erkannten Sicherheitsverletzungen
- inhärente Selbstschutzmechanismen
- Nachvollziehbarkeit der Angriffe

Neben den oben genannten Anforderungen gibt es bestimmte Eigenschaften, die bei der Föderation von Intrusion Detection Systemen in Grids von besonderer Bedeutung sind:

- Unterstützung der Grid Middleware
  Sicherheitsangriffe beginnen mit dem Abtasten und der Analyse der Software, die auf dem Zielsystem läuft. Ein Intrusion Detection System für Grids muss dieses Verhalten erkennen und die möglichen Angriffspunkte (zum Beispiel Ports) der zu schützenden Grid Middleware überwachen (entspricht Angriffen 2 und 3 aus Abbildung 2.1). Darüberhinaus muss dieses System den internen Missbrauch durch unberechtigte Nutzer durch die Überwachung der Middleware aufdecken (Angriff 1).
- Skalierbarkeit
  Grids sind in der Anzahl der teilnehmenden Ressourcen und Nutzer nicht beschränkt. Deshalb muss die Föderation der IDS skalierbar sein. Die Forderung nach geringer Inanspruchnahme von Systemressourcen gilt für alle Grids und darf nicht bei großen Systemen vernachlässigt werden. Unabhängig der Systemgröße soll die Mehrbelastung des Systems durch die Überwachung minimal sein.
- Kollaboration
  Grids sind große, verteilte Systeme, die eine Vielzahl unterschiedlicher Sensoren benötigen. Diese Sensoren sollten ihre Alarmmeldungen an andere IDS Komponenten weiterleiten können, wobei die Kommunikation verschlüsselt erfolgen muss. Bestenfalls ist der Nachrichtenaustausch nicht nur kodiert sondern derart getarnt, dass nicht festzustellen ist, ob ein Sensor einen Alarm, ein Lebenszeichen oder sonstige Informationen versendet. Aus einer derart verborgenen Zusammenarbeit, kann ein Angreifer (Angriff 2 und 3 in Abbildung 2.1) keine Informationen über das Erkennungssystem herausfinden und auch nicht feststellen, ob er bereits einen Alarm ausgelöst hat. Darüberhinaus soll das Kommunikationsformat standardisiert sein und IDMEF Nachrichten (siehe [DCF 07]) austauschen.
- Alarm Maskierung
  Aufgrund der unterschiedlichen Zuständigkeit für Grid Komponenten, die verschiedenen Organisationen angehören können, muss bei der Föderation der Intrusion Detection Systeme neben der Kollaboration mit anderen Systemen eine Möglichkeit bestehen, die zu versendenden Alarmnachrichten inhaltlich zu verändern. Gesetzes- oder Unternehmensrichtlinien, wie zum Beispiel der Datenschutz, kann die Weitergabe bestimmter Informationen in Alarmmeldungen verbieten. Deshalb müssen zum Beispiel Netzadressen, Benutzernamen und Versionsnummern ausblendbar sein. Die Aussagekraft des Alarms darf durch die Maskierung nicht verändert werden.

Nachdem in diesem Abschnitt Anforderungen an die Intrusion Detection Systeme gestellt wurden, beschreibt der nächste Unterabschnitt Ansprüche an ihre Infrastruktur und ihre Verteilung im Grid.

2.2.2 Die Zusammensetzung der IDS Architektur

Eine vollständige Überwachung aller Komponenten in verteilten Systemen ist von größter Bedeutung für die Sicherheit. Deshalb sollte jede Komponente durch ein Host IDS und jeder Knotenpunkt im Netz durch ein Network IDS kontrolliert werden (siehe Abschnitt 3.3.3, Arten der Intrusion Detection Systeme). Die einzelnen Intrusion Detection Systeme sollten zusammenarbeiten, das heißt, dass sie möglichst von zentraler Stelle aus verwaltet werden können und an diese Stelle ihre Funde berichten, da ansonsten der Verwaltungsaufwand enorme Ausmaße annimmt. Durch die zentrale Verwaltung darf die Souveränität der teilnehmenden Organisationen und ihr Datenschutz nicht beeinträchtigt werden. Wie in Abschnitt 2.2.1 beschrieben, muss die Kollaboration und Kommunikation auf gesicherten Verbindungen aufbauen.

Für eine lückenlose Sicherheitsanalyse müssen unterschiedliche Sensoren verwendet werden, die möglichst alle Erkennungsmechanismen (siehe Kapitel 3.3.4) einsetzen. Wie im Szenario beschrieben können Angriffe

nicht nur von außen durchgeführt werden, sondern auch von eigenen Mitarbeitern. Zu diesem Zweck sollten alle Komponenten durch Host IDS unter Verwendung der Protokoll-, Echtzeit- und Integritätsanalyse überwacht werden. Mit der Paketanalyse müssen darüberhinaus die Kommunikationswege kontrolliert werden.

Angriffe müssen möglichst früh erkannt werden, um Gegenmaßnahmen einleiten zu können. Deshalb sollten Ereignisse sofort am Ort ihres Auftretens analysiert werden. Weiterhin können Intrusion Prevention und Response Systeme (siehe [Spen 05] und Abschnitt 3.3.2) automatisch reagieren und Attacken abwehren.

IDS können die Sicherheit in einem System nicht garantieren. Deshalb sind zusätzliche Maßnahmen, wie Virenscanner, Firewalls, Mitarbeiter Schulungen und weitere Vorkehrungen essentiell. Weiterhin können mit Hilfe einer Monitoring Architektur zum Beispiel Leistungsdaten der Grid Dienste überwacht werden und bei ungewöhnlichem Verhalten einen Angriff andeuten.

Im folgenden Abschnitt werden gewünschte Eigenschaften für die Analyseregeln der Intrusion Detection Systeme erläutert.

2.2.3 Richtlinien zur Überwachung der Grid Middleware

In der Entwicklung von Analyseregeln für die Grid Middleware muss die spezielle Architektur berücksichtigt werden. Wie bereits im Abschnitt 2.2.2 erläutert benötigt die Einbruchserkennung in einem Grid unterschiedliche IDS, die verschiedene Verfahren (siehe Kapitel 3.3.4) einsetzen. Die allgemeinen Anforderungen an die Analyseregeln sind für alle Einbruchserkennungsmethoden ähnlich.

Die Regeln sollten alle bekannten und unbekannten Angriffe und Missbräuche erkennen können und dabei keinen Fehlalarm (false positiv) auslösen und keinen Angriff unentdeckt lassen (false negativ). Diese Anforderungen sind nicht zu erfüllen und in der Praxis gegeneinander abzuwiegen. Trotzdem müssen sie das Ziel in der Entwicklung sein. Sind die Analyseregeln an ein Angriffsmuster genau angepasst, werden Fehlalarme vermieden, aber es können leichte Variationen nicht mehr erkannt werden. Beschreiben die Analyseregeln einen Angriff nur in seinen Grundzügen, erhöht sich die Rate der Fehlalarme und es werden mehr Ereignisse als Angriffe gemeldet, die dem Normalverhalten entsprechen. Diese Regeln können aber leicht veränderte Angriffsmuster aufspüren.

Schließlich müssen die Analyseregeln an die verwendeten Dienste und Architekturen angepasst sein. Dabei sollten alte und bereits ausgebesserte Schwachstellen in Programmen ebenso beobachtet werden, wie neue Sicherheitslücken, die noch nicht im Programmcode repariert wurden. Darüberhinaus kann die Beobachtung von Schnittstellen (zum Beispiel TCP Ports), die nicht verwendet werden Aufschluss über Informationsgewinnungsversuche von Angreifern liefern. Wie bereits erläutert gibt es zur Zeit keine öffentlich zugänglichen Regelsätze zur Analyse des Globus Toolkits. Deshalb werden in dieser Arbeit drei IDS mit unterschiedlichen Erkennungsmethoden ausgewählt (siehe Kapitel 4.3) und zur Überwachung des Globus Toolkits konfiguriert.

Eine weitere Anforderung an die Analyseregeln betrifft ihre Syntax. Nicht alle IDS optimieren ihre Signaturen vor dem Einsatz auf Performanz. Deshalb sollten Analyseregeln, die Abbruchkriterien enthalten, diese mit dem Ereignis möglichst früh vergleichen. Regeln, die sich in mehreren Kriterien überschneiden, können eventuell für eine bessere Leistung zusammengefasst werden.

Der nächste Abschnitt beschäftigt sich mit der Föderation der Intrusion Detection Systeme mit Hilfe der SMONA Architektur. Zuerst werden Ansprüche an die Adapter dargelegt, die sicherheitsrelevante Nachrichten sammeln und zur Weiterverarbeitung und Darstellung an den Rich Event Composer schicken. Anschließend werden wichtige Eigenschaften der Verarbeitung der sicherheitsrelevanten Meldungen erläutert.

2.2.4 Anforderungen an die Komponenten der Service Monitoring Architecture (SMONA)

Die Service Monitoring Architecture (SMONA) ist eine geschichtete Architektur, die zur Überwachung von Diensten und Dienstattributen konzipiert wurde. In dieser Arbeit hat sie die Aufgabe, Informationen der Intrusion Detection Systeme zu kollektivieren, weiterzuverarbeiten und auszugeben. In der untersten Schicht

sammeln oder generieren Adapter Nachrichten, die anschließend von einem Rich Event Composer weiterverarbeitet und unter bestimmten Umständen an die Management Anwendung geschickt werden. Sie konfiguriert den Composer und den Adapter Configurator mit Hilfe der Service Information Specification Language (SISL). Eine detaillierte Beschreibung der Architektur ist in Kapitel 3.2 thematisiert. Der folgende Unterabschnitt stellt Anforderungen an die Adapter, bevor Ansprüche an den Rich Event Composer, die Management Anwendung und den Adapter Configurator erörtert werden.

Essentielle Eigenschaften der SMONA Adapter

Bisher wurden Anforderungen an Intrusion Detection Systeme, ihre Funktionalität und ihre Analyseregeln dargelegt. Dieser Abschnitt geht auf benötigte Eigenschaften der Adapter ein, die zur Informationsbeschaffung und dem Nachrichtentransport von den IDS zum Rich Event Composer erforderlich sind. Das Adapter Framework (siehe [Dürr 06]) listet bereits einige Anforderungen an SMONA Adapter im Allgemeinen auf, deren Bezug zu dieser Arbeit und den verwendeten Adapter erläutert wird. Anschließend werden weitere wichtige Eigenschaften beschrieben.

- Normalisierter Zugriff
  Die bereitgestellten Informationen der Adapter sollten einheitlich und standardisiert sein. Unabhängig von den verwendeten Werkzeugen (in diesem Fall der Intrusion Detection Systeme) sollten Adapter stets dieselben Informationen liefern, beziehungsweise dieselben Auswirkungen bei der Konfiguration einer Ressource erzielen. Neben dem einheitlichen Zugriff auf alle Adapter sollte die Schnittstelle der Adapter auf ihre Ressourcen ebenfalls standardisierte Formate verwenden. Beim Message Manipulator Adapter empfiehlt sich die Verwendung des Kommunikationsstandards IDMEF der Intrusion Detection Working Group (siehe [IDWG 07]).

- Programmiersprachenunabhängige Schnittstellen
  Jede Programmiersprache bietet unterschiedliche Vor- und Nachteile (zum Beispiel bei der Performanz) und so sind die Ressourcen, die in die SMONA Architektur integriert werden sollen in verschiedenen Programmiersprachen geschrieben. Um eine Unabhängigkeit zwischen ihren unterschiedlichen Implementierungen zu erzielen, müssen die Schnittstellen programmiersprachenunabhängig sein. Darüberhinaus ist die SMONA Architektur derzeit in Java programmiert. Die SMONA befindet sich aber noch in der Entwicklung und durch die Unabhängigkeit wird die weitere Konzeption und Implementierung nicht eingeschränkt. Dadurch können zu einem späteren Zeitpunkt die Vorteile unterschiedlicher Programmiersprachen genutzt werden, ohne den SMONA Entwurf zu beeinflussen.

- Asynchronität der Kommunikation
  Die Abfrage von Systeminformationen durch die Adapter kann mitunter mehrere Sekunden in Anspruch nehmen. In dieser Zeit dürfen weder der Adapter, noch der abfragende Rich Event Composer blockiert werden. Deshalb müssen asynchrone Methodenaufrufe verwendet werden. Außerdem müssen die Komponenten der Integrations- und Konfigurationsschicht über Fehlverhalten, zum Beispiel Timeouts, informiert werden. Existieren keine Sicherheitsnachrichten so muss der Message Manipulator Adapter diese Information weiterleiten, um klarzustellen, dass der Funktionsaufruf ordnungsgemäß durchgeführt wurde und der Adapter läuft.

- Autorisierung und Authentifizierung
  Unbefugte Zugriffe sind in der SMONA Architektur zu verhindern, da sonst die Architektur selbst Ziel von Angriffen werden könnte. Die Komponenten der Integrations- und Konfigurationsschicht und die Adapter müssen sich gegenseitig authentifizieren und autorisieren. Ein Angriff in diesem Zusammenhang könnte gefälschte Anfragen an einen Adapter sein, um diesen zum Absturz zu bringen (Denial of Service Angriff) oder das Versenden falscher Informationen an den Rich Event Composer, um das Management zu täuschen.

- Selbstbeschreibung der Adapterfähigkeiten
  Zur effizienten Verwaltung und Kontrolle von Adapter ist es wünschenswert, wenn sie selbständig ihre Schnittstellen, Parameter und Funktionen beschreiben können. Auf Grund der heterogenen Management und Sicherheitslösungen ist diese Anforderung jedoch nur schwer zu realisieren. Der Message Manipulator Adapter sollte zu diesem Zweck standardisierte XML IDMEF Nachrichten bearbeiten, die von

einem XML Schema beschrieben werden. Darüberhinaus wird eine textuelle Beschreibung der Adapter und dessen Datenformat in seiner Konfigurationsdatei gefordert.

- Verwaltung von Historien
  Die gelieferten Messwerte von Adaptern können natürlichen Schwankungen unterliegen. Damit dem Rich Event Composer keine die Tatsachen verfälschenden Ereignisse gemeldet werden, sollte ein Adapter früher gemessene Werte speichern, um gegebenenfalls Durchschnittswerte liefern zu können. In der vorliegenden Arbeit spielt diese Anforderung eine geringe Bedeutung, da IDS Sensoren, die ihren Bereich nach Anomalien durchforsten, nach Schwankungen und Unregelmäßigkeiten suchen müssen und dies nicht die Aufgabe der SMONA Architektur oder des Adapters ist.

Durch die Verwendung des SMONA Adapter Frameworks werden bereits die ersten drei Ansprüche erfüllt. Für die Adapter dieser Arbeit gelten darüberhinaus noch weitere Anforderungen:

- Verschlüsselte Kommunikation
  Wie in Abschnitt 2.2.1 bereits aufgezeigt wurde, müssen Adapter sicherheitsrelevante Nachrichten verschlüsseln, da der Informationsaustausch über unsichere Transfernetze geleitet werden kann. Durch die Verschlüsselung wird die Informationsgewinnung für Angriffe von außen erschwert (siehe Angriffe 2 und 3 in Abbildung 2.1). Ein Verbergen des Nachrichtenaustausches, so dass er von Angreifern nicht entdeckt werden kann, wäre darüberhinaus wünschenswert.

- Performanz
  Einbruchsversuche können in wenigen Sekunden abgeschlossen sein. Das Einlesen der Sensor Nachrichten durch den Adapter, die Verarbeitung und die Informationsweiterleitung muss unmittelbar und schnellstmöglich geschehen, um rechtzeitige Gegenmaßnahmen zu erlauben, sofern die Gegenmaßnahmen nicht von den verteilten IDS ausgeführt werden. Die Intervalle, in denen die Nachrichten des Prelude Managers ausgelesen werden, müssen dementsprechend wählbar sein.

- Unabhängige Konfiguration sensibler Einstellungen
  Die SMONA Architektur wird in dieser Arbeit in einem domänenübergreifenden Netz betrieben. Auf Grund unterschiedlicher Zuständigkeiten für die Systeme müssen Adapter zentral und teilweise lokal konfigurierbar sein. Sensible Einstellungen, wie zum Beispiel der Speicherort von Protokolldateien, müssen bei Bedarf lokal festgelegt werden können, ohne dass sie für den Adapter Configurator veränderbar sind. Darüberhinaus werden alle anderen Einstellungen über den Configurator getätigt.

- Dynamisches Konfigurieren
  In der SMONA Architektur werden die Adapter, ihre Parameter und ihre Eigenschaften in einem Dokument mit der Service Information Specification Language (siehe Kapitel 3.2.1) beschrieben. Daraus ermittelt der Adapter Configurator ihre Startkonfiguration. Die benötigten Informationen können sich im Betrieb ändern, so dass eine dynamische Einstellungsänderung notwendig wird. Adapter sollten zum Beispiel beim Eintreten bestimmter Ereignisse zur Verarbeitung zusätzlicher Informationen konfiguriert werden können, die im normalen Betrieb eine geringe Bedeutung haben. Die erneute Konfiguration sollte keinen Neustart des Adapters erforderlich machen. Daraus ergibt sich die Anforderung, dass Adapter im laufenden Betrieb dynamisch konfigurierbar sein müssen.

- Bedarfsgesteuerter Adapterstart
  Im vorliegenden Adapter Framework wird ein Adapter von der Konsole aus gestartet. Anschließend wartet er auf ein Signal des Adapter Configurators, um die Datenverarbeitung zu beginnen. Idealerweise sollten Adapter je nach Bedarf, direkt vom Adapter Configurator gestartet und beendet werden, ohne dass sie zuvor und danach in einem Wartezustand verharren.

- Allgemeine, wiederverwendbare Adapter
  Adapter dürfen nicht auf eine einzige Funktion spezialisiert sein. Sie sollten in einem Aufgabenbereich konfigurierbar sein, um die Anzahl der verschiedenen Adapter gering zu halten. In dieser Arbeit wird ein Adapter entwickelt, der Informationen von einem Prelude Manager einließt, manipuliert und anschließend weiterleitet. Er sollte sich bei der Datenaquise nicht allein auf die Prelude Architektur konzentrieren, sondern auch Daten anderer Quellen verarbeiten können.

Dieser Abschnitt hat Anforderungen an die Adapter der SMONA Architektur zur Föderation von Intrusion Detection Systemen erläutert. Ihre Realisierung wird in Kapitel 5.2.5 beschrieben und diskutiert. Im Folgenden

werden Ansprüche an die Komponenten der höheren SMONA Schichten dargelegt, die die Informationen der Adapter weiterverarbeiten und ausgeben.

Anforderungen an die SMONA Komponenten der höheren Schichten

Im vorherigen Abschnitt wurden Anforderungen an die Adapter der SMONA Architektur gestellt, die der Informationsaufnahme dienen. Diese überschneiden sich mit den geforderten Eigenschaften der Komponenten auf der Integrations- und Konfigurationsschicht und der Anwendungsschicht. Sie müssen zur Datenverarbeitung ebenfalls über programmiersprachenunabhängige Schnittstellen verfügen und asynchron kommunizieren. Aus Sicherheitsgründen wird ein verschlüsselter Datenaustausch verlangt, bei dem sich die Teilnehmer gegenseitig authentifizieren und autorisieren. Darüberhinaus müssen sie dynamisch im Betrieb konfigurierbar sein und Adapter direkt starten und beenden können. Nachstehende Auflistung beschreibt weitere Anforderungen an die SMONA Komponenten der Integrations- und Konfigurationsschicht und der Anwendungsschicht. Teilweise gehen die Punkte dabei auf spezielle Eigenschaften der Implementierung in dieser Arbeit ein.

- Lastausgleich
  In Grids, die aus mehreren tausend Rechnern bestehen, kann die Datenaggregation des Rich Event Composers viele Ressourcen benötigen. Deshalb müssen die Komponenten auf mehrere Rechner aufteilbar sein. Zum besseren Lastenausgleich sollen Komponenten mehrfach in die Architektur eingebunden werden können, so dass zum Beispiel je ein Adapter Configurator für die Einstellung der Adapter in einer Domäne zuständig ist.
- Konfiguration mit der Service Information Specification Language (SISL)
  Die SMONA Architektur sieht die Konfiguration des Rich Event Composers und des Adapter Configurator in der Service Information Specification Language (siehe Abschnitt 3.2.1) vor, die auch von der prototypischen Implementierung verwendet werden muss.
- Ausfallsicherheit
  Alle Komponenten müssen unabhängig voneinander lauffähig sein und ein Absturz einzelner Teile darf die Funktion des übrigen Systems nicht beeinträchtigen.
- Konfigurierbarkeit von Aggregationen
  Der Rich Event Composer empfängt von den Adaptern Informationen, die er in Aggregationen zusammenfasst und gegebenenfalls an die Management Anwendung verschickt. Der Composer wird nur in eingeschränkter Funktionalität für die Föderation von Intrusion Detection Systemen benötigt. In der prototypischen Umsetzung muss er trotzdem mit Funktionen und Notifikationen zur Aggregation und Benachrichtigung konfigurierbar sein und dafür rudimentäre Methoden implementieren.
- Konfiguration der Adapter
  Die Einstellungen der Adapter müssen in einem SISL Dokument angegeben und an den Adapter Configurator gesendet werden. Dieser muss die Konfiguration in das vom Adapter Framework verwendete Format umwandeln und Adapter starten und beenden können.
- Zentrale Verwaltung des Systems
  Die Steuerung der SMONA Komponenten muss über die Management Anwendung erfolgen, die die Konfiguration von Adaptern beim Adapter Configurator anstößt und den Rich Event Composer einstellt. Darüberhinaus ist die Management Anwendung für die Darstellung von Rich Events verantwortlich. In dieser Arbeit muss sie die generierten Rich Events und Statusmeldungen der SMONA Komponenten ausgeben.

Dieser Abschnitt hat grundlegende und spezielle Anforderungen an alle Teilsysteme beschrieben, die in dieser Arbeit eingesetzt werden. Sie fließen in die zu konzipierenden Komponenten ein und ihre Umsetzung wird in Kapitel 6.3.2 überprüft. Das nachfolgende Kapitel erklärt wichtige Grundlagen, die in der Konzeption und Implementierung des Sicherheitssystem von Bedeutung sind.

3 Grundlagen

Dieses Kapitel skizziert den Kontext der Diplomarbeit. Abschnitt 3.1 beschreibt die wichtigsten Grundlagen des Grid Computing. Nach einem Definitionsversuch und einem Überblick wird der Grid Standard, die Open Grid Service Architecture (OGSA, siehe [FKS+ 06]) und ihre Sicherheitsmechanismen erläutert. Der Abschnitt Monitoring im Grid Computing beschreibt unterschiedliche Softwarelösungen zur Überwachung von Leistungsdaten in Grids. Abschließend für diesen Abschnitt wird die Grid Middleware Globus Toolkit dargestellt. Nachfolgend wird die Service Monitoring Architecture (SMONA) und die Service Information Specification Language (SISL) in Abschnitt 3.2 ausführlich erläutert. Der Abschnitt 3.3 gibt eine Einführung in Intrusion Detection Systeme (IDS), ihre Komponenten und ihre Funktionsweise. Außerdem werden die Kommunikationsstandards IDMEF und IDXP vorgestellt. Beispielhaft wird das Hybrid IDS Prelude, das für die vorliegende Arbeit von großer Bedeutung ist, explizit beschrieben. Abschließend für dieses Kapitel (siehe Abschnitt 3.4) werden die Grundlagen des CORBA Standards thematisiert, der in der Implementierung verwendet wird.

3.1 Grid Computing

Dieser Abschnitt führt in das Grid Computing ein. Zu Beginn werden zwei Definitionen aus dem Artikel „The Anatomy of the Grid" und einer Gartner Studie gegeben und zusammengefasst. Der Abschnitt 3.1.2 beschreibt die Aufgaben von Grids und gibt ihren Aufbau in Kürze wieder. Außerdem wird der Begriff der Virtuellen Organisation eingeführt. In Abschnitt 3.1.3 wird auf die konzeptuelle Infrastruktur der Open Grid Service Architecture (OGSA) und seiner Dienste eingegangen. Anschließend (siehe Abschnitt 3.1.4) werden die Sicherheitsmechanismen der OGSA genauer erläutert. Das Monitoring im Grid Computing spielt für diese Arbeit eine wichtige Rolle und deshalb werden in Abschnitt 3.1.5 mehrere Lösungen vorgestellt. Die Monitoring Architektur SMONA, die von zentraler Bedeutung für die Aufgabenstellung ist, wird ausführlich in Abschnitt 3.2 behandelt. Abschließend wird das Globus Toolkit als die bedeutendste Grid Middleware erläutert (siehe Abschnitt 3.1.6).

3.1.1 Definitionen

Im Artikel „The Anatomy of the Grid" von Ian Foster, Carl Kesselman und Steven Tuecke (siehe [FTK 01]) wird Grid Computing als das Bereitstellen von direktem Zugriff auf Computer, Software, Daten und anderen Ressourcen, die zum gemeinsamen Problemlösen nützlich sind, beschrieben. Der Zugriff und das Verteilen wird von den beteiligten Ressourcenanbietern strikt kontrolliert und klar definiert. In einem späteren Artikel (siehe [Fost 02]) erweitert Foster diese Definition um eine Prüfliste. Demnach koordiniert ein Grid unter Verwendung standardisierter, offener und universeller Protokolle und Schnittstellen Ressourcen, die nicht zentral verwaltet werden. Darüberhinaus bietet Grid Computing hochwertige Dienste an.

Eine andere Definition liefern Mike Chuba und Carl Claunch in einer Untersuchung (siehe [ChCl 06]) zum Grid Computing:

> „Grids sind Sammlungen von Computerressourcen mehrerer Organisationen, die zum Lösen von gemeinsamen Problemen koordiniert werden."

Aus den oben beschriebenen Definitionen ergeben sich folgende Eigenschaften für das Grid Computing: Grid Computing ist eine Technologie, die auf offenen Standards basiert. Die Spezifikationen der verwendeten Protokolle und Schnittstellen sind offen und allgemein und beschreiben grundsätzliche Funktionen für die Authentifizierung, die Autorisierung, die Ressourcen Ermittlung und den Ressourcen Zugriff. Grid Computing

beinhaltet die enge Kooperation und Integration unterschiedlichster Ressourcen (Hardware, Software) sowie Benutzern aus verschiedenen administrativen Domänen. Das Ziel des Grid Computing ist es, eine nicht triviale Dienstgüte bereitzustellen, um Probleme in Kooperation zu lösen.

Nach dieser Definition erläutert der folgende Abschnitt kurz die Eigenschaften von Grids und führt den Begriff der Virtuellen Organisation ein.

3.1.2 Überblick über das Grid Computing

Es gibt verschiedene Grids mit unterschiedlichen Aufgaben. Am häufigsten werden Grid Technologien zur Verbesserung der Rechenleistung (Computing Grids) oder zur Vergrößerung der Speicherkapazität (Data Grids) verwendet. Darüberhinaus können Grids den Zugriff auf andere Ressourcen, wie Sensoren und Anwendungen ermöglichen. Daraus resultiert eine bessere Auslastung von Geräten und Anwendungen und es fallen geringere Kosten für neue Hardware und das IT Management an (siehe [JBFT 05] S. 7 ff). Der Schlüsselaspekt eines Grids, unabhängig seines Typs, sind Ressourcen unterschiedlicher Eigentümer, die zum Lösen eines gemeinsamen Problems koordiniert werden (siehe [ChCl 06]). Diese Zusammenarbeit dynamischer Gruppen von Personen und Institutionen mit Ressourcen wird Virtuelle Organisation (VO, siehe [FTK 01]) genannt.

Grids können, je nach Notwendigkeit, Ressourcen, Personen und Organisationen dynamisch einbinden oder herauslösen. Die eingebrachte Hard- und Software führt zu einer agilen und heterogen Struktur in Grids. Dazu können Supercomputer, einfache Rechner, Speichernetzwerke oder Sensoren wie zum Beispiel Teleskope (siehe Abbildungen 2.1 und 3.1) gehören, die über ein Netz verbunden sind. Häufig sind die Ressourcen geographisch weit verteilt und gehören zu unterschiedlichen administrativen Domänen, die keine zentrale Kontrollinstanz haben. Daraus folgt ein hoher Kommunikationsaufwand und es müssen Richtlinien im Hinblick auf Nutzerverwaltung, Sicherheitsfragen, Wartung und den Datenschutz definiert werden. Es besteht die Notwendigkeit das Verteilen der Ressourcen streng zu kontrolliert und Ressourcenanbieter und -verbraucher müssen klar definieren, was geteilt werden soll und wer darauf Zugriff erhält.

Eine Gruppe, die solche Regeln definiert und aus mehreren Personen und / oder Institutionen besteht wird Virtuellen Organisation (VO, siehe [FTK 01]) genannt. Die Virtuelle Organisation ist nicht in einer Institution manifestiert und kann neue Mitglieder dynamisch aufnehmen, oder nach Erfüllung ihrer Aufgabe verabschieden. Diese Organisationen können schnell gegründet und auch wieder aufgelöst werden. Virtuelle Organisationen können demnach unterschiedlichste Dauer, Größe und Strukturen besitzen. Der Hauptaspekt liegt in dem gemeinsamen Lösen einer Problemstellung. Dazu werden nicht nur Dokumente ausgetauscht und Daten bereitgestellt, sondern es wird den Beteiligten Zugriff auf Anwendungen, Sensoren, Computer und andere Ressourcen ermöglicht. Mitglieder und Komponenten einer Virtuellen Organisation können gleichzeitig an weiteren Virtuellen Organisation beteiligt sein.

Grundlegende Dienste, Richtlinien und Anforderungen, die zum verteilten Zugriff auf Ressourcen notwendig sind, werden in der Open Grid Service Architecture (OGSA, siehe [FKS+ 06]) beschrieben, die im folgenden Abschnitt zusammengefasst ist.

3.1.3 Die Open Grid Service Architecture

Die Open Grid Service Architecture (OGSA, siehe [FKS+ 06]) ist eine vom Global Grid Forum (GGF, siehe [GGF 07]) herausgegebene Spezifikation von Diensten und ihren Anforderungen, die ein Grid bereitstellen soll.

Die konzeptuelle Infrastruktur eines Grids besteht aus drei Schichten (siehe Abbildung 3.1). Die unterste Schicht beschreibt einfache Ressourcen (zum Beispiel Hardware, Daten, Netze, Lizenzen, usw.) und ist gekennzeichnet durch hohe Variabilität und wird lokal verwaltet, angepasst und optimiert. Sie steht nicht im Fokus der OGSA, ist für sie aber relevant. Die höchste Schicht (zum Beispiel Benutzeranwendungen, Benutzer Frameworks, usw.) ist benutzerzentriert und bietet den Geschäftswert. Dazwischen befindet sich eine dienstorientierte, virtualisierte und auf Standards basierende Schicht, die eine geringere Variabilität als die unterste Schicht aufweist. Diese Schicht abstrahiert die niederen Ressourcen und steht im Fokus der OGSA. Ihre funktionalen Komponenten basieren auf an Grids angepasste Web Service Standards (siehe [WS 03]).

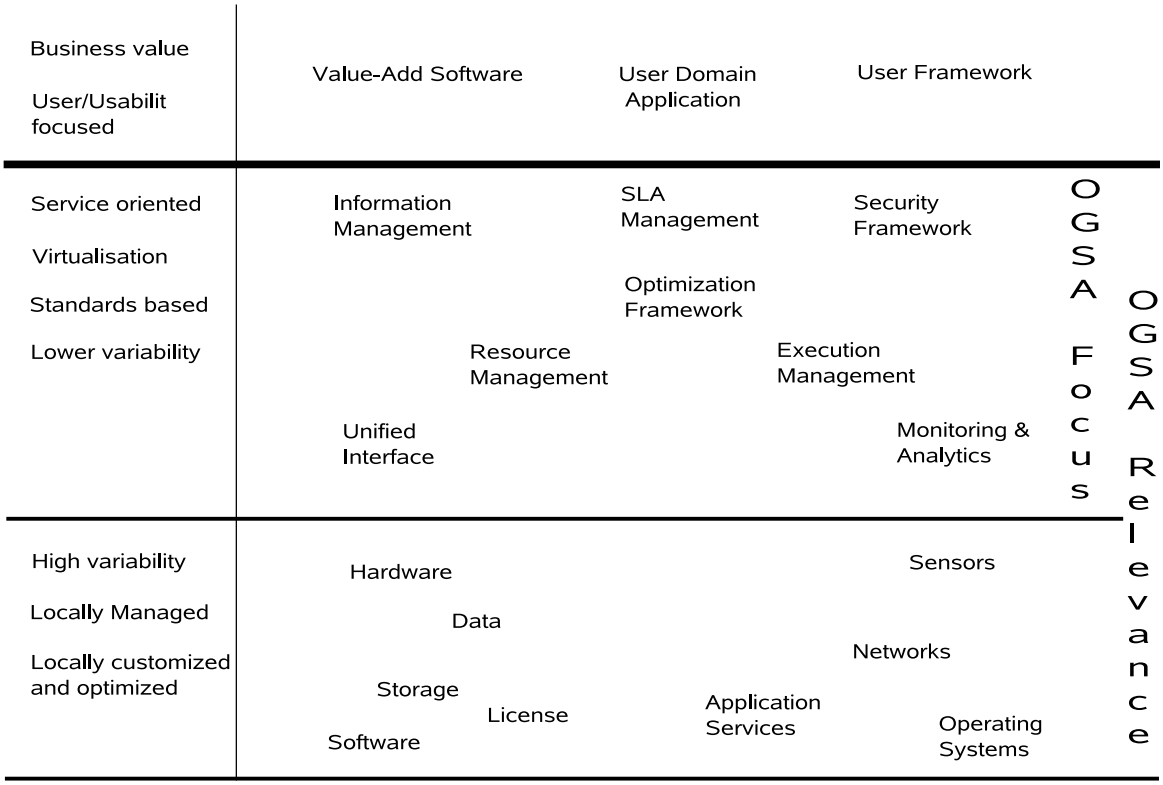

Abbildung 3.1: Überblick über die geschichtete Grid Infrastruktur nach [FKS+ 06]

Die OGSA definiert den Entwurf von Diensten, ihre Schnittstellen, ihr Verhalten und die Kommunikation der Dienste untereinander. Diese Dienste haben keine bestimmte Hierarchie und sind deshalb unabhängig voneinander. In Abbildung 3.1 sind in der mittleren Schicht einige aufgelistet. Die OGSA beinhaltet ein Kompositions Paradigma, das es Diensten ermöglichen soll, andere Dienste zu benutzen, um ihre eigene Funktionalität bereitzustellen. Nachfolgend werden die funktionellen Komponenten und Richtlinien der OGSA beschrieben:

- Infrastrukturrichtlinien
  OGSA kann als ein spezielles Anwendungsprofil der Webservice Kern Dienste angesehen werden. Das Global Grid Forum nimmt an, das Grid Systeme und Anwendungen wie dienstorientierte Architekturen strukturiert sind und das weiterhin Dienstschnittstellen in der Web Services Description Language (WSDL, [CCMW 01]) definiert werden. Nachrichten sind in der Extensible Markup Language (XML, [BPSM+ 06]) kodiert und werden nach den Richtlinien des Simple Object Access Protocols (SOAP, [GHM+ 06]) übertragen. In der OGSA Spezifikation des Global Grid Forums wird bei der Zustandsdarstellung und Komposition von Diensten, den Transaktionsmechanismen und der Benachrichtigung von Komponenten auf neue Entwicklungen im Bereich der Web Services hingewiesen.

  Darüberhinaus schreibt die OGSA eine Namensgebung für Dienste vor, die aus drei Schichten besteht. Diese sind ein auf Wörtern basierender Name, der nicht eindeutig sein muss und den Dienst beschreibt, ein abstrakter Name, der räumlich und zeitliche eindeutig ist und eine Adresse, die den Ort des Dienstes beinhaltet.

- Ausführungsmanagement Dienste
  Die Richtlinien zum Execution Management Services (OGSA-EMS) beschäftigen sich mit dem Auffinden und dem Auswählen von Ausführungskandidaten, der Ausführungsvorbereitung und der Ausführung von Arbeitseinheiten selbst. Dabei gibt es drei OGSA-EMS Klassen: Dienste, die die Verarbeitung, Speicherung, Verwaltung und Bereitstellung von Betriebsmitteln modellieren, Job Management Dienste und Auswahldienste (Execution Planning Service, Candidate Set Generator, Reservation Service).

- Datendienste
  Die in der OGSA betrachteten Datenressourcen beziehen sich auf normale Dateien, Datenströme, Datenbankmanagementsysteme, Kataloge, abgeleitete Daten und andere Datendienste. Dienstgüte Kriterien für den verlässlichen Transport, die Bandbreite, die Ankunftszeit und andere Transporteigenschaften können spezifiziert werden. Darüberhinaus schreibt die OGSA eine Datenverwaltung vor, die Metadaten, Quoten, Lebenszeiten, Verschlüsselung, Persistenz, Redundanz, Zugangsrechte und Zwischenspeicher kontrolliert.

- Ressourcenmanagement Dienste
  Die Betriebsmittelverwaltung in Grids muss physikalische und logische Ressourcen direkt oder über Web Service Schnittstellen verwalten. Das Web Service Basic Profil sieht Verwaltungsfunktionen auf der Infrastrukturebene nach dem OASIS WSDM Entwurf vor. Es wird zwischen dem Verwalten von Webservices ([Sedu 05]) und dem Verwalten mit Webservices ([Vamb 05a],[Vamb 05b]) unterschieden.

- Sicherheitsdienste
  Auf die Sicherheitsmechanismen im Grid Computing wird in Abschnitt 3.1.4 genauer eingegangen. Die OGSA schreibt kein bestimmtes Verfahren zur Authentifizierung und Autorisierung vor, definiert aber einen Dienst, der die unterschiedlichen Methoden interoperabel gestaltet. Ein Audit Dienst, der sicherheitsrelevante Ereignisse detailliert protokolliert, wird vom Global Grid Forum empfohlen.

- Selbst-Management Dienste
  Das Selbst-Management besteht aus der Selbstkonfiguration, dem Selbstreparieren und der Selbstoptimierung. Mechanismen zur Selbstkonfiguration passen das System an dynamische Veränderungen im Grid an. Die Selbstreperatur erkennt fehlerhafte Operationen der Ressourcen und Dienste und veranlasst Korrekturmaßnahmen. Die Selbstoptimierung ist eine Eigenschaft eines Grids, sich selbst effizient zu konfigurieren.

- Informationsdienste
  Die Informationsdienste teilt die OGSA in vier verschiedene Kategorien: Ein grundlegender Dienst muss das Auffindung von Ressourcen und Diensten unterstützen. Ein weiterer Informationsdienst kann Nachrichten zwischen Ressourcen und Gridnutzern zwischenspeichern und weiterleiten. Darüberhinaus sieht die OGSA Informationsdienste zur Protokollierung und Überwachung (Monitoring) von anderen Diensten vor.

Im nächsten Abschnitt wird auf die Funktionalitäten, die die OGSA zur Sicherung von Grids fordert, eingegangen.

3.1.4 Sicherheitsmechanismen im Grid Computing

Die Open Grid Service Architektur definiert mehrere Mechanismen, die die Sicherheit in Grids erhöhen sollen. Alle verwendeten Methoden und Konzepte müssen die Interoperabilität von lokalen Sicherheitsstandards über Domänengrenzen hinweg garantieren. In Abbildung 3.2 sind die konzipierten Dienste mit ihren Funktionalitäten dargestellt. Die geforderten Sicherheitsmechanismen werden im Folgenden mit den dazugehörenden Diensten beschrieben.

- Authentifizierung
  Die Authentifizierung beschäftigt sich mit dem Sicherstellen einer behaupteten Identität. Diese Funktionalität wird vom Credential Validation und dem Trust Service bereitgestellt. Die Authentifizierung kann zum Beispiel in der Evaluierung einer Benutzer-ID und eines Passwortes bestehen.

- Identity Mapping
  Die Dienste Trust, Attribute und Bridge/Translation liefern die Möglichkeit Identitäten einer Identitäts-Domäne zu transformieren, so dass sie in einer anderen Identitäts-Domäne existieren. Diese Funktionalität befasst sich nicht mit der Authentifizierung einer Identität, sondern dient einer von Richtlinien getriebenen Relation von Identitäten.

- Autorisierung
  Diese Funktionalität wird durch den gleichnamigen Dienst bereitgestellt. Der Autorisierungsdienst nimmt

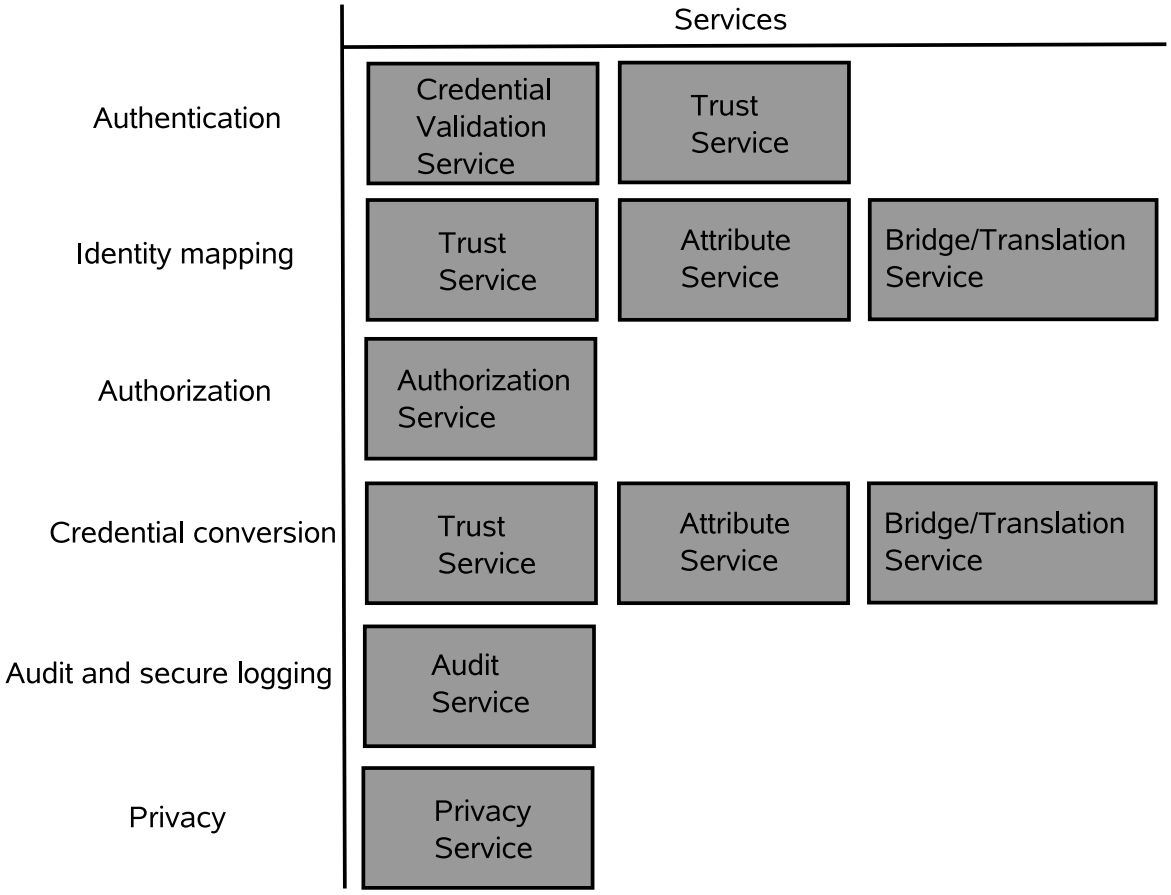

Abbildung 3.2: Komponenten des Grid Sicherheitsmodels nach [FKS+ 06] S.48

Credentials eines authentifizierten Dienstanfragenden an und entscheidet über den Zugriff auf den Dienst und die Autorisierung der benötigten Ressourcen.

- Credential Conversion
  Das Umwandeln von Credentials einer Domäne in Credentials einer anderen Domäne ist die Aufgabe dieser Funktionalität. Dazu gehören das Abgleichen von Gruppenmitgliedschaften, Privilegien, Attributen und der Annahmen die über Entitäten getroffen wurden. Die Funktionalität wird von den Trust-, Attribute- und Bridge/Translationdiensten übernommen.

- Audit and Secure Logging
  Audit und sicheres Protokollieren wird vom Auditdienst bereitgestellt. Er ist für die Protokollierung sicherheitsrelevanter Ereignisse verantwortlich und wird wie der Identity Mapping- und der Autorisierungsdienst durch Richtlinien konfiguriert.

- Privacy
  Der Datenschutz wird durch den Privacy Dienst gewährleistet. Er dient dem Schutz von persönlichen Informationen, kann diese von Dienstanbietern und Nutzern speichern und verwalten und überwacht die Einhaltung von Datenschutzrichtlinien einer Virtuellen Organisation.

Es gibt Grid Anwendungen, die eine gesicherte Kommunikation zwischen Endpunkten benötigen. Dazu unterstützt die OGSA XML Verschlüsselung und Signaturen auf höherer Ebene und TLS und IPsec in der Transportschicht. Die Autorisierung in der OGSA kann den Webservice Agreement Dienst im Zusammenhang mit OASIS Standards (z.B. SAML und XACML) benutzen, um Sicherheitsannahmen und Zugangskontrolllisten zu beschreiben.

Die Grid Middleware Globus Toolkit implementiert die hier beschriebenen Verfahren in der Grid Security Infrastructure (siehe [FKTT 98]). Das Toolkit und seine Architektur sind in Abschnitt 3.1.6 erörtert. Nachfolgend werden Monitoring Lösungen vorgestellt, die zur Überwachung der Grid Dienste und ihrer Attribute eingesetzt werden.

3.1.5 Monitoring im Grid Computing

Leistungsstarke Monitoring Lösungen sind von zentraler Bedeutung beim Aufbau und Betrieb eines Grids. Sie ermöglichen die Überwachung der einzelnen Dienste und Komponenten und erleichtern dadurch die Fehlersuche bei Störungen oder Ausfällen des Systems. Die Bereitstellung der Informationen zum Monitoring ist Aufgabe der Informationsdienste. In [TAG+ 02] beschreibt das Globus Grid Forum ein Modell zum Überwachen von Grids, die Grid Monitoring Architecture (GMA). Skalierbarkeit, Sicherheit, geringe Latenzzeiten, ein minimaler Overhead und schnelle Verarbeitung werden als Anforderungen an das Monitoring herausgestellt. Die Architektur unterscheidet zwischen drei Komponentenkategorien (siehe Abbildung 3.3): Den Produzenten, die Informationen für das Monitoring erzeugen, den Konsumenten, die Monitoring Informationen verarbeiten und einem Registrar, der die Monitoring Informationen speichert.

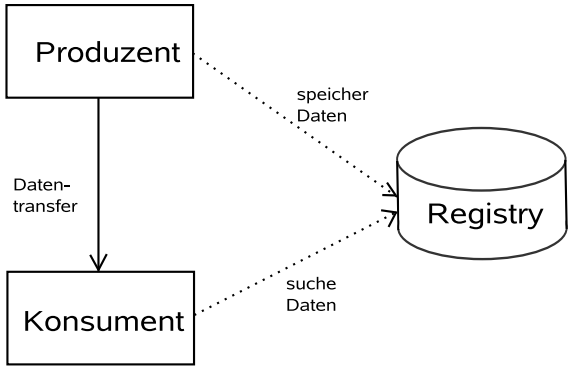

Abbildung 3.3: R-GMA Modell nach [RGMA5 07]

Es folgt eine kurze Vorstellung mehrerer exemplarischer Monitoring Lösungen:

- Relational Grid Monitoring Architecture (R-GMA)
  Im Rahmen des europäischen DataGrid (DG, siehe [DGRID 07]) wurde die R-GMA (siehe [CGM+ 03] und [RGMA 07]) als ein Grid Informations und Monitoring System entwickelt. Sie ist eine relationale Implementierung der GMA, die in ihrer Funktion einem relationalen Datenbank Management System (DBMS) ähnlich ist. Die Produzenten der R-GMA kündigen die Eigenschaften ihrer Informationen mit Hilfe eines SQL CREATE TABLE Ausdrucks an. Mit dem SQL Ausdruck INSERT werden die Monitoring Informationen veröffentlicht und Konsumenten beschaffen sich diese Daten unter Verwendung von SQL SELECT. Die R-GMA ist größtenteils in Java geschrieben und verwendet Tomcat Servlet Container.

- VisPerf
  VisPerf (siehe [LDR 03] und [VISP 07]) ist ein verteiltes, universelles Monitoring Werkzeug zur Untersuchung, Visualisierung und Kontrolle von Grids. Die Produzenten (genannt visSensor) ermitteln durch direkte und indirekte Schnittstellen den Status der Grid Middleware, ohne dass diese in großem Umfang angepasst werden muss. Die indirekten Schnittstellen verwenden Informationen aus Logdateien. visSensoren können auch direkte Schnittstellen der Grid Middleware verwenden, die interne Informationen bereitstellen. Die zentrale Komponente, zur Visualisierung, Kontrolle und Analyse wird ebenfalls VisPerf genannt.

- GridICE
  GridICE (siehe [GICE 07]) ist ein verteiltes Monitoring Werkzeug, das 2003 im europäischen DataTAG Projekt entwickelt wurde und seitdem Teil des EGEE Projektes (siehe [EGEE 07]) ist. Es verwendet standardisierte Schnittstellen, Protokolle und Datenmodelle und bietet in der Darstellung verschiedene

Ansichten für unterschiedliche Abstraktionsebenen. Die GridICE Architektur besteht aus den GridICE Sensoren, die Monitoring Informationen beschaffen und dem GridICE Server, der für die Darstellung verantwortlich ist.

- SMONA
  Die Service Monitoring Architecture (SMONA, siehe [DaSa 05]) ist ein System zur Überwachung von Diensten und Dienstattributen. Ihr Fokus liegt nicht auf Grids, sie kann aber zu deren Überwachung eingesetzt werden. Sensoren, in der Architektur Adapter genannt, befinden sind im gesamten zu überwachenden System verteilt. Sie erhalten ihre Informationen von unterschiedlichen Ressourcen und senden sie anschließend an den zentralen Rich Event Composer weiter, der die Daten aggregiert und beim Erreichen vordefinierter Zustände eine Management Komponente informiert. Eine genaue Beschreibung der Architektur befindet sich im Abschnitt 3.2.

Die im folgenden Abschnitt beschriebene Grid Middleware Globus Toolkit, verfügt über eigene Monitoring Werkzeuge. Sie sind als Information Management Komponenten zusammengefasst und werden weiter unten erläutert.

3.1.6 Das Globus Toolkit als Grid Middleware

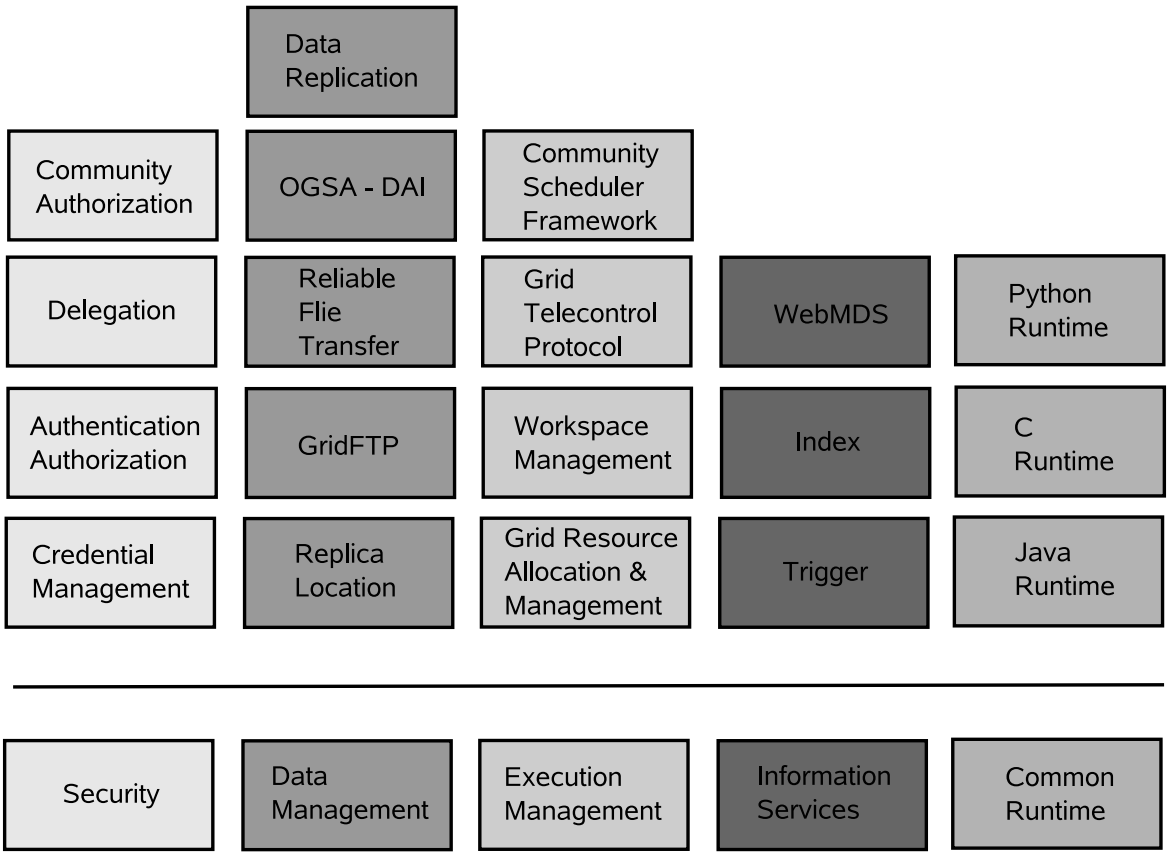

Abbildung 3.4: Komponenten des Globus Toolkit 4.0 aus [GT4COMP 07]

Die Globus Alliance (siehe [GLOBUS 07]) ist eine internationale Organisation mehrerer Universitäten und Forschungseinrichtungen. Sie entwickelt interoperable Grid-Protokolle, Programmierschnittstellen (APIs) und Entwicklungswerkzeuge (SDKs), die zusammengefasst das Globus Toolkit (GTK, siehe [GTK07]) ergeben. Das Toolkit wird nach den Vorgaben der OGSA (siehe Abschnitt 3.1.3 und [FKS+ 06]) entwickelt, auch wenn in der aktuellen Version (4.0.5) noch Komponenten enthalten sind, die nicht auf der Web Service Technologie (siehe [WS 03]) basieren. Globus ist der Standard für Grid Implementierungen und ermöglicht, Rechenleistung, Dateien und andere Ressourcen über Organisationsgrenzen hinweg zu kombinieren. Das Toolkit

stellt Komponenten zur Informationsverteilung, zum Ressourcen- und Daten Management, zur Sicherheit, zur Fehlerfindung, zur Kommunikation und zur Portabilität bereit. Außerdem enthält es eine Zertifizierungsstelle (simple-CA, siehe [SimpleCA 07]) die Benutzer- und Rechnerzertifikate erstellen und verwalten kann. Globus ist eine geschichtete Architektur, auf deren unterster Ebene lokale Dienste für die Ressourcen zuständig sind. In einer mittleren Schicht befinden sich die Toolkit Core Services, die eine Grundlage für High Level Services der Anwendungsschicht bilden. In Abbildung 3.4 sind die Komponenten des Globus Toolkit 4.0 (siehe [GTK 07]) in fünf Spalten dargestellt.

- Die Sicherheit im Globus Toolkit wird von mehreren Diensten gewährleistet und ist durch die Grid Security Infrastructure (GSI, siehe [FKTT 98]) bestimmt. Das Sicherheitskonzept basiert auf SSL Zertifikaten, die von einer zentralen Zertifizierungsstelle an Berechtigte ausgegeben werden. Die Zertifikate dienen dabei zur Authentifikation der Benutzer und der Verschlüsselung der Kommunikation. Der **Community Authorization** Dienst vergibt Zertifikate, die feingranulare Zugriffsrechte auf Ressourcen erlaubt, und von anderen Diensten anerkannt werden. Credentials können vom **Delegation** Dienst zwischengespeichert und anschließend unter Verwendung von Web Service Protokollen zur Delegation an andere Dienste genutzt werden. Zur **Autorisierung und Authentifizierung** existieren Pre-Web Service und Web Service Komponenten, die eine Public Key Infrastructure (PKI) verwenden. Sie bieten eine Programmierschnittstelle, die die Authentifizierung von Identitäten und Zugriffkontrolllisten unterstützen. Die letzte Sicherheitskomponente, das **Credential Management**, dient der Verwaltung der Credentials.

- Die Spalte des **Data Management** in Abbildung 3.4 vereint alle Komponenten, die mit der Verwaltung der Daten in Verbindung stehen. Der **GridFTP** Dienst (siehe [GFTP 07]) ist ein Datentransferdienst, der auf das GridFTP Übertragungsprotokoll aufbaut und in der Standardinstallation über den Unix Dämon Xinetd gestartet (siehe [XINE 07]) wird. Er wurde vom FTP Protokoll abgeleitet und primär um Funktionalitäten der Grid Security Infrastructure (GSI, siehe [FKTT 98]) erweitert. Darüberhinaus unterstützt der Dienst mehrere Datenkanäle zum parallelen Transfer, das Übertragen von Dateiteilen, wiederverwendbare Datenkanäle, command pipelining und den Datentransfer zu anderen GridFTP Diensten. GridFTP ist nicht OGSA konform, da er nicht die Web Service Technologie verwendet. Der **Reliable File Transfer** (RFT, siehe [RFT 07]) ist ein OGSA konformer Datentransferdienst, der den GridFTP Dienst um weitere Funktionalitäten erweitert. Der RFT verwendet eine persistente Datenbank (in der Standardkonfiguration Postgres, siehe [**?**]) zur Verwaltung der Übertragungen. Dadurch unterstützt er den Datentransfer im Disconnected Mode und es können abgebrochene Dateiübertragung wieder aufgenommen werden. Die **Data Replication** (DRS) Komponente ist in einem gleichnamigen Dienst implementiert. Sie kann Daten lokal replizieren und diese beim **Replica Location** (RLS) Dienst registrieren. Neben dem Auffinden von Datenrepliken beinhaltet er eine Relation von logischen Dateinamen auf physische Zielnamen. OGSA - DAI (siehe [ODAI 07]) steht für Open Grid Services Architecture - Data Access and Integration und bietet Schnittstellen für verschiedene Datenbanksysteme. Dieser Dienst ermöglicht SQL Anfragen an relationale Datenressourcen, sowie das Transformieren und Integrieren von Daten unterschiedlicher Quellen.

- Das **Execution Management** basiert auf vier unterschiedlichen Komponenten. Das **Grid Resource Allocation Management** (GRAM, siehe [GRAM 07]) ist eine einheitliche Dienstschnittstelle die Arbeitsaufträge annimmt und kontrolliert. GRAM erhält einen im XML Format beschriebenen Arbeitsauftrag, reserviert die benötigten Ressourcen und gibt ihn an einen Scheduler weiter. Das **Community Scheduler Framework** ist ein Metascheduling Framework, das Schnittstellen und Werkzeuge liefert, um Arbeitsaufträge in Empfang zu nehmen, Ressourcen umfassend zu reservieren und verschiedene Scheduling Richtlinien auf Grid Ebene zu definieren. Das Globus **Teleoperations Control Protocol** (GTCP) ist eine Dienstschnittstelle zur Fernsteuerung von Instrumenten und Simulationen. Der **Workspace Management Service** (WMS) erlaubt es Klienten, dynamisch Workspaces und Konten zu erstellen und zu verwalten.

- In der Spalte **Information Management** der Abbildung 3.4 sind Dienste dargestellt, die dem Monitoring der bereitgestellten Ressourcen dienen. **WebMDS** ist eine graphische Weboberfläche für standardisierte Webbrowser, um Monitoring Information zu betrachten. Der **Index Service** sammelt Monitoring Informationen des Grids und veröffentlicht sie an einem Ort. Administratoren können Regeln definieren, an Hand derer Aktionen durch den **Trigger Service** ausgeführt werden.

- Der letzte funktionelle Bereich des Globus Toolkit sind die gemeinsamen Laufzeitumgebungen (**Common Runtime**). Er enthält Bibliotheken und Werkzeuge für die Programmiersprachen C, Java und Python, um die oben beschriebenen Dienste plattformunabhängig anzubieten und damit darauf aufbauende, höherwertige Dienste implementiert werden können.

Im folgenden Abschnitt werden Grundlagen zur Service Monitoring Architecture (siehe [DaSa 05]) erläutert. Zuerst wird auf das Konzept des Systems eingegangen und anschließend die Service Information Specification Language (SISL), die zur Konfiguration der SMONA Komponenten verwendet wird, beschrieben.

3.2 Die Service Monitoring Architecture (SMONA)

Die Service Monitoring Architecture (SMONA, siehe [DaSa 05]) wird am Lehrstuhl Kommunikationssysteme und Systemprogrammierung der Ludwig-Maximilians-Universität München entwickelt. Der Ansatz geht davon aus, dass die Qualität von Diensten durch ihre Attribute, wie sie die Management Information Bases (MIB, siehe [CMRW 96]) definiert, beschrieben werden können (siehe [DgFS 07]). Die SMONA dient nicht nur der Überwachung von Ressourcen, sondern der Verwaltung und Konfiguration ganzer Dienste. Sie ermöglicht es, technische Informationen, die von Netz-, System- und Anwendungsmanagement Werkzeugen aufgegriffen werden, so zu kombinieren, dass sie den zu verwaltenden Dienst möglichst vollständig reflektieren. Dafür ist es notwendig, die Zusammenhänge zwischen den Ressourcen und auch zwischen den Diensten selbst zu kennen. Leistungsdaten einer überwachten Ressource können für unterschiedliche Dienste von Bedeutung sein. Deshalb ist Flexibilität in der SMONA von zentraler Bedeutung. Es sind Mechanismen vorgesehen, die es der Management Anwendung erlauben, in regelmäßigen Zeitabständen Ressourcen nach Daten abzufragen und opertun Konfigurationen an ihnen vorzunehmen. Andererseits sind Betriebsmittel in der Lage, Informationen selbstständig an das betreffende Dienste Management zu übermitteln. Der Informationsfluss an das Management muss sich an dessen Aufgaben orientieren und durch das Management selbst konfigurierbar sein. Die Einstellung kann über Schwellenwerte erfolgen, bei deren Überschreiten die Auslieferung einer Ereignisnachricht ausgelöst wird. Darüberhinaus können Bedingungen definiert werden, bei deren Zutreffen ebenfalls eine Mitteilung generiert wird.

Abbildung 3.5 zeigt die geschichtete SMONA Architektur. Sie veranschaulicht die einzelnen Bestandteile, die die oben beschriebene Funktionalität liefern. Die Beschaffenheit der einzelnen Schichten und ihre Komponenten werden im Folgenden erläutert:

Ressource layer
In der untersten Schicht befinden sich die unterschiedlichsten Ressourcen, die überwacht und konfiguriert werden sollen. Ressourcen können einfache Logdateien, Anwendungsprogramme oder Hardware mit verschiedensten Eigenschaften zur Informationserzeugung und Steuerung sein. Die Daten auf dieser Ebene sind unterschiedlich strukturiert und werden von den verwendeten Programmen und Datenquellen bestimmt.

Platform specific layer
Auf dieser Ebene werden die Informationen der Ressourcen und die Ressourcen selbst mit diversen, plattformabhängigen Managementwerkzeugen über technologiespezifische Schnittstellen verwaltet. Die Werkzeuge sind stark an die Ressourcen gebunden und erzeugen plattformabhängige Daten. Ihre Schnittstelle zur nächsthöheren Schicht ist abhängig, von den auf dieser Ebene verwendeten Programmen.

Platform independent layer
In dieser Schicht werden die Daten der heterogenen, plattformspezifischen Managementwerkzeuge mit Hilfe von einheitlich konfigurierbaren Adaptern homogenisiert. Abhängig von der Art der Adapter, extrahieren diese Informationen aus den verwendeten Managementwerkzeugen, um sie in einer normalisierten Form der Integrations- und Konfigurationsschicht zur Verfügung zu stellen oder sie erhalten Konfigurationen der höheren Schicht, die sie an die Managementwerkzeuge weitergegeben. Alle Adapter bieten eine vereinheitlichte Schnittstelle zur Konfiguration und liefern ihre Mitteilungen in standardisierter Form. Die Entwicklung von Adaptern für Linuxdatenquellen kann unter [Dürr 06] nachgelesen werden.

Integration and configuration layer
Zwei Komponenten dominieren diese Schicht. Nach einer Konfigurationsanfrage der Management Anwendung verändert der Adapter Configurator Einstellungen der Adapter über einheitliche Schnittstellen. Der Rich

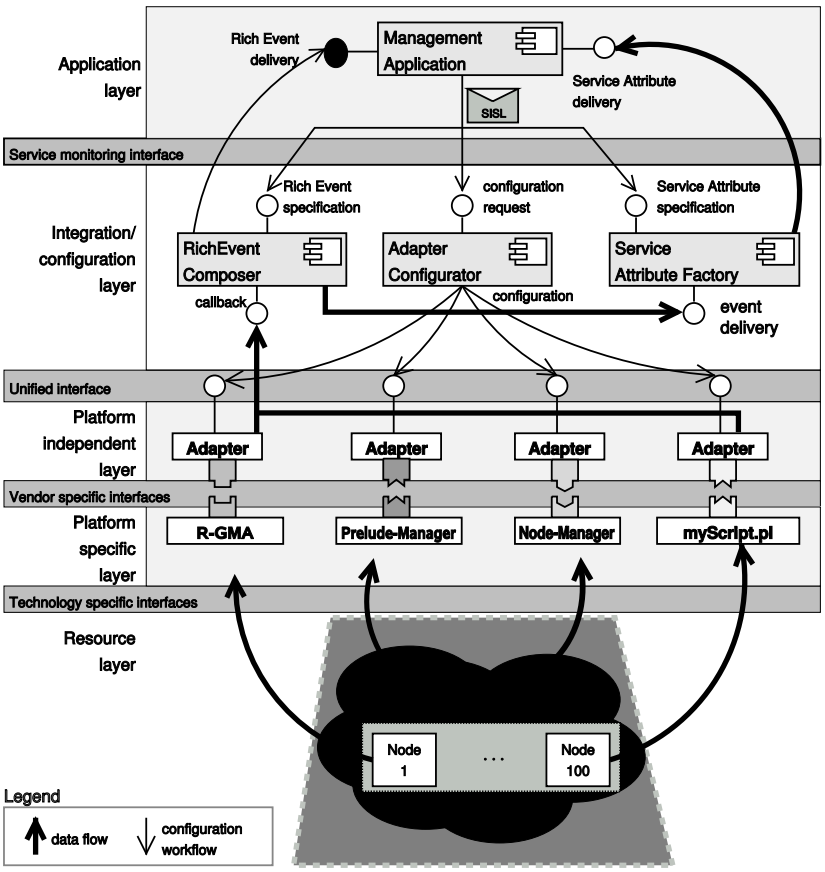

Abbildung 3.5: Die Service Monitoring Architecture aus [DgFS 07]

Event Composer bekommt von den Adaptern Ereignismeldungen der Adapter Sourcen, die er aggregiert, auswertet oder verfeinert und anschließend an die Management Anwendung weiterleitet. Die dritte Komponente dieser Schicht, die Service Attribute Factory, dient der Weiterverarbeitung von Ereignisdaten, der Konfiguration von Monitoring Optionen und soll die Wiederverwendung existierender Daten ermöglichen. Alle Komponenten dieser Schicht werden mit Hilfe der Service Information Specification Language (SISL, siehe [DgFS 07] und Abschnitt 3.2.1) durch die Management Anwendung eingerichtet.

Application layer
Die angereicherten Ereignisdaten werden von der Management Anwendung auf der Application Layer dargestellt. Darüberhinaus konfiguriert sie den Rich Event Composer, den Adapter Configurator und die Service Attribute Factory mit Hilfe der Service Information Specification Language (SISL).

In diesem Abschnitt wurden die SMONA Architektur, ihre Struktur und ihre Komponenten thematisiert, nachfolgend wird die Service Information Specification Language (SISL) erörtert, mit der die Aggregation von Dienstattributen beschrieben wird.

3.2.1 Service Information Specification Language (SISL)

Ein wichtiger Bestandteil der SMONA Architektur ist die Service Information Specification Language (SISL, siehe [DgFS 07]). Sie ist eine deklarative Sprache, die Dienstattribute in Abhängigkeit von Management Daten beschreiben kann. Ein SISL Dokument definiert eine Aggregation von Ressourcendaten. Es ist in drei Teile gegliedert: die benötigten Ressourcen, die Funktionen zur Verarbeitung der Daten und einen Benachrichtigungsteil, der bestimmt, unter welchen Umständen und in welchem Format Mitteilungen an die Management Anwendung geschickt werden.

Im Ressourcen Abschnitt werden Datenquellen beschrieben. Er wird in erster Linie vom Adapter Configurator verwendet, um Adapter und ihre Adapter Sourcen beim Starten zu konfigurieren. Funktionen, die in der SISL beschrieben sind, können von Adaptern, dem Rich Event Composer oder der Service Attribute Factory ausgeführt werden. Sie können eine Vorverarbeitung von Daten im Adapter oder die Datenaggregation im Rich Event Composer beschreiben. Darüberhinaus benutzt die Service Attribute Factory SISL Funktionen, zur Generierung von Dienstattributen. Der letzte Teil einer Aggregation beschreibt Bedingungen, die den Informationsfluss zur Management Anwendung bestimmen. Sie enthalten eine Zustandsbeschreibung als boolschen Ausdruck, bei dessen Zutreffen eine Nachrichten generiert wird. Außerdem wird in einer Deklaration das Nachrichtenformat definiert. Die Bedingungen dienen zur Einstellung des Rich Event Composers und der Service Attribute Factory.

Ein SISL Dokument wird in einer Datei im XML Format angegeben, das sich nach einem festgelegten Schema (siehe Anhang B) richtet. Zuerst werden allgemeine Informationen zur Aggregation gegeben (zum Beispiel: der Autor, das Datum, eine Kurzbeschreibung und einem Identifikationsnamen). Anschließend werden alle benötigten Ressourcen mit ihrem eindeutigen Bezeichner, ihrem Adaptertyp und ihren Attributen aufgelistet. Die Attribute enthalten ein Intervall, das einen zeitlichen Abstand zwischen den Ereignisnachrichten angibt und den Datentyp der Nachricht. Im Abschnitt der Funktionsdefinitionen wird mit grundlegenden Operatoren die Datenverarbeitung festgelegt. Eine Funktionsbeschreibung besteht aus einem eindeutigen, referenzierbaren Namen, dem Operationsnamen, dem Rückgabewert, einer optionalen Beschreibung und den Funktionsparametern. Die Parameter können Referenzen auf andere Funktionen oder Ressourcen sein. Im letzten Abschnitt der Aggregation sind Zustände definiert, bei deren Eintreten die Management Anwendung informiert werden soll. Eine Nachrichtenbeschreibung kann ein Rich Event definieren. Sie besteht aus einem eindeutigen Namen, einer optionalen Beschreibung, einer Zustandsbeschreibung und einer Deklaration. Die Zustände werden mit Boolschen Ausdrücken konkretisiert. Die Deklaration beschreibt die zu verschickende Nachricht. Sie besteht aus Referenzen auf Ressourcen oder Funktionen, deren Werte in das Rich Event übernommen werden.

Zusammengefasst, ist eine Aggregation ein abstraktes Objekt, dass alle nötigen Informationen zur Verarbeitung von Ressourcedaten und zur Erzeugung von Rich Events und Service Attributen enthält.

Der folgende Abschnitt führt kurz in die Intrusion Detection Systeme, ihre Grundlagen und ihren Aufbau ein.

3.3 Intrusion Detection Systeme

Zu Beginn definiert und erläutert dieser Abschnitt die Begriffe Intrusion und Intrusion Detection. Anschließend wird der Aufbau eines IDS und seine Komponenten in Abschnitt 3.3.2 beschrieben. An Hand der unterschiedlichen Ausprägungen der Sensorkomponente werden in Abschnitt 3.3.3 die verschiedenen Arten von IDS kategorisiert. Abschnitt 3.3.4 erläutert diverse Einbruchserkennungsverfahren, die von der Komponente Auswertungsstation eingesetzt werden können. Zur Föderation von IDS hat die Intrusion Detection Working Group (IDWG) zwei Standards entwickelt, die in Abschnitt 3.3.5 erörtert werden. Abschließend wird das Hybrid IDS Prelude vorgestellt, das für diese Arbeit von besonderer Bedeutung ist.

3.3.1 Definitionen

Die Begriffe Intrusion und Intrusion Detection sind schwer in kurzen Definitionen darzustellen. In der Literatur gibt es für beide Begrifflichkeiten eine Vielzahl unterschiedlicher Ausführungen.

Intrusion

In [Spen 05] wird eine Intrusion als eine unerlaubte, nicht autorisierte Handlung im Zusammenhang mit einem Informationssystem bezeichnet. Diese Handlungen werden in zwei Kategorien unterteilt: Einbrüche und Missbräuche. Beide können Folgen von absichtlichen Handlungen sein, oder durch unabsichtliches Fehlverhalten von Personen oder fehlkonfigurierte Komponenten ausgelöst werden. Einbrüche werden in der Literatur häufig als Angriffe von außen (von außerhalb des Teilnetzes, in dem die Komponente steht) und Missbräuche

als das gezielte Schädigen des Systems von innen (der Angreifer befindet sich innerhalb des gleichen Netzesegmentes wie das Angriffsziel) bezeichnet. Einbrüche in Netzkomponenten können auch von innen, bewusst oder unbewusst, durchgeführt werden (zum Beispiel Einbrüche in den Firmenserver). Genauso können Dienste, die von Unternehmen im Internet angeboten werden, gezielt oder aus Versehen von Außenstehenden missbraucht werden (zum Beispiel Cross Site Scripting, Google Suche nach Schlüsseln und Passwörtern).

Intrusion Detection

Das Bundesamt für Sicherheit in der Informationstechnik definiert in dem Aufsatz [BSI 02] Intrusion Detection als die aktive Überwachung von Computersystemen und -netzen auf Ereignisse, wie Angriffsversuche, Missbrauchsversuche und Sicherheitsverletzungen. Die entdeckten Ereignisse sollen zeitnah erkannt und gemeldet werden. Intrusion Detection Systeme sind eine Zusammenstellung von Werkzeugen, die von der Ereigniserkennung über die Auswertung bis zur Reaktion (Intrusion Reaktion Systeme sind einer Erweiterung der IDS) den Intrusion Detection Prozess unterstützen. Die Intrusion Detection dient somit dem Aufspüren von Einbrüchen und Missbräuchen und sollte diese in einem möglichst frühen Stadium der Ausführung erkennen.

3.3.2 Komponenten eines IDS

Das Bundesamt für Sicherheit (BSI, siehe [BSI 07]) teilt Intrusion Detection Systeme in vier funktionale Softwarekomponenten ein (siehe [BSI 02]):

- *Sensoren*
  In anderen Arbeiten wird der Begriff Sensor für ganze IDS benutzt. In dieser Arbeit sind Sensoren allein für die Datenaquise verantwortlich. Sie registrieren Ereignisse in ihrem Überwachungsbereich und leiten diese an die Auswertungsstation weiter. Es wird zwischen hostbasierten und netzbasierten Sensoren unterschieden (siehe Abschnitt 3.3.3).

- *Auswertungsstation*
  Die Auswertungsstation analysiert die empfangenen Ereignisse an Hand eines Erkennungsverfahrens (siehe Abschnitt 3.3.4) und meldet erkannte Angriffe, Anomalien und Missbräuche. Sie kann über unterschiedliche Schnittstellen bedient werden (Kommandozeile, webbasiert oder eigene, graphische Oberflächen). Dabei werden neben der Datenanalyse die Ereignisse angezeigt, sortiert, klassifiziert und gegebenenfalls zur späteren Weiterverarbeitung abgelegt. Die gespeicherten Meldungen können für Managementreports und -statistiken und zur Langzeitanalyse bislang unentdeckter Angriffe eingesetzt werden.

- *Datenbankkomponente*
  Datenbankkomponenten speichern verdächtige Ereignisse und andere anfallende Daten der Komponenten zur späteren Weiterverarbeitung und Beweissicherung. Bei geringem Datenaufkommen hält das BSI eine Speicherung in Dateiform für ausreichend. Um große Datenmengen zuverlässig verwalten zu können und Zugriffszeiten zu verringern, werden Datenbanksysteme empfohlen.

- *Managementstation*
  Die Managementstation ist für das Reporting und die Darstellung der Ereignisse verantwortlich. Über sie wird das IDS konfiguriert und verwaltet. Sie nimmt andere Komponenten und die zu Überwachenden Objekte in das IDS auf und stellt die Kommunikationsparameter (IP Adressen, Verschlüsselung, Lebenszeichen-Intervall, usw) ein. Darüberhinaus können mit der Managementstation Überwachungsregeln über eine Kommandozeile oder eine graphische Oberfläche erstellt und angepasst werden.

Die oben genannten Komponenten decken nur den Bereich der Einbruchserkennung ab. Eine Weiterentwicklung der IDS reagiert auf Angriffsversuche und kann Gegenmaßnahmen einleiten, bevor ein Schaden entsteht (Intrusion Prevention und Response Systeme, siehe [Spen 05]).

3.3.3 Arten der Intrusion Detection Systeme

Intrusion Detection Systeme werden in der Literatur meistens nach ihren Sensoren kategorisiert. Es gibt Host IDS, die einen einzelnen Rechner und Network IDS, die den Netzverkehr eines Rechners oder eines ganzen Teilnetzes auf verdächtige Ereignisse überwachen.

Host IDS

Host Intrusion Detection Systeme (HIDS) sind die ältesten Systeme und wurden entwickelt, um die Sicherheit auf Großrechnern zu untersuchen. Die Programme untersuchen Dateien und Prozesse auf einem einzelnen Rechner unter Verwendung von Integritätstests, Echtzeitanalysen oder Protokollanalysen (siehe Abschnitt 3.3.4). HIDS können Systeme umfassend überwachen und spezifische Aussagen über Angriffe treffen. Sie sind bei der Analyse auf Daten angewiesen, die vom Betriebsystem oder von einer Anwendung erzeugt wurden. Deshalb sind sie spezieller als netzbasierte IDS und erkennen typischerweise Angriffe auf Anwendungs- oder Betriebssystemebene. Für jede zu überwachende Plattform muss ein entsprechender Sensor installiert sein. Das ist kostenaufwändiger als der Einsatz reiner netzbasierter IDS. Weiterhin nachteilig für HIDS ist die aufwändige Überwachung vieler Rechner, da ein Sensor auf jedem Computer installiert sein muss und dadurch der Betrieb eines jeden überwachten Systems belastet wird. HIDS arbeiten nicht unsichtbar und können dadurch selbst Ziel eines Angriffs werden. Schließlich sind HIDS nutzlos gegen bestimmte Angriffe aus dem Netz (zum Beispiel DoS – Angriffe).
Für HIDS Beispiele siehe: Prelude-LML [PLML 07], Tripwire [TRIP 07], Logwatch [LOGW 07]

Network IDS

Network Intrusion Detection Systeme (NIDS) sind Einbruchserkennungssysteme, die auf die Überwachung des Netzverkehrs und dadurch auf die Erkennung von netzbasierten Angriffen spezialisiert sind. Die Sensoren hören permanent den Verkehr eines Netzsegments ab und prüfen ihn mittels der Paketanalyse (siehe Abschnitt 3.3.4). Zugriff auf den Datenstrom erhalten NIDS, indem sie mit Switches, die für das Portspiegeln konfiguriert sind, oder mit Hubs verbunden sind. Durch die Überwachung des Netzes von einer zentralen Komponente aus, werden andere Endsysteme nicht zusätzlich belastet. NIDS können, bei entsprechendem Einsatz und Konfiguration, für Angreifer unsichtbar arbeiten und sind deshalb schwerer angreifbar als HIDS. Nachteilig für NIDS ist die begrenzte Nachvollziehbarkeit von Angriffszielen, da sie ausschließlich die Auswirkungen eines Angriffversuchs sehen, die im Netzverkehr sichtbar sind. Bei hoher Netzlast und gleichzeitig hoher Paketdichte können, auf Grund begrenzter Rechenleistung der Analysatoren und der Switches, nicht mehr alle Pakete analysiert werden. Außerdem können verschlüsselte Kommunikationsdaten nur teilweise und dann unter hohem Aufwand überwacht werden. Die Analyse von verschlüsseltem Verkehr ist darüberhinaus rechtlich bedenklich.
Für NIDS Beispiele siehe: Snort [SNOR 07], Prelude-NIDS [PREL 07], Sancp [SANC 07], BenIDS [BIDS 07]

Hybrid IDS

Hybrid IDS vereinen Netz- und Hostsensoren in einem System. Die Managementkomponente kann unterschiedliche Sensoren gleichzeitig einbinden und deshalb die Vorteile der HIDS und NIDS nutzen und ihre Nachteile minimieren. Eine umfassende Analyse der IT Infrastruktur wird dadurch ermöglicht.
Ein Hybrid IDS Beispiel ist Prelude (siehe [PREL 07]), auf das in Unterabschnitt 3.3.6 näher eingegangen wird.

3.3.4 Einbruchserkennungsverfahren

Dieser Abschnitt beschreibt eine Unterscheidungsmethode und anschließend verschiedene Einbruchserkennungsverfahren. Die Einbruchserkennung kann nach der Art der Analyseregeln des Erkennungsmechanismus kategorisiert werden. Es gibt zwei unterschiedliche Vorgehensweisen bei der Analyse. Ereignisse können statisch nach vorher festgelegten Regeln und Signaturen oder mittels statistischen Methoden untersucht werden.

- statistische Anomalieerkennung
  Die statistische Anomalieerkennung basiert auf der Annahme, dass sich Kennwerte des überwachten Systems während eines Angriffs gravierend von ihren Normalwerten unterscheiden. Vor der eigentlichen Analyse muss in einer angriffsfreien Lernphase ein Referenzprofil des Normalbetriebs erstellt. Dabei werden die Eigenschaften und Verhaltensweisen (zum Beispiel: Anzahl von Fehlversuchen bei der Anmeldung, Nutzungshäufigkeit, Nutzungsdauer, etc.) für unterschiedliche Objekte (Nutzer, Dateien, Anwendungen, etc.) ermittelt. In der Erkennungsphase werden die Ereignisse mit dem Profil verglichen und bei einer definierten Abweichung als Angriff gewertet. Diese Methode wird auch als Anomalieerkennung bezeichnet. Sie kann bisher unbekannte Angriffe erkennen, benötigt aber eine Lernphase, in der das System nicht angegriffen wird. Verwendet das IDS keine adaptiven Verfahren, die sich an ein veränderndes Normalverhalten anpassen muss bei Veränderungen wieder ein neues Referenzprofil erstellt werden.

- Signaturanalyse
  Die statische Methode basiert auf der Annahme, dass jeder Angriff durch ein eindeutiges Muster erkennbar ist. Aus dem Muster können Signaturen erstellt werden, die von einfachen Zeichen bis zu komplexen Verhaltensmustern reichen. Die Auswertungsstation vergleicht die Signaturen mit eingehende Daten und stellt beim Blacklisting bei Übereinstimmung einen Angriff fest (Angriffssignaturen als Positiv-Listen, siehe [Spen 05]). Eine zweite Art der Signaturanalyse beschreibt das normale Verhalten des System mit Negativ-Listen, in denen alle Ereignisse beschrieben sind, die kein Angriff sind (auch Whitelisting genannt). Hier gelten alle Ereignisse als Angriffe, die keiner Signatur entsprechen. Die meisten signaturbasierten IDS beschreiben Angriffsmuster in Positiv-Listen und unterstützten die Anpassung und Neukreation der Signaturen mit einfachen Skriptsprachen. Diese Methode ist einfacher als die statistische Anomalieerkennung und sehr zuverlässig. Sind die Signaturen genau an die Angriffsmuster angepasst, werden weniger Ereignisse als Angriffe gemeldet, die dem Normalverhalten entsprechen (false positiv). Bei genau definierten Signaturen können aber leicht modifizierte Angriffe nicht mehr als solche erkannt werden (false negativ) und die Erkennungsrate sinkt. Ein weiterer Nachteil besteht darin, das für jeden Angriff eine Signatur existieren muss und neu entwickelte Angriffe nicht erkannt werden.

Die statische Signaturanalyse kann weiterhin in vier Analysearten eingeteilt werden, die auf Grund ihrer Analysedaten kategorisiert werden.

- Protokollanalyse
  Die Protokollanalyse ist die älteste Methode Angriffe und Missbräuche aufzudecken. Sie untersucht Logdateien, die vom Betriebssystem oder von Anwendungsprogrammen erstellt wurden. Die meisten IDS, die diese Analysemethode verwenden, vergleichen die Logeinträge mit Signaturen aus Positivlisten (für Beispiele siehe: Logwatch [LOGW 07], Logsurfer [LOGS 07], Logcheck [LOGC 07], Prelude-LML [PLML 07]).

- Integritätstests
  Integritätstests können Veränderungen an überwachten Dateien feststellen. Nach der Installation und Konfiguration der Programme werden für ausgewählte Informationen bestimmter Dateien Prüfsummen errechnet. Die Prüfsummen können die Dateigröße, Zugriffsrechte oder andere Dateiinformationen beinhalten. In regelmäßigen Abständen werden die Dateien mit ihren Prüfsummen verglichen. Haben sich die überwachten Dateieigenschaften geändert, wird ein Alarm ausgelöst. Nach einer Änderung an den überwachten Daten durch den Administrator müssen die Prüfsummen neu berechnet werden. Die Integritätstest können nur bereits erfolgreiche Angriffe erkennen, die zu Änderungen im Dateisystem geführt haben. (für Beispiele siehe: Tripwire [TRIP 07], Samhain [SAMH 07], AIDE [AIDE 07]).

- Paketanalyse
  Die Paketanalyse ist den Network IDS vorbehalten. Sie analysiert den Netzverkehr meistens mittels signaturbasierten Methoden. In der Paketanalyse können aber auch statistische Verfahren eingesetzt werden. Die empfangenen Daten werden in Echtzeit analysiert und das IDS kann auf Angriffe reagieren, bevor der Angriff beendet ist (die Reaktion auf Angriffe ist den Intrusion Response Systemen vorbehalten). In der Analyse können entweder die Inhalte der empfangenen Pakete oder deren Steuerinformationen untersucht werden. (für Beispiele siehe: Argus [ARGU 07], Snort [SNOR 07], Sancp [SANC 07], BenIDS [BIDS 07]).

- Echtzeitanalyse
  Die Echtzeitanalyse überwacht ständig Systemaufrufe und Dateizugriffe. Diese Analysemethode ist sehr aufwändig, da sie direkt im Betriebssystemkern verankert sein muss. Sie kann Angriffe präventiv verhindern, in dem Systemaufrufe und Dateizugriffe von bestimmten Nutzern oder Programmen blockiert werden, und der Versuch direkt an eine Managementstation gemeldet wird. (für Beispiele siehe: Snare [SNAR 07], LIDS [LIDS 07], Systrace [SYST 07]).

3.3.5 Die Kommunikationsstandards IDMEF und IDXP

Das Intrusion Detection Message Exchange Format (IDMEF, siehe [DCF 07]) und das Intrusion Detection Exchange Protocol (IDXP, siehe [FeMa 07]) wurden von der Intrusion Detection Working Group (IDWG, siehe [IDWG 07]), einer Arbeitsgruppe der Internet Engineering Task Force (IETF, [IETF 07]), entwickelt. Sie definieren ein Format und ein Transportprotokoll, um Daten zwischen Intrusion Detection-, Intrusion Prevention-, Monitoring- und Managementsystemen auszutauschen.

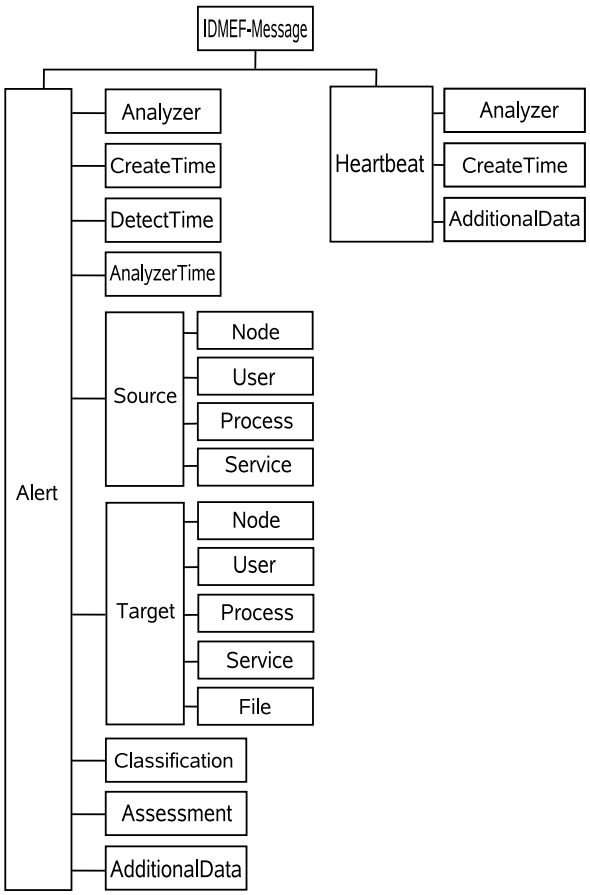

Abbildung 3.6: IDMEF Datenmodell nach [DCF 07]

In Abhängigkeit von der Art und Implementierung eines IDS enthalten Nachrichten heterogen strukturierte Informationen. Das umfassende IDMEF Datenmodell (siehe Abbildung 3.6) definiert Alarm und Heartbeat Nachrichten objektorientiert und bietet dadurch eine hohe Flexibilität. Nachrichten können durch Aggregation und Vererbung auf einfache Art erweitert werden. Darüberhinaus ist im IDMEF RFC (siehe [DCF 07]) eine Implementierung in der Extensible Markup Language (XML) beschrieben, die leicht in Intrusion Detection Systeme eingebunden werden kann. Die IDMEF Nachrichten enthalten immer Informationen über den Sensor, der sie erstellt hat, die Uhrzeit der Generierung und weitere Informationen. Alarmmeldungen enthalten darüberhinaus Informationen über die Quelle und das Ziel des Angriffs, die Uhrzeiten der Entdeckung und des Sensors und eine Einordnung und Abschätzung der Schwere des Angriffs. Der Informationstransport im XML

Format ist auf Grund der enthaltenen Datenbeschreibung ineffizient, bietet aber Vorteile bei der Lesbarkeit, der Interoperabilität und der maschinellen Verarbeitung. Der größte Nachteil des Intrusion Detection Message Exchange Formates besteht in seiner geringen Verbreitung in kommerziellen IDS Produkten.

Das Intrusion Detection Exchange Protocol ist ein Transportprotokoll auf Anwendungsschicht des OSI Modells (siehe [OSI 94]), das den Datenaustausch zwischen IDS regelt. Es ist teilweise als Blocks Extensible Exchange Protocol (BEEP, siehe [Rose 01]) Profil festgelegt und unterstützt den binären und textuellen Nachrichtenversand. Zur Authentifikation der Kommunikationspartner, zur Integritätsprüfung der empfangenen Nachrichten und zum abhörsicheren Datenaustausch setzt IDXP auf Sicherheitsprofile des BEEP Protokolls, wie zum Beispiel TLS (Transport Layer Security) oder SASL (Simple Authentication and Security Layer). Das IDXP ist ein Transportprotokoll und bietet darüberhinaus keine weiteren Dienste zur Protokollierung und Zugriffskontrolle. Durch die Verwendung eines Ports (Nummer 603) ist die Firewall Konfiguration einfach. Außerdem unterstützten viele Programmiersprachen (zum Beispiel C/C++, Java, TCL, Ruby) IDXP.

Im letzten Unterpunkt dieses Abschnitts wird die Prelude Architektur erörtert, die ein Hybrid IDS implementiert, das unterschiedliche IDS mit Hilfe des IDMEF Standards einbinden kann.

3.3.6 Beispiel Prelude

HIDS und NIDS sind zwei unterschiedliche Konzepte, die einzeln nicht alle sicherheitsrelevanten Ereignisse sammeln und analysieren können. Prelude (siehe [PREL 07]) wurde von Yoann Vandoorselaere 1998 begonnen und ist heute ein Hybrid IDS, das beide Konzepte miteinander vereinbart. Prelude ist ein verteiltes Intrusion Detection System, dessen einzelne IDS über das ganze zu überwachende Netz verteilt sind. Zu diesem Zweck ist Prelude modular aufgebaut und besteht aus mehreren Komponenten:

- **Libprelude** ist die zentrale Bibliothek für die gesamte Prelude Architektur. Sie ermöglicht eine SSL verschlüsselte Kommunikation der Komponenten im IDMEF Format (siehe [DCF 07]). Darüberhinaus bietet sie ein generisches Konfigurationsinterface, so dass alle IDS und auch der Manager über zentrale Konfigurationsdateien verwaltet werden können.
- **Preludedb** ist das Datenbank Framework des Prelude Systems. Es beinhaltet die Kommunikationsmittel, um Ereignisse in MYSQL (siehe [MYSQ 07]) oder PostgreSQL (siehe [**?**]) Datenbanksystemen zu speichern.
- Der **Prelude Manager** ist ein Hochverfügbarkeits Server, der eine Vielzahl von Verbindungen und Ereignissen verwalten, beziehungsweise verarbeiten kann. Er sammelt die Alarmmeldungen der IDS und kann die Daten in einer Datenbank speichern, als Text (als Klartext oder im XML Format) in eine Datei schreiben oder an andere Manager weiterleiten. Eine ausführlichere Beschreibung und Diskussion des Prelude Managers und seiner Ausgabe befindet sich im Konzeptionskapitel (siehe Abschnitt 4.2.3).
- **Prewikka** ist die offizielle Web Bedienoberfläche des Systems, die zur Verwaltung und Darstellung der in der Datenbank gespeicherten Ereignisse dient. Es gibt weitere grafische Schnittstellen, wie zum Beispiel Gprelude und Pylude.
- **Prelude-LML** (Prelude Log Monitor Lackey) ist ein Host IDS das lokale Protokollmeldungen überwacht, oder über ein Netz Protokolle empfängt und lokal analysiert. Der Einbruchserkennungsmechanismus des Prelude-LML stützt sich auf eine signaturbasierten, durch Plugins konfigurierten Protokollanalyse. Eine genaue Beschreibung der Funktion und der Analyseregeln wird im Abschnitt 4.3.3 diskutiert.
- **Prelude-NIDS** ist ein Network IDS, das Netzpakete in Echtzeit überwacht. Prelude-NIDS wird nicht mehr weiterentwickelt, da es von Snort abgelöst wurde.

In der Prelude Architektur (siehe Abbildung 3.7) sind verschiedene Sicherheitssysteme für die Überwachung und Analyse der Ereignisse zuständig. Sie nennt alle Informationsquellen Sensoren, auch wenn diese vollwertige IDS sind. Prelude kann unterschiedliche IDS und andere Sicherheitskomponenten (siehe Auflistung weiter unten) einbinden und dadurch alle im Abschnitt 3.3.4 beschriebenen Einbruchserkennungsverfahren einsetzen. Angriffe, Missbräuche und Anomalien melden die IDS über TLS/SSL (siehe [DiRe 06]) verschlüsselte Verbindungen an eine zentrale Managementstation, den Prelude Manager, der alle Alarmmeldungen abspeichert oder

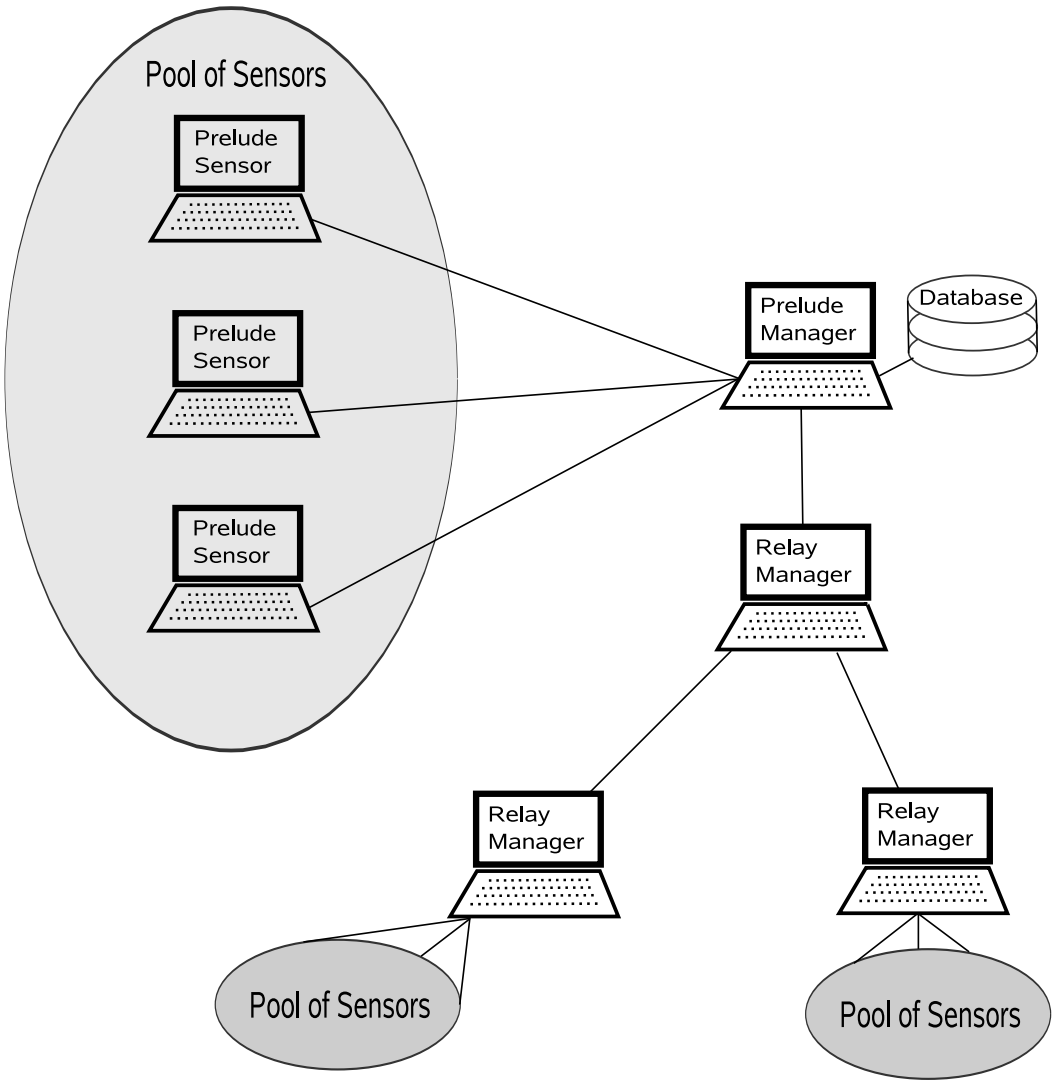

Abbildung 3.7: Prelude Architektur nach [PREARCH 07]

an andere Managementstationen weiterleitet. Die nachfolgende Auflistung skizziert die bekanntesten Sicherheitslösungen, die ihre Alarmnachrichten an den Prelude Manager schicken können und deren Funktionsweise.

- **Snort** (siehe [SNOR 07]) ist ein weitverbreitetes und umfangreiches Network Intrusion Detection und Prevention System, das den Netzverkehr wie ein Sniffer abhört und die Daten mit einer signaturbasierten Paketanalyse (dem Aho-Corasick-Algorithmus) untersucht. Neben der Alarmierung bei verdächtigen Paketen kann Snort auch Gegenmaßnahmen einleiten. Die Snort Entwickler veröffentlichen in regelmäßigen Abständen Regeldateien, die an neue Angriffe angepasst sind. Der Aufbau von Analyseregeln wird in den Kapiteln 4.3.5 und 5.3.3 beschrieben.

- **Samhain** (siehe [SAMH 07]) ist ein Multiplattform, Open Source, Host IDS, das unter Verwendung von Integritätstests Unix, Linux und Cygwin/Windows Systeme überwachen kann. Samhain kann von einem zentralen Rechner aus mehrere Dateisysteme unterschiedlicher Rechner überwachen, oder auf einer einzelnen Einheit arbeiten. Die zentrale Konfigurationsdatei (/etc/samhainrc) enthält neben den Samhain Einstellungen auch eine Auflistung der Dateien, die überwacht werden sollen. Die kontrollierten Eigenschaften der Dateien können die Dateigröße, Zugriffsrechte, Zeitstempel (des letzten Zugriffs oder der letzten Änderung), der Besitzer und weitere Eigenschaften sein. Eine weiterführende Beschreibung der Samhain Funktionalität und ihre Bedeutung für diese Arbeit ist in den Abschnitten 4.3.4 und 5.3.2 zu finden.

- Der **Prelude-LML** ist fester Bestandteil des Prelude Systems und wird auch vom Prelude Team weiterentwickelt. Er analysiert Protokolldateien mittels signaturbasierten Positivlisten. Das Host IDS kann nicht nur lokale Protokolle untersuchen, sondern auch syslog (siehe [LONV 01]) Nachrichten empfangen und anschließend analysieren. Im den Abschnitten 5.3.1 und 4.3.3 wird der Aufbau von Prelude-LML Analyseregeln erläutert.
- Der vielseitige Honeypot **Nepenthes** (siehe [NEPE 07]) sammelt Schadprogramme (englisch: Malware). Er arbeitet passiv, indem er bekannte Sicherheitslöcher emuliert und Schadprogramme herunterlädt. Die Funktionalität entspricht nicht der eines typischen IDS. Honeypots versuchen Angriffe auf sich zu lenken und dadurch Angriffe auf Computernetze zu entdecken.
- **NuFW** (siehe [NUFW 07]) ist eine Firewall, die auf Netfilter beruht und IP Datagramme, die eine Verbindung aufbauen, authentisieren.
- Das Sicherheitswerkzeug **Sancp** (siehe [SANC 07]) sammelt statistische Daten über den Netzverkehr und kann je nach Konfiguration die Datenpakete auch speichern. Es können Regeln definiert werden, die normalen von abnormalen Verkehr unterscheiden und Verbindungen markieren.
- **Libsafe** (siehe [LIBS 07]) bietet die Möglichkeit buffer overflow Angriffe mittels der Echtzeitanalyse zu erkennen. Dazu fängt es alle Funktionsaufrufe auf verwundbare Bibliotheksfunktionen ab und leitet sie an sichere Bibliotheken weiter.

Durch die Einbindung verschiedener IDS ist die Prelude Architektur ein umfassendes, verteiltes Sicherheitssystem. Bei lokalen Ausfällen oder Netzstörungen können die Sensorkomponenten Nachrichten zwischenspeichern und die Überwachung des gesamten Systems wird nicht beeinträchtigt.

Im Abschnitt Voraussetzungen der Prelude Architektur (siehe Kapitel 4.3.2) werden weitere Eigenschaften von Prelude diskutiert, die für das Sicherheitssystem von Bedeutung sind. Eine genauere Erläuterung zu den Ausgabemöglichkeiten des Prelude Managers ist in Abschnitt 4.2.3.

Der letzte Abschnitt dieses Kapitels thematisiert die Grundlagen des CORBA Standards, der in der Implementierung zur Kommunikation in der SMONA Architektur verwendet wird.

3.4 Der CORBA Kommunikationsstandard

Die Common Object Request Broker Architecture (CORBA, siehe [Grou 04]) ist eine objektorientierte, plattform- und programmiersprachenunabhängige Architektur, die von dem SMONA Adapter Framework (siehe [Dürr 06]) bereits verwendet wird. Auf Grund seiner Flexibilität unterstützt es eine standardisierte, unabhängige Zusammenarbeit von Objekten und Anwendungen über Rechnergrenzen hinweg.

Die Verwaltung und Weiterentwicklung des CORBA Standards obliegt der Object Management Group (OMG). CORBA Implementierungen werden von verschiedenen anderen Organisationen angeboten und nennen sich Object Request Broker (ORB, zum Beispiel JacORB [JACORB 07]).

Neben der guten Unterstützung vieler unterschiedlicher Programmiersprachen (zum Beispiel: Java, C/C++, Smalltalk, Cobol usw.) zur Implementierung diverser Komponenten sprechen weitere Argumente für die Verwendung von CORBA. Die Spezifikation der Schnittstellen und der kommunizierenden Instanzen erfolgt in der Interface Description Language (IDL) deklarativ und objektorientiert. Dadurch können alle Vorteile der objektorientierten Entwicklung genutzt werden. Clients und Server sind als Objekte modelliert und profitieren von Vererbung und Polymorphismus. Dadurch, dass vom ORB ein verteilter Kommunikationsbus bereitgestellt wird, müssen Kommunikationspartner nicht wie beim Remote Procedure Call (RPC) eine Zieladresse angeben. Es können bei symmetrischer und asynchroner Kommunikation n zu m Verbindungen aufgebaut werden. Der ORB bewerkstelligt die Weiterleitung von Methodenaufrufen auf entfernte Objekte und es ist für die Kommunikationsteilnehmer irrelevant, an welchem Ort sich die Methoden und Objekte befinden. Die Methodenaufrufe sind transparent und unterscheiden nicht, ob es sich um einen Aufruf einer lokalen oder einer entfernten Instanz handelt (Ortstransparenz). Beim Versenden der Objekte wird das General Inter-ORB Protocol (GIOP) verwendet, das die Konfiguration von Firewalls auf Grund der vielen verwendeten Ports,

erschwert. Der CORBA Standard spezifiziert verschiedene Dienste, die Object Request Broker anbieten (Security Service, Naming Service, Event Service, Life Cycle Service, usw).

Der Security Service ist im Entwurf dieses Sicherheitssystems von besonderer Bedeutung, da CORBA für die Kommunikation über Domänengrenzen hinweg verwendet wird. Die OMG hat bei der Spezifikation des Sicherheitsdienstes die Vertraulichkeit, Integrität, Verantwortlichkeit und Verfügbarkeit der Daten als Ziele gesetzt. Dafür stellt der Sicherheitsdienst verschiedene Mechanismen zur Authentisierung, Autorisation, Zugriffsüberwachung, Protokollierung und Verschlüsselung bereit. Die Architektur des Sicherheitsdienstes besteht aus einem Principal Authenticator (siehe Kapitel 5.1.1), der für die Authentisierung von Klienten zuständig ist und Credentials zu diesem Zweck an diese zurück liefert. Bei jedem entfernten Methodenaufruf eines Klienten werden dessen Credentials in den Aufruf kopiert. Der Domain Manager verwaltet innerhalb von Sicherheitsdomänen die Sicherheitsrichtlinien, die als Policyobjekte gekapselt werden. AccessDecision und AuditDecision Objekte, die über Request Interceptoren eingebunden sind, benutzen die Policies, um über Aufrufe und ihre Protokollierung zu entscheiden. Zu Beginn handeln Interceptoren ein SessionContext Objekt aus, mit dem Aufrufe durch Verschlüsselung gesichert werden. Die CORBA Spezifikation empfiehlt allen Implementierungen die SSL/TLS Verschlüsselungsalgorithmen (siehe [DiRe 06]) zu verwenden.

Aufbauend auf die in diesem Kapitel beschriebenen Grundlagen folgt die Konzeption des Sicherheitssystems zur Föderation von Intrusion Detection Systemen in Grid Infrastrukturen.

4 Konzeption und Entwurf des Sicherheitssystems

In diesem Kapitel wird ein Sicherheitssystem konzipiert, das die Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen ermöglicht. Zuerst erfolgt ein Überblick über das gesamte Sicherheitssystem. Es wird das in Abschnitt 2.1 gegebene Szenario in Abbildung 4.1 erweitert und die einzelnen Komponenten und der Informationsfluss des föderativen Systems kurz beschrieben. Anschließend erfolgt in Unterabschnitt 4.1 die Auswahl und Anpassung von Komponenten der höheren Schichten der SMONA Architektur, die für diese Arbeit von Bedeutung sind. Desweiteren wird der Kommunikationsfluss und das Kommunikationsformat der IDS bis zur Managementanwendung bestimmt. In Abschnitt 4.2 werden Adapter entworfen, die sicherheitsrelevante Meldungen aus Protokolldateien auslesen und manipulieren oder analysieren können und ihre Ergebnisse an den Rich Event Composer (siehe Abbildung 4.4) weiterleiten. Außerdem werden die exemplarischen Push und Pull Adapter des SMONA Frameworks an die Föderation von IDS angepasst. Nachfolgend wird in Abschnitt 4.3 auf mehrere IDS eingegangen, die mit Hilfe der Prelude Architektur eingebunden werden können und für eine umfassende Überwachung des Grids relevant sind. Es werden Analyseregeln für drei Intrusion Detection Systemen konzipiert, die die Grid Middleware Globus Toolkit auf Angriffe analysieren.

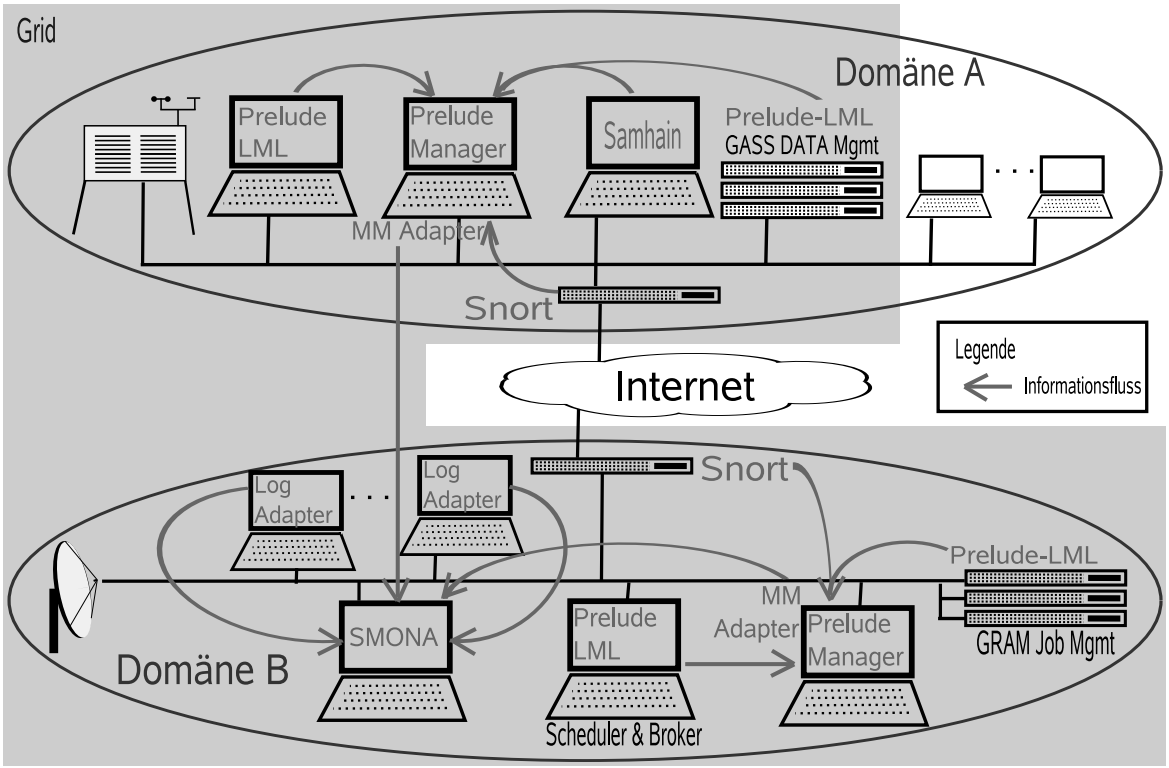

Abbildung 4.1: Föderation von IDS zur Erkennung von Angriffen

In Abbildung 4.1 ist das in Abschnitt 2.1 beschriebene Szenario um die Föderation der IDS zu einem umfassenden Sicherheitssystem erweitert. In jeder administrativen Domäne überwachen die einzelnen Intrusion Detection Systeme ihren Bereich und schicken die IDMEF Alarm und Heartbeat Nachrichten (siehe Kapitel 3.3.5) verschlüsselt an einen zentralen Prelude Manager der Domäne. Die Logdateien der Gridkomponenten

werden jeweils von einem Prelude-LML untersucht, der seine Funde in einem binär codierten IDMEF Format an den Manager meldet. Darüberhinaus wird der Netzverkehr an den Gateways durch Snort kontrolliert. In Domäne A existiert ein Samhain IDS, das mittels Integritätstest das Dateisystem überwacht. In jeder Domäne läuft ein Prelude Manager, der die empfangenen Daten normalisiert und lokal abspeichert. Hier liest ein SMONA Message Manipulator Adapter (in der Abbildung 4.1 als MM Adapter gekennzeichnet) die Alarm und Heartbeat Nachrichten ein, verändert sie und schickt sie an den zentralen Rich Event Composer (in der Abbildung unter dem Sammelbegriff SMONA bezeichnet). In Domäne B erhält dieser zusätzlich Ergebnisse von zwei Log Adaptern, die lokale Protokolldateien analysieren. Der Rich Event Composer ist die wesentliche Sammelstelle für alle sicherheitsrelevanten Ereignisse im Grid. Er gibt die erhaltenen Informationen an eine Management Anwendung weiter, mittels der sich Sicherheitsbeauftragte über den Sicherheitszustand des Grid informieren können. Der Adapter Configurator und die Management Anwendung befinden sich in der Abbildung in Domäne B und sind unter dem Namen SMONA zusammengefasst.

Im Folgenden wird der Entwurf des Frameworks der SMONA Architektur, seine Komponenten und ihre Kommunikation näher erläutert.

4.1 Konzeption des SMONA Frameworks

Dieser Abschnitt beschreibt die Kommunikation von den Intrusion Detection Systemen bis zur Managementanwendung und die Funktionalitäten von SMONA Komponenten, die für die Aufgabenstellung benötigt werden. Zuerst werden grundlegende Eigenschaften der Kommunikation erläutert (siehe Abschnitt 4.1.1) und damit die Verwendung von CORBA als Kommunikationsmittel und IDMEF als Kommunikationsformat begründet. In Abschnitt 4.1.2 werden schließlich Komponenten der SMONA Architektur ausgewählt und an die Aufgaben dieser Arbeit angepasst.

4.1.1 Kommunikation

Die SMONA Architektur dient dem Überwachen von verschiedenen Komponenten und den darauf aufbauenden Diensten. Die Informationen, die auf der ressourcen- und der plattformspezifischen Schicht erzeugt werden, sind verteilt und müssen zu den Komponenten der Integrations- und Konfigurationsschicht gegebenfalls über unsichere Transfernetze transportiert werden. Auf Grund der vielfältigen Komponenten muss die Kommunikation des Sicherheitssystems innerhalb der SMONA standardisiert und über plattform- und programmiersprachenunabhängige Schnittstellen stattfinden (siehe Anforderungen an die SMONA Komponenten und ihre Funktionalität in Abschnitt 2.2.4). Ein verlässlicher Nachrichtenaustausch, der effizient und ohne merkliche Mehrbelastung für die Infrastruktur erfolgt, ist von höchster Priorität für ein Sicherheitssystem. Im Idealfall ist der Informationsaustausch nicht nur verschlüsselt, sondern auch versteckt, so dass er von Angreifern nicht registriert werden kann.

Bei der Konzeption des Frameworks müssen die Anforderungen an die Kommunikation gegeneinander abgewogen werden. Im Folgenden wird die Verwendung von CORBA anstatt IDXP als Kommunikationsmittel und von IDMEF als Kommunikationsformat in der SMONA Architektur diskutiert.

IDXP (siehe Kapitel 3.3.5) wurde speziell zur Kommunikation zwischen Intrusion Detection Systemen konzipiert und bietet den Vorteil, dass die Firewallkonfiguration einfacher ist als bei CORBA (siehe Kapitel 3.4), da nur ein Port verwendet wird. Die SMONA Architektur ist für das Überwachen von Diensten und ihren Attributen konzipiert und benötigt dafür unterschiedliche Dienste (zum Beispiel einen Namensdienst) und eine einfache Beschreibung von Schnittstellen, das nur von CORBA geleistet wird. Weiterhin benötigt die Verschlüsselung der Kommunikation durch JacORB keine Anpassungen am Programmcode und ist damit unabhängig von der verwendeten Programmiersprache und den Schnittstellen der Komponenten. Deshalb wird in diesem Entwurf der SMONA Architektur die CORBA Implementierung JacORB für alle Verbindungen verwendet. Jede erstellt Komponente seinen eigenen ORB, der für die Verschlüsselung der Kommunikation zuständig ist. Die dazu benötigten Einstellungen werden in Abschnitt 5.1.1 erläutert. Außerdem soll ein selbständiger ORB einen Namensdienst anbieten, bei dem sich alle Kommunikationsparteien der SMONA Komponenten registrieren und dadurch auffindbar sind.

IDMEF ist ein Standard, der ein Kommunikationsformat bestimmt. Es wurde konzipiert, um Nachrichten zwischen Intrusion Detection Systemen, die unterschiedliche Informationen enthalten, zu vereinheitlichen. Wie in Abschnitt 4.2.3 näher erläutert, verwendet der Message Manipulator Adapter die Dateiausgabe des Prelude Managers im IDMEF Format, um sicherheitsrelevante Informationen einzulesen. Er verschickt die veränderten Logeinträge im selben Format an den Rich Event Composer weiter. Dadurch kann in einer Weiterentwicklung der SMONA Architektur eine Sicherheitskomponente eingefügt oder angebunden werden, die aus den Meldungen alle Informationen eines Angriff entnimmt und diese erneut analysiert. Die Nachrichten der anderen Adapter entsprechen nicht dem IDMEF Standard. Der Log Adapter verschickt zwei vorkonfigurierte Mitteilungen, die von einem Administrator eingestellt werden. Je nach Aufgabe des Adapters kann hier das IDMEF Format verwendet werden. Der Nachrichtenversand der Push und Pull Adapter wird durch die Ausgabe der jeweils eingestellten Programme bestimmt (siehe Abschnitt 4.2.2).

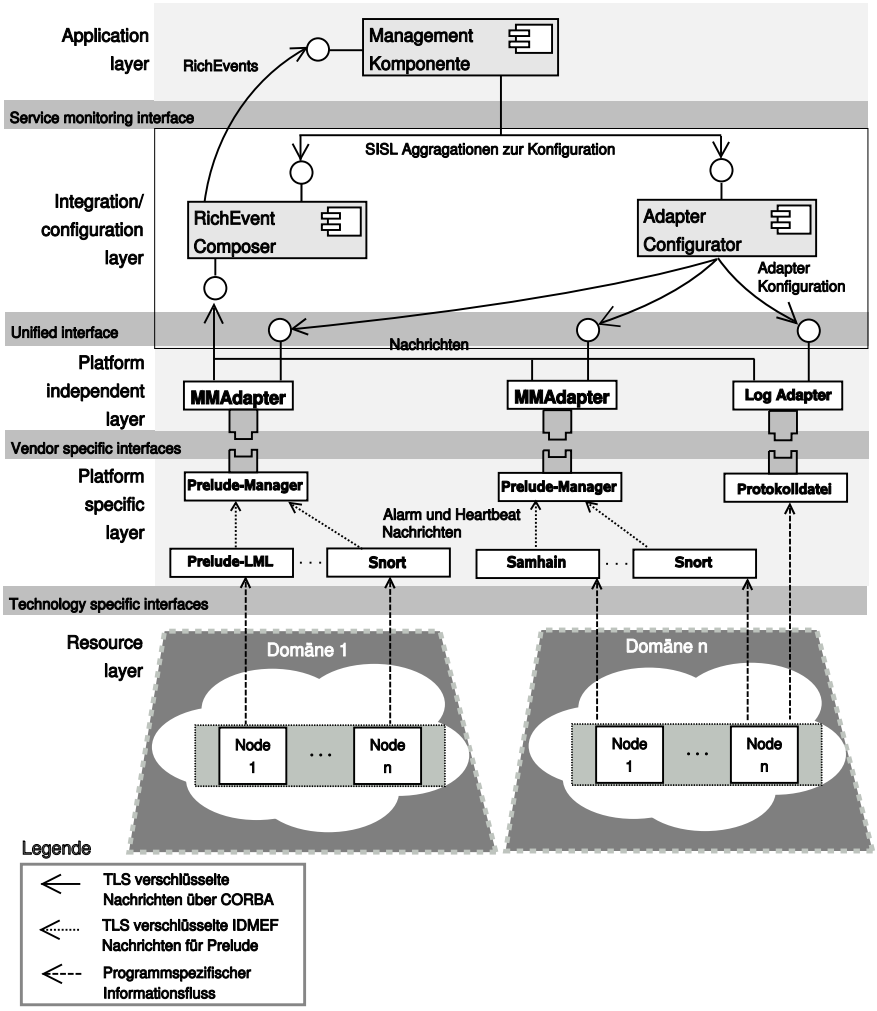

Abbildung 4.2: Die Kommunikation im Sicherheitssystem

Abbildung 4.2 zeigt den Kommunikationsfluss des Sicherheitssystems. Die Sicherung des Datenaustauschs vom IDS bis zur Management Anwendung wird durch die Verwendung der TLS / SSL Verschlüsselung (siehe [DiRe 06]) gewährleistet. Die IDMEF Alarm- und Heartbeatnachrichten der Intrusion Detection Systeme werden dabei direkt an den Prelude Manager geschickt, der im Bild auf der Plattformspezifischen Schicht der SMONA Architektur angesiedelt ist. Die im Schaubild dargestellten Log und Message Manipulator Adapter lesen ihre Informationen aus unverschlüsselten Protokolldateien aus und verschicken ihre Daten über CORBA ORBs an den Rich Event Composer. Das Format der Nachrichten ist für jeden Adapter speziell. Die Message Manipulator Adapter verwendet den IDMEF Standard, während die Nachrichten des Log Adapters bei dessen Konfiguration angegeben werden. Die Kommunikation innerhalb der SMONA findet nur über TLS /

SSL verschlüsselte Verbindungen der ORBs statt. Der Composer versendet Rich Events an die Management Anwendung, deren Objektstruktur in Abbildung 5.1 dargestellt ist. Ebenso enthält das Klassendiagramm 5.1 die Objekte, die von der Management Anwendung zur Konfiguration des Rich Event Composers und des Adapter Configurator verwendet werden. Zur Einstellung der Adapter durch den Configurator werden Property Objekte des SMONA Adapter Frameworks verschickt.

4.1.2 Die SMONA Komponenten der höheren Schichten

Die SMONA Architektur und seine Komponenten befinden sich derzeit noch in der Entwicklung und sie existieren in unterschiedlich detaillierten Entwürfen (vergleiche [DaSa 05], [DSHH 06], [DgFS 07] und [Lang 06]). Die Architektur wird in Abschnitt 3.2 ausführlich beschrieben. Ihre Anforderungen für diese Arbeit sind unter Abschnitt 2.2.4 erläutert. Das Design der SMONA Adapter wird in Abschnitt 4.2 aufgezeigt. Dieser Abschnitt legt die Konzeption der SMONA Komponenten der Integrations- und Konfigurationsschicht und der Anwendungsschicht dar. Dabei werden zuerst an Hand zweier Diagramme die wichtigsten Prozesse erläutert und anschließend auf die Funktionalitäten der einzelnen Einheiten eingegangen, die für diese Arbeit benötigt werden.

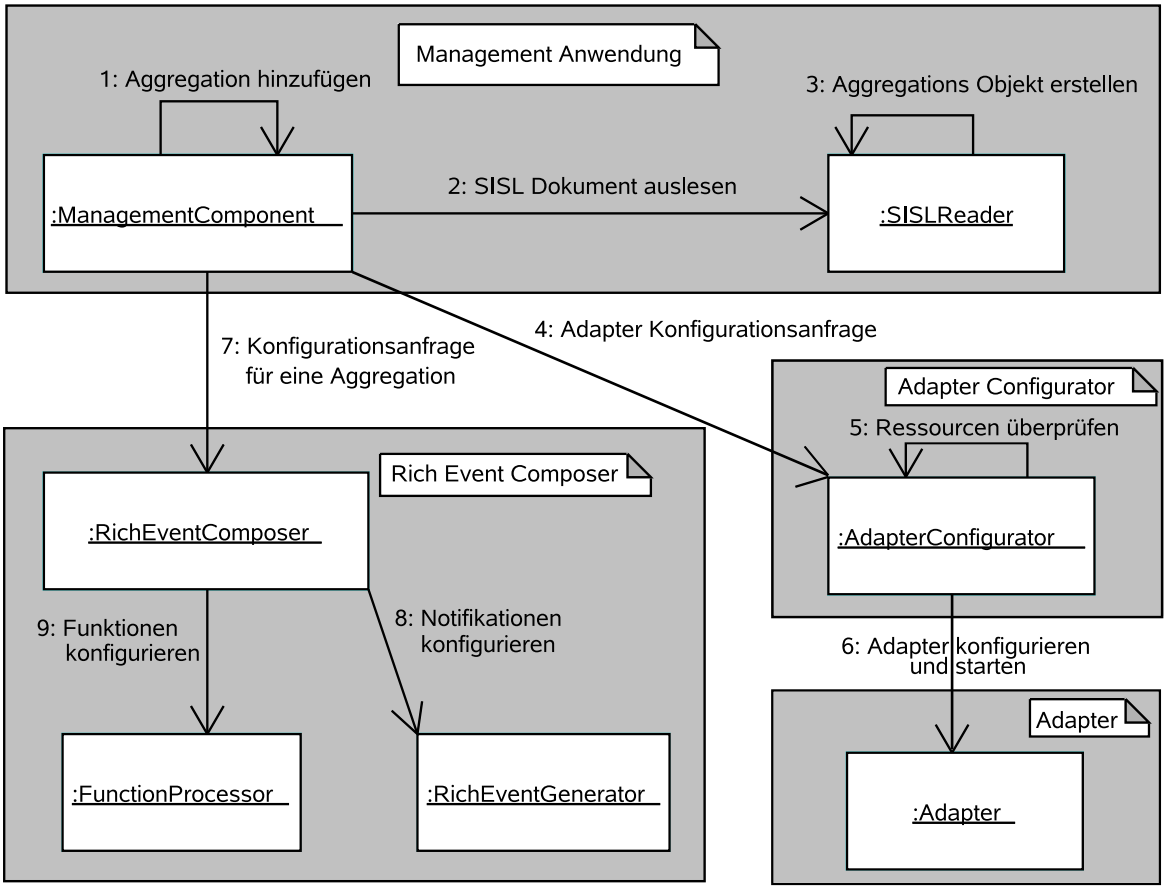

Abbildung 4.3: Verkürztes Kollaborationsdiagramm für das Hinzufügen einer neuen Aggregation

SMONA ist eine verteilte Architektur, die große Datenmengen von verschiedenen Quellen und unterschiedlichen Orten verarbeiten muss. Nicht nur die Ressourcen und ihre Adapter sind auf mehrere Rechner verteilt, sondern auch die Komponenten der Anwendungsschicht und der Integrations- und Konfigurationsschicht können, zum Beispiel zur Lastverteilung, auf verschiedenen Computern arbeiten. In Abschnitt 4.1.1 wurde die Verwendung von CORBA zur standardisierten Kommunikation aller Komponenten motiviert. In dieser Arbeit werden für die Management Anwendung, den Rich Event Composer und den Adapter Configurator Schnittstellen mit der Interface Description Language (IDL) definiert (siehe Anhang C). Sie soll die einzelnen

funktionalen Einheiten austauschbar und flexibler machen. In späteren Implementierungen können dann die Komponenten redundant auf mehrere Systeme verteilt werden. In diesem Entwurf besteht die SMONA aus jeweils einer Management Anwendung, einem Rich Event Composer und einem Adapter Configurator. Die Kommunikation zwischen ihnen wird mit TLS/SSL verschlüsselt. Die Sicherung der Verbindungen wird vom JacORB bereitgestellt und in dessen Konfiguration eingestellt (siehe Abschnitt 5.1.1).

In Abbildung 4.3 ist ein vereinfachtes Kollaborationsdiagramm dargestellt, dass das Hinzufügen einer neuen Aggregation beschreibt. Objekte die nur der Kommunikation dienen sind nicht abgebildet. Die grau hinterlegten Flächen kennzeichnen die Zusammengehörigkeit der einzelnen Komponenten, die in eigenen Programmen entworfen werden. Zu Beginn muss von einem Benutzer der Dateiname eines SISL Aggregationsdokumentes in der graphischen Oberfläche der Management Anwendung angegeben werden. Anschließend wird aus der XML Beschreibung vom *SISLReader* ein Aggregationsobjekt (die Struktur ist in Abbildung 5.1 aufgezeigt) erstellt und an den Adapter Configurator verschickt. Dieser untersucht die Ressourcen der Aggregation auf bereits laufende Adapter und konfiguriert und startet nur neue Adapter. Wird ein Adapter vom CORBA Namensdienst nicht gefunden, oder kann er nicht eingerichtet werden, müssen alle bereits gestarteten Adapter der Aggregation wieder beendet und eine Fehlermeldung ausgegeben werden. Sind alle Adapter erfolgreich gestartet, wird die Aggregation gespeichert. Die Funktions- und Notifikationsobjekte werden von der Management Komponente an den Rich Event Composer versendet, der damit den *RichEventGenerator* und den *FunctionProcessor* einrichtet. Beide überprüfen, ob bereits eine Funktion oder eine Notifikation mit dem selben Namen existiert und generieren gegebenenfalls eine Fehlermeldung. Tritt bei der Konfiguration ein Fehler auf, wird die gesamte Aggregation zurückgesetzt. Es werden alle Funktionen, Notifikationen und Adapter der Aggregation gestoppt und entfernt.

Das Kollaborationsdiagramm 4.4 stellt den Informationsfluss und die Datenaggregation in seinen Grundzügen dar. In diesem Schaubild sind ebenfalls alle Komponenten, die gemeinsam in einem Prozess ausgeführt werden in einer grauen Fläche. Die Kommunikation der Prozesse untereinander erfolgt über CORBA ORBs. In der Abbildung wird die Aggregation durch neue Daten eines Adapters angestoßen, der eine Nachricht über einen Listener an den Rich Event Composer schickt. Dieser speichert die Information und den Namen der Datenquelle und leitet beides an den *FunctionProcessor* weiter. Hier werden alle gespeicherten Funktionen überprüft, ob sie Daten dieser Quelle zur Bearbeitung benötigen. Ist eine Funktion gefunden, wird diese mit Hilfe der *ComposerFunctionLib* ausgeführt. Dazu wird der Methodentyp aus der Funktionsdefinition mit den Methoden aus der Funktionsbibliothek verglichen. Bei Übereinstimmung startet der *FunctionProcessor* die Methode mit allen benötigten Parametern. Erfordert eine Funktion Argumente mehrerer Adapter oder anderer Funktionen, wird der jeweils letzte Wert verwendet. Fehlt dieser, wird die Bearbeitung abgebrochen. Der Rückgabewert der Funktion wird gespeichert. Anschließend überprüft der *FunctionProcessor* rekursiv alle Funktionen und führt diejenigen aus, die den Rückgabewert einer soeben berechneten Methode als Parameter benötigen. Nachdem alle Funktionen, deren Argumente sich geändert haben, ausgeführt wurden, startet der *RichEventGenerator* die Auswertung der boolschen Ausdrücke aller Notifikationen. Dazu werden dem *ConditionInterpreter* alle Nachrichten der Adapter und alle Rückgabewerte der Funktionen zusammen mit dem Ausdruck einer Bedingung übergeben. Wird eine Bedingung erfüllt, generiert der Rich Event Generator ein neues Rich Event und verschickt es an die Management Anwendung.

Es folgt eine detaillierte Konzeption und Beschreibung der Erweiterungen der Service Information Specification Language und der SMONA Komponenten.

SISL Erweiterungen

Die Service Information Specification Language (SISL) wurde in Abschnitt 3.2.1 ausführlich erläutert. In diesem Unterabschnitt werden notwendige Erweiterungen zur Konfiguration der Adapter und zur Einstellung von Funktionen beschrieben. Das vollständige XSD Schema der SISL befindet sich in Anhang B.

In die SISL Grammatik wurde ein Initialparameter (siehe Listing 5.6) eingefügt, der aus einem Parameternamen und einem festen Wert besteht. Er kann in *Source* Attributen mehrmals angegeben werden und dient der zentralen Konfiguration der Adapter. Zum Beispiel wird der Message Manipulator Adapter in Listing 5.11 (Zeile 15 und 16) über den neuen Parameter mit den Speicherorten einer Regeldatei und einer Protokolldatei eingestellt. Um Sicherheitsverletzungen zu vermeiden kann die Einstellung eines Initialparameters lokal durch eine Adapterkonfigurationsdatei überschrieben werden.

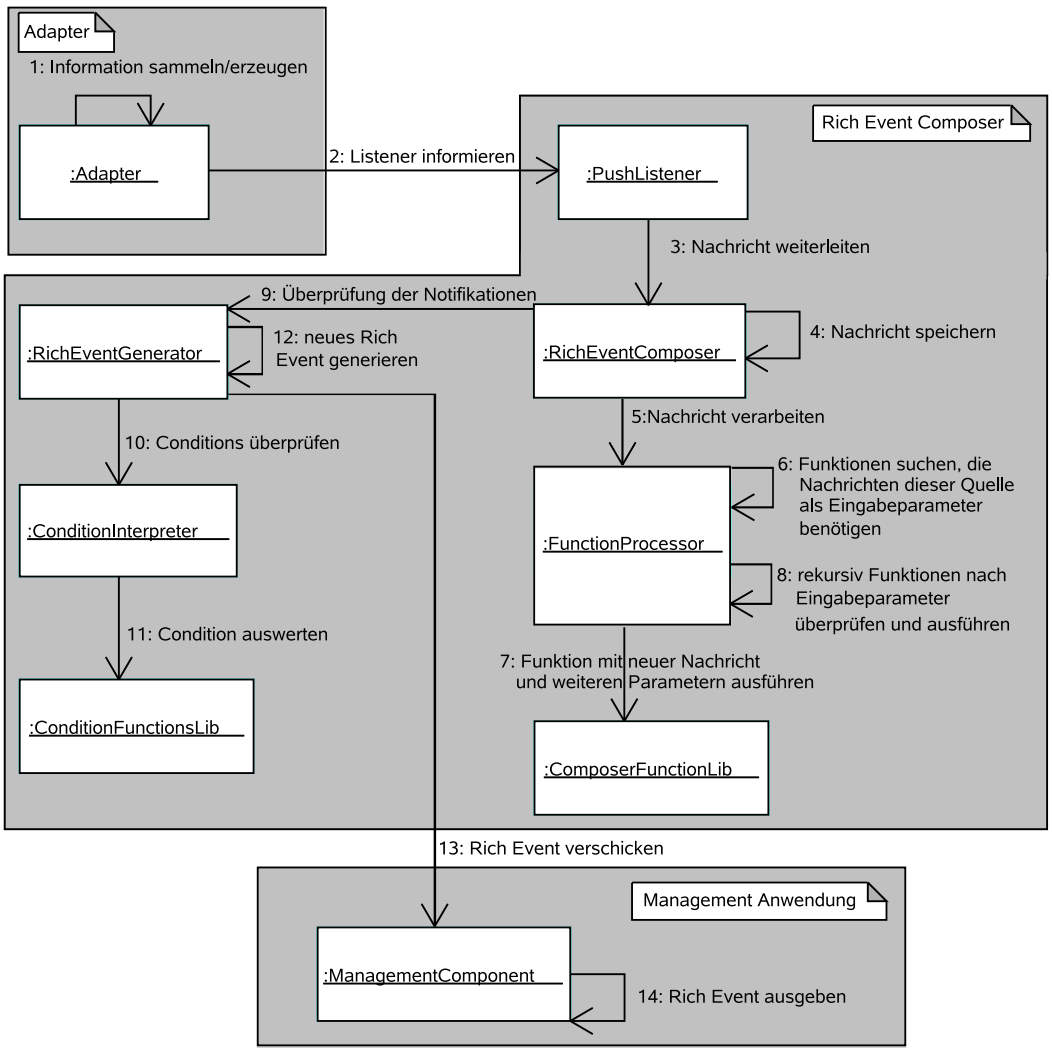

Abbildung 4.4: Verkürztes Kollaborationsdiagramm über die Datenverarbeitung

In der SISL ist vorgesehen, dass Funktionen von Adaptern, der Service Attribute Factory oder dem Rich Event Composer ausgeführt werden. Dafür fehlt ein Attribut, das den Ausführungsort bestimmt. Nachdem in dieser Arbeit alle Funktionen vom Rich Event Composer bearbeitet werden, wurde kein neues Attribut eingeführt. Stattdessen erweiterten sich die Parameter der Funktionen. Bisher können sie nur Referenzen auf andere Funktionen oder auf Ressourcen sein. Für die unten beschriebenen Operationen des *Function Processors* werden weiterhin konstante Parameter benötigt. In Listing 5.8 ist eine exemplarische Funktionsdefinition, die in Zeile 15 den neuen Parameter enthält.

Darüberhinaus wurden die Ausdrücke, die in einer *Condition* angegeben werden können erweitert, so dass der Rückgabewert einer Funktion, die neu bearbeitet wurde, sofort ausgegeben werden kann. Ähnlich dem bestehenden *timeout* Ausdruck, mit dem Rich Events in regelmäßigen Abständen verschickt werden können, ist für den neuen Ausdruck *hasResult* eine eigene Hintergrundfunktion erforderlich. Sie erhält als Parameter eine Referenz und gibt den boolschen Ausdruck *wahr* zurück, wenn die referenzierte SISL Funktion ein neues Ergebnis hat. In Listing 5.9 ist eine beispielhafte Notifikation dargestellt, die den neuen Ausdruck enthält. Sie generiert eine Nachricht, wenn die Funktion *returnAlertMessages* einen neuen Rückgabewert hat. Außerdem wurde der Ausdruck *match* entworfen, der einen regulären Ausdruck (siehe [JPATTERN 07] und Beispiele in Listing 5.12) und eine Referenz auf eine Resource in den Parametern erwartet. Er ist dann wahr, wenn der reguläre Ausdruck in der letzten Nachricht enthalten ist.

Aus den XML Aggreagtionsbeschreibungen generiert die Management Anwendung CORBA Objekte (siehe

Abbildung 5.1), die an den Rich Event Composer und den Adapter Configurator verschickt werden.

Die folgenden Unterabschnitte behandeln die Auswahl und den Entwurf der SMONA Komponenten der höheren Schichten, die teilweise über SISL Aggregationen konfiguriert werden.

Die Service Attribute Factory

In der Integrations- und Konfigurationsschicht befindet sich die Service Attribute Factory. Sie ist verantwortlich für die Weiterverarbeitung von Ereignisdaten, die Konfiguration von Monitoring Optionen und die Wiederverwendung existierender Daten. Diese Funktionalität wird für die Föderation von Intrusion Detection Systemen in Grids vorerst nicht benötigt und eine Konzeption dieser Komponente entfällt. In nachfolgenden Arbeiten kann sie die IDMEF Nachrichten aller IDS auf Zusammenhänge untersuchen.

Die Management Anwendung

Die Management Anwendung ist eine wichtige Komponente in der SMONA Architektur. Sie konfiguriert mit Hilfe einer Aggregation der Service Information Specification Language (SISL, siehe [DaSa 05], [DSHH 06] und 3.2.1) die Komponenten der Integrations- und Konfigurationsschicht. Dafür bietet die Management Anwendung die Möglichkeit, dass Benutzer im Betrieb SISL Dokumente angeben, die anschließend vom *SISL-Reader* eingelesen und in Aggregationsobjekte überführt werden. Die Ressourcenobjekte der Aggregation enthalten neben den in der SISL definierten Attributen zusätzliche Initialparameter (siehe Abschnitt SISL Erweiterungen) und eine Verknüpfung zu einem *Listener* des Rich Event Composers, an den die Adapter ihre Nachrichten schicken sollen. Der Adapter Configurator richtet die Adapter nach einer Anfrage der Management Anwendung ein, startet und stoppt sie. Können einzelne Adapter einer Aggregation nicht gestartet werden, so wird die gesamte Aggregation zurückgesetzt und eine Fehlermeldung ausgegeben. In diesem Fall wird der Rich Event Composer gar nicht erst konfiguriert. Rich Events und Statusinformationen stellt die Management Anwendung auf einer graphischen Oberfläche dar. Im Klassendiagramm 5.2 sind die konzipierten Komponenten der Anwendung dargestellt. Anhang C.1 enthält ein Listing der IDL Schnittstellen Beschreibung.

Der Rich Event Composer

Alle konzipierten Unterkomponenten des Rich Event Composers sind in Abbildung 4.4 im grauen Kästchen dargestellt (siehe auch Klassendiagramm 5.5). Funktionale Einheiten, die ausschließlich der Kommunikation dienen, wurden aus Gründen der Übersichtlichkeit ausgelassen. In dieser Arbeit hat der Composer die Aufgabe die Föderation von Intrusion Detection Systemen zu unterstützen und eine Weiterverarbeitung der Informationen zu ermöglichen. Dafür empfängt er vom Message Manipulator Adapter die gesammelten Nachrichten des Prelude Managers und bietet durch die dynamische Konfiguration rudimentärer Funktionen eine Erweiterungsmöglichkeit. Aus der Einstellung mit Aggregationsobjekten durch die Management Anwendung werden Funktions- und Notifikationsobjekte extrahiert, die dem *FunctionProcessor* beziehungsweise dem *RichEvent-Generator* zur Konfiguration übergeben werden. In diesem Entwurf werden alle definierten SISL Funktionen einer Aggregation vom Function Processor des Rich Event Composers ausgeführt. Sie können kombiniert werden, so dass eine Funktion auf andere Funktionen aufbauen kann und deren Ausgaben weiterverarbeitet. SISL Funktionen besitzen ein *method* Attribut, das die Operation der Funktion bestimmt (siehe Listing 5.8). In den bisherigen Entwürfen der Service Information Specification Language sind nur mathematische Funktionen vorgesehen, die nicht auf die Nachrichten von Intrusion Detection Systemen angewendet werden können. Deshalb werden in Tabelle 4.1 neue, rudimentäre Operationen vorgestellt, die in einer Funktionsbibliothek (*ComposerFunctionLib*, siehe Abbildung 4.4) implementiert werden und eine einfache Weiterverarbeitung der Meldungen erlauben. Darüberhinaus ist der Funktionsumfang durch die Trennung der Funktionsbibliothek vom Function Processor unkompliziert zu erweitern.

Weitere Funktionen könnten in späteren Implementierungen die Nachrichten nach der Zugehörigkeit zu einer Virtuellen Organisation sortieren (zum Beispiel an Hand der Ziel IP Adresse der IDMEF Nachricht, oder der Analyzer ID bei HIDS).

Methodenname	Parameter	Funktion	Beispielhafte Anwendung in dieser Arbeit
contain	• eine oder mehrere Funktions- oder Ressourcenreferenzen • ein regulärer Ausdruck	gibt die eingegebene Nachricht zurück, wenn sie den regulärer Ausdruck beinhaltet	zur einfacheren Verwendung der *match* Operation; Beispiele sind in Anhang D
count	• eine oder mehrere Funktions- oder Ressourcenreferenzen	zählt Nachrichten und gibt die Anzahl zurück	um Alarmnachrichten zu zählen
extract	• eine oder mehrere Funktions- oder Ressourcenreferenzen • ein regulärer Ausdruck	extrahiert eine Zeichenkette, die einem regulären Ausdruck entspricht, aus einer Nachricht	zur Weitergabe bestimmter Nachrichteninhalte an andere Funktionen
match	• eine oder mehrere Funktions- oder Ressourcenreferenzen • ein regulärer Ausdruck	untersucht, ob eine Nachricht einem regulären Ausdruck entspricht und gibt sie bei Übereinstimmung wieder zurück	um bestimmte Alarmmeldungen an andere Funktionen weiterzugeben
timediff	• eine oder mehrere Funktions- oder Ressourcenreferenzen • ein regulärer Ausdruck • eine Zahl, die eine Zeitspanne in Sekunden angibt	kategorisiert Nachrichten an Hand eines regulären Ausdrucks und gibt die letzte Meldung zurück, wenn der Zeitabstand zwischen ihr und der vorherigen Nachricht der selben Kategorie kleiner als eine definierte Zeitspanne ist. Weitere Nachrichten der Kategorie werden unterdrückt bis eine bestimmte Zeit vergangen ist. Sie hat eine ähnliche Funktionsweise wie der Log Adapter in Abschnitt 4.2.4	zur Erkennung von Angriffen, die in kurzer Zeit viele Alarme erzeugen und der Unterdrückung einer wiederholten Berichterstattung; ein Beispiele ist in Abschnitt 6.2.4
timeout	• eine oder mehrere Funktions- oder Ressourcenreferenzen • ein regulärer Ausdruck • eine Zahl, die eine Zeitspanne in Sekunden angibt	überprüft, ob innerhalb einer definierten Zeitspanne keine Nachricht einer definierten Form empfangen wurde	zur Erkennung ausbleibender IDS Heartbeat Meldungen; ein Beispiel ist in Listing 5.8

Tabelle 4.1: Methoden der Rich Event Composer Funktionsbibliothek

Erreicht den Rich Event Composer eine Nachricht eines Adapters, werden alle SISL Funktionen untersucht, ob sie Informationen dieser Quelle als Eingabe verwenden und gegebenenfalls bearbeitet. Da Funktionen kombiniert werden können, muss der Function Processor, nachdem er für die eingehende Nachricht alle betreffenden Methoden ausgeführt hat, rekursiv alle Funktionen überprüfen. Wird die Rückgabe einer soeben bearbeiteten Funktion von einer anderen Funktion benötigt, wird diese gestartet. Hat der Function Processor seine Arbeit beendet, startet der Composer den *RichEventGenerator*. Dieser lässt jede Bedingung aller Notifikationen vom *ConditionInterpreter* auswerten. Nachdem Intrusion Detection Systeme in der Regel „Zeichenketten" ausgeben, die mit boolschen Operatoren nur stark eingeschränkt überprüft werden können, beschränken sich die Ausdrücke, die der Condition Interpreter verarbeiten kann auf *hasResult* und *match* (siehe vorherigen Unterabschnitt „SISL Erweiterungen"). Der Ausdruck *hasResult* ist dann wahr, wenn die angegebene Funktion einen neuen Rückgabewerte hat. Die Bedingung *match* überprüft, ob ein vorgegebener regulärer Ausdruck in der letzten Rückgabe einer Funktion oder der letzten Nachricht eines Adapters enthalten ist. Trifft dies zu, ist der Ausdruck wahr. Beide Ausdrücke verlangen eine Hintergrundfunktion, die von der *ConditionFunctionLib* geliefert und bearbeitet wird. Durch die gemeinsame Auswertung der *Condition* Objekte des Condition Interpreters und der Condition Function Lib, können die auswertbaren Ausdrücke leicht erweitert werden.

Trifft eine Bedingung zu, generiert der Rich Event Generator mit Hilfe der Definition aus dem Notifikationsobjekt ein Rich Event. Es wird mit dem Bezeichner der Benachrichtigung, seiner Erläuterung und einem Array von Namen/ Werte Paaren konzipiert. Der Erläuterungstext entspricht dabei dem Text der Beschreibung der Condition. Das Array stellt sich aus den Ressourcen-, beziehungsweise den Funktionsnamen und ihren Werten zusammen. Es wird jeweils nur der letzte Wert einer Funktion oder Ressource übermittelt.

Der Rich Event Composer benötigt mehrere Schnittstellen (siehe Listing in Anhang C.1) zur Kommunikation mit den Adaptern und der Management Anwendung. Für das Empfangen der Adapternachrichten werden aus dem SMONA Adapter Framework (siehe [Dürr 06]) der *Push-* und der *PullListener* eingebunden und beim CORBA Namensdienst registriert. Zum Hinzufügen und Entfernen der Funktionen und Notifikationen müssen zwei Schnittstellen für die Management Anwendung entworfen werden, die Aggregationsobjekte entgegennehmen und anschließend die gewünschte Aufgabe ausführen. Darüberhinaus werden die Rich Events und Statusmeldungen an eine Schnittstelle der Management Anwendung versandt.

Der Adapter Configurator

Der Adapter Configurator hat die Verwaltung der Adapter zur Aufgabe. Im Klassendiagramm 5.4 sind die Funktionalen Komponenten des Adapter Configurator dargestellt. Für die Kommunikation sind die *ACCCommunication* und *ACCCommunicationServant* zuständig. ACCCommunication registriert den Adapter Configurator beim Namensdienst und stellt die Verbindungen zur Management Anwendung und den Adaptern her. Der ACCCommunicationServant dient der Entgegennahme von Konfigurationsanfragen für Aggregationen. Für jeden Pull Adapter erstellt der Adapter Configurator eine *PullSourceTask*, die in vorkonfigurierten Intervallen aufgerufen wird und den Pull Adapter zur Informationsverarbeitung anstößt. Die Komponente *AdapterConfigurator* beinhaltet die Funktionalität zur Verwaltung der Adapter.

Zur Konfiguration und dem Starten der Adapter einer Aggregation erhält der Adapter Configurator von der Management Anwendung Aggregationsobjekte aus denen er die Ressourcen extrahiert. Nach einem Vergleich mit den bereits laufenden Adaptern werden nur die neuen Ressourcen konfiguriert und gestartet. Ebenso werden nur Adapter beendet, die von keiner anderen Aggregation mehr benötigt werden. Im SMONA Adapter Framework müssen Adapter zuerst lokal initialisiert werden, bevor sie durch den Configurator gestartet werden können. Dabei werden sie mit einem eindeutigen Namen beim CORBA Namensdienst registriert. Der Adapter Configurator sucht beim Namensdienst nach den benötigten Adaptern an Hand der *Source Attribute ID*, die mit dem Adapternamen übereinstimmen muss. In der Konfiguration eines Adapters werden initiale Parameter (siehe Abschnitt SISL Erweiterungen) und eine Verknüpfung mit einem *Listener* des Rich Event Composers übermittelt. Die Initialparameter müssen an jeden Adapter speziell angepasst sein und können zum Beispiel den Speicherort einer Logdatei enthalten. Der Listener dient dem Adapter als Schnittstelle beim Rich Event Composer. In dieser Arbeit werden nur Adapter vom Typ Push- oder Pull Adapter unterstützt, da eine zentrale Konfiguration der verteilten Intrusion Detection Systeme durch Set Adapter, die Schutzrichtlinien und die Souveränität der einzelnen Domänen verletzten können. Der Informationsversandt von Pull Adaptern an den Rich Event Composer wird vom Adapter Configurator, in den jeweils eingestellten Intervallen angestoßen.

Eine dynamische Konfiguration von Adaptern, die bereits laufen, ist in diesem Entwurf nicht vorgesehen, da das SMONA Adapter Framework keine Möglichkeit dazu bietet. Die Konfiguration eines Adapters, der von zwei Aggregationen verwendet wird, wird von der Aggregation eingestellt, die als erstes gestartet wurde.

Zur Bewältigung seiner Aufgabe benötigt der Adapter Configurator zwei Schnittstellen (siehe Listing in Anhang C.3). Sie nehmen Aggregationen an, um Adapter zu starten oder beziehungsweise zu beenden. Zur Ausgabe auf der graphischen Benutzeroberfläche verschickt der Configurator seine Statusinformationen an eine Schnittstelle der Management Anwendung. Die Konfiguration der Adapter wird an ihre Schnittstellen versendet.

In diesem Abschnitt wurde die Konzeption der SMONA Komponenten der Integrations- und Konfigurationsschicht und der Anwendungschicht erläutert. Der folgende Abschnitt geht nun auf die Entwicklung mehrerer Adapter ein.

4.2 Konzeption der Adapter

Die in Abschnitt 2.2.4 beschriebenen Anforderungen gelten allen Adaptern. Im Unterabschnitt 4.2.1 wird das SMONA Adapter Framework vorgestellt und auf bereits vorhandene und zu dieser Aufgabenstellung passende Komponenten untersucht. Abschnitt 4.2.2 geht auf die Anpassungen bereits bestehender Adapter des Frameworks ein. Zur Föderation von Intrusion Detection Systemen wird ein Message Manipulator Adapter (siehe Abschnitt 4.2.3) konzipiert, der Alarm- und Heartbeatnachrichten des Prelude Managers einließt, manipuliert und an den Rich Event Composer verschickt. Darüberhinaus wird in diesem Abschnitt ein Log Adapter (siehe Abschnitt 4.2.4) entworfen, der Protokolldateien auf reguläre Ausdrücke im Zusammenhang mit ihren Zeitstempeln untersuchen kann. Der Log Adapter ist als eine Ergänzung zur Protokollanalyse der Host Intrusion Detection Systeme gedacht und bietet die Möglichkeit Einträge, die einem regulären Ausdruck entsprechen und sich innherhalb einer bestimmten Zeitspanne wiederholen zu melden.

4.2.1 Das SMONA Adapter Framework

Das SMONA Adapter Framework (siehe [Dürr 06]) wurde von Michael Dürr in einem Fortgeschrittenen Praktikum an der Ludwig Maximilians Universität entwickelt. Es unterscheidet zwischen Adaptern und Adapter Sourcen.

Adapter sind die Schnittstellen der Integrations- und Konfigurationsschicht zur Ressource. Sie sind in drei funktional unterschiedliche Klassen eingeteilt: Push, Pull und Set Adapter. Pull Adapter sind konzipiert, um es dem Rich Event Composer zu ermöglichen Daten von Ressourcen abzufragen. Set Adapter bieten die Möglichkeit die Konfiguration einer Ressource zu verändern. Für die vorliegende Arbeit ist die Kategorie der Push Adapter relevant, die der permanenten Überwachung von Netz- und Systemelementen dient und Nachrichten in regelmäßigen Abständen selbständig versenden.

Die **Adapter Source** bewerkstelligt intern den Zugriff auf Ressourcen und ist deshalb für jeden Datentyp einer Ressource unterschiedlich konzipiert und implementiert. Im Beispiel eines Push Adapters implementiert die Klasse *PushSource* allgemeine Methoden. Für den Datentyp *String* existiert eine Klasse *PushStringSource*, die von Push Source erbt und sie um eine Methode erweitert, die Daten vom Typ *String* an die Integrations- und Konfigurationsschicht schickt. Im SMONA Adapter Framework rufen die Adapter Sourcen in regelmäßigen Abständen externe Programme auf, deren Rückgabewerte mittels datenspezifischer Objekte versendet werden.

Adapter sind immer von einem bestimmten Typ (pull, push oder set) und können auch dementsprechend nur eine oder mehrere Adapter Sourcen der selben Kategorie starten. *Sourcen* werden vom Adapter Configurator zentral, mittels der Einstellungen eines SISL Dokuments konfiguriert. Auf Grund der unterschiedlichen Zuständigkeiten in den Domänen eines Grids und somit der Adapter, müssen diese auch lokal mit einer Konfigurationsdatei eingerichtet werden können. Dabei sind die lokalen Einstellungen schwerwiegender und können nicht vom Adapter Configurator überschrieben werden. Die Konfigurationsdatei dient darüberhinaus dem Adapter als Hilfe, welche Adapter Sourcen auf die Konfiguration und den Start vorbereitet werden sollen.

Das Adapter Framework sieht für alle Push und Pull Adapter ein einheitliches *SourceEnviroment* Objekt vor, das alle einstellbaren Variablen einer Adapter Source enthält. Diese sind für den Message Manipulator Adapter und den Log Adapter nicht ausreichend. Es fehlen zum Beispiel Parameter zur Einstellung der Protokolldateien und der Regeldatei. Deshalb wird eine neue Klasse (*AllSourceEnviroment*) entwickelt, die zusätzlich zu den bisherigen Variablen ein Array mit freikonfigurierbaren Namen/ Werte Paare enthält. Der Name kennzeichnet dabei den Bezeichner einer Einstellung und der Wert die Einstellung selbst. Darüberhinaus muss die Initialisierung der neuen Klasse durch die Objekte *XmlPushEnviroment* beziehungsweise *XmlPullEnviroment* angepasst werden. Durch die erweiterten Konfigurationsmöglichkeiten der Adapter Sourcen ist eine Veränderung des XSD Schemas der Konfigurationsdatei unumgänglich. Hier wird ein *attribute* Tag eingeführt, dass den Namen eines Konfigurationsparameters und die Einstellung umfasst. Zur Initialisierung der neuen Adapter Sourcen werden die Klassen *PushSourceFactory* und *PullSourceFactory* modifiziert. Schließlich wird der *AdapterServer* verändert, der das Adapter Programm startet.

Durch die Verwendung des SMONA Adapter Framework werden bereits die ersten Anforderungen aus Abschnitt 2.2.4 erfüllt. Die Schnittstellen zwischen den Adaptern und den Komponenten der Integrations- und Konfigurationsschicht werden in der Interface Description Language (IDL) des CORBA Standards beschrieben und sie können dadurch in verschiedenen Programmiersprachen implementiert werden. Der normalisierte Zugriff auf Adapter wird durch die Verwendung von *Property* Arrays, die aus einfachen Namen/ Werte Paaren bestehen, gewährleistet. Die neu konzipierten Initialparameter und alle anderen Einstellungen eines SISL Resource Objektes transformiert der Adapter Configurator in *Property* Objekte, aus denen sich die Adapter ihre Konfiguration extrahieren. Die Selbstbeschreibung der Adapterfähigkeiten, das dynamische Konfigurieren, der bedarfsgesteuerte Adapterstart und die Verwaltung von Historien sind in den Adaptern dieser Arbeit nicht vorgesehen. Auf Grund der TLS/SSL verschlüsselten Kommunikation von JacORB, wird eine Autorisierung und Authentifizierung aller SMONA Komponenten vorgenommen. Die Anforderung an die Performanz fließt in die Konzeption der zu entwerfenden Adapter ein, ist aber in dieser Arbeit zweitrangig.

Adapter werden beim Starten mit Hilfe einer lokalen Konfigurationsdatei eingerichtet. Darüberhinaus können Sie durch den zentralen Adapter Configurator eingestellt werden, wobei die lokalen Konfigurationen nicht veränderbar sind. Dadurch können Administratoren sensible Einstellungen an den Adaptern ihrer Domäne vornehmen, ohne dass diese von zentraler Stelle aus überschrieben werden. Der Anspruch an Adapter allgemein und wiederverwendbar zu sein wird in den folgenden Abschnitten näher erläutert.

Im nächsten Abschnitt wird die Anpassung der Push und Pull Adapter des SMONA Adapter Frameworks dargestellt.

4.2.2 Anpassungen der bisherigen Push und Pull Adapter

In dieser Arbeit wird davon ausgegangen, dass alle Ressourcen, die zur Überwachung von Grids auf sicherheitsrelevante Ereignisse den Datentyp „Zeichenkette" zurückgeben. Deshalb werden nur Adapter, die diesen Datentyp verarbeiten, betrachtet. Set Adapter, die von zentraler Stelle aus Einstellungen an den Intrusion Detection Systemen vornehmen können, verletzten die Souveränität der Domänen und werden aus diesem Grund nicht unterstützt.

Die Push und Pull Adapter des SMONA Adapter Frameworks von Michael Dürr (siehe [Dürr 06]) rufen in regelmäßigen Abständen externe Programme auf und verschicken deren Rückgabe an den Rich Event Composer. Sie können die Föderation von Intrusion Detection Systemen in Grid Infrastrukturen unterstützten, in dem sie zum Beispiel die Ausgabe von Shell Scripten, an die SMONA Architektur weiterleiten. Dazu sind mehrere Anpassungen zur Kompatibilität mit den in Abschnitt 4.1.2 beschriebenen SMONA Komponenten notwendig. Außerdem wird die Anforderungen, dass Adapter durch eine Konfigurationsdatei und durch den Adapter Configurator eingerichtet werden können (siehe Abschnitt 2.2.4) umgesetzt.

Derzeit ist keine Verschlüsselung der Meldungen durch die Push und Pull Adapter vorgesehen. Durch die Verwendung der TLS/SSL Verschlüsselung des JacORBs kann diese Anforderung, ohne Änderungen am Adaptercode vorzunehmen, erfüllt werden. Abgesehen davon wird dadurch eine Autorisierung und Authentifizierung erreicht. Weiterhin wird der Anspruch der lokalen und der zentralen Konfigurationsmöglichkeit an Adapter erhoben. Zu dessen Umsetzung müssen die Klassen *PushStringSource* und *PullSource*, die intern im Adapter die Konfiguration des Adapter Configurators übernehmen, erweitert werden, so dass alle Einstellungen auch

zentral vorgenommen werden können. Lokale Einstellungen dürfen auch in diesem Fall nicht überschrieben werden. Die neue Klasse *AllSourceEnviroment* ist mit der alten Klasse *SourceEnviroment* kompatibel, so dass in den Adaptern keine Modifikationen notwendig sind.

Aufbauend, auf die Erläuterungen zum SMONA Adapter Framework dieses Abschnitts legt der folgende Unterpunkt den Entwurf eines Message Manipulator Adapters dar, der Protokolleinträge eines Intrusion Detection Systems auslesen, manipulieren und weiterleiten soll.

4.2.3 Der Message Manipulator Adapter

Im Unterabschnitt „Der Prelude Manager" wird zuerst auf die Ausgabemöglichkeiten des Prelude Managers eingegangen und dabei eine adäquate Schnittstelle für den Message Manipulator Adapter gesucht. Darauf folgt sein Entwurf und eine Beschreibung seiner Komponenten und seines Aufbaus. Der Message Manipulator Adapter kann nicht nur zur Anbindung der Prelude Architektur an die SMONA verwendet werden, sondern auch Protokolldateien anderer Intrusion Detection Systeme auslesen, deren Einträge manipulieren und anschließend die Ergebnisse an den Rich Event Composer weiterleiten.

Der Prelude Manager

Die Prelude Architektur wurde in Kapitel 3.3.6 bereits in ihren Grundzügen erläutert. Dieser Abschnitt erörtert die Eigenschaften des Prelude Managers, seine Ausgabeformate und ihre Vor- und Nachteile.

Zur Föderation der Intrusion Detection Systeme ist in jeder administrativen Domäne eines Grids ein Prelude Manager vorgesehen, der die Alarm und Heartbeat Nachrichten der Sensoren konzentriert. Im Prelude Hybrid IDS ist der Prelude Manager die zentrale Komponente, die Informationen der verteilten Sensoren sammelt, normalisiert, filtert, dauerhaft speichert und/oder an andere Manager weiterleitet. Er ist über TLS (siehe [DiRe 06]) verschlüsselte Verbindungen mit seinen Sensoren verbunden, die ihm IDMEF Nachrichten (siehe Abschnitt 3.3.5) über UNIX Domain Sockets oder Netzwerkschnittstellen schicken. Das Nachrichtenformat ist binär und wurde vom Prelude Entwicklerteam in Anlehnung an den IDMEF Standard konzipiert. Empfangene Alarm und Heartbeat Meldungen werden je nach Priorität disponiert und können anschließend anhand einfacher Regeln gefiltert werden. Beispielsweise können Nachrichten, die nicht das Grid betreffen, aussortiert werden. Eine Manipulation der Daten während dem Filterprozess ist nicht möglich. Zur Speicherung der Ereignisse gibt es Ausgabeplugins für *Dateien*, *Datenbanken* (MySQL und PostgreSQL), *SMTP* (nur käuflich erhältlich) und *Relaying* (zum Weiterleiten an andere Manager). Für die Ausgabe in Dateien existieren zwei Formate: als Klartext, in Anlehnung an den IDMEF Standard und als IDMEF normiertes XML.

Für den Message Manipulator Adapter wird die XML Ausgabe in eine Datei verwendet. Die Datenbankschnittstelle ist langsam, da die Nachrichten in mehreren Tabellen gespeichert sind und erst rekonstruiert werden müssten. Außerdem wird das Prelude System ständig weiterentwickelt, so dass sich das Datenbankschema in zukünftigen Versionen ändern kann. Die IDMEF Meldungen aus einer Klartextdatei auszulesen ist ebenfalls langsam, da die Nachricht auf mehrere Zeilen verteilt sind und erst zusammengefügt werden müssen. Die SMTP Schnittstelle ist nur käuflich erhältlich und kann somit im Rahmen dieser Arbeit nicht verwendet werden. Das Relaying Plugin verschickt binäre Nachrichten dessen Format in der *libprelude* Funktionsbibliothek definiert ist. Diese ist in der Programmiersprache C geschrieben und kann nicht in das in Java programmierte SMONA Framework importiert werden.

Aus Gründen der Performanz und der Standardkonformität wird in dieser Arbeit die Dateiausgabe im IDMEF XML Format verwendet. Darüberhinaus ist der Message Manipulator Adapter durch das Einlesen von Protokolldateien nicht nur auf das Prelude System spezialisiert und kann auch andere Intrusion Detection oder Monitoring Systeme als Eingabe verwenden.

Entwurf des Message Manipulator Adapters

In jeder administrativen Domäne eines Grids befindet sich ein Prelude Manager, der die Ereignisse der verteilten Intrusion Detection Systeme konzentriert. Die Hauptaufgabe des Message Manipulator Adapters ist

das Einlesen, die Manipulation und die Weiterleitung der vom Prelude Manager gespeicherten Alarm und Heartbeat Nachrichten.

In Folge der Dringlichkeit der IDS Meldungen wird der Adapter als Push Adapter konzipiert, der selbständig und in regelmäßigen Abständen die Mitteilungen des Prelude Managers einliest und verarbeitet. Der zeitliche Abstand wird in der Konfiguration angegeben. Weiterhin wird der Adaptertyp, der eindeutige Adaptername, eine Beschreibung und die Speicherorte der Protokolldatei und einer Datei mit Manipulationsregeln eingestellt. Mit dem Namen wird der Adapter beim Namendienst ORB registriert und ist dort für den Adapater Configurator auffindbar. Das Einlesen der Protokolldatei ist nicht auf ein bestimmtes Format angewiesen, solange sich ein Eintrag in einer Zeile befindet. Dadurch können auch Logdateien anderer Programme zur Eingabe verwendet werden. Die Regeldatei besteht aus Tupeln, die einen regulären Ausdruck und eine Zeichenkette beinhalten. Ein Administrator sollte die Manipulationsregeln, die beim Initialisieren des Adapters eingelesen werden, nach seinen Bedürfnissen selbst erstellen. Die Syntax der verwendbaren regulären Ausdrücke ist in der Java Dokumentation (siehe [JPATTERN 07]) definiert. Beispiele sind in Listing 5.12 dargestellt und werden in Abschnitt 5.2.5 erläutert. Jeder Logeintrag wird nach den Ausdrücken durchsucht und bei Übereinstimmung der entsprechende Teil durch die zugehörige Zeichenkette ersetzt. Manipulierte Nachrichten werden direkt an den Rich Event Composer versendet.

Die Aufgaben des Adapters werden im nachfolgenden Unterabschnitt auf drei funktionale Einheiten verteilt und dort genauer erläutert.

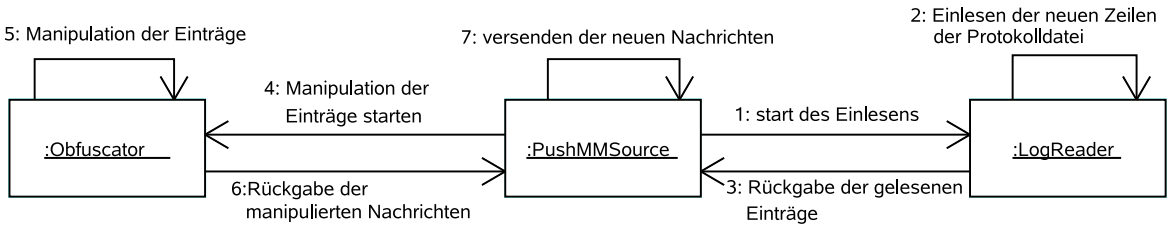

Abbildung 4.5: Vereinfachtes Kollaborationsdiagramm des Message Manipulator Adapters

Der Aufbau und die Komponenten des Message Manipulator Adapters

Der Message Manipulator Adapter gliedert sich hauptsächlich in drei funktionale Komponenten: die *PushMMSource*, den *LogReader* und den *Obfuscator*. In Abbildung 4.5 ist in einem Kollaborationsdiagramm die Datenverarbeitung des Adapters dargestellt. Komponenten, die nicht an der Bewältigung der Hauptaufgabe beteiligt sind, sind nicht abgebildet.

Die zentrale Komponente des Adapters ist die **PushMMSource**. Sie initialisiert und konfiguriert die anderen Komponenten. Die Push MM Source lässt neue Protokolleinträge vom Log Reader einlesen und anschließend vom Obfuscator manipulieren. Darauf verschickt sie die Nachrichten zusammen mit dem Adapternamen über eine TLS/SSL verschlüsselte Verbindung an den Push Listener des Rich Event Composers. Nach dem Versenden wartet die Push MM Source eine Zeit lang. Die Dauer wird über die Konfiguration eingestellt. Schließlich beginnt die Verarbeitung wieder von vorne.

Der **LogReader** liest beim ersten Start alle Nachrichten zeilenweise aus der Logdatei und leitet je nach Konfiguration die Einträge der letzten zehn Minuten weiter. Anschließend werden nach festen Zeitintervallen nur die neuen Einträge eingelesen und an die Push MM Source zurückgegeben. Die Logdatei muss einen Eintrag pro Zeile enthalten. In dieser Arbeit wird das standardisierte IDMEF XML Format vom Prelude Manager verwendet. Der Zeitabstand zwischen den Lesezugriffen wird vom Adapter Configurator während der Konfiguration des Adapters eingestellt und muss sich innerhalb eines von der Adapterkonfiguration vorgegebenen Intervalls halten.

Nachdem die neuen Einträge vom Log Reader eingelesen wurden, verändert der **Obfuscator** die IDMEF Nachrichten entsprechend voreingestellter Regeln. Die Regeln zur Manipulation sind als reguläre Ausdrücke in einer Datei anzugeben und werden beim Start des Adapters konfiguriert. Die Grammatik der regulären Ausdrücke richtet sich nach der JAVA Pattern Definition (siehe [JPATTERN 07]). In dieser Konzeption obliegt die

Verantwortung zur Informationsveränderung beim zuständigen Administrator und es wird nicht nachgeprüft, ob die Daten nach der Manipulation denselben Informationsgehalt haben. Alternativ könnten die Nachrichten vor der Manipulation in IDMEF Objekte gespeichert werden. Dies verringert aber die Performanz und schränkt die Möglichkeiten der Verschleierung ein. Darum wird die direkte Manipulation und Weiterleitung als Zeichenkette bevorzugt.

Die Umsetzung des in diesem Abschnitt beschriebenen Entwurfs befindet sich im Kapitel 5.2.5. Der nächste Abschnitt legt die Konzeption des Log Adapters dar.

4.2.4 Der Log Adapter

Bisherige Host Intrusion Detection Systeme, die der Überwachung von Protokolldateien dienen überprüfen einzelne Einträge nur an Hand eines regulären Ausdrucks und melden jede Übereinstimmung, ohne Beziehungen zwischen den Nachrichten zu beachten. Dies ist ein Mangel, der durch diesen Adapter behoben wird. Im Entwurf des Log Adapters wird davon ausgegangen, dass ein Angriff mehrere ähnlich lautende Nachrichten zur Folge haben kann. Deshalb kategorisiert er bestimmte Einträge von Protokolldateien nach einem regulären Ausdruck und vergleicht die Zeitstempel der letzten beiden Nachrichten einer Sparte. Ist der Zeitabstand innerhalb einer definierten Spanne, wird der Rich Event Composer informiert. Trifft innerhalb kurzer Zeit eine dritte Meldung der selben Rubrik ein, wird keine weitere Nachricht generiert. Einzelne Meldungen einer Kategorie, die einen festgelegten Zeitabstand zu den anderen Nachrichten der selben Sparte überschreiten werden ebenfalls dem Composer berichtet. Mit diesem Analysealgorithmus können sich wiederholende Angriffe, wie sie zum Beispiel der SSH Wurm durchführt, erkannt werden. Durch die Unterdrückung von Folgenachrichten werden die Systemressourcen nicht unnötig in Anspruch genommen.

Der nächste Unterpunkt erläutert den Entwurf des Log Adapters und geht genauer auf den Algorithmus ein, bevor die Komponenten dargestellt werden.

Entwurf des Log Adapters

Der Log Adapter ist ein Push Adapter, der ständig eine Protokolldatei überwacht. Er wird mit einem eindeutigen Bezeichner, dem absoluten Dateinamen der Logdatei, zwei regulären Ausdrücken, zwei Nachrichtentexten und zwei Zeitintervallen konfiguriert. Den Bezeichner übermittelt der Adapter bei der Registrierung dem CORBA Namensdienst, damit er für den Adapter Configurator auffindbar ist. Der erste reguläre Ausdruck dient dem herausfiltern bestimmter Nachrichten aus der Protokolldatei. Zum Beispiel aller Meldungen eines Programms. Der weitere Ausdruck wird verwendet, um eine Zeichenkette aus dem Eintrag zu extrahieren, die zur Unterscheidung der Nachrichtentypen herangezogen wird. Ein Intervall bestimmt die Periode, in der die Protokolldatei ausgelesen wird. Das Zweite definiert den Zeitabstand, der zwischen zwei Logeinträgen der selben Kategorie verstreichen muss, damit sie nicht zu einem einzigen Auslöser gewertet werden. Ein einziger Auslöser bedeutet hier, dass zum Beispiel aufgrund eines Angriffs beide Nachrichten erzeugt wurden. Werden zwei Meldungen einem Angriff zugeordnet, wird ein Nachrichtentext verschickt. Handelt es sich bei dem Protokolleintrag um eine Nachricht, die mit keinem Angriff korrespondiert, wird der zweite Nachrichtentext an den Rich Event Composer übermittelt.

Das Aktivitätsdiagramm aus Abbildung 4.6 beschreibt den Algorithmus des Log Adapters. Zu Beginn wird ein Eintrag aus der Protokolldatei ausgelesen und überprüft, ob er einen regulären Ausdruck beinhaltet. Ist der Ausdruck nicht enthalten, wir der Eintrag verworfen. Ansonsten wird mit Hilfe eines zweiten regulären Ausdrucks eine Zeichenkette aus dem Eintrag extrahiert, der die Nachricht kategorisiert. Anschließend wird der Zeitstempel der letzten Nachricht dieser Sparte gesucht. Wird er nicht gefunden, wird die Zeichenkette mit der Zeit ihres Fundes gespeichert und eine Nachricht an den Rich Event Composer übermittelt, die bei der Initialisierung für ein einzelnes Ereignis konfiguriert wurde. Trat die Zeichenkette in einer vorhergehenden Nachricht auf, werden der damalige mit dem jetzigen Zeitstempel verglichen. Ist der Zeitabstand größer als das eingestellte Intervall, wird der Zeitstempel aktualisiert und der Composer mit dem Nachrichtentext eines einzelnen Vorfalls informiert. Sind beide Zeitstempel innerhalb des Intervalls, werden die Einträge als Nachrichten eines Angriffs gewertet. In diesem Fall wird überprüft, ob der Composer bereits über den Angriff

unterrichtet wurde. Trifft dies nicht zu wird der Nachrichtentext für diesen Umstand versendet. In jedem Fall wird der Zeitstempel für diese Zeichenkette aktualisiert.

Nachdem der Text beider Nachrichten lokal eingestellt werden kann und der Protokolleintrag nicht verschickt wird, müssen ihre Inhalte nicht aus Datenschutzgründen manipuliert werden.

Der Aufbau und die Komponenten des Log Adapters

Der oben beschriebene Algorithmus wird in drei Komponenten konzipiert: der *PushLogSource*, dem *LogReader* und dem *EntryAnalyzer*.

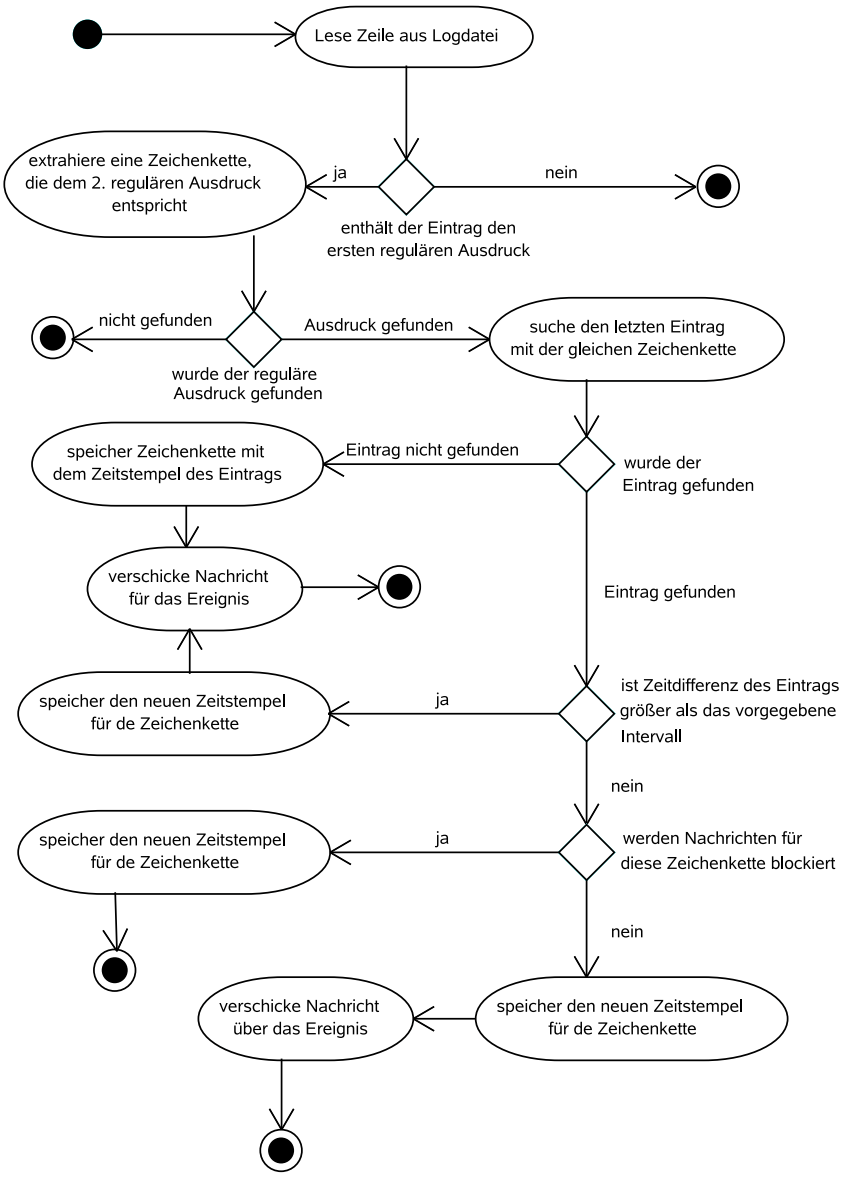

Abbildung 4.6: Algorithmus des Log Adapters

Die **PushLogSource** initialisiert und konfiguriert die beiden anderen Komponenten. Sie startet den Log Reader, um neue Einträge aus der Protokolldatei auszulesen und gibt seine Rückgabewerte an den Entry Analyzer weiter. Hat der Entry Analyzer Nachrichten generiert, werden diese zusammen mit dem Adapternamen an den Rich Event Composer verschickt. Anschließend verharrt die Push Log Source für die Dauer des eingestellten Intervalls im Wartezustand und das Einlesen und Verarbeiten der Protokolleinträge beginnt wieder von vorne.

Beim ersten Start des **LogReader** werden die Zeitstempel aller Einträge der Protokolldatei überprüft und alle Nachrichten der letzten 10 Minuten weiterverarbeitet. Dieser Zeitabschnitt ist im Programmcode in den Konstanten konfigurierbar. Anschließend werden nur die neuen Einträge eingelesen. Der Log Reader verwirft alle Zeilen der Logdatei, die nicht den ersten regulären Ausdruck beinhalten und extrahiert aus den übrigen Meldungen eine Zeichenkette mit Hilfe des zweiten regulären Ausdrucks. Die Zeichenkette und den Zeitstempel der Nachricht übergibt er an die Push Log Source. Die Einträge der Protokolldatei müssen dem Syslog Standard entsprechen und zu Beginn einen Zeitstempel enthalten.

Die **EntryAnalyzer** Komponente untersucht, ob die ihr übergebenen Zeichenketten in einer vorherigen Nachricht enthalten waren und überprüft gegebenenfalls die Zeitstempel. Erkennt der Entry Analyzer, dass mehrere Zeichenketten zu einem Angriff gehören und wurde der Rich Event Composer darüber noch nicht informiert, wird eine Nachricht generiert. Steht die Zeichenkette in keinem Zusammenhang mit einem anderen Angriff, wird eine andere Meldung erzeugt (siehe Abbildung 4.6). Die Mitteilungen werden an die Push Log Source überreicht, die sie an den Composer versendet.

Der in diesem Abschnitt konzipierte Adapter kann insbesondere dazu eingesetzt werden, bestimmte DoS Angriffe und SSH Wurm Angriffe zu erkennen. Seine Implementierung ist in Abschnitt 5.2.5 erläutert.

Der anschließende Abschnitt wählt drei Intrusion Detection Systeme zur Überwachung der Grid Middleware Globus aus und beschreibt den Entwurf von Analyseregeln.

4.3 Föderation der Intrusion Detection Systeme zur Überwachung des Globus Toolkits

Dieser Abschnitt thematisiert die Konzeption von Analyseregeln für drei Intrusion Detection Systeme, die an die Überwachung des Globus Toolkits angepasst sind. Zu Beginn wird in einer kurzen Sicherheitsanalyse des Toolkits auf Probleme der Grid Middleware eingegangen (siehe Abschnitt 4.3.1). Darauf aufbauend wird in Abschnitt 4.3.2 die Prelude Architektur und ihre Möglichkeiten erläutert, den Bedrohungen entgegen zu treten. Anschließend werden drei IDS (Prelude-LML, Samhain und Snort) erörtert und aufgezeigt, wie mit ihnen das Globus Toolkit überwacht werden kann. Abschnitt 4.3.3 geht dabei konkret auf die Protokollanalyse durch Prelude-LML ein. Nachfolgend (siehe Abschnitt 4.3.4) werden Dateien zur Integritätstestüberwachung mit Hilfe von Samhain ausgewählt. Abschließend für dieses Kapitel beschreibt Abschnitt 4.3.5 die Überwachung des Globus Netzverkehrs mit Snort.

4.3.1 Sicherheitsanalyse des Globus Toolkits

Das Security Engineering (siehe [Ecke 00]) beschäftigt sich mit dem Erfassen sicherheitsrelevanter Systemeigenschaften, der Ermittlung des Schutzbedarfs, der Bedrohungs- und Risikoanalyse, Sicherheitsstrategien und der Entwicklung von Sicherheitsarchitekturen. In der Strukturanalyse wird eine Ist-Aufnahme der vorhandenen Infrastruktur vorgenommen, in die auch bereits bestehende Sicherheitsmechanismen einfließen. Zur Ermittlung des Schutzbedarfs werden Eigenschaften des System, seine Einsatzumgebung und sein Verwendungszweck erfasst. Anschließend werden in der Bedrohungsanalyse Gefährdungsbereiche klassifiziert und in einer Bedrohungsmatrix oder einem Bedrohungsbaum festgehalten. Darauf aufbauend werden in der Risikoanalyse die identifizierten Bedrohungen bewertet. Darin fließen Bedrohungsrisiken und Wahrscheinlichkeiten für das Eintreten einer Bedrohung ein. Darüberhinaus wird der potentielle Schaden von Bedrohungen eingeschätzt. Aus den vorherigen Analysen werden Sicherheitsstrategien entwickelt, die der Abwehr der gefundenen Bedrohungen dienen.

Eine genaue Sicherheitsanalyse des Globus Toolkits kann auf der oben beschriebenen Grundlage nicht durchgeführt werden, da es sich in dieser Arbeit um ein abstraktes System handelt und keine Architektur vorgegeben ist. Deshalb beschränkt sie sich alleine auf das Globus Toolkit und geht auf keine Infrastruktur ein. Außerdem wird nur ein grober Überblick über die Bedrohungen gegeben.

Das Globus Toolkit ist ein spezielles, verteiltes System, dessen Komponenten weit verteilt sind und trotzdem eng zusammen arbeiten. Durch diese Eigenschaften ist das Toolkit besonders durch Netzangriffe bedroht. Nach

dem Ausspionieren der Infrastruktur (zum Beispiel durch sniffing) kann mit DoS und DDoS Angriffen versucht werden, wichtige Komponenten (zum Beispiel Gateways zwischen den Netzen) oder Dienste auf besonderen Komponenten (zum Beispiel der Grid Resource Allocation Management Dienst auf einem Supercomputer) zu blockieren. Darüberhinaus können Angreifer versuchen, sich mit Hilfe von Würmern und Trojanern Zugriff auf Grid Ressourcen zu verschaffen und diese anschließend ausnutzen oder Daten ausspionieren. Das Globus Toolkit ist eine komplexe Werkzeugsammlung, die viele Dienste über Netzwerkschnittstellen anbietet. Diese sind durch Buffer Overflows bedroht. Das Toolkit verfügt über verschiedene Sicherheitsmechanismen (siehe Abschnitte 3.1.4 und 3.1.6), um Angriffe abzuwehren. Zertifikate und Schlüssel sind ein Teil davon. Ein Angreifer kann versuchen Zugriff, auf das Zertifikat und den privaten Schlüssel eines Grid Benutzers zu bekommen und sich mit diesen beim Grid anmelden. Als Benutzer des Globus Toolkits kann er dann das System von innen mit Missbräuchen schädigen, wichtige Daten stehlen oder versuchen Ressourcen zu blockieren.

Aus dieser Sicherheitsanalyse ergibt sich die Anforderung einer umfassenden Überwachung des gesamten Systems. Dazu gehört nicht nur die Analyse des Toolkits durch ein spezialisiertes Intrusion Detection System, sondern es müssen auch unterschiedliche Erkennungsmechanismen zum Einsatz kommen. Die Dateien des Toolkits müssen mit der Integritätsanalyse überwacht werden, während System- und Globusprotokolle mit der Protokollanalyse auf Angriff- und Missbrauchsspuren untersucht werden. Schließlich muss die Paketanalyse den Netzverkehr des Globus Toolkits und seiner Komponenten kontrollieren. Der folgende Abschnitt beschreibt, wie die Prelude Architektur zu diesem Zweck eingesetzt werden kann.

4.3.2 Voraussetzungen der Prelude Architektur

In Abschnitt 3.3.6 wurden die Grundlagen der Prelude Architektur erläutert. Dieser Abschnitt diskutiert die Eigenschaften des Systems, die für diese Arbeit von Bedeutung sind. Der Datenaustausch der Intrusion Detection Systeme zum Prelude Manager wurde bereits in Kapitel 4.1.1 kurz beschrieben und wird im Folgenden nicht näher betrachtet.

Die Verwendung der Prelude Architektur bietet mehrere Vorteile. Das Prelude Team hat bereits einheitliche Schnittstellen in verschiedene Sicherheitslösungen integriert und durch die Verwendung von IDMEF standardisiert. Es ist nicht notwendig, für jeden Sensor einen eigenen Adapter zu entwickeln, der auf IDMEF basiert und die unterschiedlichen Ausgaben der Intrusion Detection Systeme vereinheitlicht. Durch den Einsatz des Prelude Managers können kooperierende IDS zur Überwachung einer Domäne installiert werden. Die Filterfunktion des Managers kann alle Alarmnachrichten auf einer Komponente sammeln und anschließend die Nachrichten, die nicht von zentraler Bedeutung sind, aussortieren. Bei der Kooperation von IDS zur Überwachung eines Grids kann diese Funktionalität verwendet werden, um IDMEF Nachrichten, die nicht das Grid betreffen, auszufiltern. Alle Nachrichten, die das Grid betreffen, werden in die XML Protokolldatei geschrieben. Ein weiterer Aspekt ist die andauernde Weiterentwicklung von Prelude und die zunehmende Integration neuer Sensoren in das System. Neue Intrusion Detection Systeme die in die Prelude Architektur eingebunden werden, benötigten keine Neuentwicklung eines SMONA Adapters und können sofort genutzt werden. Darüberhinaus liefert das Prelude Projekt ein Framework in den Programmiersprachen C, Python, and Perl und eine Anleitung, um eigene Sicherheitslösungen auf einfache Art in das Prelude System mit Hilfe der libprelude Bibliothek zu integrieren. Die Prelude Architektur eignet sich deshalb gut für die Föderation von Intrusion Detection Systemen und zur Erkennung von Angriffen auf eine Domäne. Durch die Implementierung des Message Manipulator Adapters wird die Funktionalität um den Aspekt der Nachrichten Manipulation erweitert. Unter Berücksichtigung des Datenschutzes und organisationsinterner Sicherheitsrichtlinien können damit die Alarmnachrichten an eine zentrale Autorität des Grids weitergeleitet werden.

Aus dem in Abschnitt 3.3.6 beschriebenen Sicherheitslösungen werden in den nächsten Abschnitten drei IDS (Prelude-LML, Samhain, Snort) ausgewählt, die sich besonders für die Überwachung des Globus Toolkits eignen. Dabei wird zuerst im folgenden Abschnitt der Entwurf von Protokollanalyse Regeln thematisiert.

4.3.3 Konzeption von Regeln zur Protokollanalyse

Die Protokollanalyse war die erste Methode der Einbruchserkennungsverfahren (siehe Abschnitt 3.3.4). Sie ist immernoch ein wichtiger Bestandteil der IDS. In diesem Abschnitt werden Regeln zur Protokollanalyse

für Prelude-LML (siehe [PLML 07]) entwickelt, die das Globus Toolkit überwachen. In der Literatur werden bisher Sicherheitslöcher und die darauf folgenden Logeinträge des Globus Toolkit nicht betrachtet. Deshalb werden hier einfache Regeln entworfen, die nicht zwangsläufig auf einen Angriff hindeuten.

Der Java Webservice Core des Globus Toolkits verwendet zum Erstellen der Protokolle die Log4j ([LOG4J 07]) Bibliothek der Apache Software Foundation (siehe [APACHE 07]). Die Protokollierung kann in den Dateien `container-log4j.properties` und `log4j.properties` (im Globus Toolkit Stammverzeichnis) eingestellt werden. Die Ereignisse schreibt das Toolkit in unterschiedlich formatierten Einträgen in die Datei container.log. Die Globus Dienste GridFTP, RFT und MDS schreiben in der Standardinstallation keine Ereignisse in die Protokolldatei und können deshalb nicht untersucht werden.

Der Regelsatz zur Überwachung des Toolkits besteht aus vier einfachen Regeln, die das Starten und Stoppen, auftretende Java Ausnahmen und Statusinformationen des GRAM analysieren. Die ersten beiden Regeln lösen einen Alarm niedriger Priorität aus, wenn das Globus Toolkit gestartet oder beendet wird. Treten im Betrieb des Toolkits Fehler auf, werden diese ausführlich in der Logdatei dokumentiert und Prelude-LML schickt einen Alarm mittlerer Priorität an den Prelude Manager. In der IDMEF Nachricht steht ein Teil der Fehlermeldung. Die letzte Protokollanalyseregel sucht nach Einträgen des Grid Resource Allocation Management (GRAM). Wird ein Auftrag an den GRAM Dienst geschickt, werden drei Einträge geschrieben. Der erste Eintrag dokumentiert die Annahme, der zweite die Verarbeitung und der dritte das Ergebnis des Auftrags. Die Analyseregel löst bei der Annahme einen Alarm niedriger Priorität aus. Nachdem die weiteren Globus Dienste, wie GridFTP und RFT keine Logeinträge schreiben, können sie auch nicht überwacht werden. Die Implementierung der Regeln ist im Abschnitt 5.3.1 zu finden.

Ergänzend zur Analyse der Protokolleinträge des Globus Toolkits, werden dessen Dateien mit der Integritätsüberwachung untersucht. Die Auswahl der Dateien und der Entwurf der Analyseregeln werden im nächsten Abschnitt beschrieben.

4.3.4 Auswahl von Dateien zur Integritätsüberwachung

In der Sicherheitsanalyse (siehe Abschnitt 4.3.1) wurde auf die Gefahren hingewiesen, die entstehen, wenn ein Angreifer Zugriff auf einen Rechner hat, auf dem Grid Zertifikate oder Globus Komponenten gespeichert sind. An dieser Stelle setzt die Integritätsüberwachung von Dateien an. Mit diesem Einbruchserkennungsmechnismus werden Dateieigenschaften überwacht. Wichtige Daten (zum Beispiel Private Keys) können nur durch privilegierte Benutzer gelesen werden. Ein Angreifer kann versuchen, Zugriff auf diese Dateien zu erlangen und gegebenfalls dabei die Eigenschaften (Zugriffsrechte, Zeitstempel der letzten Änderung, usw.) verändern. Samhain (siehe [SAMH 07] und Kapitel 3.3.6) ist ein Intrusion Detection System, das mit der Integritätsanalyse diese Veränderungen erkennen kann.

Das Globus Toolkit verwendet Benutzerzertifikate, die im Heimverzeichnis eines Anwenders gespeichert sind. Außerdem werden Hostzertifikate und andere sicherheitsrelevante Informationen in einem zentralen Verzeichnis gespeichert (/etc/gird-security). Die Eigenschaften (Zeitstempel der letzten Änderung, Dateigröße, etc) der Dateien werden bei ihrer Verwendung nicht verändert und können durch Samhain darauf überwacht werden. Im Globus Verzeichnis befinden sich Konfigurationsdateien (zum Beispiel $GLOBUS_LOCATION/etc/globus_wsrf_rft/jndi-config.xml und $GLOBUS_LOCATION/etc/globus-user-env.sh, wobei $GLOBUS_LOCATION das Stammverzeichnis der Globusinstallation ist), die nur selten geändert werden müssen. Ebenso werden Scripte verwendet, die Globus und seine Komponenten starten (zum Beispiel $GLOBUS_LOCATION/start-stop) und sich nicht ändern. Änderungen an diesen Dateien können auf einen Angriff hindeuten und durch Samhain analysiert werden. Die Globus Protokolldatei befindet sich im Globus Verzeichnis ($GLOBUS_LOCATION/var/container.log) und sollte ständig größer werden. Ein Angreifer der seine Spuren verwischen will, muss diese und andere Protokolldateien manipulieren (zum Beispiel /var/log/messages). Samhain kann jedoch die Größe und Zugriffsrechte dieser Datei überwachen.

Darüberhinaus benutzt das Globus Toolkit die PostgreSQL Datenbank (siehe [POST 07]) und den Unix Dämon Xinetd (siehe [XINE 07]), deren Dateien ebenfalls überwacht werden müssen. PostgreSQL wird über die Datei pg_hba.conf konfiguriert. Diese Datei sollte nur für den Benutzer postgres les- und schreibbar sein und mit Samhain überwacht werden. Die PostgreSQL Programmdateien können je nach Distribution in unterschiedlichen Verzeichnissen liegen und es darf sich nur der Zeitstempel des letzten Zugriffs ändern. Die Dateien, in

denen die Datenbank gespeichert sind, verändert sich ständig, so dass diese nur auf ihre Zugriffsrechte und Eigentumsverhältnisse überwacht werden. Der Unix Dämon Xinetd hört verschiedene Ports ab und startet bei Anfragen den zugehörigen Dienst. Die Konfigurationen für diesen Dienst befinden sich im Verzeichnis /etc/-xinetd.d/. Die Xinetd Konfigurationsdatei für den GridFTP Dienst nennt sich gridftp und ihre Eigenschaften dürfen sich nicht verändern (ausgenommen des Zeitstempels des letzten Lesezugriffs). Für PostgreSQL und Xinetd gibt es bereits Überwachungsregeln, die an die lokale Installation angepasst werden müssen.

Das Globus Toolkit, Xinetd und PostgreSQL werden normalerweise als Dämonen gestartet. Die Startdateien befinden sich im Verzeichnis /etc/init.d (globus, xinetd und postgres) und sollten im Betrieb nicht modifiziert werden. Eine Veränderung der Dateieigenschaften bis auf den Zeitstempels des letzten Lesezugriffs ist verdächtig und kann auf einen Angriff oder einen Missbrauch hindeuten.

Im folgenden Abschnitt wird die Überwachung des Globus Toolkits auf Angriffe durch die Analyse des Netzverkehrs komplettiert.

4.3.5 Überwachung des Globus Toolkit Netzverkehrs

In Abschnitt 4.3.1 wurde erläutert, warum das Globus Toolkit anfällig für netzbasierte Angriffe ist. Snort ist ein NIDS, das den Netzverkehr unter Verwendung mehrerer Regeldateien überwacht. Sie sind größtenteils auf spezielle Dienste ausgerichtet und werden in der Konfigurationsdatei angegeben. Die meisten Analyseregeln sind auf einzelne Angriffe spezialisiert. Damit Snort den gesamten Netzverkehr überwachen kann, muss es entweder auf zentralen Rechnern (zum Beispiel Gateways) installiert sein, oder in einem Netz mit Switches alle Pakete über dedizierte Spiegel Ports empfangen.

Der Entwurf der Snort Analyseregeln für das Globus Toolkit beschränkt sich auf Grund fehlender Literatur auf die Analyse von Verbindungsanfragen an Globus Dienste. Die Dienste GRAM, MDS, GridFTP und MyProxy des Toolkits bieten ihre Funktionalitäten über Netz an (siehe [GT4PORTS 07]). Die Verbindungen sind mit SSL verschlüsselt und können nur unter hohem Aufwand analysiert werden. Außerdem ist die Analyse verschlüsselter Kommunikation rechtlich bedenklich. Der Globus RFT Dienst ist auf den PostgreSQL Server angewiesen, der seine Dienste ebenfalls dem gesamten Grid zur Verfügung stellen muss. Im Lieferumfang von Snort sind bereits Analyseregeln für SQL Server enthalten.

Die Überwachung der Globusdienste untersucht den Netzverkehr auf Anfragen an die Dienste GRAM (Port 8443), MDS (Port 8443), GridFTP (Port 2811) und MyProxy (Port 7512). Greift ein Computer auf einen Globus Dienst zu, wird ein Alarm niedriger Priorität von Snort an den Prelude Manager gesendet. In der Alarmklassifizierung wird eine Beschreibung, die den Dienst und den Zugriff dokumentiert, gegeben.

Auf die Konzeption des Systems zur Föderation von Intrusion Detection Systemen zur Überwachung von Grids folgt im nächsten Kapitel dessen Implementierung.

5 Prototypische Implementierung

Dieses Kapitel dokumentiert die Implementierung des in Kapitel 4 konzipierten Sicherheitsystems. Zuerst wird die Konfiguration des JacORBs, der für die verschlüsselte Kommunikation verantwortlich ist (siehe Abschnitt 5.1.1), erläutert. Anschließend folgt die Umsetzung der Service Information Specification Language (siehe Abschnitt 5.2.1) und der SMONA Komponenten der höheren Schichten (siehe Abschnitte 5.2.2 bis 5.2.4). In Abschnitt 5.2.5 wird die Realisierung von Änderungen am SMONA Adapter Framework und die Umsetzung des Message Manipulator und des Log Adapters erörtert. Abschließend folgt eine Darstellung der Implementierung von Analyseregeln für das Globus Toolkit (siehe Abschnitt 5.3). Dabei werden die Protokollanalyseregeln, die für Prelude-LML (siehe Abschnitt 5.3.1) konzipiert wurden und der Überwachung der Protokolldateien des Toolkits dienen, behandelt. Abschnitt 5.3.2 enthält die Umsetzung von Samhain Regeln zur Kontrolle der Eigenschaften der wichtigsten Globus Toolkit Dateien. Der letzte Abschnitt dieses Kapitels thematisiert die Konfiguration von Snort zur Überwachung von Verbindungsanfragen an Globus Dienste (siehe Abschnitt 5.3.3).

5.1 Konfiguration der Umgebung

Das Globus Toolkit verwendet zur Kommunikation eine eigene SSL Infrastruktur, in die sich die SMONA Architektur nicht einbinden lässt. In dieser Implementierung wird auf Grund der genannten Vorteile aus Abschnitt 4.1 auf die CORBA Middleware JacORB (siehe [JACORB 07], in Version 2.3) zum Nachrichtentransport aufgebaut. Dieser Abschnitt erläutert zuerst die Konfiguration von JacORB zur Verschlüsselung der Kommunikation und behandelt anschließend kurz den Start des Namensdienstes.

5.1.1 Einstellungen zur verschlüsselten Kommunikation

Die TLS/SSL Verschlüsselung der Verbindungen zwischen allen SMONA Komponenten wird in der JacORB Konfiguration eingerichtet und muss nicht im Quellcode programmiert werden. Dieser Abschnitt beschreibt zuerst die Erstellung eines Keystores und erläutert das Generieren und Einbinden von Zertifikaten. Nachfolgend werden die relevanten JacORB Einstellungen beschrieben.

Erzeugung von TLS Zertifikaten

Die Transport Layer Security (TLS, siehe [DiRe 06]) verwendet Public Key Zertifikate im standardisierten X.509 Format. Zu deren Generierung und Verwaltung benutzt der JacORB ein Java Keystore (siehe [JKES 07]), das in einer Datei gespeichert ist. Jede SMONA Komponente und jeder Adapter benötigen ein eigenes Keystore, das zugleich ein Truststore darstellt. In ihm werden eigene und vertrauenswürdige Zertifikate anderer Komponenten gespeichert. Als erstes muss für jede Komponente ein eigenes Zertifikat generiert werden, das anschließend in die Keystores der Kommunikationspartner als vertrauenswürdiger Schlüssel importiert wird. Es ist möglich ein einziges Zertifikat für alle Komponenten zu verwenden. Aus Sicherheitsgründen sollte aber für jede ein eigenes erstellt werden. Auf Kommandozeilenebene wird der Keystore mit dem Keytool (siehe [JKET 07]) gesteuert. Nach der Eingabe des in Listing 5.1 angegebenen Befehls müssen ein Passwort und ein Distinguished Name (DN) zur Generierung des Public/Private Key Paares eingegeben werden. Mit dem Parameter `genkey` generiert das Keytool ein neues Schlüsselpaar. Zusätzlich muss ein Alias angegeben werden, mit dem der Schlüssel benannt wird. Mit dem letzten Argument `keystore` wird ein Dateiname für den Keystore festgelegt.

Listing 5.1: Erstellen eines neuen Keystores mit einem TLS/SSL Public/Private Key Paares

```
1 keytool -genkey -alias smona -keystore neuerKeystore
```

Das neue, selbst signierte Zertifikat muss nun in jeden Truststore der Kommunikationspartner importiert und dadurch als vertrauenswürdig gekennzeichnet werden. Dafür wird es aus dem Keystore exportiert (siehe Listing 5.2). Der Parameter `keystore` gibt den Dateinamen des Keystores an, in dem das Zertifikat zuvor gespeichert wurde. Anschließend werden der Name (Parameter `alias`) des Zertifikats und eine Zieldatei (Parameter `file`) übergeben. Der Kommandozeilen Befehl in Listing 5.3 zeigt das Importieren des Zertifikats in einen anderen Truststore. Er besteht ebenfalls aus vier Parametern, die das Importieren, den Keystore, den Namen und schließlich die Datei des Zertifikats angeben.

Listing 5.2: Exportieren des selbst signierten Zertifikats aus dem Keystore

```
1 keytool -export -keystore neuerKeystore -alias smona -file testZertifikat
```

Listing 5.3: Importieren des selbst signierten Zertifikats in ein Java Truststore

```
1 keytool -import -keystore truststore -alias smona -file testZertifikat
```

Konfiguration des JacORB

Der JacORB erhält seine Konfiguration aus der Datei jacorb.properties. Diese muss sich im *Classpath* oder im aktuellen Verzeichnis des ausgeführten Programms befinden. In dieser Implementierung ist sie im etc/ Ordner enthalten und wird beim Starten der Anwendung im *Classpath* angegeben. In Listing 5.4 sind die für die SSL verschlüsselte Kommunikation verantwortlichen Einstellungen abgebildet.

In der ersten Zeile wird die SSL Unterstützung eingeschaltet. Dadurch werden beim Start des JacORB die SSL Klassen geladen. In den folgenden vier Zeilen wird die Verwendung der SSL Socket Factory und der JSSE SSL Server Socket Factory aus der Java Secure Socket Extension (JSSE, siehe [JSSE 07]) eingestellt. Alternativ kann JacORB auch das IAIK (siehe [IAIK 07]) Framework einbinden. Anschließend werden der Principal Authenticator und das Access Decision Objekt angegeben (vergleiche Abschnitt 3.4). Die nachfolgenden vier Zeilen sind IIOP/SSL Parameter (siehe Zeilen 10 bis 13). Der zugewiesene Wert 20 schreibt dem Server und dem Client die Verwendung von SSL vor. Anschließend wird der Ort der Keystoredatei und das Passwort eingegeben. Die letzte Einstellung ist speziell für JSSE und bedeutet, dass anstatt eines dedizierten Truststores, vertrauenswürdige Zertifikate aus der Keystoredatei verwendet werden (Zeile 16).

Listing 5.4: Auszug aus der Datei jacorb.properties

```
1  jacorb.security.support_ssl=on
   jacorb.ssl.socket_factory= \\
3          org.jacorb.security.ssl.sun_jsse.SSLSocketFactory
   jacorb.ssl.server_socket_factory= \\
5          org.jacorb.security.ssl.sun_jsse.SSLServerSocketFactory
   jacorb.security.principal_authenticator= \\
7          org.jacorb.security.level2.PrincipalAuthenticatorImpl
   jacorb.security.access_decision= \\
9          org.jacorb.security.level2.AccessDecisionImpl
   jacorb.security.ssl.client.supported_options=20
11 jacorb.security.ssl.client.required_options=20
   jacorb.security.ssl.server.supported_options=20
13 jacorb.security.ssl.server.required_options=20
   jacorb.security.keystore=keystore
15 jacorb.security.keystore_password=******
   jacorb.security.jsse.trustees_from_ks=on
```

5.1.2 Der Namensdienst

In dieser Umsetzung wird von einem Java ORB ein Namensdienst bereitgestellt, bei dem sich alle Komponenten der SMONA mit einem eindeutigen Namen registrieren und dadurch für andere auffindbar sind. Der Namensdienst läuft als separates Programm und wird über die Kommandozeile mit dem Befehl aus Listing 5.5 gestartet. Der Port, unter dem der Dienst zu erreichen ist, wird als Parameter eingestellt (`ORBInitialPort`) und muss den SMONA Programmen beim Starten übergeben werden.

Listing 5.5: Start des Namensdienstes

```
orbd -ORBInitialPort 1050
```

Nach dieser Beschreibung zur Verschlüsselung der Kommunikation in der gesamten SMONA Architektur wird im anschließenden Abschnitt die Implementierung der Komponenten ausgeführt.

5.2 Die SMONA Komponenten

In Abschnitt 4.1.2 wurde die Auswahl und Konzeption der zur Föderation von Intrusion Detection Systemen benötigten SMONA Komponenten ausführlich dargelegt. Abschnitt 5.2.1 geht auf die Umsetzung der Service Information Specification Language ein. Die Abschnitte 5.2.2, 5.2.3 und 5.2.4 behandeln die Implementierung der Management Anwendung, des Rich Event Composers und des Adapter Configurators. Die Implementierung der SMONA Adapter befindet sich im Unterabschnitt 5.2.5.

5.2.1 Das SISL XML Schema und die SISL Objekte

Die Service Information Specification Language dient der Beschreibung von Aggregationen. In dieser Arbeit wird sie zur Konfiguration der Adapter und des Rich Event Composers eingesetzt. Zu diesem Zweck wurde das bestehende XML Schema von Martin Sailer angepasst und um die in Abschnitt 4.1.2 erläuterten Erweiterungen ergänzt. In der Konfiguration einer Aggregation wird es von der Management Anwendung zur Validierung der Einstellungen herangezogen. Ferner wurde in der CORBA Interface Description Language ein Aggregations Objekt definiert, das von der Management Anwendung aus dem XML Aggregationsdokument generiert und über ein Netz an den Rich Event Composer und den Adapter Configurator versendet wird.

Listing 5.6: Ausschnitte aus dem SISL XML Schema

```
<xsd:element name="parameter" type="parameter"/>
<xsd:complexType name="parameter">
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="sourceAttrib">
    ...
        <xsd:element name="parameter" type="parameter" \
                        minOccurs="0" maxOccurs="unbounded"/>
    ...
</xsd:complexType>

<xsd:complexType name="valueset">
        <xsd:choice>
            <xsd:element name="resourceRef" type="resourceRef"/>
            <xsd:element name="functionRef" type="functionRef"/>
            <xsd:element name="parameter" type="parameter"/>
        </xsd:choice>
</xsd:complexType>
```

SISL XML Schema

Das SISL XML Schema definiert die Struktur und das Format eines XML Dokumentes, das zur Einstellung einer Aggregation benutzt wird. Die wichtigsten Modifikationen an der Dokumentendefinition sind in Listing 5.6 dargestellt. Nach der Präzisierung des Parameters (siehe Zeilen 1 bis 5) besitzt er zwei Attribute vom Typ Zeichenkette: einen Namen und einen Wert. Er kann beliebig oft für ein *Source* Attribute angegeben werden (siehe Zeilen 7 bis 12) und dadurch Adaptern in der Konfiguration konstante Initialwerte übermitteln. In den Zeilen 14 bis 20 ist die Definition eines *Valuesets* zu sehen, das die Parameter einer Funktion bestimmt. Neben der Übergabe von Referenzen auf andere Funktionen und Ressourcen wurde hier ebenfalls der neu festgelegte Parameter eingefügt. Er darf höchsten einmal in einem *Valueset* angegeben werden, wenn es nicht bereits eine Referenz enthält. Das komplette SISL XML Schema befindet sich im Anhang B. Beispiele für XML Konfigurationen, die dem Schema entsprechen sind in den Listings 5.7, 5.8 und 5.9.

Nachdem ein Benutzer eine XML Aggregationsdefinition gemäß dem Schema angefertigt und bei der Management Anwendung eingegeben hat, wird diese in ein Aggregationsobjekt geparst.

Die SISL Kommunikationsobjekte

In Abbildung 5.1 ist das Datenmodell eines Aggregations- und eines RichEvent Objektes dargestellt, wie sie in dieser Implementierung eingesetzt werden. Die Aggregationsobjekte werden vom SISLReader der Management Anwendung aufgrund ihrer XML Definition erstellt und anschließend an den Rich Event Composer und den Adapter Configurator übermittelt. Nachrichten, die durch den Rich Event Composer nach dem Zutreffen einer Bedingung generiert werden, versendet er als RichEvent Objekte an die Management Anwendung.

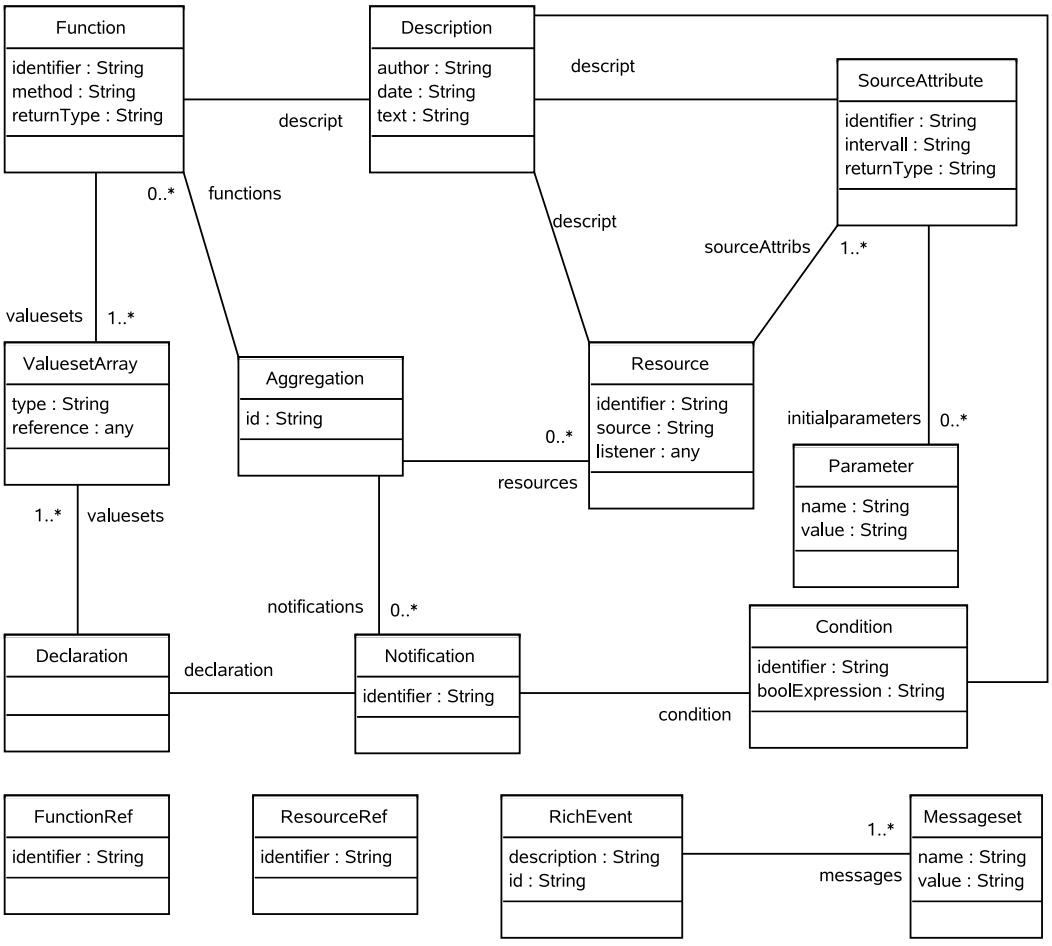

Abbildung 5.1: Klassendiagramm der Kommunikationsobjekte

Zusätzlich, zu den im XML Schema definierten Attributen besitzt eine *Resource* ein Objekt vom Typ Any. Es beinhaltet entweder einen *Push Listener* oder einen *Pull Listener*, der in der Adapterkonfiguration der Einstellung des Kommunikationspartners einer Datenquelle dient. Außerdem besitzt ein *Valueset* ein Any Objekt, das eine Funktions- oder Ressourcenreferenz oder einen Parameter beinhaltet. Die Art des Objektes wird im *type* Attribut angegeben.

Das in der Darstellung abgebildete Rich Event wird vom Rich Event Generator des Rich Event Composers erzeugt und an die Management Anwendung verschickt. Das Attribut *id* entspricht dem eindeutigen Bezeichner der Notifikation und das Attribut *description* umfasst den Beschreibungstext der Bedingung, die für dieses Rich Event verantwortlich sind. Das *Messageset* setzt sich aus den *Valuesets* der Deklaration zusammen. Es enthält einem Referenznamen und den Wert der referenzierten Quelle zusammen.

Aufbauend, auf diesen Abschnitt folgt in den nächsten drei Punkten eine Darlegung der Implementierung der SMONA Komponenten der obersten beiden Schichten.

5.2.2 Die Management Anwendung

Die Management Anwendung ist ein eigenständiges Programm, das der Steuerung aller SMONA Komponenten und der Darstellung von *RichEvents* und Statusmeldungen dient. Durch sie können neue Aggregationen und somit neue Adapter, Funktionen und Notifikationen hinzugefügt oder aktive entfernt werden.

Die funktionalen Einheiten

Die Management Komponente besteht hauptsächlich aus fünf Klassen (siehe Abbildung 5.2). Die zu den *Listenern* gehörigen Klassen werden hier nicht beschrieben, da sie aus dem Adapter Framework entnommen sind und nicht verändert wurden. Gestartet wird die Anwendung mit der Klasse `ManagementComponent`, die für den Programmablauf und die graphische Oberfläche (siehe Abbildung 5.3) verantwortlich ist. Der `SISLReader` validiert die übergebene XML Konfigurationsdatei mit Hilfe des SISL XML Schemas und liest die Aggregationen, wenn keine Fehler erkannt wurden, aus. Im weiteren Verlauf wird ein Aggregationsobjekt erstellt, zu deren *Resourcen* jeweils eine Referenz auf einen Push- oder Pull Listener des Rich Event Composers hinzugefügt wird. Anschließend werden die Einstellungen über den `ManagementClient` an den Adapter Configurator und den Rich Event Composer versendet. Der `ManagementServant` implementiert die eingehende Schnittstelle zu den Komponenten der Integrations- und Konfigurationsschicht und nimmt deren Statusinformationen und Rich Events an. Die Nachrichten werden von der `ManagementComponent` auf der graphischen Oberfläche ausgegeben. Beim Start der Management Anwendung registriert der `Management-Server` den `ManagementServant` beim Namensdienst.

Die graphische Oberfläche

Die Management Anwendung bietet eine dreigeteilte graphische Bedienoberfläche (siehe Abbildung 5.3) bestehend aus einer Bedienleiste (oben), einer Ausgabe von eingebundenen Aggregationen (links) und einer Ausgabe von Nachrichten (rechts). In der oberen Knopfleiste können in der Textzeile Aggregationsdateien angegeben werden. Nach einem Klick auf „Add Aggregation" wird vom *SISLReader* versucht die Konfiguration auszulesen und daraus ein *Aggregation* Objekt (siehe Abbildung 5.1) erstellt, das zur Konfiguration an den Adapter Configurator und den Rich Event Composer versendet wird. Anschließend wird der Dateiname der Aggregation in das linke Textfeld übernommen. Mit der Schaltfläche „Remove Aggregation" wird die in der Textzeile benannte Aggregation wieder entfernt. Dabei werden alle ihre Adapter, Funktionen und Notifikationen beendet und anschließend ihr Name aus der Liste der aktiven Aggregationen entfernt. Das letzte Bedienelement („Quit") dient dem Schließen der Anwendung. Die eingehenden Statusinformationen aller SMONA Komponenten und empfangene Rich Events werden im rechten Textfeld mit der Uhrzeit ihres Eintreffens ausgegeben.

Auf die Beschreibung der Implementierung der Management Anwendung wird im nächsten Abschnitt die Realisierung der funktionalen Einheiten des Adapter Configurators erläutert.

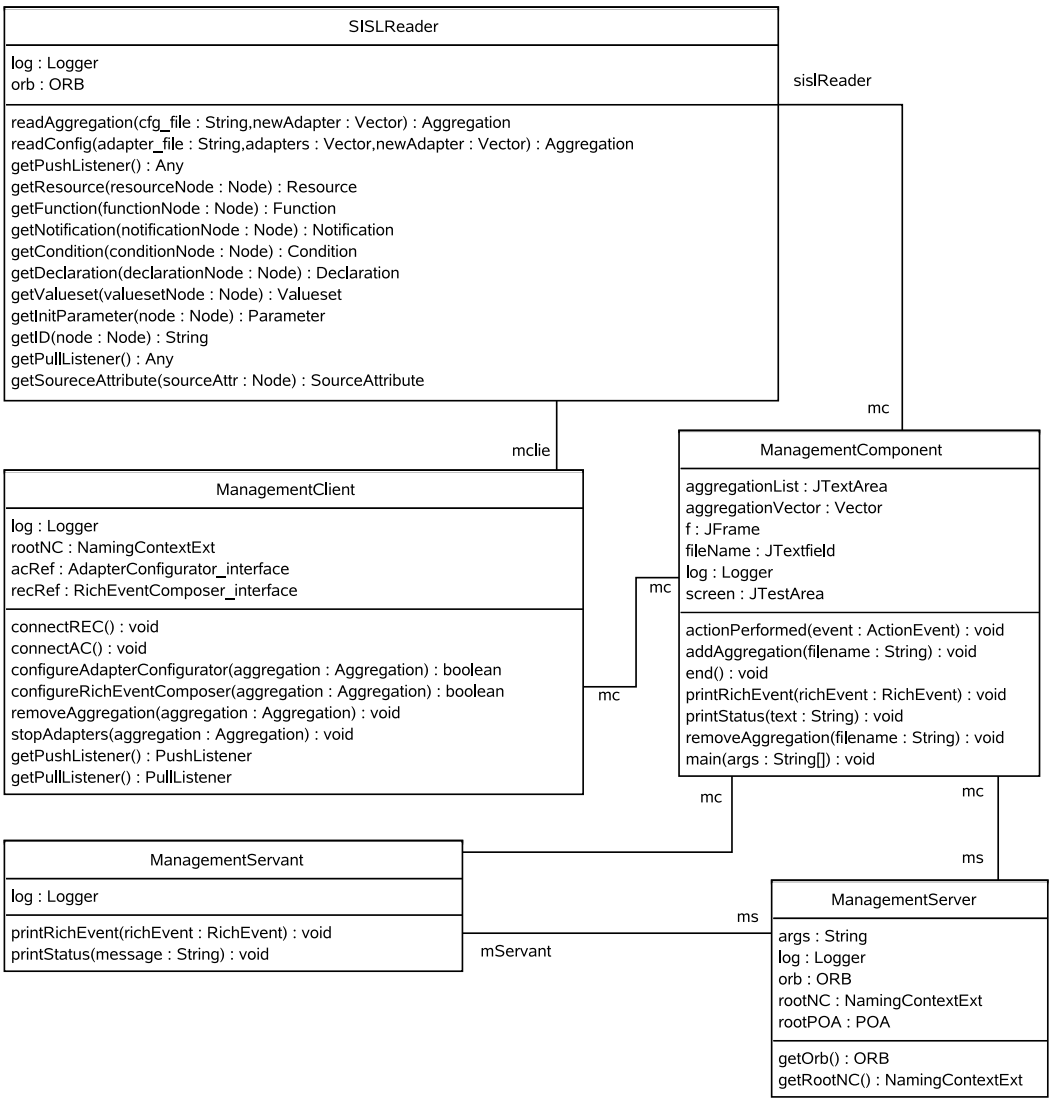

Abbildung 5.2: Klassendiagramm der Management Anwendung

5.2.3 Der Adapter Configurator

In der SMONA Architektur hat der Adapter Configurator die Aufgabe die zu den Aggregationen benötigten Adapter zu verwalten. Durch sie werden von der Management Anwendung die Adapter einer Aggregation gestartet und gestoppt. Der folgende Unterpunkt behandelt die Umsetzung der Anwendung und erläutert die erforderlichen Klassen. Darauf aufbauend wird eine exemplarische Adapterkonfiguration beschrieben.

Komponenten des Adapter Configurators

Das Programm „Adapter Configurator" besteht aus vier Klassen (siehe Abbildung 5.4). Der wichtigste Teil der Funktionalität befindet sich in der Klasse `AdapterConfigurator`. Sie konfiguriert, startet und beendet die Adapter. Dafür erhält sie über ein Objekt der Klasse `ACCommunicationServant` eine Aggregation, aus der Ressourcenobjekte extrahiert und in für Adapter verständliche `Property` Arrays transformiert werden. Vor dem Starten eines Adapters wird untersucht, ob er bereits einer aktiven Aggregationen Daten liefert. Läuft er noch nicht werden die Attribute `source` und `listener` der Resource zusammen mit dem *SourceAttribute* Objekt in ein Datenfeld transformiert und an den Adapter übermittelt. Diese Datenreihe wird vom SMONA Adapter Framework zur Konfiguration verwendet und besteht aus Property Objekten, die den Namen einer

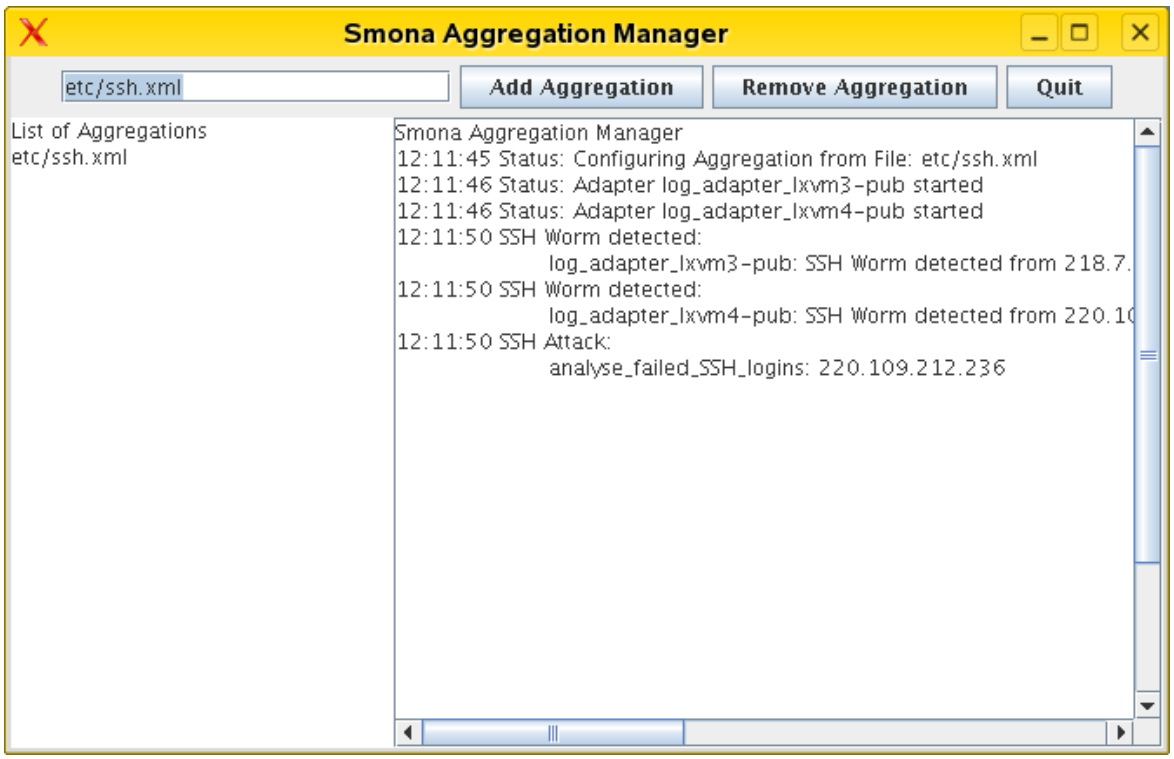

Abbildung 5.3: Die graphische Oberfläche der Management Anwendung

Einstellung und deren Wert enthalten. Kann ein Adapter nicht konfiguriert oder gestartet werden, beendet der Adapter Configurator alle Datenquellen der Aggregation und verschickt eine Fehlermeldung. Wurden alle Adapter einer Aggregation aktiviert, wird diese in einem *Vector* gespeichert. Beim Entfernen einer Aggregation werden alle dazugehörigen Adapter beendet. Dabei überprüft der Adapter Configurator an Hand des eindeutigen Adapternamens alle Aggregationen, ob dieser noch benötigt wird. Ist dies nicht der Fall, wird er gestoppt.

Der Adapter Configurator löst in dem für einen Pull Adapter eingestellten Intervall den Nachrichtenversandt an den Rich Event Composer aus. Dazu wird für jeden Adapter dieses Typs ein `PullSourceTask` Objekt erstellt, das die Informationsverarbeitung beim Adapter startet. Es wird beim Adapterstart beim `Timer` Objekt des `AdapterConfigurators` registriert und seine `run` Methode in den eingestellten Zeitabständen ausgeführt.

Die Kommunikation zu den Adaptern und die Berichterstattung an die Management Anwendung wird in der Klasse `ACCommunication` realisiert. Sie ist für den abgehenden Netzverkehr zuständig und registriert darüberhinaus den ACCommunicationServant beim Namensdienst. Die Schnittstelle für eingehende Konfigurationsanfragen ist in der Klasse ACCommunicationServant umgesetzt. Ihre Aufgabe besteht in der Weiterleitung von entfernten Methodenaufrufen an den *AdapterConfigurator*. Die IDL Beschreibung der Schnittstelle, die von der Klasse ACCommunicationServant implementiert wird ist in Anhang C.3.

Im anschließenden Unterpunkt wird eine exemplarische Adapterkonfiguration dargestellt und der Bezug ihrer Elemente zu den Adaptereinstellungen erläutert.

Beispielhafte Ressourcenkonfiguration

In Listing 5.7 ist ein Ausschnitt aus einer exemplarischen SISL Aggregationsbeschreibung dargestellt, der die Konfiguration eines Message Manipulator Adapters (siehe Abschnitt 5.2.5) beinhaltet. In diesem Segment wird ein Adapter mit einer Adapter Source definiert. Die *Resource* (Zeilen 1 bis 9) entspricht dem Adapter und das darin enthaltene *SourceAttribute* (Zeilen 3 bis 8) stellt die Adapter Source dar. Der Name der Resource aus Zeile eins wird ignoriert. In Zeile zwei wird der Typ des Adapters angegeben. Der eindeutige Bezeichner

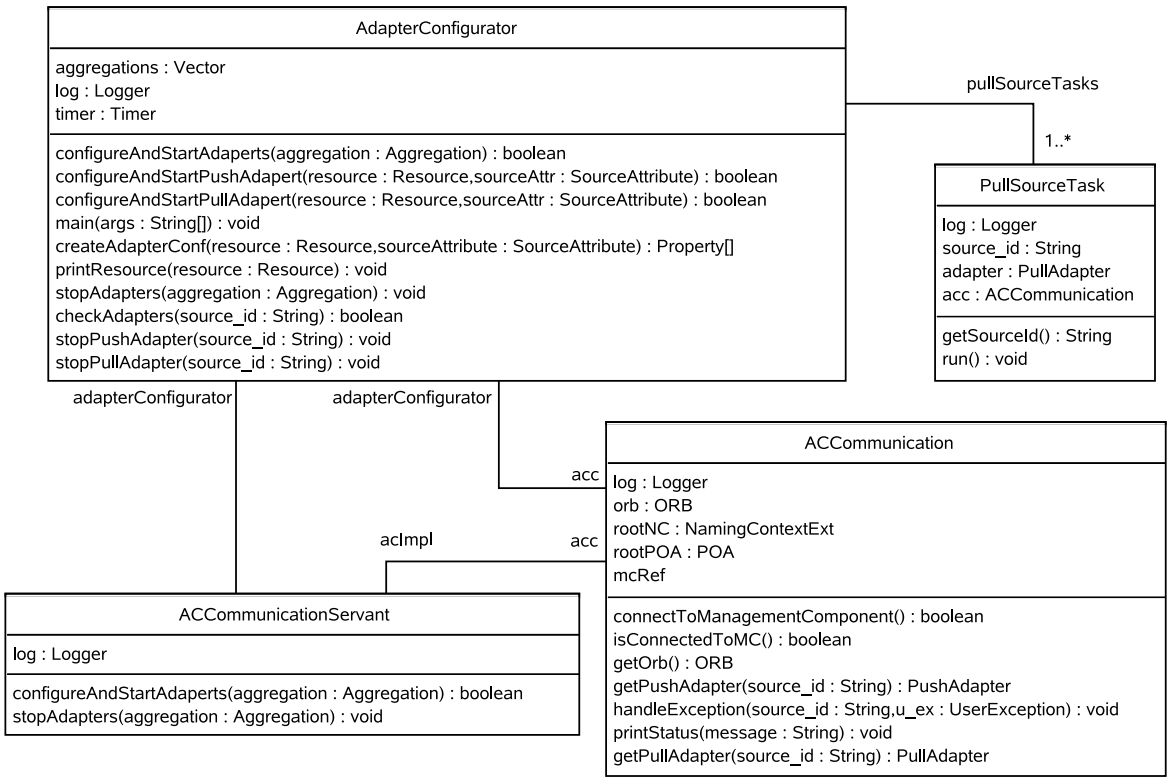

Abbildung 5.4: Klassendiagramm des Adapter Configurator

des *Source Attributs* (Zeile 3) muss dem Namen entsprechen, mit dem die Adapter Source beim Namensdienst registriert ist. Mit ihm bezieht der Adapter Configurator eine Referenz auf den entsprechenden Adapter. Die nachfolgenden Parameter (Zeilen 4 und 5) und die Intervall Einstellung (Zeile 6) sind Argumente, die der Adapter Source beim Start übergeben werden. Die Parameter sind optional und können auch lokal eingerichtet werden. Lokale Einstellungen sind durch die hier definierten Werte nicht überschreibbar. In diesem Beispiel werden dem Message Manipulator Adapters der absolute Pfad der Prelude Manager Logdatei und der relative Pfad zur Datei mit den Manipulationsregeln übermittelt. Darüberhinaus wird eine Referenz auf einen Pull- oder Push Listener des Rich Event Composers übermittelt, an den die Datenquelle ihre Nachrichten schicken soll. Der Datentyp der des Adapters (Zeile 7) wird derzeit nicht verwendet.

Listing 5.7: Exemplarische Adapterkonfiguration

```
<resource id="MM_Adapter_Domain_A">
    <source>push_adapter</source>
    <sourceAttrib id="MM_Adapter_Domain_A">
        <parameter name="logfile" value="/var/log/prelude-xml.log"/>
        <parameter name="rulefile" value="etc/rules"/>
        <interval>2000</interval>
        <return>string</return>
    </sourceAttrib>
</resource>
```

Der Adapter Configurator spielt in der Föderation von Intrusion Detection Systemen eine eher untergeordnete Rolle. Im folgenden Abschnitt wird die Implementierung des Rich Event Composers thematisiert, der mit Funktionen zur Analyse von sicherheitsrelevanten Meldungen konfiguriert werden kann.

5.2.4 Der Rich Event Composer

Der Rich Event Composer ist in der Föderation von Intrusion Detection Systemen eine zentrale Komponente. Sie konzentriert die Nachrichten der IDS, die zum Beispiel durch den Message Manipulator Adapter eingelesen werden. Darüberhinaus kann sie mit Funktionen konfiguriert werden, die die Meldungen weiterverarbeiten. Außerdem werden über Notifikationen Zustände festgelegt, bei deren Eintreten die Management Anwendung informiert wird. Die folgenden Unterpunkte gehen zuerst auf die Umsetzung der funktionalen Einheiten des Rich Event Composers ein und führen anschließend die Verarbeitung von Funktionen und die Auswertung von Bedingungen aus.

Die funktionalen Einheiten des Rich Event Composers

Der Rich Event Composers teilt seine Funktionalität auf acht Klassen auf (siehe Abbildung 5.5). Gestartet wird das Programm über die Klasse `RichEventComposer`. Sie umfasst eine Methode, die das Hinzufügen und Entfernen von Funktionen und Notifikationen beim `FunctionProcessor` beziehungsweise beim `RichEventGenerator` veranlasst. Außerdem speichert sie alle eintreffenden Mitteilungen und kontrolliert deren Verarbeitung. Dabei wird zuerst der Adaptername der letzten Meldung und alle gespeicherten Nachrichten dem Function Processor übergeben und erst anschließend der Rich Event Generator gestartet, der die eingerichteten Bedingungen überprüften lässt. In der Berechnung von Funktionen greift der Function Processor auf Methoden der `ComposerFunctionLibrary` zurück. Sie enthält alle Methoden, die durch eine SISL Funktion ausgeführt werden können. Außerdem initialisiert sie ein `Timer` Objekt, das für die Methode `timeout` jede Sekunde überprüft, ob Nachrichten fehlen. Bevor ein Rich Event erzeugt wird, lässt der Rich Event Generator alle eingestellten boolschen Ausdrücke von einem Objekt der Klasse `ConditionInterpreter` auswerten. Benötigt ein Ausdruck eine Hintergrundfunktion (zum Beispiel *hasResult*), wird diese von der Klasse `ConditionFunctionsLib` bereitgestellt und auch in ihr berechnet. Der detaillierte Ablauf während der Funktionsberechnung und der Auswertung der Ausdrücke ist in den nachfolgenden Unterpunkten beschrieben.

Die Klasse `RECCommunication` verschickt Rich Events und Statusinformationen an die Management Anwendung und initialisiert je ein Objekt der Klassen `RECServant`, `PullListener` und `PushListener`, die sie beim Namensdienst des JacORBs registriert. Der REC Servant realisiert eine weitere Schnittstelle (die IDL ist in Anhang C.2) zur Management Anwendung und leitet ihre Aggregations Anfragen an den Rich Event Composer weiter. Die Daten der Adapter werden von einem Pull- oder Push Listener entgegen genommen und über ein Objekt der Klasse REC Communication an den Rich Event Composer übergeben.

Die Konfiguration und Verarbeitung von SISL Funktionen

Listing 5.8 zeigt einen Ausschnitt aus einer Aggregation, mit dem eine SISL Funktion (*timeout*) präzisiert wird. Sie kategorisiert die Nachrichten des Adapters *MM_Adapter_Domain_A* an Hand der enthaltenen *AnalyzerID* eines IDS und generiert eine Nachricht, falls der Adapter innerhalb von 610 Sekunden keine weitere Meldung des selben Intrusion Detection Systems einliest. Mit der Funktion können IDS, die keine Heartbeatnachrichten mehr verschicken entdeckt werden. Unter dem Bezeichner aus Zeile eins wird die Funktion beim Function Processor registriert. Er muss für alle Funktionen und Ressourcen eindeutig sein. Die zweite Zeile legt die Methode (*timeout*) fest, die in dieser Funktion ausgeführt werden soll. Der Datentyp des Rückgabewertes ist eine Zeichenkette (Zeile 3). In den Zeilen vier bis sechs wird ein Beschreibung, die nur aus einem Text besteht gegeben. Anschließend werden die Parameter der Funktion definiert (Zeilen 7 bis 17). Als erstes wird eine Datenquelle eingestellt, deren Meldungen nach dem regulären Ausdruck (*analyzerid="[\d]+"* , Zeile 15) des dritten Parameters kategorisiert werden sollen. Der maximale Zeitabstand zwischen zwei Nachrichten einer Kategorie wird in Zeile 12 mit 610 Sekunden definiert.

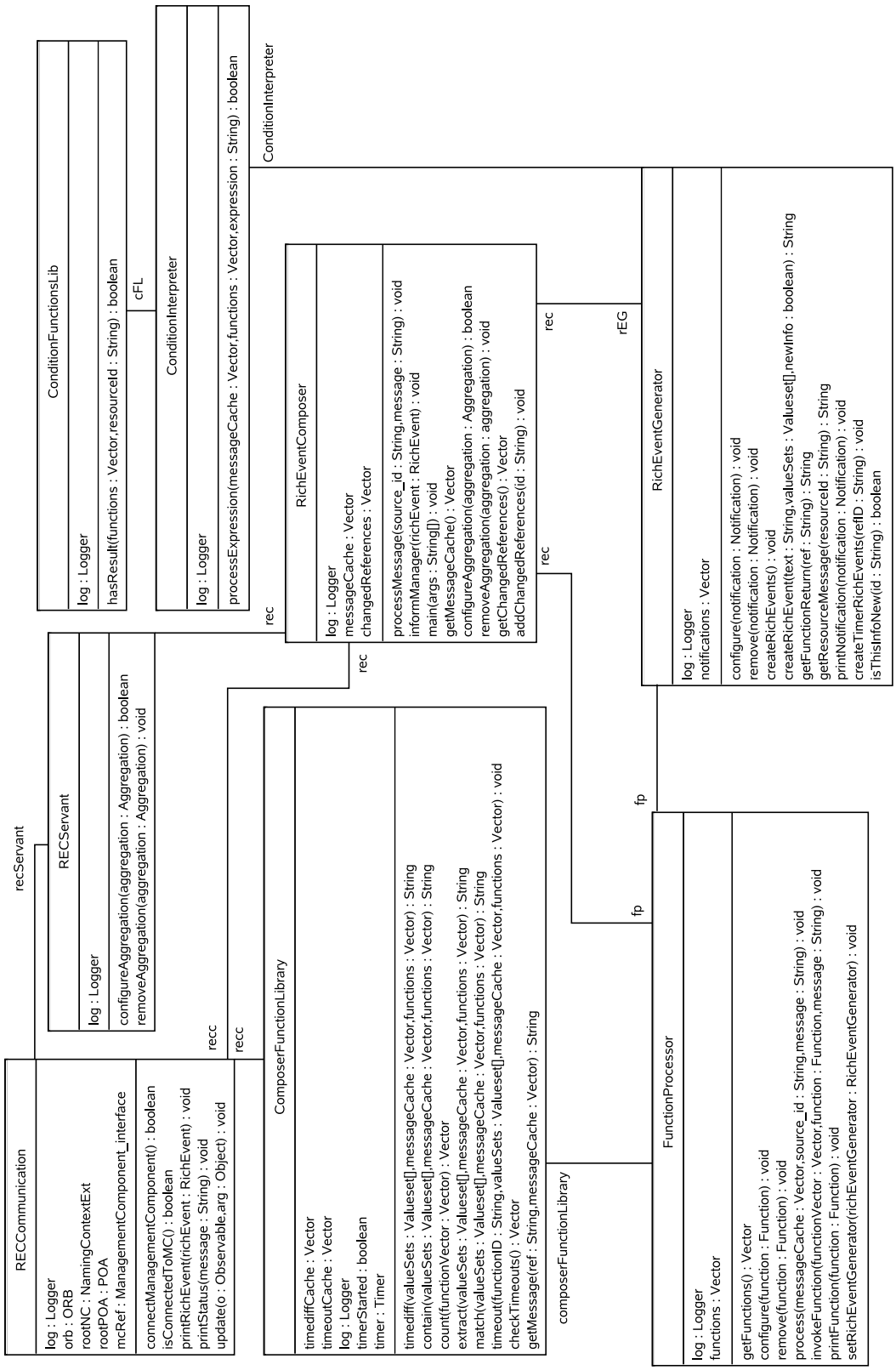

Abbildung 5.5: Klassendiagramm des Rich Event Composers

Listing 5.8: Exemplarische Funktionsdefinition

```
<function id="checkLiveSignal">
    <method>timeout</method>
    <return>string</return>
    <description>
        <text>check for missing Heartbeats</text>
    </description>
    <parameters>
        <valueset>
            <resourceRef id="MM_Adapter_Domain_A"/>
        </valueset>
        <valueset>
            <parameter name="timedifference" value="610"/>
        </valueset>
        <valueset>
            <parameter name="differentiator" value="analyzerid=\"[\d]+\"␣"/>
        </valueset>
    </parameters>
</function>
```

Die Funktionsverarbeitung beginnt im Function Processor, der vom Rich Event Composer mit dem Namen der Datenquelle, der letzten erhaltenen Nachricht und allen empfangenen Meldungen gestartet wird. Er lässt daraufhin alle Funktionen berechnen, die Mitteilungen der Datenquelle als Eingabe benötigen und speichert deren Ergebnis und Bezeichner. Dabei werden der in der SISL Funktion festgelegten Methode (siehe Zeile 3 in Listing 5.8) alle gespeicherten Funktionswerte und Nachrichten und das Valueset der SISL Funktion übergeben. Die Methode wird von einem Objekt der Klasse *ComposerFunctionLibrary* ausgeführt. Sie bezieht alle konstanten Parameter aus dem Valueset. Weitere Daten von Ressourcen oder anderen Funktionen können aus den übergebenen Argumenten entnommen werden. Nachdem eine Funktion ausgeführt wurde, speichert der Function Processor den Rückgabewert und den Funktionsnamen. Anschließend werden rekursiv alle Funktionen untersucht, ob sie Rückgabewerte einer gerade ausgeführten Funktion als Eingabe verwenden. Trifft dies zu, wird die Funktion ausgeführt und als soeben bearbeitet gekennzeichnet.

Notifikations - Verarbeitung

Listing 5.9: Exemplarische Notifikationsdefinition

```
<notification id="foundAlert">
    <condition id="foundAlert">
        <description>
            <author>Thomas Binder</author>
            <date>04.12.07</date>
            <text>Received an alert message</text>
        </description>
        <boolExpression>
            <expression>
                hasResult(returnAlertMessages)
            </expression>
        </boolExpression>
    </condition>
    <declaration>
       <valueset>
            <functionRef id="returnAlertMessages"/>
        </valueset>
    </declaration>
</notification>
```

Das Listing 5.9 zeigt eine beispielhafte Definition einer Notifikation. In der ersten Zeile wird ein eindeutiger Bezeichner für die Benachrichtigung konfiguriert. Eindeutig bedeutet hier, dass keine andere Notifikation den

selben Namen haben darf. Anschließend folgt die Definition der Bedingung, unter der ein Rich Event generiert wird. Sie hat zufällig den selben Namen (Zeile 2) wie die Benachrichtigung. In den Zeilen drei bis sieben wird eine Beschreibung angegeben. In diesem Beispiel besteht sie aus den optionalen Angaben des Autors und des Datums und der obligatorischen Einstellung des Beschreibungstextes. In Zeile 10 wird eine Bedingung präzisiert, die vom *ConditionInterpreter* ausgewertet wird und bei deren Eintreffen ein Rich Event erzeugt wird. In Listing 5.9 steht der in dieser Arbeit entworfene Ausdruck *hasResult* mit der Funktionsreferenz *returnAlertMessages*. Das heißt, dass die Management Anwendung dann informiert wird, wenn die Funktion *returnAlertMessages* einen neuen Rückgabewert hat. In der abschließenden Deklaration (Zeile 14 bis 18) wird bestimmt welche Informationen in das Rich Event übernommen werden sollen. Auf Grund der Zeile 16 enthält es den Namen der Funktion *returnAlertMessages* und ihren letzten Rückgabewert. Weiterhin wird der Name der Benachrichtigung aus Zeile eins und der Beschreibungstext aus Zeile sechs in die Mitteilung übernommen.

Die Bearbeitung der Notifikationen durch den *RichEventGenerator* kann auf zwei Arten gestartet werden. Entweder wird sie durch eine Funktion des Function Processors oder vom Rich Event Composer angestoßen. Meldet eine *timeout* Funktion des Function Processors, dass Nachrichten einer Quelle ausbleiben, dann wird der Rich Event Generator direkt aufgerufen, alle Bedingungen der Benachrichtigungen zu überprüfen. Enthält eine *Condition* eine Referenz auf die aufrufende *timeout* Funktion, wird sie vom *ConditionInterpreter* mit Hilfe der *ConditionFunctionLib* ausgewertet. In dieser Implementierung kann der Condition Interpreter nur die Ausdrücke *hasResult* und *match* (siehe Unterpunkt SISL Erweiterungen in Abschnitt 4.1.2) verarbeiten. Dabei wird entweder die übergebene Funktion auf neue Rückgabewerte überprüft, oder der letzte Wert einer Funktion oder einer Ressource mit einem regulären Ausdruck verglichen. Ergibt die Auswertung, dass die Bedingung erfüllt ist, erzeugt der Rich Event Generator eine Nachricht.

In den meisten Fällen wird die Überprüfung der Benachrichtigungen vom Rich Event Composer gestartet. Nachdem eine neue Meldung den Composer erreicht hat, werden alle Funktionen, die Mitteilung dieser Quelle verarbeiten, ausgeführt. Anschließend ruft der Composer den Rich Event Generator auf, alle Condition Objekte zu überprüfen. Trifft eine Bedingung zu, erstellt der Rich Event Generator ein Rich Event. Dabei werden der Beschreibungstext der Notifikation und ihr Bezeichner in die Nachricht übernommen. Außerdem werden alle Referenzen der *Deklaration* mit ihrem Namen und ihrem letzten Wert in *Messageset* Objekte geschrieben, die dem Rich Event angehängt werden. Abschließend wird das Event an die Management Anwendung verschickt.

Dieser Abschnitt hat die Implementierung der SMONA Komponenten der Integrations- und Konfigurationsschicht und der Anwendungsschicht thematisiert. Im Anschluss wird die Realisierung mehrerer Adapter der plattformunabhängigen Schicht erörtert.

5.2.5 Implementierung der Adapter

Im vorherigen Kapitel, in Abschnitt 4.2 wurden Anpassungen an das SMONA Adapter Framework und der Entwurf mehrerer Adapter konzipiert. In den folgenden Unterabschnitten werden deren Umsetzungen beschrieben. Abschnitt 5.2.5 legt die Realisierung verschiedener Modifikationen am Adapter Framework und an den Push und Pull Adaptern dar. Anschließend wird die Verwirklichung des Message Manipulator Adapters (siehe Abschnitt 5.2.5) und des Log Adapters (siehe Abschnitt 5.2.5) erläutert.

Anpassungen des Adapter Frameworks

Im bestehenden SMONA Adapter Framework von Michael Dürr ([Dürr 06]) mussten verschiedene Änderungen erdacht werden. Sie werden an Hand eines beispielhaften Initialisierungsvorgangs eines Push Adapters erklärt. Die nachstehende Tabelle 5.1 beinhaltet darüberhinaus eine Auflistung der wichtigsten Abwandlungen der Klassen.

Im Wiederspruch zur Anforderung, dass Adapter Programme über Netz vom Adapter Configurator gestartet werden, wird die Ausführung lokal gestartet. Dies geschieht über die Klasse `AdapterServer`. Sie bereitet alle Adapter einer Konfigurationsdatei auf die Initialisierung vor. Zu Beginn erzeugt der Adapter Server eine `XmlPushEnviroment`, die aus der lokalen Adapterkonfigurationsdatei ein `AllSourceEnviroment` Objekt erstellt. Sie wurde für das Einlesen der neu entworfenen Parameter, die beliebige Einstellungen neuer

Name der Klasse	Modifikation
`AllSourceEnviroment`	erbt von der Klasse SourceEnviroment alle Attribute und enthält zusätzlich ein Datenfeld aus Namen/ Werte Paaren, das die neuen Parameter (siehe Listing 5.10) speichert
`PushSourceFactory`	wurde um die Initialisierung des Log Adapters und des Message Manipulator Adapters erweitert und verwendet nun die Klasse AllSourceEnviroment zur Adapterkonfiguration
`PullSourceFactory`	verwendet nun die Klasse AllSourceEnviroment zur Adapterkonfiguration
`XmlPushEnviroment`	kann aus der Konfigurationsdatei eines Adapters die neuen Parameter auslesen und erzeugt daraus ein Objekt der Klasse AllSourceEnviroment
`XmlPullEnviroment`	siehe XmlPushEnviroment
`PushStringSource`	wurde um die Einstellung der neuen Parameter (siehe Listing 5.6) durch den Adapter Configurator erweitert
`PullSource`	siehe PushStringSource

Tabelle 5.1: Veränderungen an den Klassen des SMONA Adapter Frameworks

Adapter enthalten können (siehe Listing 5.10), verändert. Die All Source Enviroment speichert die Einstellungen und Umgebungsvariablen der Adapter. Aus Kompatibilitätsgründen erbt sie von der Klasse `Source-Enviroment` alle bisherigen Attribute und bietet darüberhinaus ein Datenfeld, in dem neue Parameter mit ihren Namen und Werten gespeichert werden. Mit Hilfe der Konfiguration generiert der Adapter Server ein Objekt der Klasse `PushSourceFactory`. Sie erzeugt, konfiguriert und startet nach Aufforderung des Adapter Configurators, die entsprechende Push Source. Die neue Push Source Factory kann in der zentralen Adapterinitialisierung durch den Configurator alle Umgebungsvariablen eines Adapters einstellen, aber vorhandene Werte nicht verändern. Schließlich erstellt der Adapter Server Objekte der Klassen `SmonaAdapterPublisher` und `PushAdapter`. Der Push Adapter dient dabei als Schnittstelle für den Adapter Configurator und wird vom Smona Adapter Publisher beim Namensdienst registriert.

Veränderungen am XML Schema der lokalen Adapter Konfiguration

Auf Grund der benötigten Einstellungen der neuen Adapter wurde das XML Schema der Konfigurationsdatei verändert. Listing 5.10 zeigt die relevanten Ausschnitte. In den Zeilen eins bis sechs und sieben bis zwölf werden die Konfigurationselemente der Pull- und Push Adapter erweitert. Sie können beliebig oft neue XML Tags vom Typ *parameter* enthalten. Ein Parameter (Zeilen 13 bis 16) besteht aus zwei Zeichenketten Attributen, die den Namen einer Einstellung und deren Wert beinhalten sollen.

Listing 5.10: Auszug aus dem XML Schema der Adapter Konfigurationsdatei

```
1  <xsd:complexType name="pullAdapterType">
     ...
3      <xsd:element name="parameter" type="parameter"
           minOccurs="0" maxOccurs="unbounded" />
5    ...
   </xsd:complexType>
7  <xsd:complexType name="pushAdapterType">
     ...
9      <xsd:element name="parameter" type="parameter"
           minOccurs="0" maxOccurs="unbounded" />
11   ...
   </xsd:complexType>
13 <xsd:complexType name="parameter">
       <xsd:attribute name="name" type="xsd:string" use="required"/>
15     <xsd:attribute name="value" type="xsd:string" use="required"/>
   </xsd:complexType>
```

Exemplarische Konfigurationsdatei eines Adapters

In Listing 5.11 ist eine beispielhafte Konfigurationsdatei eines Adapters abgebildet, der eine Adapter Source (Zeilen 7 bis 17) startet. Bei der Adapter Source handelt es sich um einen Message Manipulator Adapter (siehe Abschnitt 5.2.5), zur Überwachung einer Prelude Manager Protokolldatei. Die erste Zeile enthält die XML Version und die Kodierung der Datei. Anschließend wird der Adapter definiert. Dabei wird am Anfang der *Namespace*, der Speicherort des zugehörigen Schemas (siehe Listing 5.10) und ein Bezeichner des Adapters angegeben. Der Name des Adapters aus Zeile fünf wird ignoriert. In Zeile sechs wird der Typ des Adapters präzisiert. Darauf folgen Einstellungen für den Message Manipulator Adapter. Unter dem Bezeichner aus Zeile sieben wird der Message Manipulator Adapter beim Namensdienst registriert und er kann dadurch in eine Aggregation eingebunden werden. Der Datentyp (Zeile 8) einer *Source* bestimmt zusammen mit seinem Namen welcher Adapter gestartet wird. Die Beschreibung in Zeile neun ist in dieser Implementierung unwichtig. Mit der Einstellung einer oberen und einer unteren Grenze (Zeilen 10 bis 14) wird das vom Adapter Configurator eingestellte Zeitintervall, in dem der Adapter seine Arbeit ausführt, überprüft. Es muss sich innerhalb des vorgegeben Rahmens (Zeile 12 und 13) befinden. Die Protokolldatei des Prelude Managers und die Datei mit den Manipulationsregeln werden in den Zeilen 15 und 16 festgelegt und können vom Adapter Configurator nicht verändert werden.

Listing 5.11: Exemplarische Konfigurationsdatei eines Adapter

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="Adapter.xsd"
    id="Message_Manipulator_Domain_A">
    <push_adapter>
        <source id="Message_Manipulator_Domain_A">
            <src_type>string</src_type>
            <description>Push Prelude messages</description>
            <interval>
                <type>milliseconds</type>
                <max_allowed>60000</max_allowed>
                <min_allowed>1000</min_allowed>
            </interval>
            <parameter name="rulefile" value="etc/rules"/>
            <parameter name="logfile" value="testData/prelude-xml.log"/>
        </source>
    </push_adapter>
</adapter>
```

Auf die Beschreibung der Veränderungen am SMONA Adapter Framework und Erörterung der Adapter Konfigurationsdatei folgt im nächsten Unterabschnitt eine Erläuterung der Implementierung des Message Manipulator Adapters.

Der Message Manipulator Adapter

In diesem Abschnitt wird die Umsetzung der Konzeption aus Abschnitt 4.2.3 beschrieben. Es wird zuerst ein verkürztes Klassendiagramm des Adapters dargestellt und daran die Funktionen der Teilkomponenten erklärt. Dieser Abschnitt schließt mit einer Beschreibung der Manipulationsregeln, die die Veränderung der Protokolleinträge definieren.

Der Message Manipulator Adapter besteht hauptsächlich aus drei Klassen, die die Funktionalität zum Auslesen von Meldungen aus einer Protokolldatei, der Manipulation der Einträge und dem Versenden der Ergebnisse an den Rich Event Composer liefern. Gestartet wird der Adapter wie alle anderen über die Klasse `AdapterServer`, die den Message Manipulator Adapter mit Hilfe einer Konfigurationsdatei generiert, konfiguriert und beim Namensdienst registriert. Der Adapter muss für den Betrieb mit einem Intervall und den Speicherorten der Protokolldatei und der Regeldatei konfiguriert werden.

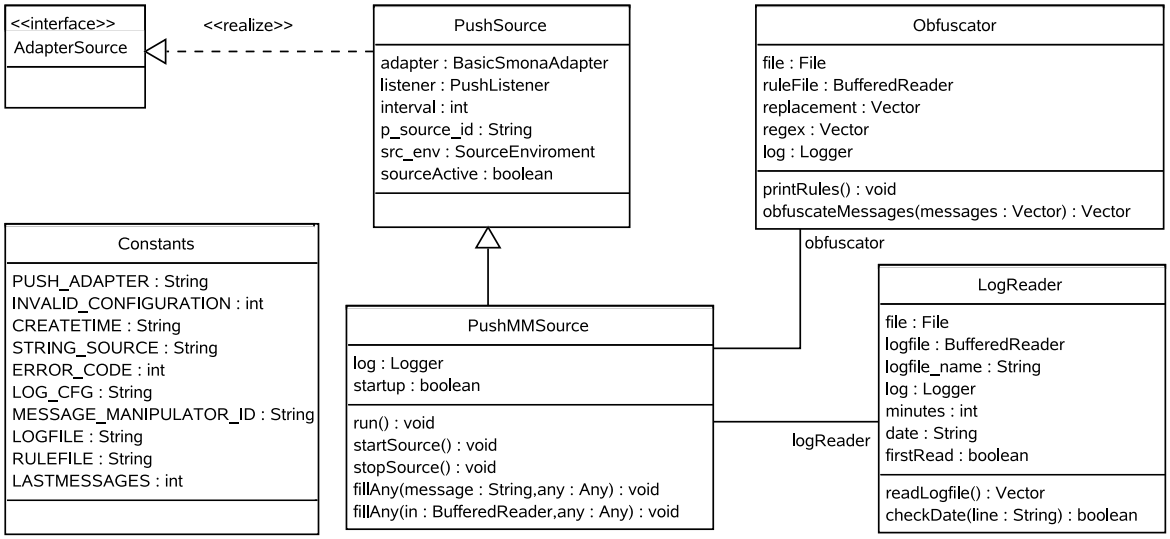

Abbildung 5.6: Verkürztes Klassendiagramm des Message Manipulator Adapters

Der Arbeitsablauf wird durch ein Objekt der Klasse **PushMMSource** gesteuert. Sie ist die zentrale Komponente des Adapters. Aus Gründen der Kompatibilität und Anpassung an diese Aufgabenstellung erweitert sie die Klasse `PushSource` und überschreibt mehrere Methoden. Im Entwurf des SMONA Adapter Frameworks (siehe [Dürr 06]) ruft die Push Source ein externes Programm auf, das Daten vorverarbeitet und dessen Rückgabewerte weitergesendet werden. In dieser Implementierung ist ein Objekt der Klasse Push MM Source für die Verarbeitung selbst zuständig. Es wird mit dem ORB, einem eindeutigen Namen, den Einstellungen des Adapter Configurators, einem Adapter und der All Source Enviroment der lokalen Einstellungen konfiguriert. In der Initialisierung werden aus der Konfiguration durch den Adapter Configurator ein Push Listener und ein Zeitintervall extrahiert und eingestellt. Weitere Parameter, wie zum Beispiel die Speicherorte der Protokolldatei und der Regeldatei, die in der Aggregation optional angegeben werden können, richtet die Push MM Source ebenfalls ein, wenn sie nicht bereits durch die lokale Konfiguration in der All Source Enviroment definiert sind. Das Intervall bestimmt einen Zeitabstand, in dem die Push MM Source den *LogReader* die Protokolldatei auslesen und die Einträge vom *Obfuscator* manipulieren lässt. Anschließend werden die Nachrichten an den Rich Event Composer in einem Any Objekt an den Rich Event Composer versendet.

Die Klasse **LogReader** ist für das Auslesen der Protokolldatei zuständig und wird mit dem relativen Dateinamen initialisiert. Beim ersten Auslesen der Datei werden zuerst alle Einträge gelesen und je nach Konfiguration alle Nachrichten der letzten zehn Minuten (siehe Klasse `Constants` in Abbildung 5.6) an den Rich Event Composer weitergeleitet. Jeder weiterer Aufruf liefert nur die neuen Nachrichten.

Ein Objekt der Klasse **Obfuscator** bekommt von der Push MM Source in der Initialisierung den Speicherort einer Datei, die Regeln zur Manipulation von Protokolleinträgen enthalten. Diese wird zu Beginn ausgelesen und die regulären Ausdrücke in Java Pattern Objekte überführt. Ein Vektor (`regex`) enthält alle Muster, die ersetzt werden sollen. Ein zweiter Vektor (`replacement`) umfasst die Ersetzungen. Das Format der Regeldatei wird in einem eigenen Unterpunkt beschrieben. Im Betrieb übernimmt der Obfuscator die gelesenen Nachrichten des Log Readers und verändert sie an Hand der vorkonfigurierter Regeln. Jede Nachricht wird auf Muster aus dem `regex` Vektor untersucht und bei Übereinstimmung der entsprechende Abschnitt durch den zugehörigen Eintrag aus dem `replacement` Vektor ersetzt.

Zuletzt wird hier die Klasse **Constants** beschrieben. Sie enthält alle Konstanten des Message Manipulator Adapters. Die wichtigsten Konstanten und ihre Bedeutung werden in Tabelle 5.2 beschrieben.

Die Regeldatei und die Manipulationsregeln

Dieser Abschnitt enthält zuerst eine kurze Beschreibung des Aufbaus der Regeldatei und erläutert anschließend mehrere Anwendungsbeispiele.

Die Regeldatei ist wie eine gemeine Linux Konfigurationsdatei aufgebaut. Kommentare sind mit einem # Zei-

Name der Konstante	Bedeutung
MESSAGE_MANIPULATOR_ID	enthält eine Zeichenkette, die von der `PushSourceFactory` verwendet wird, um einen Message Manipulator Adapter zu erzeugen
LASTMESSAGES	bestimmt beim ersten Auslesen welche Einträge der letzten LASTMESSAGES Minuten eingelesen werden
LOGFILE	enthält eine Zeichenkette, die zur Erkennung des Parameters verwendet wird, der den Speicherort der Protokolldatei definiert
RULEFILE	enthält eine Zeichenkette, die zur Erkennung des Parameters verwendet wird, der den Speicherort der Regeldatei festlegt

Tabelle 5.2: Die wichtigsten Konstanten des Message Manipulator Adapters aus der Klasse Constants

chen gekennzeichnet. Alles nach dem # Zeichen wird ignoriert. Ersetzungsregeln richten sich in ihrer Syntax nach Java Pattern (siehe [JPATTERN 07]) und bestehen aus zwei Teilen, die durch einen Schrägstrich (/) getrennt werden. Zuerst wird ein regulärer Ausdruck gegeben, der ersetzt werden soll. Dieser Ausdruck wird vom Obfuscator gelesen und in ein Pattern Objekt überführt. Beim Manipulieren der Meldungen wird ihr Text nach dem regulären Ausdruck durchsucht und durch den zweiten Teil der Manipulationsregel ersetzt. Der zweite Teil der Ersetzungsregel enthält eine Substitution, die an die Stelle des regulären Ausdrucks in der Nachricht geschrieben wird. Die Substitution ist ein einfacher Text.

Der Obfuscator liest die Regeln zeilenweise ein und speichert sie auch in der selben Reihenfolge. Beim Manipulieren der Nachrichten werden die Ersetzungsregeln in der gleichen Reihenfolge angewendet.

In Listing 5.12 ist eine exemplarische Regeldatei abgebildet, die mehrere Beispiele enthält. Die erste Regel in Zeile zwei lässt den Obfuscator alle Benutzernamen, die in einer IDMEF Nachricht des Prelude Managers als Zielbenutzer angegeben werden, durch „hiddenusername" ersetzen. In den Zeilen fünf und neun sind zwei Regeln dargestellt, die IP Adressen der Nachricht in „hiddenip" umwandeln. Die Ersetzungsregel in der fünften Zeile verändert nur IP Adressen, die in der IDMEF Nachricht als solche gekennzeichnet sind. Durch Zeile neun werden alle weiteren IP Adressen verändert. Nachdem Anwenden der Regel in Zeile zwölf sind alle Betriebssystem Versionsnummern verschleiert. Weitere Versionsnummern werden durch Zeile 15 ersetzt. Der Protokollanalysator Prelude-LML verschickt in der IDMEF Nachricht den kompletten Eintrag, der die Alarmnachricht ausgelöst hat. Die Ersetzungsregel in Zeile 18 blendet diesen Eintrag wieder aus. Die letzten beiden Zeilen des Listings sind für die Verschleierung von TCP Ports verantwortlich. Zuerst werden Ports, die das Ziel oder die Quelle eines Angriffs angeben, ersetzt. Die letzte Regel kaschiert alle restlichen Portangaben.

Listing 5.12: Eine exemplarische Regeldatei mit Manipulationsregeln

```
1  # the following line replaces target user names
   type="target-user"><name>[^<]*/type="target-user"><name>hiddenusername
3
   # the following line replaces ip addresses from the address tag into "hiddenip"
5  <address>[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}/<address>hiddenip

7  # the following line replaces all ip addresses into "hiddenip";
   # not only in the address tag
9  [\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}/hiddenip

11 # the following line replaces the OS versionnumber into "hiddenversion"
   osversion="[^"]*"/osversion="hiddenosversion"
13
   #␣the␣following␣line␣replaces␣all␣analyzer␣versionnumbers␣into␣"hiddenversion"
15 version="[^"]*" class/version="hiddenversion" class

17 # the following line replaces the original Log entry in Prelude-LML Alert messages
   Original Log"><string>[^<]*</Original␣Log"><string>hidden log<
19
   # the following lines replace all port numbers
21 <port>[\d]+/<port>hiddenport
   port [\d]+/port hiddenport
```

Nach der Beschreibung der Realisierung des Message Manipulator Adapters folgt im anschließenden Abschnitt die Umsetzung der Konzeption des Log Adapters.

Der Log Adapter

Dieser Abschnitt thematisiert die Implementierung des Log Adapters, der in Kapitel 4.2.4 entworfen wurde. Zuerst wird an Hand des verwendeten Algorithmus die obligatorische Konfiguration des Adapters erläutert. Weiterführend erfolgt eine Beschreibung seiner Klassen.

Der Log Adapter dient der Erkennung ähnlicher Protokolleinträge, die sich innerhalb eines bestimmten Zeitabstandes wiederholen. Er untersucht die Einträge aus einer Protokolldatei auf das Vorhandensein eines ersten regulären Ausdrucks und verwirft sie wieder, wenn er nicht enthalten ist. Anschließend werden aus den verbleibenden Nachrichten mit Hilfe eines zweiten regulären Ausdrucks eine Zeichenkette extrahiert und diese dadurch kategorisiert. Treffen zwei Mitteilungen der selben Sparte innerhalb einer festgelegten Zeit ein, wird eine definierte Nachricht verschickt und weitere Meldungen für das Eintreffen einer Mitteilung dieser Sparte ausgesetzt. Existiert eine Kategorie noch nicht, oder war die letzte Nachricht länger als der festgelegte Zeitraum her, wird eine zweite definierte Mitteilung an den Rich Event Composer verschickt (siehe Abschnitt 4.2.4 und Abbildung 4.6). Durch diese Analyse kann der Log Adapter zum Beispiel SSH Wurmangriffe oder bestimmte DoS Angriffe auf Webserver erkennen. Er benötigt dafür sechs Parameter, die vom Adapter Configurator oder über eine lokale Konfigurationsdatei eingestellt werden können. Zwei Parameter sind reguläre Ausdrücke. Außerdem müssen zwei unterschiedliche Nachrichten, eine für einzelne Ereignisse und eine für die Entdeckung eines Angriffs eingestellt werden. Darüberhinaus muss eine Zeitspanne und der Speicherort der Protokolldatei übergeben werden.

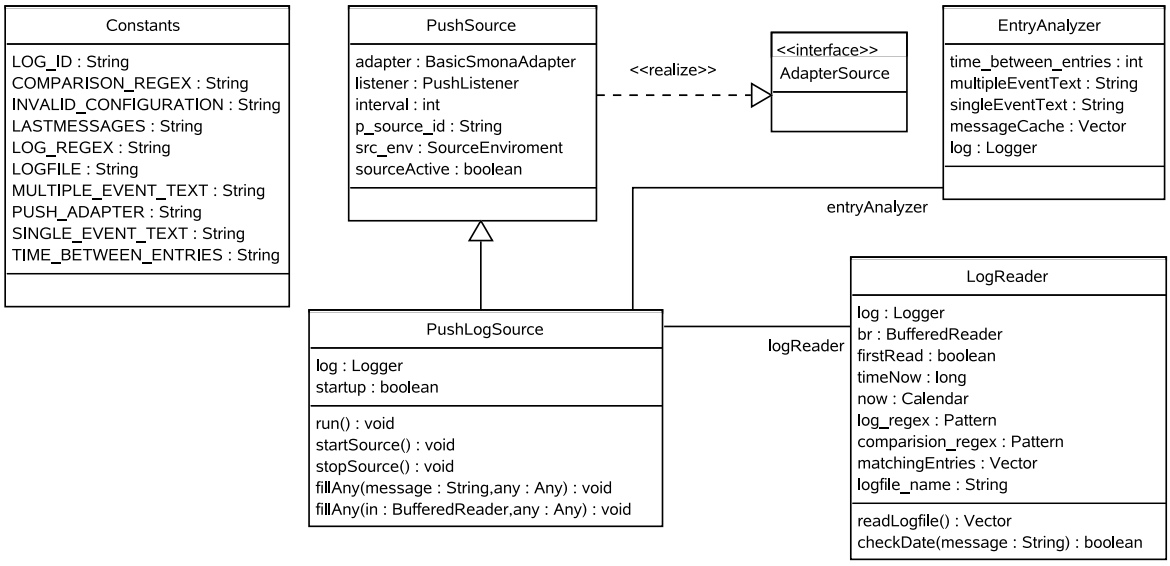

Abbildung 5.7: Verkürztes Klassendiagramm des Log Adapters

Aufbauend auf den oben kurz beschriebenen Analysealgorithmus und die Konfiguration folgt die Erläuterung der wichtigsten Klassen des Log Adapters.

Abbildung 5.7 stellt diese in einem verkürzten Klassendiagramm dar. Der Adapter wird in der Initialisierung der Klasse `AdapterServer` generiert, konfiguriert und beim Namensdienst registriert. Die Analyse der Protokolldatei wird vom Adapter Configurator in einem Objekt der Klasse **PushLogSource** gestartet. Sie steuert den Arbeitsablauf und lässt in einem definierten Zeitabstand dem *LogReader* die Protokolldatei auslesen und die Einträge vom *EntryAnalyzer* analysieren. Ergebnisse der Analyse schickt sie zusammen mit seinem eindeutigen Namen an den Push Listener des Rich Event Composers.

Ein **LogReader** Objekt liest die Protokolldatei aus und filtert alle Einträge heraus, die nicht den ersten regulären Ausdruck enthalten. Daraufhin extrahiert sie mit Hilfe des zweiten Ausdrucks eine Zeichenkette und

Name der Konstante	Bedeutung
LOG_ID	enthält eine Zeichenkette, die von der `PushSourceFactory` verwendet wird, um einen Log Adapter zu erzeugen
COMPARISON_REGEX	zur Erkennung des Parameters, der den regulären Ausdruck zur Kategorisierung der Protokolleinträge präzisiert
LASTMESSAGES	bestimmt beim ersten Auslesen welche Einträge der letzten LASTMESSAGES Minuten eingelesen werden
LOG_REGEX	zur Erkennung des Parameters, der den regulären Ausdruck zur Filterung von Protokolleinträgen definiert
LOGFILE	zur Erkennung des Parameters, der den Speicherort der Protokolldatei festlegt
MULTIPLE_EVENT_TEXT	zur Erkennung des Parameters, der einen Nachrichtentext für Angriffe konkretisiert
SINGLE_EVENT_TEXT	zur Erkennung des Parameters, der einen Nachrichtentext für einzelne Ereignisse präzisiert
TIME_BETWEEN_ENTRIES	zur Erkennung des Parameters, der die Zeitdifferenz zwischen zwei Nachrichten einer Kategorie festlegt, bevor sie als zusammengehörig erkannt werden

Tabelle 5.3: Die wichtigsten Konstanten des Log Adapters aus der Klasse Constants

übergibt sie zusammen mit dem Zeitstempel des Eintrags an die Push Log Source, die damit den Entry Analyzer startet. Beim ersten Auslesen werden alle Nachrichten der letzten zehn Minuten (siehe Klasse `Constants` in Abbildung 5.7) untersucht. Im folgenden Betrieb werden nur neu hinzugekommene Meldungen überprüft und weitergeleitet.

Die Klasse **EntryAnalyzer** implementiert die zeitliche Analyse der erhaltenen Zeichenketten, die aus den Protokolleinträgen entnommen wurden und generiert gegebenenfalls eine Nachricht die an die Push Log Source übergeben wird. Die Nachricht setzt sich aus dem festgelegten Text und der Zeichenkette zusammen.

Die Klasse **Constants** enthält alle Konstanten des Log Adapters. Die wichtigsten davon sind in Tabelle 5.3 mit ihrer Bedeutung beschrieben.

Nachdem in diesem Abschnitt die Implementierungen verschiedener Adapter zur Informationsaufnahme in die SMONA Architektur behandelt wurden, führt der nächste Unterabschnitt die Realisierung von Analyseregeln zur Angriffserkennung aus.

5.3 Überwachung des Globus Toolkits mit drei exemplarischen IDS

In Kapitel 4.3 wurde die Föderation der IDS zur Überwachung des Globus Toolkits konzipiert. Dieser Abschnitt stellt die Umsetzung des Entwurfs dar. Zuerst wird in Abschnitt 5.3.1 die Konfiguration des Prelude-LML beschrieben und auf die Implementierung der Protokollanalyseregeln eingegangen. Abschnitt 5.3.2 beschreibt Anfangs den Aufbau von Analyseregeln und anschließend die Überwachung von sicherheitsrelevanten Dateien des Globus Toolkits mit Hilfe von Samhain. Schließlich werden Analyseregeln für Snort, zur Überwachung des Globus Netzverkehrs erläutert (siehe Abschnitt 5.3.3).

5.3.1 Protokollanalyse Regeln für den Prelude-LML

In Kapitel 4.3.3 wurde die Protokollierung des Globus Toolkits und der Entwurf von Analyseregeln zur Überwachung der Logeinträge beschrieben. Dieser Abschnitt führt die Konzeption aus und erklärt den Aufbau einer Prelude-LML Regel.

Zur Implementierung der Globus Toolkit Regeln muss eine neue Regeldatei erstellt werden (siehe Listing 5.14). Anschließend muss sie in der Konfigurationdatei prelude-lml.conf angegeben werden. In Listing 5.13 ist ein Abschnitt der Konfigurationsdatei, in dem Prelude-LML mit der Globus Protokolldatei konfiguriert. Die erste Zeile beschreibt eine neue Formatvorlage für die Logeinträge. Darauf folgt ein Präfix, mit dem jeder Eintrag beginnt. Das Globus Toolkit verwendet für sein Protokoll keine einheitliche schreibweise und deshalb ist der Präfix leer. Wird kein Präfix angegeben, schreib Prelude-LML eine Fehlermeldung für jeden Protokolleintrag in seine Logdatei. In der letzten Zeile steht schließlich der Pfad und Dateiname der zu überwachenden Protokolldatei.

Listing 5.13: Auszug aus prelude-lml.conf zur Konfiguration der neuen Logdatei

```
[format=globus]
prefix-regex = "␣"
file = /usr/local/globus-4.0.5/var/container.log
```

Im Listing 5.14 ist ein Auszug der Protokollanalyse Regeln aus der neu erstellten Regeldatei. Darin werden vier Analyseregeln beschrieben, die einen Alarm auslösen, wenn das Globus Toolkit gestartet oder gestoppt wird, einen Laufzeitfehler protokolliert oder GRAM (Grid Resource Allocation Management) einen Arbeitsauftrag annimmt.

In der ersten Zeile jeder Regel steht ein regulärer Ausdruck, der mit einem Logeintrag verglichen wird. Der Ausdruck kann Variablen definieren, die später in die IDMEF Nachricht übernommen werden. Anschließend werden IDMEF Nachrichten mit Informationen zum Alarm konfiguriert. Jede Analyseregel schließt mit dem Schlüsselwort `last` ab und Prelude-LML beendet die Analyse des Logeintrags. Eine genaue Beschreibung zum Aufbau von Analyseregeln kann unter [PLML 07] nachgelesen werden.

Die erste Regel in Zeile eins löst einen Alarm aus, wenn ein Logeintrag die Zeichenfolge „Starting SOAP server at: https://" gefolgt von einer IP Adresse und ein Port enthält. Die regulären Ausdrücke `([\d\ .]+)` und `(\d+)` stehen für beliebige Zahlen und definieren zwei Variablen, die in der IDMEF Nachricht referenziert werden können. Eine Variable kann mit einem $ Zeichen und einer Zahl verwendet werden (zum Beispiel $1). Die Zahl gibt dabei die Variablennummer an. In der zweiten Zeile wird eine kurze Klassifizierung des Alarms gegeben. Darin steht zu Beginn eine eindeutige Regelidentität auf die eine Kurzbeschreibung folgt. Die Klassifizierung wird in die IDMEF Nachricht übernommen. Die Zeilen drei und vier enthalten eine eindeutige Regelidentität zur Unterscheidung in Prelude-LML und eine Versionsnummer der Regel. Beide Informationen werden nicht in den IDMEF Alarm geschrieben. Anschließend werden Informationen zum Analysator beschrieben. Der erste Analysator ist in diesem Fall das Globus Toolkit, das die Protokolleinträge erstellt hat. In den Zeilen neun bis zwölf wird eine genauere Beschreibung des Alarms und seine schwere definiert. In Zeile 14 wird die IP Adresse des Rechners mit $1 übergeben. Die erste Regel endet mit einer Definition des „Angriffziels". Dazu werden die IP Adresse, der Port und das Transportprotokoll des neu gestarteten Dienstes eingestellt.

Die zweite Analyseregel (Zeilen 20 - 36) löst einen Alarm aus, sobald das Globus Toolkit beendet wird. Sie ist analog zur ersten Regel aufgebaut und unterscheidet sich nur im regulären Ausdruck und der Alarmbeschreibung. In den Zeilen 38 bis 48 des Listings steht eine Regel für Logeinträge des GRAM. Sie ist für einen Alarm verantwortlich, der ausgelöst wird, sobald GRAM einen neuen Arbeitsauftrag angenommen hat. In der Protokolldatei wird der Benutzer der Anfrage und seine Herkunft angegeben und in der Zeile 47 in die Alarm Beschreibung durch die Variablen $5 und $6 aufgenommen. Die letzte Regel erkennt einen Laufzeitfehler des Globus Toolkits und verschickt daraufhin eine IDMEF Nachricht. Treten im Betrieb des Toolkits Fehler auf, werden diese in mehreren Zeilen ausführlich in der Logdatei dokumentiert. Der Prelude-LML kann nur einzeilige Einträge verarbeiten und es wird deshalb nur die erste Zeile der Fehlernachricht an Hand der Zeichenkette „ERROR" erkannt und in den Alarm aufgenommen. In der Alarmbeschreibung wird mit der Variablen $1 das Globus Modul angegeben, das den Fehler ausgelöst hat.

Listing 5.14: Auszug aus der Prelude-LML Regeldatei für das Globus Toolkit

```
1 regex=Starting SOAP server at: https://([\d\.]+):(\d+); \
   classification.text=1001011: SOAP server started; \
3  id=1001011; \
   revision=1; \
5  analyzer(0).name=Globus Toolkit; \
```

```
 analyzer(0).manufacturer=globus.org; \
 analyzer(0).class=grid middleware; \
 analyzer(0).version= 4.0.5\;
 assessment.impact.severity=low; \
 assessment.impact.completion=succeeded; \
 assessment.impact.type=admin; \
 assessment.impact.description=Globus Toolkit container started on $1; \
 target(0).node.address(0).category=ipv4-addr; \
 target(0).node.address(0).address=$1; \
 target(0).service.port=$2; \
 target(0).service.iana_protocol_name=tcp; \
 target(0).service.iana_protocol_number=6; \
 last;

regex=Stopped SOAP Axis server at: https://([\d\.]+):(\d+); \
 classification.text=1001012: SOAP Axis server stopped; \
 id=1001012; \
 revision=1; \
 analyzer(0).name=Globus Toolkit; \
 analyzer(0).manufacturer=globus.org; \
 analyzer(0).class=grid middleware; \
 assessment.impact.severity=low; \
 assessment.impact.completion=succeeded; \
 assessment.impact.type=admin; \
 assessment.impact.description=Globus Toolkit container stopped on $1; \
 target(0).node.address(0).category=ipv4-addr; \
 target(0).node.address(0).address=$1; \
 target(0).service.port=$2; \
 target(0).service.iana_protocol_name=tcp; \
 target(0).service.iana_protocol_number=6; \
 last;

regex=INFO  exec.StateMachine (\S+) (\S+) (\S+) (\S+) for (\S+) user (\S+); \
 classification.text=1001013: Globus Toolkit GRAM accepted a job; \
 id=1001013; \
 revision=1; \
 analyzer(0).name=Globus Toolkit; \
 analyzer(0).manufacturer=globus.org; \
 analyzer(0).class=grid middleware; \
 assessment.impact.severity=low; \
 assessment.impact.type=admin; \
 assessment.impact.description=GRAM $4 a job for $5 user $6; \
 last;

regex=ERROR (\S+).(\S+); \
 classification.text=1001014: Globus Toolkit Container Exception in $1; \
 id=1001014; \
 revision=1; \
 analyzer(0).name=Globus Toolkit; \
 analyzer(0).manufacturer=globus.org; \
 analyzer(0).class=grid middleware; \
 assessment.impact.severity=medium; \
 assessment.impact.type=admin; \
 assessment.impact.description=The Globus Toolkit Container threw an \
        Exception for the $1; \
 last;
```

Nach der Ausführung zur Überwachung der Protokolleinträge spricht der folgende Abschnitt die Realisierung der Analyse von Dateieigenschaften der Globus Toolkit Dateien an.

5.3.2 Integritätsanalyse des Globus Toolkits mit Samhain

Das Globus Toolkit verwendet Zertifikate, Konfigurationsdateien und eine Protokolldatei die schützenswert sind (siehe Kapitel 4.3.4). Darüberhinaus verwendet das Toolkit in der Standardkonfiguration den Xinetd Dienst (siehe [XINE 07]) um den GridFTP Dienst (siehe Kapitel 3.1.6) zu starten. Der RFT Dienst (siehe Kapitel 3.1.6) des Toolkits benutzt eine PostgreSQL Datenbank (siehe [POST 07]), die ebenfalls überwacht werden muss.

Samhain (siehe [SAMH 07]) liest die zu überwachenden Dateien und Verzeichnisse aus der Konfigurationsdatei /etc/samhainrc (siehe Listing 5.15). Es können verschiedene Informationen der Objekte überprüft werden. Zu diesem Zweck sind in samhainrc verschieden vordefinierte Bereiche enthalten. In den Bereich `[ReadOnly]` werden alle Dateien und Verzeichnisse geschrieben, deren Eigenschaften sich nicht ändern (bis auf das Lesezugriffsdatum). Im Bereich `[GrowingLogFiles]` stehen zu überwachende Objekte, deren Änderung einen Alarm auslöst mit Ausnahme ihres Wachstums, ihrem Zeitstempel und ihrer Signatur. Der dritte verwendete Bereich ist mit `[Attributes]` gekennzeichnet. Hier werden nur Veränderungen des Eigentümers, der Gruppe oder der Zugriffsrechte berichtet. Nachdem alle sicherheitsrelevanten Dateien und Verzeichnisse (siehe Kapitel 4.3.4 und Listing 5.15) in der Konfigurationsdatei angegeben wurden, muss für diese die Samhain Datenbank initialisiert werden. Anschließend können in regelmäßigen Abständen die Dateien und ihre Eigenschaften mit der Datenbank verglichen werden. Bei Regelverstößen löst Samhain einen Alarm aus.

Listing 5.15: Auszug aus der Datei /etc/samhainrc

```
1  [ReadOnly]
   dir=4/home/*/.globus/
3
   dir=/etc/grid-security
5  file=/etc/services
   file=/etc/xinetd.d/gridftp
7  file=/etc/xinetd.conf
   file=/etc/init.d/postgresql
9  file=/etc/init.d/xinetd
   file=/etc/init.d/globus
11
   dir=3/usr/lib64/postgresql
13 dir=99/usr/local/globus-4.0.5
   dir=3/usr/share/postgresql
15 file=/usr/sbin/xinetd
   file=/usr/bin/postgres
17
   file=/var/lib/pgsql/data/pg_hba.conf
19 file=/var/lib/postgres/postmaster.conf

21 [GrowingLogFiles]
   file=/usr/local/globus-4.0.5/var/container.log
23 file=/var/lib/pgsql/data/pg_log/*

25 [Attributes]
   dir=3/var/lib/pgsql
```

In Listing 5.15 ist ein Auszug der Konfigurationsdatei samhainrc mit den Überwachungsregeln für das Globus Toolkit. Der Ausschnitt enthält drei Abschnitte für die unterschiedlichen Überwachungsregeln. Im ersten Bereich sind alle Dateien und Verzeichnisse aufgelistet, deren Eigenschaften sich nicht verändern dürfen. Davon ausgenommen ist der Zeitstempel des letzten Lesezugriffs. Ab Zeile 21 sind im zweiten Abschnitt Regeln, die wachsende Logdateien überwachen. Die letzte Zeile gehört zum Bereich Attributes und überprüft die Dateien der PostgreSQL Datenbank. Die im Listing angegebenen Pfade zu den Dateien und Verzeichnissen können für verschiedene Linux Distributionen unterschiedlich sein. Für PostgreSQL und Xinetd gibt es bereits Überwachungsregeln, die an die lokale Installation angepasst werden müssen. Darüberhinaus überschneiden sich die Regeln des Listings teilweise mit vorkonfigurierten Samhain Regeln. Sie werden zur Vollständigkeit trotzdem

genannt. Im Folgenden werden die wichtigsten Regeln beschrieben.

In der zweiten Zeile des Listings steht eine Regel, die Veränderungen der Benutzerzertifikate und Schlüssel meldet. Das Globus Toolkit liefert eine Zertifizierungsstelle, die bei der Standardinstallation Dateien im Verzeichnis `/home/globus/.globus/simpleCA` ablegt. Diese Dateien werden durch die erste Regel ebenfalls geschützt. Die zweite Regel überwacht Hostzertifikate und Sicherheitseinstellungen des Globus Toolkits. Die Regel `file=/etc/xinetd.d/gridftp` meldet Veränderungen der GridFTP Einstellung des Xinetd Dienstes. In der dreizehnten Zeile wird das Installationsverzeichnis des Globus Toolkits überwacht. In diesem Verzeichnis befindet sich auch eine Protokolldatei, deren Überprüfung in der zweiundzwanzigsten Zeile überschrieben wird.

Die Analyse der Globus Toolkit Protokolleinträge und die Überwachung seiner Dateien werden im nächsten Abschnitt durch die Beschreibung der Implementierung der Analyseregeln des Netzverkehrs ergänzt.

5.3.3 Analyse des Globus Toolkit Netzverkehrs mit Snort

Die Analyse des Grid Netzverkehrs dient der Sicherung der von Globus bereitgestellten Dienste. Snort (siehe [SNOR 07]) ist ein NIDS, das bereits für viele unterschiedliche Angriffe und Dienste Analyseregeln bereitstellt. Nach der Standardinstallation werden die Regeldateien in das Verzeichnis `/etc/snort/rules` kopiert. Die Datei `snort.conf` enthält die Konfiguration des Snort NIDS. Hier muss der Pfad zu den Analyseregeln angegeben und die Zusammenarbeit mit Prelude eingestellt werden. Für den von Globus verwendeten PostgreSQL Server existieren Analyseregeln in der Datei `sql.rules`. Eigene Snortregeln werden in die Datei `local.rules` geschrieben.

Listing 5.16: Auszug aus der Snort Regeldatei für das Globus Toolkit

```
1  config classification: \
   grid-connection,A TCP connection with a Grid Service was detected,4
3
   # GRAM and MDS connections
5  alert tcp any any -> any 8443 \
   (msg:"A␣TCP␣connection␣with␣the␣Globus␣GRAM␣or␣MDS␣was␣detected"; \
7  flowbits:isnotset,GRAM_MDS_request; flowbits:set,GRAM_MDS_request; \
   classtype:grid-connection; sid:1001001;)
9
   # GridFTP connections
11 alert tcp any any -> any 2811 \
   (msg:"A␣TCP␣connection␣with␣the␣Globus␣GridFTP␣was␣detected"; \
13 flowbits:isnotset,GFTP_request; flowbits:set,GFTP_request; \
   classtype:grid-connection; sid:1001002;)
15
   # MyProxy connections
17 alert tcp any any -> any 7512 \
   (msg:"A␣TCP␣connection␣with␣the␣Globus␣MyProxy␣was␣detected"; \
19 flowbits:isnotset,MP_request; flowbits:set,MP_request; \
   classtype:grid-connection; sid:1001003;)
```

Listing 5.16 beinhaltet einen Auszug der Snort Regeldatei (local.rules) für das Globus Toolkit. Die ersten beiden Zeilen definieren eine Klassifizierungen, mit der jeder Alarm der nachfolgenden Regeln, beschriftet wird. Zuerst steht das Schlüsselwort zur Definition der neuen Klassifizierung. Nach dem Doppelpunkt wird ein Kurzname, eine kurze Beschreibung und zuletzt die Priorität des Alarms angegeben. Die Priorität gibt von 1 (schwer) bis 4 (leicht) die schwere des Alarms wieder.

Die Zeilen fünf bis acht des Listings definieren die erste Regel, die einen Alarm auslöst, sobald ein Paket an den Port des GRAM und MDS Dienstes geschickt wird. Zuerst wird mit `alarm` angegeben, dass es sich um eine Alarmdefinition handelt. Das Schlüsselwort `tcp` schränkt die Verbindungsanfragen auf das TCP Transportprotokoll ein. Anschließend werden die Quellrechner, der Quellport, der Zielrechner und der Zielport angegeben. `Any` steht hier für beliebige Quell- und Zielrechner und einen beliebigen Quellport. Nachfolgend

steht der Zielport 8443, den die Dienste GRAM und MDS verwenden. In den Zeilen sechs und acht werden Informationen angegeben, die in die IDMEF Nachricht geschrieben werden. Zuerst steht eine Kurzbeschreibung des Alarms. Daraufhin wird die Klassifizierung und eine eindeutige Regelidentität des Alarms angegeben. In der siebten Zeile wird die Verbindung markiert, damit nachfolgende Pakete der Verbindung keine weiteren Alarme auslösen.

Die zweite und dritte Regel sind analog zur ersten und unterscheiden sich nur im Zielport und der Alarmbeschreibung. Die zweite Regel löst einen Alarm bei Verbindungen zum GridFTP Port aus. Die letzte Regel überwacht den Port 7512 des MyProxy Dienstes aller Rechner auf Verbindungen.

Dieses Kapitel hat die Implementierung eines Systems zur Föderation von Intrusion Detection Systemen zur Erkennung von Angriffen auf Grid Infrastrukturen erörtert. Im folgenden Kapitel werden Tests zur Evaluation des System beschrieben und deren Ergebnisse dargelegt.

6 Evaluation und Test des Sicherheitssystems

In den vorherigen Kapiteln wurde der Entwurf des Sicherheitssystems und die Implementierung seiner Komponenten beschrieben. In diesem Kapitel wird die prototypische Umsetzung in einem kleinen Testumfeld realisiert und auf Schwachstellen überprüft. Zuerst wird der Testaufbau in seinen Grundzügen beschrieben. Anschließend gehen die Abschnitte 6.1.1, 6.1.2 und 6.1.3 detailliert auf die Infrastruktur und die Konfiguration der Teilsysteme ein. In den Abschnitten 6.2.1, 6.2.2 und 6.2.3 wird mit eigens entwickelten Programmen versucht die Implementierung der Adapter und der SMONA Komponenten anzugreifen. Nachfolgend ist ein Anwendungsbeispiel mit einer exemplarischen Aggregation erläutert, das der Erkennung von SSH Würmern und SSH Angriffen dient. Abschließend erfolgt in Abschnitt 6.3 eine Bewertung der Konzeption und der prototypischen Implementierung.

6.1 Testaufbau

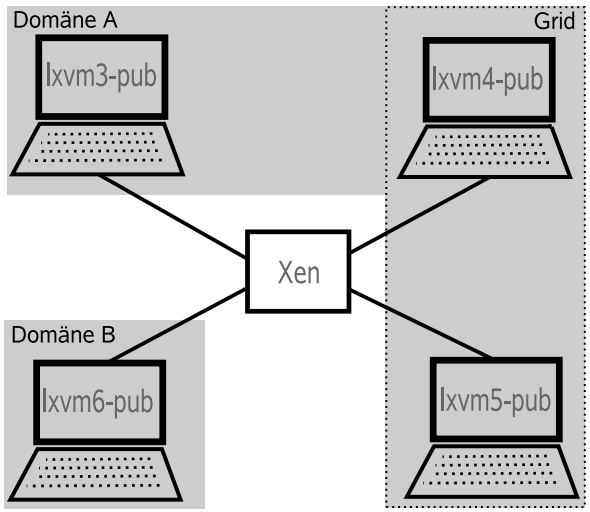

Abbildung 6.1: Testaufbau des Globus Toolkits und der SMONA mit mehreren IDS

Der gesamte Testaufbau wird in virtuellen Computern auf einem Rechner mit Hilfe des Virtuellen-Maschinen-Monitors Xen ([XEN 07]) virtualisiert. Abbildung 6.1 skizziert die Infrastruktur des Testumfelds in seinen Grundzügen. Im Testaufbau befinden sich vier Computer (lxvm3-pub, lxvm4-pub, lxvm5-pub und lxvm6-pub). Obwohl alle Rechner direkt über Xen miteinander kommunizieren können, sind sie in zwei Domänen aufgeteilt, die durch die grauen Flächen gekennzeichnet sind. Das Grid, bestehend aus lxvm4-pub und lxvm5-pub, ist mit einer gepunkteten Linie umrandet. Alle Rechner werden von Xen mit einem Dual Core AMD Opteron 2,5 GHz Prozessor und 512 MB Arbeitspeicher bereitgestellt. Auf jedem virtuellen Rechner ist SuSE Linux 10.1 mit dem Kernel 2.6.16.13-4-xen installiert. Die unterschiedlichen Funktionen und Anwendungsprogramme der Komponenten werden in der Tabelle 6.1 kurz beschrieben und sind in den folgenden Unterabschnitten ausführlich dargelegt. Der Aufbau der IDS Architektur und ihr Informationsfluss bis zum Rich Event Composer sind in Abbildung 6.3 dargestellt. Das Schaubild 6.4 zeigt die Verteilung und Zusammenarbeit der SMONA Komponenten.

Rechnername	Funktion	Anwendungen
lxvm3-pub	Der Rechner lxvm3-pub dient als zentrale Sammelstelle der sicherheitsrelevanten Nachrichten aus den Domänen A und B. Er konzentriert die Alarmnachrichten der IDS aus Domäne A (siehe Abbildung 6.3) und verarbeitet sie in den SMONA Komponenten der höheren Schichten. Außerdem befindet sich hier die graphische Benutzeroberfläche der Management Anwendung.	• SMONA Management Komponente • SMONA Adapter Configurator • SMONA Rich Event Composer • SMONA Message Manipulator Adapter • SMONA Log Adapter • Prelude Manager
lxvm4-pub	Dieser Rechner nimmt am Grid teil und bietet mehrere Dienste an. Er wird von zwei IDS und einem Log Adapter überwacht, die ihre Funde an den Prelude Manager und den Rich Event Composer auf dem Computer lxvm3-pub versenden (siehe Abbildungen 6.3 und 6.4).	• GTK4 GridFTP • GTK4 RFT • GTK4 GRAM • Prelude-LML • Samhain • SMONA Log Adapter
lxvm5-pub	lxvm5-pub ist der zentrale Grid Rechner, auf dem Benutzer- und Hostzertifikate erstellt und verwaltet werden. Außerdem stellt er die Datenbank PostgreSQL für den RFT Dienst beider Gridrechner zur Verfügung. Er wird von zwei IDS überwacht, die ihre Funde an den Prelude Manager auf lxvm3-pub (siehe Abbildung 6.3) verschicken.	• GTK4 Simple-CA • PostgreSQL • GTK4 GridFTP • GTK4 RFT • GTK4 GRAM • Prelude-LML • Samhain • Snort
lxvm6-pub	lxvm6-pub ist in keinem Grid und befindet sich in einer anderen Domäne als die übrigen Rechner. Er dient als Ausgangspunkt eines Angriff (siehe Abschnitt 6.2.1)	• SMONA Message Manipulator Adapter • Prelude Manager • Prelude-LML

Tabelle 6.1: Rechner des Testaufbaus, ihre Aufgaben und Anwendungen

6.1.1 Konfiguration des Globus Toolkits

Dieser Abschnitt erläutert die Installation und Konfiguration des Globus Toolkits im Testaufbau. In Abbildung 6.2 sind die eingerichteten Dienste veranschaulicht. Die gepunkteten Linien umrahmen die Computer lxvm4-pub und lxvm5-pub und stellen ihre Grenzen dar.

Das Globus Toolkit (siehe Kapitel 3.1.6) wurde in der Version 4.0.5 nach der Beschreibung aus [GTQS 07] kompiliert und installiert. Eine detailliertere Installationsanweisung ist unter [GTAD 07] zu finden. Es folgt eine Kurzzusammenfassung der Einrichtung des Toolkits auf dem Computer lxvm5-pub:

Zuerst wurde auf dem Rechner lxvm5-pub ein neuer Benutzer (globus) eingerichtet, der Eigentümer der Globus Toolkit Dateien ist und dessen Funktion darin besteht, die Zertifizierungsstelle *SimpleCA* zu kontrollieren. Anschließend wurde der Globus Quellcode mit den Befehlen `configure --prefix=/usr/local/globus-4.0.5/`, `make` und `make install` kompiliert und installiert. Zur Erstellung von Benutzer- und Rechnerzertifikaten wurde die im Toolkit integrierte SimpleCA eingerichtet. Nachfolgend wurden Zertifikate für einen Testbenutzer und den Rechner erzeugt und von der SimpleCA signiert. Diese Zertifikate dienen der Authentifizierung und Autorisierung von Benutzern und werden zur Verschlüsselung des Netzver-

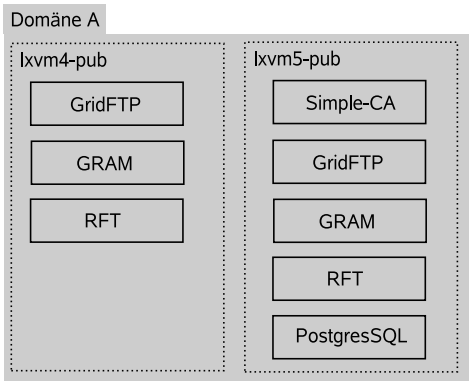

Abbildung 6.2: Die installierten Globus Toolkit Dienste

kehrs benötigt. Für den *Xinetd* Dämon wurde eine Konfigurationsdatei zum Starten und Stoppen des GridFTP Dienstes erschaffen. Neben der Porteinstellung in der Datei `/etc/services` benötigt GridFTP keine weiteren Änderungen. Zur Automatisierung des Start und Stopvorganges des Webservice Containers wurden zwei Scripte in den Verzeichnisse `/etc/init.d` und `/usr/local/globus-4.0.5/` erstellt. Das Script im Globus Stammverzeichnis übergibt dem Toolkit mehrere Optionen und wird vom Startskript (in `/etc/init.d`) aufgerufen. Nachfolgend wurde der RFT Dienst konfiguriert. Dazu musste zuerst die PostgreSQL Datenbank eingerichtet werden. Hierfür wurde ein neuer Benutzer (globus) in PostgreSQL erstellt, die Zugriffsrechte in der Konfigurationsdatei `pg_hba.conf` definiert und eine neue Datenbank generiert. Mit Hilfe des RFT Schemas aus der Datei `rft_schema.sql` wurde anschließend die neue Datenbank des RFT eingerichtet. Der Benutzername des Datenbank Administrators und sein Passwort müssen in der Konfigurationsdatei (`jndi-config.xml`) eingetragen sein. Zur Einrichtung des GRAM Dienstes wurden nur die Einstellungen des Programms *sudo* verändert und dem Benutzer globus root Rechte für das Programm `globus-gridmap-and-execute` gegeben. Die Globus Dienste WebMDS und MyProxy wurden nicht eingerichtet.

Auf dem Rechner lxvm4-pub sind die selben Dienste wie auf lxvm5-pub installiert. Die Benutzer- und Rechnerzertifikate wurden auf lxvm5-pub erstellt, signiert und anschließend an lxvm4-pub zurückgesendet. Der RFT Dienst auf diesem Computer verwendet die PostgreSQL Datenbank auf lxvm5-pub.

Der folgende Abschnitt legt die Überwachung des hier erörterten Grids durch eine IDS Infrastruktur mit Prelude dar.

6.1.2 Konfiguration der Prelude Architektur

Die Infrastruktur des Prelude Systems ist in beiden Domänen A und B unterschiedlich. Abbildung 6.3 illustriert die Verteilung der Intrusion Detection Systeme und ihre Kommunikation. Rechnergrenzen sind mit gepunkteten Linien gekennzeichnet und alle Computer einer Domäne befinden sich in einer grauen Fläche. In Domäne A arbeiten Intrusion Detection Systeme auf allen Rechnern und melden ihre Ereignisse an den Prelude Manager auf lxvm3-pub. Da die andere Domäne nur aus einem Rechner besteht, empfängt der Prelude Manager auf lxvm6-pub nur Informationen von dem lokalen Prelude-LML. Die folgenden beiden Unterabschnitte gehen genauer auf die installierten Komponenten aus dem Schaubild 6.3 ein.

Domäne A

In der Domäne A befinden sich unterschiedliche Intrusion Detection Systeme auf verschiedenen Rechnern (siehe Tabelle 6.1). Auf den am Grid beteiligten Computer (lxvm4-pub und lxvm5-pub) sind jeweils ein Protokoll- (Prelude-LML, Version 0.9.8.1) und ein Integritätsanalysator (Samhain, Version 2.3.3) eingerichtet. Auf lxvm5-pub überwacht zusätzlich noch Snort (Version 2.6.1.4) mit der Paketanalyse den Netzverkehr. Alle IDS verschicken ihre IDMEF Nachrichten verschlüsselt an den Prelude Manager (Version 0.9.8.1) auf lxvm3-pub. Lxvm3-pub wird in diesem Testumfeld nicht auf Angriffe überprüft. Zusätzlich ist auf allen Rechnern die

Prelude Bibliothek libprelude in der Version 0.9.13.2 installiert. Die IDS verwenden neben den Analyseregeln der Standardinstallation, die in den Abschnitten 5.3.1, 5.3.2 und 5.3.3 beschriebenen Regelsätze.

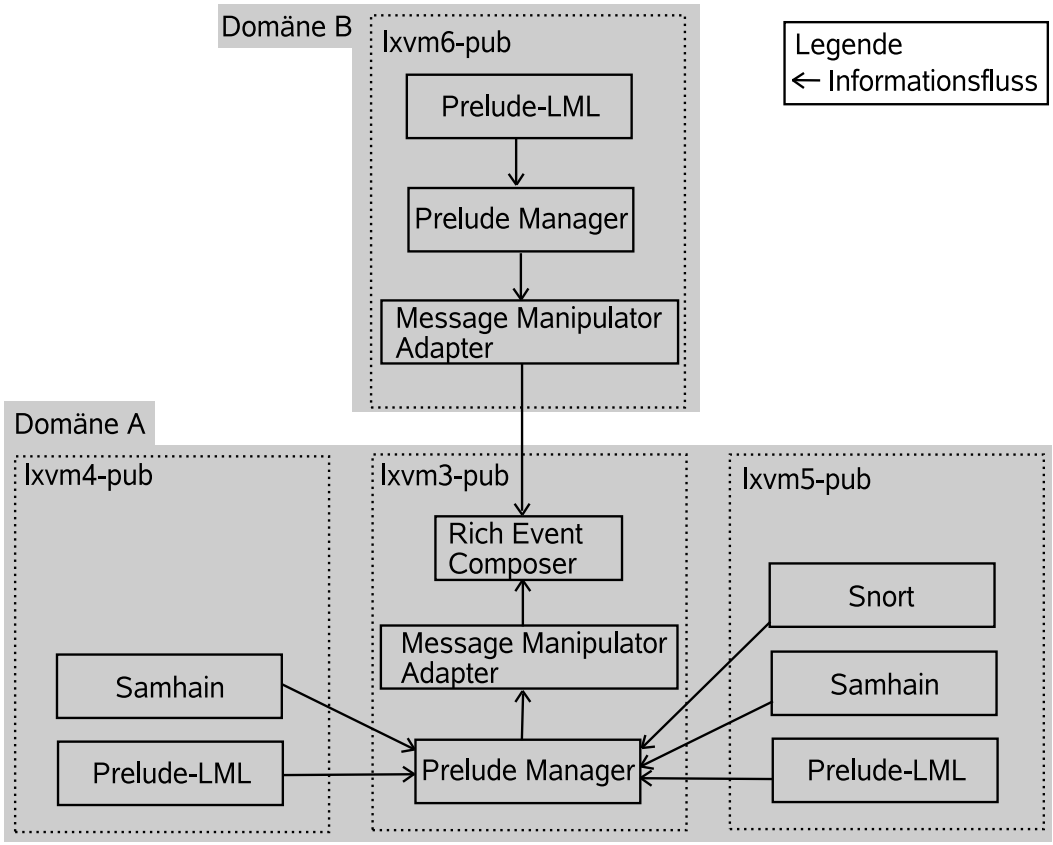

Abbildung 6.3: Verteilung und Kommunikation der IDS bis zum Rich Event Composer

Domäne B

Die Domäne B besteht aus nur einem Rechner. Auf ihm ist ein IDS, Prelude-LML (Version 0.9.8.1), installiert, das seine Informationen an den Prelude Manager (Version 0.9.8.1) auf dem selben Computer sendet. Beide verwenden die Bibliothek libprelude (Version 0.9.13.2).

6.1.3 Konfiguration der SMONA

Die SMONA Architektur wurde entwickelt um auf mehreren Rechnern gleichzeitig zu laufen. Wie in Abbildung 6.4 zu sehen ist, befinden sich im Testumfeld alle Komponenten der Anwendungsschicht und der Integrations- und Konfigurationsschicht auf dem Computer lxvm3-pub. Rechnergrenzen sind im Schaubild mit gepunkteten Linien und Domänen mit grauen Flächen gekennzeichnet. In den schwarzen Umrandungen stehen die Komponenten. Die verteilten Message Manipulator und Log Adapter senden alle ihre Nachrichten an den Rich Event Composer auf lxvm3-pub. Die Message Manipulator Adapter auf lxvm3-pub und lxvm6-pub untersuchen die Protokolldateien der Prelude Manager auf IDMEF Nachrichten, die sie nach Datenschutzrichtlinien verändern und weiterleiten. Die Log Adapter auf den Computer lxvm3-pub und lxvm4-pub analysieren die Systemprotokolldateien auf SSH Würmer und SSH Angriffe. Ihre Konfiguration ist in Abschnitt 6.2.4 in einem Anwendungsbeispiel erläutert. Ein eigenständiger JacORB, der in der Abbildung nicht zu sehen ist, stellt den Namensdienst auf lxvm3-pub für alle SMONA Komponenten beider Domänen zur Verfügung.

Der nachfolgende Abschnitt beschreibt mehrere Angriffe und ein Anwendungsbeispiel, mit denen der Testaufbau untersucht wurde.

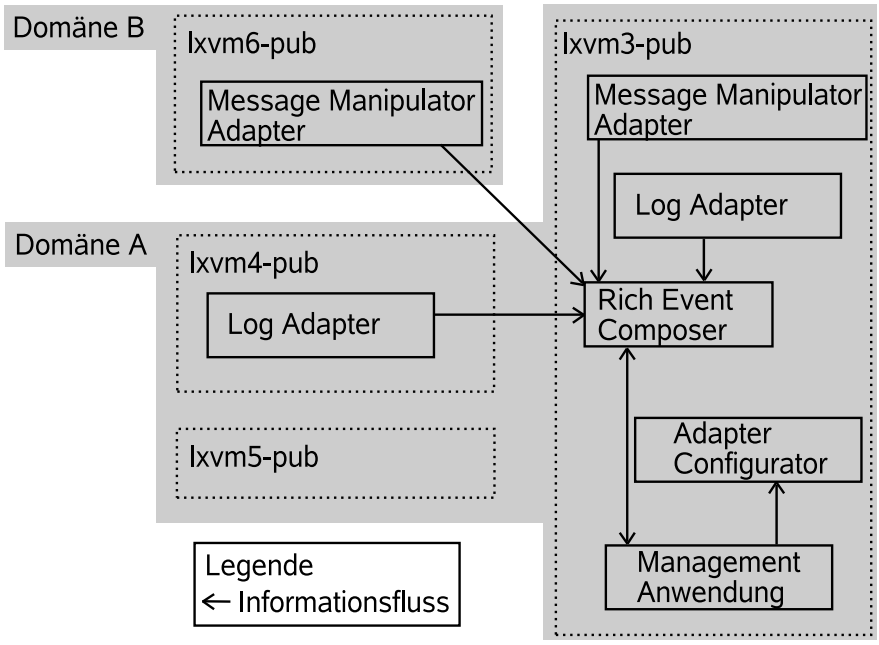

Abbildung 6.4: Verteilung und Kommunikation der SMONA Komponenten

6.2 Test des Sicherheitssystems

Dieser Abschnitt beschreibt mehrere Angriffe, die die Funktionalität und die Sicherheit der SMONA Implementierung überprüfen sollen. Zuerst wird versucht Falschinformationen an den Rich Event Composer zu versenden (siehe Abschnitt 6.2.1). Darauf aufbauend wird ein DoS Angriff beschrieben, der die Management Anwendung außer Betrieb setzt. Nachfolgend beschreibt Unterabschnitt 6.2.3 die Auswirkungen eines Angriffs, der das Beenden eines bestimmten Adapters zum Ziel hat. Abschließend wird (siehe Abschnitt 6.2.4) in einem Anwendungsbeispiel die Erkennung von SSH Würmern und SSH Angriffen mit Hilfe des Log Adapters erläutert.

6.2.1 Einschleusen von Fehlmeldungen

Dieser Abschnitt beschreibt das Einschleusen von falschen Informationen in die SMONA Architektur. Der Versuch wird auf dem Rechner lxvm6-pub (siehe Abbildung 6.1) durchgeführt, auf dem ein Message Manipulator Adapter läuft. Die übrigen SMONA Komponenten werden auf dem Computer lxvm3-pub ausgeführt. Ein Hindernis stellt die Verwendung des Trust- und Keystores durch die SMONA dar, der vom JacORB benutzt wird, um die Verbindungen zu verschlüsseln und Kommunikationspartner zu authentifizieren.

Nach der in Abschnitt 5.1.1 beschriebenen Einrichtung der Verschlüsselung beinhaltet jeder Keystore des Sicherheitssystems einen eigenen Schlüssel und vertrauenswürdige Zertifikate. Das verwendete Programm *keytool* schränkt standardmäßig die Zugriffsrechte auf die Datei nicht ein und sie ist für alle Benutzer des Computers lesbar. Die Konfigurationsdatei des JacORBs (*jacorb.properties*) ist normalerweise ebenfalls für alle Benutzer lesbar und enthält das Passwort im Klartext und den Speicherort des Keystores. Dies kann von einem Schadprogramm, das lokal ausgeführt wird ausgenutzt werden.

Befindet sich in einer Konfigurationsdatei der absolute Pfad des Keystores reicht es aus, dem Schadprogramm den Namen und Pfad der Datei zu übergeben, um sie zu verwenden. Wurde der relative Pfad angegeben, muss der Keystore in ein entsprechendes Verzeichnis des Schadprogramms kopiert werden. Durch die Verwendung der JacORB Konfiguration und eines SMONA Keystores kann die Verschlüsselung und Authentifizierung umgangen werden.

Bei diesem Angriff laufen die SMONA Komponenten auf dem Rechner lxvm3-pub und das Schadprogramm

(*DoSAdapter*) wird auf dem Computer lxvm6-pub ausgeführt. Der DoSAdapter verwendet zum Einschleusen von falschen Informationen die JacORB Konfigurationsdatei und den Keystore des Message Manipulator Adapters und importiert dessen Bibliothek. Er erstellt einen eigenen ORB und bezieht vom Namensdienst des Sicherheitsystems eine Referenz auf den Rich Event Composer. Die IP Adresse und der Port des Namensdienstes und der Pfad zur Konfigurationsdatei des Message Manipulator Adapter ORBs müssen beim Start eingestellt werden. Anschließend wird eine Nachricht mit der Quellidentität des Message Manipulator Adapters und einem beliebigem Text an der Rich Event Composer versendet, der die Nachricht verarbeitet und gegebenenfalls ein Rich Event an die Management Komponente weiterleitet. Wird die beim DoSAdapter benutzte Quellidentität von keiner Funktion im Rich Event Composer verwendet, speichert er die Meldungen ohne die Management Anwendung zu informieren.

Dieser Angriff kann verhindert werden, indem die Zugriffsrechte auf die JacORB Konfigurationsdatei und den Keystore verändert werden, so dass sie nicht mehr für alle lesbar sind. Außerdem sollte die Adapter Konfigurationsdatei nicht für alle lesbar sein, so dass ein Angreifer seine Nachrichten nicht mit einem gültigen Adapternamen versenden kann. Schließlich benötigt ein Angreifer detailliertes Wissen zur Implementierung der Schnittstellen des Rich Event Composers, um diese neu zu programmieren, oder er importiert die Bibliothek eines Adapters. Um dies zu verhindern, sollten dessen Zugriffsrechte ebenfalls eingeschränkt werden.

Auf den erfolgreichen Angriff in diesem Abschnitt beschreibt der nächste Unterabschnitt den Versuch die SMONA Architektur im Betrieb zu beeinträchtigen.

6.2.2 DoS Angriffe auf die SMONA Architektur

Die in diesem Abschnitt beschriebenen Angriffe basieren auf den im Unterabschnitt 6.2.1 erläuterten Sicherheitslücken und versuchen die SMONA Architektur an der Ausgabe von Nachrichten zu hindern. Voraussetzung dafür ist der Zugriff auf eine JacORB Konfigurationsdatei und ein Keystore innerhalb der SMONA Architektur. Außerdem müssen die IP Adresse und der Port des Namensdienstes und der Name der Management Anwendung, mit dem sie registriert ist, bekannt sein. Zur Implementierung des *DoSAdapters* werden die Java Bibliotheken der Adapter und der SMONA benötigt. Für beide Versuche wurde das bereits kurz erläuterte Programm erweitert. Diesmal werden anstatt einzelner Fehlmeldungen, viele, große Nachrichten verschickt.

Im diesem Test wurden in einer iterativen Schleife zwanzigtausend 1 kB große Rich Events, ohne Pause, an die Management Anwendung verschickt. Dieser Versuch wurde mehrmals wiederholt und es stellte sich heraus, dass das Versenden durchschnittlich eine Minute dauerte. Die CPU Auslastung der Management Anwendung blieb jedes Mal unter 60 Prozent. Nach circa 13398 Rich Events erschien eine Fehlermeldung, dass die Management Anwendung nicht mehr genug Speicher hat, um weiter Nachrichten darzustellen und es wurden keine Meldungen mehr ausgegeben. Daraufhin wurde eine neue Aggregation mit einem Test Adapter gestartet, dessen Rich Events nach wenigen Meldungen ebenfalls nicht mehr dargestellt wurden und die grafische Oberfläche nicht mehr funktionierte. Die Aggregation konnte nicht mehr beendet werden und jeder Versuch führte zu einer Fehlermeldung über zu wenig Speicher.

Daraufhin wurde die Ausgabe der grafischen Oberfläche auskommentiert, die Management Anwendung neu kompiliert und Nachrichten nur noch auf die Konsole geschrieben. Durch einen wiederholten Test mit zwanzigtausend 1 kB großen Rich Events konnte der Betrieb nicht mehr gestört werden.

Daraus lässt sich folgern, dass die Bildschirmausgabe in der verwendeten Implementierung ein Sicherheitsrisiko darstellt und die Leistungsfähigkeit des Systems stark beeinträchtigt. Zur Entschärfung des Problems können die Zugriffsrechte auf den Keystore, die Konfigurationsdateien und die SMONA Bibliothek eingeschränkt werden, so dass ein Angreifer keinen gültigen Schlüssel erhält und die Bibliothek nicht verwenden kann. Außerdem sollten die Namen, mit denen die SMONA Komponenten beim Namensdienst registriert sind nicht öffentlich und nicht leicht zu erraten sein. In einer späteren Implementierung sollte die graphische Oberfläche alte Nachrichten und Statusmeldungen archivieren und vom Bildschirm entfernen, damit sie weniger Speicher benötigt.

Nachdem in diesem Angriff die SMONA Architektur erfolgreich in ihrem Betrieb gestört wurde, beschreibt der folgende Abschnitt, wie ein Adapter von einem falschen Adapter Configurator beendet werden kann.

6.2.3 Beenden eines Adapters als Betriebsstörung

Basierend auf den in Abschnitt 6.2.1 beschriebenen Sicherheitlücken bezüglich der Leserechte des Keystores wurde versucht einen aktiven Message Manipulator Adapter mit einem falschen Adapter Configurator zu beenden. Für diesen Angriff wurde ein neues Programm (*FakeAC*) geschrieben, das auf dem Rechner lxvm3-pub lief. Die Komponenten der SMONA Architektur und ein Message Manipulator Adapter wurden auf dem selben Rechner ausgeführt.

FakeAC importiert die Java Bibliothek der Adapter und benötigt Zugriff auf den Keystore des Adapter Configurators um sich mit dem Adapter verbinden zu können. Darüberhinaus muss es den registrierten Namen des Zieladapters kennen. Mit diesem Namen kann vom Namensdienst eine Referenz auf den Adapter geholt und der Adapter mit der Methode *stop* beendet werden.

Nach dem Stoppen eines Push Adapters in diesem Angriff wird er trotzdem in der Liste der Adapter des Adapter Configurators geführt. Für die Management Anwendung bleibt der Angriff ebenfalls unbemerkt und die Aggregation kann nicht mehr korrekt bearbeitet werden. Erkannt wird die Attacke nur, wenn die Nachrichten des Adapters von einer *timeout* Funktion im Rich Event Composer überprüft werden. Wird ein Pull Adapter mit dem Programm *FakeAC* beendet, berichtet der Adapter Configurator mit einer Fehlernachricht, dass keine Informationen von ihm mehr abgefragt werden können.

Dieses Sicherheitsproblem kann ebenfalls durch einen Leseschutz des Keystores und der Konfigurationsdateien verhindert werden. Darüberhinaus sollte in einer späteren Implementierung jeder Adapter Ereignisse, wie das Starten oder Stoppen einer Datenquelle, direkt an die Management Anwendung melden.

6.2.4 Anwendungsfall: Erkennung von SSH Wurm Angriffen

In diesem Abschnitt wird ein Anwendungsfall für das in dieser Arbeit konzipierte und implementierte System zur Föderation von Intrusion Detection Systemen in Grid Infrastrukturen erläutert. Er beschreibt wie das System zur Erkennung von SSH Würmern und SSH Angriffen eingesetzt werden kann. Zuerst werden die Eigenschaften einer solchen Attacke erläutert. Anschließend wird eine exemplarische Aggregation dargelegt, mit der die SMONA Architektur konfiguriert werden kann, um diese Angriffe aufzuspüren.

Der SSH Wurm versucht sich mit Brute-Force Angriffen über SSH (siehe [YlLo 06]) auf einem Rechner einzuloggen. Dazu generiert er verschiedene Benutzernamen und Passwörter, die in kurzen Zeitabständen an einen SSH Server geschickt werden. Dabei hinterlässt er, in Abhängigkeit vom verwendeten SSH Server und seiner Einstellung, eine Spur in der Systemprotokolldatei von ein bis mehreren Einträgen je Anmeldung. Ein Angriff kann mehrere hundert Loginversuche umfassen. Ein signifikantes Merkmal des Wurm Angriffs ist der kurze Zeitabstand zwischen den Anmeldungen. In zwei vorliegenden Logdateien waren es durchschnittlich 4 Sekunden. In einer Domäne eines Grids können sich normalerweise Benutzer auf den meisten Rechnern mit dem selben Benutzernamen und Passwort anmelden. Diese Eigenschaft kann in einer Weiterentwicklung des SSH Wurms genutzt werden, um Angriffe zu verstecken. Dazu wird nicht ein Rechner in kurzer Zeit mit Brute-Force Attacken angegriffen, sondern die Anmeldeversuche auf alle Rechner der Domäne verteilt. Dadurch können die Loginversuche auf einem Computer in größeren Abständen erfolgen, ohne den Angriff insgesamt zu verlängern.

Zur Erkennung eines Angriffs werden in der Testumgebung zwei Log Adapter auf den Rechnern lxvm3-pub und lxvm4-pub installiert, die lokal die Systemprotokolldateien auf sich wiederholende Einträge von fehlgeschlagenen SSH Anmeldungen untersuchen. Dadurch können bereits SSH Würmer aufgespürt werden. SSH Angriffe, die ihre Anmeldeversuche auf alle Rechner einer Domäne verteilen benötigen eine zentrale Analyse der einzelnen Einträge der SSH Server. Diese wird mit Hilfe einer Aggregation beim Rich Event Composer konfiguriert und durchgeführt.

In Anhang D ist einem Listing die Aggregation zur Identifikation von SSH Angriffen dargestellt. Zu Beginn wird eine kurze Beschreibung der Aggregation angegeben. Anschließend folgt eine Konfiguration zweier Adapter (auf lxvm3-pub und lxvm4-pub), die alle benötigten Parameter enthalten. Die Log Adapter sollen Einträge aus den exemplarischen Systemprotokolldateien herausfiltern, die einen misslungenen Login enthalten und die Quell IP Adressen speichern. Sie dient als Unterscheidungsmerkmal der Nachrichten. Treffen zwei Meldungen mit der selben IP Adresse innerhalb von 20 Sekunden ein, wird ein Wurmangriff festgestellt. Für

Einträge, die außerhalb dieses Zeitrahmens sind, wird der Nachrichtentext „Failed SSH Login from" zusammen mit der IP Adresse versendet und im Rich Event Composer analysiert. Im Funktionsabschnitt werden drei Methoden präzisiert. Die erste Funktion untersucht die Meldungen der Ressourcen auf Nachrichten über einzelne fehlgeschlagene Anmeldungen. Die zweite Funktion baut auf die Rückgabe der ersten auf. Sie verwendet die Methode timediff, um den zeitlichen Zusammenhang der Mitteilungen beider Adapter zu überprüfen. Zur Unterscheidung der einzelnen Anmeldeversuche wird ihre Quell IP Addresse benutzt. Stimmt die IP Adresse zweier Nachrichten überein und wurden sie innerhalb von 15 Sekunden empfangen, wird ein SSH Angriff erkannt. Die letzte Funktion stützt sich auf die Methode contain und gibt alle Meldungen zurück, die von einem SSH Wurm Angriff zeugen. Im letzten Teil der Aggregation sind zwei Notifikationen definiert, die die Ergebnisse der Funktionen in einem Rich Event an die Management Anwendung schicken. Die erste Notifikation prüft in ihrer Bedingung, ob ein SSH Angriff von einer Funktion erkannt wurde und meldet ihn. Die zweite Notifikation generiert ein Rich Event, falls eine Nachricht über einen SSH Wurm von einem Adapter empfangen wurde.

Die oben beschriebene Aggregation wurde im Testumfeld installiert und überprüft. Die analysierten Protokolldateien enthalten jeweils einen Wurmangriff und verschiedene fehlgeschlagene Anmeldeversuche von unterschiedlichen IP Adressen. Dabei entsprechen fünf Einträge einem SSH Angriff nach den oben erläuterten Kriterien an. Die Ausgabe der Management Anwendung ist in Abbildung 5.3 dargestellt. Es wurden alle SSH Würmer und Angriffe richtig erkannt.

Nach der Erörterung verschiedener Test und ihrer Ergebnisse wird im nächsten Abschnitt eine Bewertung des Systems zur Föderation von Intrusion Detection Systemen gegeben.

6.3 Bewertung der Konzeption und ihrer Umsetzung

Zu Beginn dieses Abschnitts werden allgemeine Probleme des Entwurfs und der Umsetzung erläutert. Anschließend folgt eine Gegenüberstellung der Anforderungen mit dem entworfenen und realisierten Sicherheitssystem. Im abschließenden Unterabschnitt wird eine kurze Bewertung abgegeben.

6.3.1 Bemerkungen

Dieser Abschnitt beschreibt allgemeine und spezielle Probleme der Konzeption und der Implementierung. Dabei wird auf die einzelnen Komponenten des Sicherheitssystems eingegangen und Schwachstellen diskutiert.

Kritikpunkte am Entwurf und der Realisierung

Die SMONA Architektur wurde zur Überwachung von Diensten und ihren Attributen entwickelt und sieht deshalb mehrere Mechanismen, die in einem Sicherheitssystem benötigt werden, nicht vor. Zum Beispiel wird durch JacORB eine Authentifizierung und Autorisierung aller Komponenten erreicht. Wie aber in den Abschnitten 6.2.1, 6.2.2 und 6.2.3 erläutert, kann dies umgangen werden und deshalb sollten sich die SMONA Komponenten gegenseitig Authentifizieren und Autorisieren, ohne auf CORBA oder JacORB angewiesen zu sein. Dazu müssen umfassende Leitlinien entworfen werden, die definieren, welche Komponenten und Benutzer Zugriff auf welche Funktionalitäten und in welcher Form benötigen. Anschließend können Mechanismen und funktionale Einheiten für die Programme der SMONA Komponenten zur Gewährleistung der Sicherheitsrichtlinien ausgesucht und konzipiert werden.

In dem entworfenen Sicherheitssystem sind die Komponenten relativ fest miteinander verbunden. Fällt eine SMONA Komponente aus, oder ist sie nicht mehr erreichbar, so kann dies dazu führen, das andere beendet und neu gestartet werden müssen. Alternativ könnte vor jedem Nachrichtenversandt ein Adapter die Verbindung erneut herstellen. Dies ist bei einer hohen Anzahl von Meldungen ineffizient. In einer erweiterten Konzeption des Kommunikationsmodells muss die Ausfallsicherheit stärker berücksichtigt werden. Zum Beispiel könnten die Adapter einen Ausfall des Rich Event Composers an die Management Anwendung melden, der daraufhin einen neuen Composer startet und den Adaptern über den Adapter Configurator die neue Referenz einstellt. Außerdem sollten die SMONA Komponenten der höheren Schichten eine Schnittstelle zu einer persistenten

Datenbanken enthalten, die Daten und Konfigurationen dauerhaft speichern, so dass bei einem Ausfall die bisherigen Daten nicht verloren gehen.

Die Performanz des Rich Event Composers wurde in ihrem Entwurf nur teilweise berücksichtigt und lässt sich noch steigern. Die Auswertung von boolschen Ausdrücken des *ConditionInterpreters* und der Funktionumfang des *FunctionProcessors* sind stark eingeschränkt und bedürfen der Erweiterung und Leistungssteigerung. Nachdem der Rich Event Composer eine zentrale Sammelstelle für alle sicherheitsrelevanten Nachrichten und Monitoring Attribute ist, könnte er, wie der GIDS Server im GIDS Entwurf von Ong Tian Choon und Azman Samsudin (siehe Abschnitt 1.2 und [ChSa 03]) alle die Daten auf Anomalien und verteilte Angriffe untersuchen. Dafür werden zusätzliche Funktionen benötigt und es müssen erweiterte boolsche Ausdrücke ausgewertet werden können.

Administratoren sind in diesem Entwurf selbst für die Manipulation von Alarmmeldungen verantwortlich und können die Nachrichten ausversehen verstümmeln. Es sollten Richtlinien entworfen werden, welche Informationen einer Alarmnachricht zur weiteren Analyse unabdingbar sind, und welche verändert werden dürfen. Außerdem wäre ein Werkzeug wäre hilfreich, mit dem die Veränderungen an den Nachrichten getestet werden können, ohne dass dafür eine eigene SMONA Testumgebung benötigt wird.

Ein großes Problem der Implementierung besteht bei der graphischen Oberfläche und ihrem begrenzten Speicher. Hier sollte eine Erweiterung alte Nachrichten archivieren und entfernen und dadurch die Speichernutzung begrenzen.

IDMEF im XML Format

IDMEF ist ein umfassender Standard zur Formatierung von IDS Meldungen. In ihm ist ein XML Schema definiert, das den Aufbau von IDMEF Alarm und Heartbeat Nachrichten beschreibt. Durch die Verwendung von XML entsteht ein beachtlicher Overhead an Steuerinformationen, der die Verarbeitung der Nachrichten verlangsamt. In kleinen Systemen, in denen nur wenige Meldungen generiert und verschickt werden ist dies nur ein geringfügiges Problem. In Grids werden viele Intrusion Detection Systeme benötigt, deren Informationen ohne XML Tags bereits ein großes Datenvolumen ausmachen können.

Globus Toolkit

Auf der Seite des Toolkits bestehen mehrere Probleme im Bereich der Protokollierung. Das Format der Logdatei entspricht nicht dem Unix Standard. Das Datum- und Uhrzeitformat unterscheiden sich von denen in der Protokolldatei *messages*. Darüberhinaus wird beim Start des Globus Containers keine Uhrzeit angegeben und stattdessen in mehreren Zeilen die gestarteten Dienste aufgelistet. Fehlermeldungen eines Dienstes werden ebenfalls in mehreren Zeilen ausführlich beschrieben. Die meisten Protokollanalysatoren erwarten einen Logeintrag in einer Zeile und können mehrere Zeilen nicht korrelieren. Weiterhin besteht nur eine eingeschränkte Möglichkeit Einfluss auf den Umfang der Protokollierung zu nehmen. Während der RFT Dienst keine Einträge schreibt, protokolliert der Ausführungsdienst für jeden angenommenen Auftrag drei Ereignisse. Schließlich wird bei jedem Neustart des Globus Servers die Logdatei überschrieben, so dass alte Einträge nicht mehr vorhanden sind und nicht mehr analysiert werden können.

Corba

Das objektorientierte, plattform- und programmiersprachenunabhängige CORBA Framework ist gut geeignet, um universelle Schnittstellen zu implementieren. Außerdem bietet CORBA verschiedene Dienste an. In dieser Arbeit wurde der Namensdienst verwendet, um Verbindungen zwischen den Komponenten aufzubauen. Dazu verwendet CORBA unterschiedliche TCP Ports, die im domänenübergreifenden Verkehr an der Firewall geöffnet werden müssen. Dadurch entsteht ein erhöhter Konfigurationsaufwand und es bilden sich neue Sicherheitrisiken. Die verwendete CORBA Implementierung ist über das Internet ansprechbar und ein Angreifer kann diese nach Sicherheitslücken untersuchen. Der zentrale Namensdienst gründet eine weitere Schwachstelle. Ist er nicht erreichbar, können keine neuen Verbindungen aufgebaut und keine neuen Adapter in die SMONA Architektur integriert werden.

Nach der Erörterung diverser Probleme des Sicherheitssystems thematisiert der folgende Abschnitt die Erfüllung der Anforderungen.

6.3.2 Überprüfung der Anforderungen

Dieser Abschnitt diskutiert die Umsetzung der in Kapitel 2.2 erörterten Anforderungen an das System zur Föderation von Intrusion Detection Systemen in Grid Infrastrukturen.

Die Einhaltung der geforderten Eigenschaften der Intrusion Detection Systeme und ihrer Architektur lässt sich nicht überprüfen, da in dieses Sicherheitssystem beliebige IDS integriert werden können und die Konzeption der Überwachung Aufgabe eines Administrators ist. Durch die Verwendung der Prelude Infrastruktur und die allgemeine Implementierung des Message Manipulator Adapters können viele, unterschiedliche IDS in das System integriert werden, aus denen sich ein Administrator die geeignetsten aussuchen kann.

In dieser Arbeit wurden drei IDS ausgewählt und für die Überwachung des Globus Toolkits Analyseregeln entworfen. Den Anforderungen aus Abschnitt 2.2.3 konnte auf Grund fehlender Literatur und der eingeschränkten Protokollierung des Globus Toolkits nur zum Teil genüge geleistet werden. Die Regeln zur Protokoll- und Paketanalyse sind sehr weit gefasst und produzieren viele Fehlalarme (false positiv). Die Einstellungen zur Überwachung der Globus Dateien können nur die Auswirkungen eines Angriffs erkennen, die Spuren im Dateisystem hinterlassen. Dafür sind sie aber speziell angepasst und gut geeignet.

Die in Abschnitt 2.2.4 beschriebenen Ansprüche an die Adapter wurden bereits teilweise durch die Verwendung des SMONA Adapter Frameworks erfüllt. Dazu gehören der normalisierte Zugriff, die asynchrone Kommunikation und die Definition programmiersprachenunabhängiger Schnittstellen. In dieser Arbeit wurde zusätzlich durch die verschlüsselte Kommunikation über JacORB eine Autorisierung und Authentifizierung der Adapter eingeführt. Diese Funktionalität sollte aber auch in den SMONA Komponenten konzipiert werden. Außerdem können Adapter lokal mit einer Konfigurationsdatei und zentral durch den Adapter Configurator eingerichtet werden. Lokale Einstellungen können durch die zentrale Konfiguration nicht überschrieben werden, so dass die Souveränität der Domänen gewahrt bleibt. Der Message Manipulator Adapter, der Log Adapter und die erneuerten Push und Pull Adapter sind nicht auf die Verwendung bestimmter Programme auf der Ressourcenebene der SMONA spezialisiert und können dadurch für andere Problemstellungen verwendet werden. In zukünftigen Entwürfen und Implementierungen müssen die Anforderungen der Performanz, der Selbstbeschreibung der Fähigkeiten durch einen Adapter, das dynamische Konfigurieren der Adapter während dem Betrieb und der bedarfsgesteuerte Adapterstart im Entwurf berücksichtigt und umgesetzt werden.

Die geforderten Eigenschaften der SMONA Komponenten der Integrations- und Konfigurations und der Anwendungsschicht wurde größtenteils in dieser Arbeit realisiert. programmiersprachenunabhängige Schnittstellen, die asynchrone, verschlüsselte Kommunikation und die Autorisierung und Authentifizierung werden durch die Verwendung von CORBA und JacORB erfüllt. Drüberhinaus bietet die Management Anwendung eine zentrale Benutzerschnittstelle, über die Aggregationen und dadurch Adapter mit der Service Information Specification Language konfiguriert werden können. Ein bedarfsgesteuerter Adapterstart über den Adapter Configurator ist durch die Angabe einer Aggregation noch nicht möglich. Ebenso wurde die Möglichkeit eine Komponente gleichzeitig auf mehreren Rechnern zum Lastausgleich auszuführen zwar berücksichtigt aber nicht realisiert. Das größte Problem stellt die Ausfallsicherheit des Systems dar. Im SMONA Adapter Framework ist nicht vorgesehen, dass Adapter dynamisch für das Versenden an einen Rich Event Composer konfiguriert werden können. Dadurch verlangt der Ausfall des Rich Event Composers einen Neustart aller Aggregationen und Adapter.

Auf die Diskussion über die Realisierung der Anforderungen folgt im abschließenden Abschnitt eine Bewertung der Konzeption und Implementierung.

6.3.3 Abschließende Bewertung

Dieser Abschnitt erläutert eine kurze, abschließende Beurteilung der Konzeption, der Implementierung und der in diesem Kapitel erläuterten Schwachstellen.

Die Überwachung der Grid Middleware ist auf Grund fehlender Literatur und mangelhaften Einstellungsmöglichkeiten zur Protokollierung des Toolkits nur bedingt aussagekräftigt. Sie produziert viele Nachrichten, unter denen auch viele Fehlalarme (false positivs) sind. Eine Überwachung des Toolkits benötigt eine tiefgreifende Analyse seine Komponenten, Funktionsweise und ihrer Schwachstellen, die über den Umfang dieser Diplomarbeit hinausgeht.

Die meisten der in den Abschnitten 6.2.1, 6.2.2 und 6.2.3 beschriebenen Schwachstellen lassen sich durch einfache Konfigurationen beheben. Zur Durchführung der Angriffe muss ein lokaler Zugriff auf verschiedene Konfigurationsdateien, den Keystore und die SMONA Bibliotheken bestehen. Außerdem muss der Angreifer die Architektur und die beim Namensdienst registrierten Namen kennen. Durch strikte Sicherheitsrichtlinien kann all das verhindert werden.

Schließlich wurde die Performanz in der Implementierung wenig berücksichtigt. Diese ist aber nur prototypisch und erbringt den Nachweis der Tragfähigkeit der Konzeption. Im Entwurf fehlen funktionale Einheiten, die eine Optimierung der Verarbeitung der SISL Funktionen und Notifikationen durchführt.

Mit den in den vorherigen Abschnitten erläuterten Kritikpunkten leistet das in dieser Arbeit entwickelte System die Föderation von Intrusion Detection Systemen in Grid Infrastrukturen. Sie ermöglicht die Überwachung der Grid Middleware Globus Toolkit auf Angriffe und konzentriert die Nachrichten der IDS im Rich Event Composer. Dabei können die sicherheitsrelevanten Meldungen nach Datenschutzrichtlinien verändert, in Aggregationen nachbearbeitet und von der Management Anwendung ausgegeben werden. Für eine Analyse verteilter Angriffe auf ein Grid muss der Funktionsumfang des *Function Processors* noch erweitert und die Auswertung der boolschen Ausdrücke komplettiert werden. Die in dieser Arbeit entworfenen Methoden bieten dafür eine Grundlage. Das Kommunikationsmodell der SMONA sollte zur höheren Sicherheit modifiziert werden, so dass zum Beispiel die Authentifizierung und Autorisierung in den SMONA Komponenten konzipiert wird. Außerdem sollten die Daten der Adapter, der SISL Funktionen und die Rich Events in persistenten Datenbanken gespeichert werden, damit bei Ausfällen und einem Neustart alte Daten nicht verloren gehen.

Im letzten Kapitel dieser Arbeit wird eine Zusammenfassung gegeben und in einem Ausblick zukünftige Erweiterungsmöglichkeiten und nachfolgende Arbeiten erläutert.

7 Zusammenfassung und Ausblick

In diesem letzten Kapitel der Diplomarbeit wird die Ausarbeitung der Aufgabenstellung überblickartig resümiert. Zurerst werden die Ergebnisse der Arbeit festgehalten und anschließend fasst Abschnitt 7.1 die letzten Kapitel und ihre Resultate zusammen, bevor abschließend in Abschnitt 7.2 ein Ausblick auf mögliche Folgearbeiten gegeben wird.

7.1 Zusammenfassung

Aufgabe der Diplomarbeit war die Überwachung der Grid Middleware Globus Toolkit auf Angriffe und die Föderation der sicherheitsrelevanten Nachrichten. Dazu gehörte die Konzentration der Meldungen an einer zentralen Stelle und die Veränderung der Alarmnachrichten um Datenschutzverletzungen zu vermeiden. Es wurden Komponenten aus der SMONA Architektur ausgewählt und für diese Aufgabenstellung erweitert und implementiert.

Zu Beginn wurde in Kapitel 2 ein Grid Szenario dargestellt und daran Anforderungen an die verschiedenen Teilsysteme zur Bewältigung der Aufgabenstellung erarbeitet. Die geforderten Eigenschaften flossen in die Konzeption und Implementierung ein und ihre Umsetzung wurde in Abschnitt 6.3.2 diskutiert.

Die notwendigen Grundlagen erklärte Kapitel 3. Dabei wurden Grid Konzepte, die Open Grid Service Architecture und ihre Sicherheitsmechanismen erörtert. Anschließend erfolgte die Auflistung verschiedener Monitoring Lösungen und eine Beschreibung der Grid Middleware Globus Toolkit. Speziell wurde auf die Service Monitoring Architecture (SMONA) und die Service Information Specification Language (SISL) eingegangen, deren Komponenten zur Erfüllung der Aufgabenstellung weiterentwickelt und implementiert wurden. Schließlich führte das Kapitel kurz in die Thematik der Intrusion Detection Systeme ein, die kategorisiert und deren Erkennungsmechanismen erläutert wurden.

Aufbauend auf die Grundlagen teilte Kapitel 4 die Konzeption des Systems zur Föderation sicherheitsrelevanter Nachrichten in drei Abschnitte auf. Zuerst wurden SMONA Komponenten ausgewählt und an die Aufgabenstellung angepasst. Dazu erweiterten sich die SISL Ressourcen um Parameter, die in der Konfiguration der Adapter initiale Einstellungen vornehmen. Außerdem wurden die Parameter der SISL Funktionen verändert, um den Methoden konstante Werte übergeben zu können. Die Management Anwendung entwickelte sich zur rudimentären Kontrollstation, in der Aggregationen eingegeben werden können und die die Darstellung von Rich Events zur Ausgabe hat. Der Adapter Configurator erhält zum Starten und Konfiguration der Adapter ein Aggregationsobjekt, dessen Ressourcen er in ein durch das Adapter Framework vorgegebenes Format transformiert. Die Funktionen die der Rich Event Composer ausführen kann, wurden um sechs Methoden erweitert, mit denen Alarmnachrichten der Intrusion Detection Systeme nachverarbeitet und aggregiert werden können. Auf die Auswertung der boolschen Ausdrücke der Bedingungen zur Nachrichtenerzeugung wurde weniger Wert gelegt. Der nachfolgende Abschnitt erläuterte Anpassungen des SMONA Adapter Frameworks und den Entwurf zweier neuer Adapter. Durch die Modifikationen erleichtert sich die Entwicklung neuer Adapter und es können flexibel Parameter an sie übergeben werden. Außerdem führte die Adaption der Push und Pull Adapter zu der Möglichkeit beliebige Programme zur Informationsgewinnung ausführen zu können. Die neu entworfenen Adapter können Protokolldateien auslesen und entweder deren Einträge nach Datenschutzrichtlinien verändern, oder sie auf zeitliche Beziehungen untersuchen. Die Ergebnisse werden an den Rich Event Composer weitergeleitet und dort verarbeitet. Der letzte Abschnitt des Kapitels behandelte die Entwicklung von Analyseregeln zur Überwachung der Grid Middleware. Es wurden Regeln zur Integritätsüberwachung der Globus Dateien, zur Auswertung der Protokolldatei und der Begutachtung des Netzverkehrs auf Angriffe konzipiert. Die Überwachung des Globus Toolkits stellte sich dabei auf Grund fehlender Literatur und mangelnder Einstellungsmöglichkeiten bei der Protokollierung als schwierig heraus und erzeugt deshalb viele Fehlalarme

niedriger Priorität. Die Tragfähigkeit der Konzeption und ihre Leistungsfähigkeit wurden in der Implementierung nachgewiesen und im Anwendungsbeispiel in Abschnitt 6.2.4 aufgezeigt.

In Kapitel 5 erhielt der Leser einen Einblick in die prototypische Umsetzung des Entwurfs. Es wurde erörtert, warum in dieser Arbeit das Kommunikationsformat IDMEF und das CORBA Framework verwendet werden. Anschließend wurde die Einstellung der verschlüsselten Kommunikation der SMONA Komponenten aufgezeigt. Die folgenden Unterabschnitte beschrieben an Hand von UML Klassendiagrammen die Umsetzung der SMONA Komponenten. Die Aggregationen zur Konfiguration des Rich Event Composers und der Adapter werden in einem XML Format angegeben, dessen Schema erläutert wurde und in Anhang B dargestellt ist. Die daraus entstehenden CORBA Aggregations Objekte wurden in einem Diagramm illustriert. Die Implementierung der funktionalen Einheiten der SMONA erfolgte in eigenständigen Programmen der Programmiersprache Java. Ihre Schnittstellen wurden mit der Interface Description Language beschrieben und in den jeweiligen Abschnitten erläutert. Mit Hilfe von SISL Beispielen veranschaulichte dieses Kapitel die Adapterkonfiguration und die Funktionsverarbeitung des Rich Event Composers. In der Beschreibung der Umsetzung der Adapter wurden zuerst die nötigen Anpassungen erörtert. Dazu gehörte vor allem die Möglichkeit Adapter lokal durch eine Konfigurationsdatei und zusätzlich durch den zentralen Adapter Configurator einstellten zu können. Aus Gründen der Souveränität der Domänen, die an einem Grid beteiligt sind, können die lokalen Einstellungen nicht vom Adapter Configurator überschrieben werden. Darüberhinaus können nun beliebige Werte in der Adapter Konfiguration eingerichtet und gespeichert werden. Die Darlegung der Implementierung der neuen Adapter erfolgte ebenfalls an Hand von Klassendiagrammen. Der Message Manipulator Adapter liest Manipulationsregeln, die sich in ihrem Format an Java Pattern anlehnen, aus einer Datei aus und wendet sie auf jeden Eintrag einer eingestellten Protokolldatei an bevor eine Nachricht an den Rich Event Composer verschickt wird. Der Log Adapter analysiert Ereignisnachrichten aus Logdateien auf Grund einer einstellbaren Kategorisierung und ihres Zeitstempels. Der letzte Abschnitt dieses Kapitel thematisierte die Konfiguration von Analyseregeln zur Überwachung des Globus Toolkits. Mit Hilfe von Listings wurden die Regeln der Intrusion Detection Systeme und deren Aufbau erklärt.

Im darauf folgenden Kapitel 6 wurde die Konzeption und ihre Implementierung mit Hilfe von verschiedenen Test und einer beispielhaften Anwendung in einer Testumgebung evaluiert und bewertet. Die dabei entdeckten Schwachstellen sind zum größten Teil auf Leserechte von Konfigurationsdateien und des Keystores zur Verschlüsselung zurückzuführen und können durch strikte Sicherheitsrichtlinien verhindert werden. Das Anwendungsbeispiel erläutert an Hand einer exemplarischen Aggregation die Erkennung von SSH Würmern und SSH Angriffen. Die abschließende Bewertung erläutert nocheinmal allgemeine Probleme und überprüft die in Kapitel 2 erörterten Anforderungen.

Zum Abschluss dieses Abschnitts werden noch einmal die Ergebnisse zusammengefasst.

7.1.1 Ergebnisse

In dieser Diplomarbeit wurde aufgezeigt, dass eine umfassende Überwachung von Grid Infrastrukturen notwendig und mit der Monitoring Architektur SMONA möglich ist. Dafür waren und sind Modifikationen essentiell, die die Sicherung der Architektur und Erweiterungen ihres Funktionsumfangs zum Ziel haben. Die SMONA kann relevante Nachrichten von Intrusion Detection Systemen unter Berücksichtigung von Datenschutzrichtlinien sammeln und weiterverarbeiten. Darüberhinaus können verteilte Angriffe im Rich Event Composer an Hand der gesammelten Nachrichten entdeckt werden. Die gefundenen Schwachstellen sind größtenteils auf den prototypischen Charakter der Implementierung zurückzuführen und können auf einfachem Wege vermieden werden. Für eine bessere Überwachung der Grid Middleware sind weitere Arbeiten, die sich mit dessen Sicherheit beschäftigen, notwendig.

Der nächste Abschnitt gibt einen Ausblick auf offen gebliebene Fragen und mögliche Erweiterungen des Sicherheitssystems.

7.2 Ausblick

Auf die, in dieser Arbeit konzipierte Föderation von Alarmmeldungen kann in verschiedenen Richtungen aufgebaut werden.

In den Abschnitten 6.3.2 und 6.3.3 wurde auf den eingeschränkten Funktionsumfang des Function Processors und des Condition Interpreters hingewiesen. Dabei stellt sich die Frage welche Methoden und Ausdrücke weiterhin benötigt werden, um generell verteilte Angriffe mit der SMONA Architektur erkennen zu können. Dazu könnten dann neben Alarmnachrichten von Intrusion Detection Systemen auch Monitoringinformationen und Dienstattribute zur Überwachung von verteilten Systemen herangezogen und zum Beispiel mit der statistischen Anomalieerkennung ausgewertet werden. Die Frage bleibt bestehen, in welcher Form Gegenmaßnahmen durch die SMONA rechtzeitig und vielleicht automatisch eingeleitet werden können. Zu klären bleibt auch inwieweit die Architektur der Sensoren, der IDS und der SMONA Komponenten selbst vor Angriffen geschützt werden können.

Eine weitere Forschungsarbeit könnte die Untersuchung der Akzeptanz eines Intrusion Detection Systems für Grids zum Thema haben. Sicherheit in einer Domäne herzustellen ist die Aufgabe der Administratoren. Dabei geben diese nur ungern Informationen über die Infrastruktur, die Sicherheitskomponenten, die Gefahren und über entdeckte Angriffe weiter. Eine Kooperation, in der der Verdacht auftaucht, dass in die Systeme eines Partnerunternehmens eingebrochen wurde, kann in Gefahr geraten. Die Arbeit müsste mittels Befragungen klären, inwiefern Unternehmen bereit sind Informationen über Angriffe auf ihre IT Infrastruktur mit anderen Organisationen des Grids zu teilen.

Schließlich wurde in dieser Arbeit nur eine prototypische Implementierung des System dargestellt, die um mehrere Annehmlichkeiten erweitert werden kann. Dazu kann ein Werkzeug gehören, mit die Manipulationsregeln von Protokolleinträgen getestet werden. Die graphische Oberfläche der Management Anwendung kann um verschiedene Funktionen vergrößert werden, die zum Beispiel einen Editor beinhalten, in dem Aggregationen eingegeben und validiert werden können. Außerdem sind in der Umsetzung keine Funktionen enthalten die dem Monitoring von Diensten und ihren Attributen dienen.

In dieser Arbeit wurde für die oben beschriebenen Erweiterungsmöglichkeiten eine Basis konzipiert, auf die in verschiedenen nachfolgenden Arbeiten aufgebaut werden kann.

A Installation und Konfiguration der Komponenten

A.1 Das SMONA Framework

Die SMONA Komponenten sind in einer Datei Namens `smona.tar.bz2` gepackt. Zur Installation muss diese mit dem Kommandozeilen Befehl `tar -xjf smona.tar.bz2` entpackt werden. Anschließend können, nach einem Wechsel in das Verzeichnis `smona`, die Programme mit dem Befehl `make` kompiliert werden.

Zur Konfiguration müssen die Startskripte der Komponenten (`start-composer`, `start-management` und `start-adapterConfigurator` aus dem Verzeichnis `bin` angepasst werden. Sie dienem dem Starten der einzelnen Komponenten und enthalten die IP Adresse (Variable `HOST_ADDR`) und den Port (Variable `PORT`), unter denen der Namensdienst erreichbar ist.

Im Verzeichnis `etc` sind mehrere beispielhafte Aggregationen, die an das eigene System angepasst werden können. Die Datei `ssh.xml`, zum Beispiel enthält die Aggregation des Anwendungsbeispiels aus Abschnitt 6.2.4 mit zwei Log Adaptern, drei Funktionen und zwei Notifikationen wird die SMONA zur Überwachung auf SSH Würmer und SSH Angriffe konfiguriert. Ihr Aufbau wurde in Abschnitt 6.2.4 beschrieben. Die exemplarischen Aggreagtionen können nach belieben an die eigenen Bedürfnisse angepasst werden. Dabei ist es wichtig, dass die Identität jedes Source Attributes mit der Identität eines Adapters übereinstimmt.

Vor der Inbetriebname der SMONA Komponenten sollte ein eigenes Keystore erstellt (siehe Kapitel 5.1.1) und in der Datei `etc/jacorb.properties` mit dem Passwort eingetragen werden. Zu Testzwecken ist bereits ein Keystore vorkonfiguriert.

Anschließend kann der Namensdienst mit dem Befehl `bin/start-orb` gestartet werden. Die SMONA Komponenten werden mit ihren Startskripten aus dem Verzeichnis `bin` gestartet.

A.2 Die Adapter

Der Quellcode für die Adapter befindet sich in der komprimierten Datei `Adapter.tar.bz2`. Zur Installation muss sie mit dem Kommandozeilen Befehl `tar -xjf Adapter.tar.bz2` entpackt werden. Anschließend kann der Quelltext, nach einem Wechsel in das Verzeichnis `Adapter`, mit dem Befehl `make` kompiliert werden.

Zur Konfiguration eines Adapters muss sein Startskript (zum Beispiel: `bin/start-LogAdapter`) angepasst werden. Es enthält die IP Adresse (Variable `HOST_ADDR`) und den Port (Variable `PORT`) des Namensdienstes. Außerdem bestimmt die Variable `ENV_FILE`, welche Konfigurationsdatei für den Adapter verwendet wird.

Im Verzeichnis `etc` befinden sich mehrere exemplarische Konfigurationsdateien (zum Beispiel `Log_MM_AdapterA.xml`) für Adapter, die an das eigene System angepasst werden müssen. Ihr Aufbau ist im Listing 5.11 erläutert. Hier müssen mindestens der Adaptername und der Adaptertyp eingestellt werden.

Vor dem Start eines Adapters sollte nach der Beschreibung aus Abschnitt 5.1.1 ein eigenes Keystore erstellt und in der Datei `etc/jacorb.properties` mit dem Passwort eingetragen werden. Zu Testzwecken ist bereits ein Keystore vorkonfiguriert. Außerdem muss für den Betrieb eines Adapters ein ORB mit einem Namensdienst laufen.

Ein Adapter kann schließlich mit seinem Startskript aus dem Verzeichnis `bin` gestartet werden.

B XML Schema der Service Information Specification Language

Listing B.1: XML Schema der Service Information Specification Language (SISL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="resource" type="resource"/>
    <xsd:complexType name="resource">
        <xsd:sequence>
            <xsd:element name="source" type="xsd:string"/>
            <xsd:element name="description" type="description" minOccurs="0" \
                         maxOccurs="1"/>
            <xsd:element name="sourceAttrib" type="sourceAttrib" minOccurs="0" \
                         maxOccurs="unbound"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:element name="function" type="function"/>
    <xsd:complexType name="function">
        <xsd:sequence>
          <xsd:element name="method" type="xsd:string"/>
          <xsd:element name="return" type="xsd:string"/>
          <xsd:element name="description" type="description"/>
          <xsd:element name="parameters" type="parameters"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:element name="notification" type="notification"/>
    <xsd:complexType name="notification">
        <xsd:sequence>
            <xsd:element name="condition" type="condition"/>
            <xsd:element name="declaration" type="declaration"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:element name="sourceAttrib" type="sourceAttrib"/>
    <xsd:complexType name="sourceAttrib">
        <xsd:sequence>
            <xsd:element name="parameter" type="parameter" minOccurs="0" \
                         maxOccurs="unbounded"/>
            <xsd:element name="interval" type="xsd:float"/>
            <xsd:element name="return" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:element name="aggregation" type="aggregation"/>
    <xsd:complexType name="aggregation">
        <xsd:sequence>
            <xsd:element name="description" type="description" minOccurs="0" \
                         maxOccurs="1"/>
            <xsd:element name="resource" type="resource" minOccurs="1" \
                         maxOccurs="unbounded"/>
```

```
            <xsd:element name="function" type="function" minOccurs="0" \
50                      maxOccurs="unbounded"/>
            <xsd:element name="notification" type="notification" minOccurs="1" \
52                      maxOccurs="unbounded"/>
        </xsd:sequence>
54      <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
56  <xsd:element name="description" type="description"/>
    <xsd:complexType name="description">
58      <xsd:sequence>
            <xsd:element name="author" type="xsd:string" minOccurs="0" \
60                      maxOccurs="1"/>
            <xsd:element name="date" type="xsd:string" minOccurs="0" \
62                      maxOccurs="1"/>
            <xsd:element name="text" type="xsd:string"/>
64      </xsd:sequence>
    </xsd:complexType>
66  <xsd:element name="parameters" type="parameters"/>
    <xsd:complexType name="parameters">
68      <xsd:sequence>
            <xsd:element name="valueset" type="valueset" minOccurs="1" \
70                      maxOccurs="unbounded"/>
        </xsd:sequence>
72  </xsd:complexType>
    <xsd:element name="valueset" type="valueset"/>
74  <xsd:complexType name="valueset">
        <xsd:choice>
76          <xsd:element name="resourceRef" type="resourceRef"/>
            <xsd:element name="functionRef" type="functionRef"/>
78          <xsd:element name="parameter" type="parameter"/>
        </xsd:choice>
80  </xsd:complexType>
    <xsd:element name="resourceRef" type="resourceRef"/>
82  <xsd:complexType name="resourceRef">
        <xsd:attribute name="id" type="xsd:string" use="required"/>
84  </xsd:complexType>
    <xsd:element name="parameter" type="parameter"/>
86  <xsd:complexType name="parameter">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
88      <xsd:attribute name="value" type="xsd:string" use="required"/>
    </xsd:complexType>
90  <xsd:element name="functionRef" type="functionRef"/>
    <xsd:complexType name="functionRef">
92      <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
94  <xsd:element name="condition" type="condition"/>
    <xsd:complexType name="condition">
96      <xsd:sequence>
            <xsd:element name="description" type="description"/>
98          <xsd:element name="boolExpression" type="boolExpression"/>
        </xsd:sequence>
100     <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>
102 <xsd:element name="declaration" type="declaration"/>
    <xsd:complexType name="declaration">
104     <xsd:sequence>
            <xsd:element name="valueset" type="valueset" minOccurs="1" \
106                     maxOccurs="unbounded"/>
        </xsd:sequence>
108 </xsd:complexType>
    <xsd:element name="boolExpression" type="boolExpression"/>
```

```
110     <xsd:complexType name="boolExpression">
            <xsd:sequence>
112             <xsd:element name="expression" type="xsd:string" minOccurs="1" \
                        maxOccurs="1"/>
114         </xsd:sequence>
        </xsd:complexType>
116     <xsd:complexType name="timediff">
            <xsd:attribute name="timedifference" type="xsd:string" \
118                     use="required"/>
            <xsd:attribute name="ref" type="xsd:string" use="required"/>
120         <xsd:attribute name="regex" type="xsd:string" use="required"/>
        </xsd:complexType>
122     <xsd:complexType name="match">
            <xsd:attribute name="regex" type="xsd:string" use="required"/>
124     </xsd:complexType>
</xsd:schema>
```

C Die Schnittstellen der SMONA Komponenten

C.1 Die Schnittstelle der Management Anwendung

Listing C.1: Die Schnittstelle der Management Anwendung

```
/**
 *      Package for the SMONA Managemenent Component
 */

#include "sisl.idl"

module de {
module lmu {
module ifi {
module nm {
module smona {
module Management {

        interface ManagementComponent_interface{

                /** Method to print out Rich Events
                 */
                void printRichEvent(in RichEvent richEvent);

                /** Method to print out statusinformation and exceptions
                 */
                void printStatus(in string message);

        };

    };
};};};};};};
```

C.2 Die Schnittstelle des Rich Event Composers

Listing C.2: Die Schnittstelle des Rich Event Composers

```
/**
 *      Package for the SMONA Rich Event Composer
 */

#include "sisl.idl"
#include "SmonaAdapter.idl"

module de {
module lmu {
module ifi {
```

```
module nm {
module smona {
module Composer {

        interface RichEventComposer_interface{

                /** Method to configure an Aggregation
                 */
                boolean configureAggregation(in Aggregation aggregation);

                /** Method to remove an Aggregation
                 */
                void removeAggregation(in Aggregation aggregation);

        };

    };
};};};};};};
```

C.3 Die Schnittstelle des Adapter Configurators

Listing C.3: Die Schnittstelle des Adapter Configurators

```
/**
 *      Package for the SMONA Adapter Configurator
 */

#include "sisl.idl"
#include "SmonaAdapter.idl"

module de {
module lmu {
module ifi {
module nm {
module smona {
module AdapterConfigurator {

        interface AdapterConfigurator_interface{

                /** Method to start and configure new Adapters from an Aggregation
                 */
                boolean configureAndStartAdapters(in Aggregation aggregation);

                /** Method to stop Adapters from an Aggregation
                 */
                void stopAdapters(in Aggregation aggregation);

        };

    };
};};};};};};
```

D Exemplarische Aggregation zur Erkennung von SSH Würmern und SSH Angriffen

Listing D.1: Exemplarische Aggregation zur Erkennung von SSH Würmern und SSH Angriffen

```
<?xml version="1.0" encoding="UTF-8"?>
<aggregation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="etc/sislSchema.xsd"
    id="SSH_Aggregation">
    <description>
        <author>Thomas Binder</author>
        <date>04.12.07</date>
        <text>Aggregation zur Erkennung von SSH Würmern und SSH Angriffen</text>
    </description>
    <!--   Resource Part   -->
    <resource id="lxvm3-pub">
        <source>push_adapter</source>
        <sourceAttrib id="log_adapter_lxvm3-pub">
            <parameter name="log␣regex" value="Failed␣password"/>
            <parameter name="differentiator"
                       value="[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}"/>
            <parameter name="time␣between␣entries" value="20"/>
            <parameter name="logfile"  value="testData/messages"/>
            <parameter name="Single␣Event␣Text" value="Failed␣SSH␣Login␣from"/>
            <parameter name="Multiple␣Event␣Text"
                       value="SSH␣Worm␣detected␣from"/>
            <interval>2000</interval>
            <return>string</return>
        </sourceAttrib>
    </resource>
    <resource id="lxvm4-pub">
        <source>push_adapter</source>
        <sourceAttrib id="log_adapter_lxvm4-pub">
            <parameter name="log␣regex" value="Failed␣password"/>
            <parameter name="differentiator"
                       value="[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}"/>
            <parameter name="time␣between␣entries" value="20"/>
            <parameter name="logfile"  value="testData/messagesB"/>
            <parameter name="Single␣Event␣Text" value="Failed␣SSH␣Login␣from"/>
            <parameter name="Multiple␣Event␣Text"
                       value="SSH␣Worm␣detected␣from"/>
            <interval>2000</interval>
            <return>string</return>
        </sourceAttrib>
    </resource>
<!--   Function Part   -->
    <function id="get_failed_SSH_entries">
        <method>contain</method>
        <return>string</return>
        <description>
            <text>This function is used to filter out log entries which
```

```
47                  does not report a failed ssh login</text>
        </description>
49      <parameters>
            <valueset>
51              <resourceRef id="log_adapter_lxvm3-pub"/>
            </valueset>
53          <valueset>
                <resourceRef id="log_adapter_lxvm4-pub"/>
55          </valueset>
            <valueset>
57              <parameter name="regex" value="Failed␣SSH␣Login"/>
            </valueset>
59      </parameters>
    </function>
61  <function id="analyse_failed_SSH_logins">
        <method>timediff</method>
63      <return>string</return>
        <description>
65          <text>This function is used to analyse failed
                SSH logins for attacks</text>
67      </description>
        <parameters>
69          <valueset>
                <functionRef id="get_failed_SSH_entries"/>
71          </valueset>
            <valueset>
73              <parameter name="timedifference" value="15"/>
            </valueset>
75          <valueset>
                <parameter name="differentiator"
77                          value="[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}\.[\d]{1,3}"/>
            </valueset>
79      </parameters>
    </function>
81  <function id="get_SSH_worms">
        <method>contain</method>
83      <return>string</return>
        <description>
85          <text>This function is used to filter out log entries which does
                not report a ssh worm</text>
87      </description>
        <parameters>
89          <valueset>
                <resourceRef id="log_adapter_lxvm3-pub"/>
91          </valueset>
            <valueset>
93              <resourceRef id="log_adapter_lxvm4-pub"/>
            </valueset>
95          <valueset>
                <parameter name="regex" value="SSH␣Worm␣detected␣from"/>
97          </valueset>
        </parameters>
99  </function>
    <!--   Notification Part   -->
101 <notification id="analyse_failed_SSH_logins">
        <condition id="foundSSHAttack">
103         <description>
                <text>SSH Attack</text>
105         </description>
            <boolExpression>
107             <expression>
```

```
                    hasResult(analyse_failed_SSH_logins)
109                 </expression>
                </boolExpression>
111         </condition>
            <declaration>
113             <valueset>
                    <functionRef id="get_failed_SSH_entries"/>
115             </valueset>
            </declaration>
117     </notification>
        <notification id="report_SSH_Worms">
119         <condition id="foundSSHWorm">
                <description>
121                 <text>SSH Worm detected</text>
                </description>
123             <boolExpression>
                    <expression>
125                     hasResult(get_SSH_worms)
                    </expression>
127             </boolExpression>
            </condition>
129         <declaration>
                <valueset>
131                 <functionRef id="get_SSH_worms"/>
                </valueset>
133         </declaration>
        </notification>
135 </aggregation>
```

Abkürzungsverzeichnis

ADS Anomaly Detection System

API Aplication Programming Interface

BEEP Blocks Extensible Exchange Protocol

BSI Bundesamt für Sicherheit in der Informationstechnik

CA Certificate Authorization

CAIDS Cooperative Anomaly and Intrusion Detection System

CORBA Common Object Request Broker Architecture

DBMS Datenbank Management System

DDoS Distributed Dinial of Service

DHT Distributed Hash Table

DoS Dinial of Service

EGEE Enabling Grids for E-Sience

GGF Global Grid Forum

GIDS Grid-based Intrusion Detection System

GMA Grid Monitoring Architecture

GRAM Grid Resource Allocation Management

GSI Grid Security Infrastructure

GTCP Teleoperations Control Protocol

GTK Globus Toolkit

HIDS Host Intrusion Detection System

IBM International Business Machines

ID Identität

IDL Interface Description Language

IDMEF Intrusion Detection Message Exchange Format

IDS Intrusion Detection System

IDWG Intrusion Detection Working Group

IDXP The Intrusion Detection Exchange Protocol

IETF Internet Engineering Task Force

IIOP Inter Orb Protocoll

IPS Intrusion Prevention System

IPsec Internet Protocol Security

ISO International Organization for Standardization

IT Information Technology

MIB Managementinformationsbasis

NIDS Network Intrusion Detection System

OASIS Organization for the Advancement of Structured Information Standards

OGSA Open Grid Service Architecture

OGSA - DAI Open Grid Services Architecture - Data Access and Integration

OGSA-EMS OGSA Execution Management Services

ORB Object Request Broker

OSI Open Systems Interconnection

PKI Public Key Infrastructure

R-GMA Relational Grid Monitoring Architecture

RFC Request for Comment

RFT Reliable File Transfer

RPC Remote Procedure Call

SISL Service Information Specification Language

SMONA Service Monitoring Architecture

SOAP Simple Object Access Protocols

SQL Structured Query Language

SSH Secure Socket

SSL Secure Socket Layer

TCP Transport Control Protocoll

TLS Transport Layer Security

UML Unified Modelling Language

VO Virtuelle Organisation

WMS Workspace Management Service

WS Web Service

WSDL Web Services Description Language

WSDM OASIS Web Services Distributed Management

WSRF Web Service Resource Framework

XML Extensible Markup Language

XSD XML Schema Definition

Literaturverzeichnis

[AIDE 07] *Webseite des AIDE IDS*, http://sourceforge.net/projects/aide.

[APACHE 07] *Webseite der Apache Software Foundation*, http://apache.org.

[ARGU 07] *Webseite des Argus Projektes*, http://qosient.com/argus/security.htm.

[BIDS 07] *Webseite des Benids Projektes*, http://www.duke.edu/~ttoomey/benids/.

[BPSM+ 06] BRAY, TIM, JEAN PAOLI, C. M. SPERBERG-MCQUEEN, EVE MALER, FRANCOIS YERGEAU und JOHN COWAN: *Extensible Markup Language (XML) 1.1 (Second Edition)*, 2006, http://www.w3.org/TR/2006/REC-xml11-20060816/.

[BSI 02] BSI: *Einführung von Intrusion-Detection-Systemen*, 2002, http://www.bsi.de/literat/studien/ids02/dokumente/Grundlagenv10.pdf.

[BSI 07] *Webseite des Bundesamt für Sicherheit (BSI)*, http://www.bsi.de.

[CCMW 01] CHRISTENSEN, ERIK, FRANCISCO CURBERA, GREG MEREDITH und SANJIVA WEERAWARANA: *Web Services Description Language (WSDL) 1.1*, 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[CGM+ 03] COOKE, ANDREW W., ALASDAIR J. G. GRAY, LISHA MA, WERNER NUTT, JAMES MAGOWAN, MANFRED OEVERS, PAUL TAYLOR, ROB BYROM, LAURENCE FIELD, STEVE HICKS, JASON LEAKE, MANISH SONI, ANTONY J. WILSON, RONEY CORDENONSI, LINDA CORNWALL, ABDESLEM DJAOUI, STEVE FISHER, NORBERT PODHORSZKI, BRIAN A. COGHLAN, STUART KENNY und DAVID O'CALLAGHAN: *R-GMA: An Information Integration System for Grid Monitoring*. In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, Seiten 462–481. Springer, ISBN 3-540-20498-9, 2003.

[ChCl 06] CHUBA, MIKE und CARL CLAUNCH: *What Grid Computing Is Really About*, 2006, http://www.gartner.com/DisplayDocument?doc_cd=138888.

[ChSa 03] CHOON, ONG TIAN und AZMAN SAMSUDIN: *Grid-based Intrusion Detection System*, 2003, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber1274254.

[CMRW 96] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)*, 1996, http://www.ietf.org/rfc/rfc1907.txt.

[DaSa 05] DANCIU, VITALIAN A. und MARTIN SAILER: *A monitoring architecture supporting service management data composition*, 2005, http://www.mnm-team.org/pub/Publikationen/dasa05/PDF-Version/dasa05.pdf.

[DCF 07] DEBAR, HERVE, DAVID A. CURRY und BENJAMIN S. FEINSTEIN: *The Intrusion Detection Message Exchange Format (IDMEF)*, 2007, http://www.ietf.org/rfc/rfc4765.txt.

[DgFS 07] DANCIU, VITALIAN, NILS GENTSCHEN FELDE und MARTIN SAILER: *Declarative Specification of Service Management Attributes*. In: *Moving From Bits to Business Value: Proceedings of the 10th International IFIP/IEEE Symposium on Integrated Management (IM 2007), Munich, Germany*, 2007, http://wwwmnmteam.informatik.uni-muenchen.de/pub/Publikationen/dgs07/PDF-Version/dgs07.pdf.

[DGRID 07] *Webseite des europäischen DataGrid Projektes*, http://eu-datagrid.web.cern.ch/eu-datagrid/ .

[DiRe 06] DIERKS, T. und E. RESCORLA: *The Transport Layer Security (TLS) Protocol Version 1.1*, 2006, http://tools.ietf.org/html/rfc4346 .

[DSHH 06] DANCIU, VITALIAN A., MARTIN SAILER, ANDREAS HANEMANN und HEINZ-GERD HEGERING: *IT Service Management: Getting the View*. Springer, ISBN 978-3-540-34128-4, 2006, http://www.springerlink.com/index/l787m7077444117x.pdf .

[Dürr 06] DÜRR, MICHAEL: *Entwicklung von Adaptern für Datenquellen auf Linux-Systemen*, 2006, http://www.mnm-team.org/pub/Fopras/duer06/PDF-Version/duer06.pdf .

[Ecke 00] ECKERT, CLAUDIA: *IT- Sicherheit. Konzepte, Verfahren, Protokolle*. Oldenbourg, ISBN 3-4862-5298-4, 2000.

[EGEE 07] *Webseite des EGEE Projektes*, http://public.eu-egee.org .

[EXPE 07] *Webseite der Experton Group*, http://www.experton-group.de/press/releases/pressrelease/article/neue-studie-der-experton-group-zur-it-sicher
html .

[FeMa 07] FEINSTEIN, B. und G. MATTHEWS: *The Intrusion Detection Exchange Protocol (IDXP)*, 2007, http://www.ietf.org/rfc/rfc4767.txt .

[FKS+ 06] FOSTER, IAN, H. KISHIMOTO, A. SAVVA, D. BERRY, A. DJAOUI, A. GRIMSHAW, B. HORN, F. MACIEL, F. SIEBENLIST, R. SUBRAMANIAM, J. TREADWELL und J. VON REICH: *The Open Grid Services Architecture, Version 1.5*, 2006, http://www.gridforum.org/Public_Comment_Docs/Documents/Apr-2006/draft-ggf-ogsa-spec-1.5-008.pdf .

[FKTT 98] FOSTER, IAN T., CARL KESSELMAN, GENE TSUDIK und STEVEN TUECKE: *A Security Architecture for Computational Grids*. In: *ACM Conference on Computer and Communications Security*, Seiten 83–92. ACM Press, 1998, ftp://ftp.globus.org/pub/globus/papers/security.pdf .

[Fost 02] FOSTER, IAN: *What is the Grid? A Three Point Checklist*. GridToday, 1(6), June 2002, http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf .

[FTK 01] FOSTER, IAN, STEVEN TUECKE und CARL KESSELMAN: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. In: *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, Seite 500, Washington, DC, USA, 2001. IEEE Computer Society, http://www.globus.org/alliance/publications/papers/anatomy.pdf . 6 S.

[GFTP 07] *GT 4.0 GridFTP*, http://www.globus.org/toolkit/docs/4.0/data/gridftp/index.pdf .

[GGF 07] *Webseite des Global Grid Forums*, http://www.ggf.org .

[GHM+ 06] GUDGIN, MARTIN, MARC HADLEY, NOAH MENDELSOHN, JEAN-JACQUES MOREAU, HENRIK FRYSTYK NIELSEN, ANISH KARMARKAR und YVES LAFON: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, 2006, http://www.w3.org/TR/2006/PER-soap12-part1-20061219/ .

[GICE 07] *GridICE Webseite*, http://gridice.forge.cnaf.infn.it .

[GLOBUS 07] *Webseite der Globus Alliance*, http://www.globus.org .

[GRAM 07] *GT 4.0 WS GRAM*, http://www.globus.org/toolkit/docs/4.0/data/gridftp/index.pdf .

[Grou 04] GROUP, THE OBJECT MANAGEMENT: *Common Object Request Broker Architecture: Core Specification*, 2004, http://www.omg.org/docs/formal/04-03-12.pdf .

[GT4COMP 07] *Webseite: Globus Toolkit 4.0 Documentation Overview*, http://www.globus.org/toolkit/docs/4.0/doc_overview.html.

[GT4PORTS 07] *Globus Toolkit Wiki mit einer Auflistung der von GT4 verwendeten Ports*, http://dev.globus.org/wiki/FirewallHowTo.

[GTAD 07] *Webseite mit einer ausfühlrlichen Installationsanleitung zum Globus Toolkit 4*, http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch04.html.

[GTK 07] *Webseite GT 4.0 Dokumentation*, http://globus.org/toolkit/docs/4.0/.

[GTK07] *Webseite der Globus Alliance*, http://www.globus.org/toolkit/.

[GTQS 07] *Webseite mit einer kurzen Installationsanleitung zum Globus Toolkit 4*, http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html.

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[HKS+ 06] HWANG, KAI, YU-KWONG KWOK, SHANSHAN SONG, MIN CAI, YU CHEN und YING CHEN: *DHT-based Security Infrastructure for Trusted Internet and Grid Computing*, 2006, http://gridsec.usc.edu/files/publications/IJCIS-Hwang-2006.pdf.

[IAIK 07] *Webseite des IAIK Projektes*, http://jce.iaik.tugraz.at.

[IDWG 07] *Webseite der Intrusion Detection Working Group (IDWG) der IETF*, http://www.ietf.org/ids.by.wg/idwg.html.

[IETF 07] *Webseite der The Internet Engineering Task Force (IETF)*, http://www.ietf.org.

[JACORB 07] *Webseite des JacORB Projektes*, http://www.jacorb.org.

[JBFT 05] JACOB, BART, MICHAEL BROWN, KENTARO FUKUI und NIHAR TRIVEDI: *Introduction to Grid Computing*. IBM Redbooks, ISBN 0-7384-9400-3, 2005.

[JKES 07] *Dokumentations Webseite des Java Keystore*, http://java.sun.com/j2se/1.5.0/docs/api/java/security/KeyStore.html.

[JKET 07] *Dokumentations Webseite des Java Keytool*, http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html.

[JPATTERN 07] *Webseite mit der Definition von Java Pattern*, http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html.

[JSSE 07] *Dokumentations Webseite der Java Secure Socket Extension*, http://java.sun.com/products/jsse/.

[Lang 06] LANGE, S.: *Formalisierung von Aggregationsvorschriften für Dienstinformationen*, 2006, http://www.nm.ifi.lmu.de/pub/Diplomarbeiten/lang06/PDF-Version/lang06.pdf.

[LDR 03] LEE, D., J. DONGARRA und R. RAMAKRISHNA: *Visperf: Monitoring tool for grid computing*. 2003, http://icl.cs.utk.edu/projectsfiles/netsolve/pubs/visperf.pdf.

[LIBS 07] *Webseite von libsafe*, http://directory.fsf.org/libsafe.html.

[LIDS 07] *Webseite des LIDS IDS*, http://www.lids.org.

[LOG4J 07] *Projektwebseite des Log4j Projektes*, http://logging.apache.org/log4j/docs/index.html.

[LOGC 07] *Webseite des Logcheck IDS*, http://logcheck.org.

[LOGS 07] *Webseite des Logsurf IDS beim DFN Cert*, http://www.dfn-cert.de/eng/logsurf/index.html.

[LOGW 07] *Webseite des Logwatch Projektes*, http://freshmeat.net/projects/logwatch/.

[LONV 01] C., LONVICK: *The BSD syslog Protocol*, 2001, http://tools.ietf.org/html/rfc3164.

[MYSQ 07] *Webseite der Mysql AB*, http://www.mysql.de.

[NEPE 07] *Webseite des Nepenthes Honeypots*, http://nepenthes.mwcollect.org.

[NUFW 07] *Webseite der NuFW firewall*, http://www.nufw.org/-English-.html.

[ODAI 07] *Webseite des OGSA-DAI Projektes*, http://www.ogsadai.org.uk.

[OSI 94] *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, 1994, http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[PLML 07] *Webseite des Prelude-LML*, https://trac.prelude-ids.org/wiki/PreludeLML.

[POST 07] *Webseite des Postgres Projektes*, http://www.postgresql.org.

[PREARCH 07] *Webseite: Prelude Architecture overview*, https://trac.prelude-ids.org/wiki/PreludesArchi-tecture.

[PREL 07] *Webseite des Prelude Projektes*, http://prelude-ids.org.

[RFT 07] *GT 4.0 Reliable File Transfer (RFT) Service*, http://www.globus.org/toolkit/docs/4.0/data/rft/index.pdf.

[RGMA 07] *Webseite des R-GMA Projektes*, http://www.r-gma.org.

[RGMA5 07] *Webseite: R-GMA in 5 Minuten*, http://www.r-gma.org/fivemins.html.

[Rose 01] ROSE, MARSHALL T.: *The Blocks Extensible Exchange Protocol Core*, 2001, http://www.rfc-editor.org/rfc/rfc3080.txt.

[SAMH 07] *Webseite des Samhain IDS*, http://la-samhna.de/samhain.

[SANC 07] *Webseite des Sancp Projektes*, http://sourceforge.net/projects/sancp.

[Sedu 05] SEDUKHIN, I. (ED.): *Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.0*, 2005, http://docs.oasis\discretionary{-}{}{}open.org/wsdm/2004/12/wsdm-mows-1.0.pdf.

[SimpleCA 07] *Webseite der Globus Alliance*, http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch07.html#s-simpleca-admin-installing.

[SNAR 07] *Webseite des SNARE IDS*, http://www.intersectalliance.com/projects/index.html.

[SNOR 07] *Webseite des Snort IDS*, http://www.snort.org.

[Spen 05] SPENNEBERG, RALF: *Intrusion Detection und Prevention mit Snort2 und Co.* Addison-Wesley, ISBN 3-8273-2134-4, 2005.

[SYST 07] *Webseite des Systrace IDS*, http://www.systrace.org.

[TAG+ 02] TIERNEY, B., R. AYDT, D. GUNTER, W. SMITH, M. SWANY, V. TAYLOR und R. WOLSKI: *A Grid Monitoring Architekture*, 2002, http://www.ggf.org/documents/GFD.7.pdf.

[TRIP 07] *Webseite des Tripwire Projektes*, http://sourceforge.net/projects/tripwire/.

[Vamb 05a] VAMBENEPE, W. (ED.): *Web Services Distributed Management: Management using Web Services (MUWS 1.0) Part 1*, 2005, http://docs.oasi\discretionary{-}{}{}open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf.

[Vamb 05b] VAMBENEPE, W. (ED.): *Web Services Distributed Management: Management using Web Services (MUWS 1.0) Part 2*, 2005, http://docs.oasi-open.org/wsdm/2004/12/wsdm-muws-part2-1.0.pdf.

[VISP 07] *Webseite des visPerf Projektes*, http://icl.cs.utk.edu/netsolvedev/applications/visperf.html.

[WS 03] *Webseite der W3C zur Web Service Architektur*, http://www.w3.org/TR/2003/WD-ws-arch-20030808/.

[XEN 07] *Webseite des Xen Projektes*, http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[XINE 07] *Webseite des xinetd Projektes*, http://www.xinetd.org.

[YlLo 06] YLONEN, T. und C. E. LONVICK: *The Secure Shell (SSH) Protocol Architecture*, 2006, http://tools.ietf.org/html/rfc4251.