Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Fakultät für **Informatik**

Institut für Telematik
Cooperation & Management

Policy-basiertes Management einer IT-Infrastruktur am Beispiel des VPN-Dienstes

Diplomarbeit
von

Thomas Poisl

Verantwortlicher Betreuer:	Prof. Dr. Sebastian Abeck
Betreuender Mitarbeiter:	Dipl.-Math. Klaus Scheibenberger

Bearbeitungszeit: 15. März 2006 – 14. September 2006

Universität Karlsruhe (TH) · Institut für Telematik · Geb. 20.20 · Zirkel 2 · 76131 Karlsruhe

Ehrenwörtliche Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Karlsruhe, den 14. September 2006

Thomas Poisl

Dank

Wenn du ein Schiff bauen willst, dann trommle nicht Männer zusammen, um Holz zu beschaffen, Aufgaben zu vergeben und die Arbeit einzuteilen, sondern lehre sie die Sehnsucht nach dem weiten, endlosen Meer.

Zitat von Antoine de Saint-Exupéry
(Schriftsteller)

Diese Diplomarbeit wurde in der Forschungsgruppe Cooperation & Management (C&M) am Institut für Telematik der Fakultät für Informatik an der Universität Karlsruhe verfasst.

Für die Ermöglichung dieser Arbeit bedanke ich mich bei Professor Sebastian Abeck, der mit seinen wertvollen Anregungen zum Gelingen der Arbeit beigetragen hat.

Besonderer Dank gilt auch Klaus Scheibenberger, der mir während der Erstellung der Arbeit stets mit gutem Rat zur Seite stand.

Weiterhin danke ich Dr. Christian Mayerl, Stefan Popescu und Kai Hühner für die Arbeitsgemeinschaft „Integriertes Management“, dort wurden stets gemeinsam wichtige Fragestellungen zum Thema Management vorgestellt und diskutiert. Dieser gegenseitige und offene Wissensaustausch war von der Leidenschaft des obigen Zitats geprägt und hat die Qualität der Arbeit maßgeblich gesteigert. Für seine wertvollen Hinweise bedanke ich mich an dieser Stelle auch besonders bei Dr. Oliver Mehl.

Mein herzlicher Dank gilt auch Olaf Hopp, der mich bei der Visualisierung von Ressourcenparametern unterstützt hat und Jürgen Bensching und Akram Radwan, deren Rezensionen der Arbeit den letzten Schliff verliehen haben.

Nicht zuletzt bedanke ich mich besonders bei meiner Frau, Helen Schumacher-Poisl, die mich stets entlastet und bei meiner Arbeit unterstützt hat.

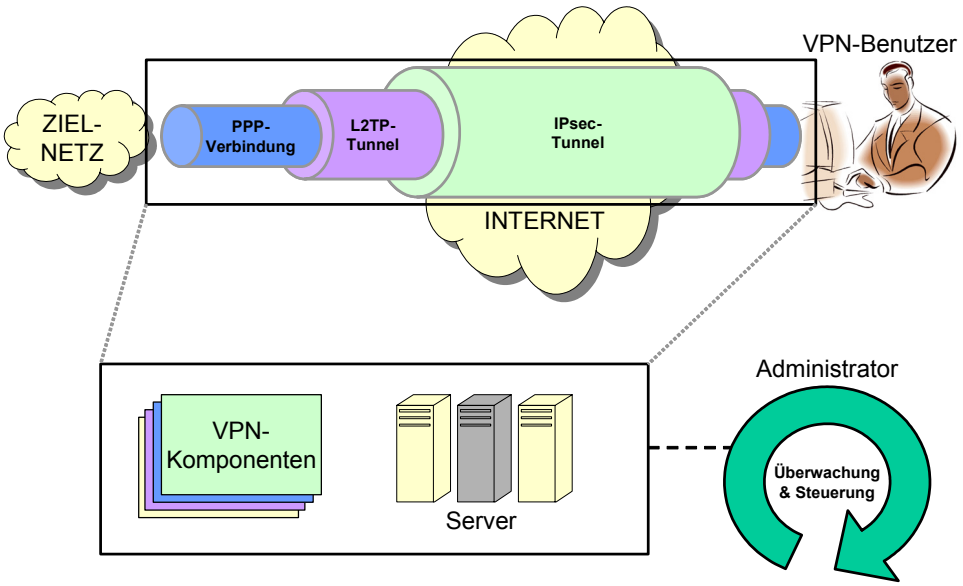
Inhaltsverzeichnis

1 EINLEITUNG

1.1 Einführung in das Themengebiet

In der heutigen Zeit gewinnt IT-Unterstützung immer mehr an Bedeutung, da viele Geschäftsprozesse dadurch wesentlich effizienter und kostengünstiger gestaltet werden können. Bezieht ein Kunde diese IT-Unterstützung von einem Betreiber (Dienstleister), wird von einem IT-Dienst gesprochen. Dadurch entsteht zwischen Kunde und Betreiber ein dienstorientiertes Verhältnis, das durch ein *Service Level Agreement* (SLA) vertraglich fixiert wird. Das SLA legt fest, welche funktionalen und qualitativen Merkmale der Dienst hat. Diese Vereinbarung gibt dem Kunden die Sicherheit, dass der Dienst wie im SLA festgelegt bereitgestellt werden muss und er sich auf die korrekte Diensterbringung verlassen kann. Der Betreiber verpflichtet sich durch das SLA, den Dienst innerhalb der vereinbarten Funktions- und Qualitätsgrenzen zu erbringen. Für den Betreiber bedeutet ein SLA einerseits eine Eingrenzung der zu erbringenden Leistung, andererseits eine Verpflichtung zur Erbringung der Dienstleistung innerhalb der geforderten Parameter. Um seine Verpflichtungen diesbezüglich einhalten und nachweisen zu können, muss der Dienst bzw. müssen die Ressourcen (System, Netzwerk) die den Dienst funktional und qualitativ erbringen, überwacht und gesteuert werden.

Unter dem Begriff Qualität ist in der vorliegenden Arbeit immer die Qualität technischer Merkmale zu verstehen. Qualität beispielsweise im Sinne der Reaktionszeit des Second Level Supports für einen Dienst wird hier nicht betrachtet.



Information 1: Vereinfachte schematische Darstellung des VPN-Dienstes

Die Abteilung Technische Infrastruktur (ATIS) stellt ihren Kunden unter anderem einen *Virtual Private Network* (VPN) Netzzugangsdienst zur Verfügung. Der VPN-Dienst ermöglicht es, das Datennetz der Fakultät für Informatik der Universität Karlsruhe von Zuhause aus zu nutzen, als würde ein Computerarbeitsplatz innerhalb der Fakultät verwendet. Dabei wird das Computersystem des entfernten Nutzers in das Fakultätsnetz eingebunden, als wäre es tatsächlich physikalisch an dieses Netz angeschlossen (s.

Information 1). Um diese virtuelle Verbindung herzustellten, wird ein Tunnel zwischen Nutzersystem und *Remote Access Server* (RAS) aufgebaut. Da der Tunnel über das öffentlich zugängliche Internet realisiert wird, ist es nötig, die übertragenen Daten zu verschlüsseln, um zu verhindern, dass sensible Daten durch mitlauschende Angreifer kompromittiert werden können. Außerdem ist sicherzustellen, dass nur berechtigte Nutzer einen Tunnel in das Fakultätsnetz aufbauen können, um Missbrauch Dritter auszuschließen. Verschlüsselung und Authentifikation für das Internetprotokoll (IP) wurde im Jahr 1998 von der *Internet Engineering Task Force* (IETF) mit IPsec standardisiert, welches mittlerweile in einer im Dezember 2005 überarbeiteten Fassung [KS05] vorliegt.

Für den VPN-Dienst der ATIS sollen zukünftig mit den Kunden SLAs abgeschlossen werden, in denen qualitative Merkmale wie beispielsweise Antwortzeit, Durchsatz und Verfügbarkeit vereinbart werden. Um qualitativen Anforderungen an den VPN-Dienst einhalten zu können, sollen Policies – die sich bereits im Bereich des Infrastrukturmanagements etabliert haben – als Werkzeuge zur Überwachung und Steuerung von Ressourcen eingesetzt werden. Im weiteren Verlauf wird exemplarisch der Parameter „Verfügbarkeit“ herangezogen, da Verfügbarkeit die Grundvoraussetzung für die Überwachung anderer Qualitätsparameter ist. Eine Policy bezeichnet eine Regel, die festgelegte Bedingungen mit gewissen Aktionen verknüpft. Diese Regel stellt dabei die Formalisierung des Verhaltens eines Systemadministrators dar, der im Falle eines Fehlers gewisse Aktionen zur Behebung ausführt. Um Policies zum Management von Ressourcen zu verwenden, werden eine Policy-Spezifikation und eine Policy-Architektur benötigt. Die Policy-Spezifikation legt fest, wie Policies formuliert werden müssen, wohingegen die Policy-Architektur die nötigen Komponenten und Schnittstellen definiert, um Policies zu verarbeiten. Für die Spezifikation und Architektur existieren verschiedene Ansätze und Modelle.

Die Implementierung einer Policy-Architektur durch ein policy-basiertes Managementsystem hat das Ziel, Managementaufgaben des Betreibers wesentlich zu automatisieren. Dazu trifft ein policy-basiertes Managementsystem anhand der spezifizierten Regeln selbstständig Entscheidungen und führt diese aus.

1.2 In der Arbeit behandelte Fragestellungen

Die IETF hat bereits in verschiedenen RFCs eine Policy-Architektur und -Spezifikation standardisiert. Bislang wurde deren Praxistauglichkeit meist nur an reinen Netzwerkszenarien – wie *Quality of Service* (QoS) in *Differentiated Services* (DiffServ) bzw. *Integrated Services* (IntServ) Netzwerken nachgewiesen. In diesen Szenarien geht es darum, Datenpakete nutzerbezogen zu priorisieren und zu parametrisieren. Dazu wird überprüft, ob Datenfelder der Pakete auf vorhandenen Policies zutreffen, um dann die Aktionen auf den Ressourcen auszuführen. Verschiedene Arbeiten beschäftigen sich mit dem IETF-Ansatz, um beispielsweise Bandbreiten oder bestimmte Routen für Nutzer zu reservieren oder die Einhaltung bestimmter Sicherheitsmechanismen von Nutzern zu gewährleisten.

In dieser Arbeit wird der Fokus jedoch auf die generelle Verfügbarkeit des VPN-Dienstes gelegt. Um eine im SLA vereinbarte Verfügbarkeit gewährleisten zu können, muss der Betreiber des Dienstes verschiedene Ressourcen überwachen und gegebenenfalls steuernd eingreifen. Im Gegensatz zum Kunden muss der Betreiber dazu genau wissen, welche Systeme und Komponenten für die Diensterbringung verantwortlich

sind. Ein einfacher Vergleich von Datenfeldern ist für dieses Szenario nicht möglich. Vielmehr müssen Hard- und Softwarekomponenten aktiv überprüft werden, um eventuelle Fehler dieser Ressourcen festzustellen. Der Auslöser für eine Policy-Entscheidung ist hier also nicht auf ein eingehendes Datenpaket eines Nutzers zurückzuführen, sondern wird aufgrund eines Fehlers einer Ressource ausgelöst.

Wie in Information 1 dargestellt, umfasst der VPN-Dienst nicht nur die reine Netzwerkebene, sondern stellt einen erweiterten Netzwerkdienst dar, der sich über mehrere verschiedenen Komponenten und Rechnersysteme erstreckt.

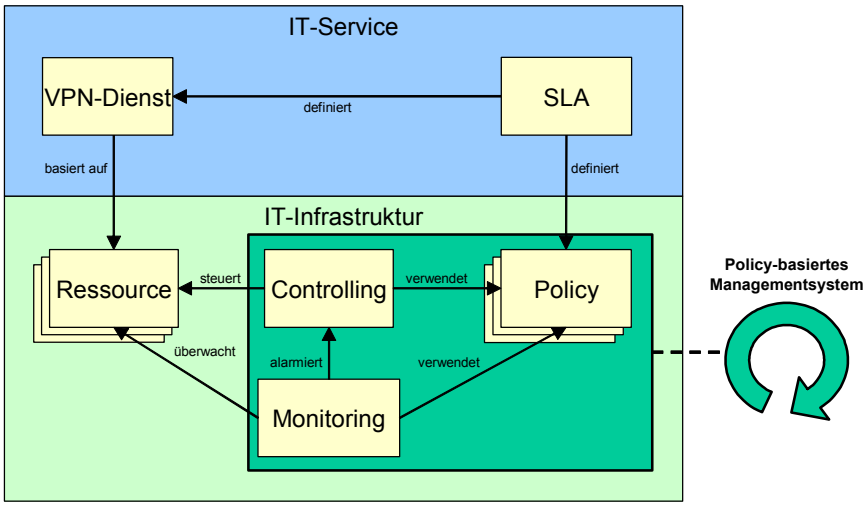
Um den einwandfreien Betrieb des VPN-Dienstes sicherzustellen, ist es notwendig einzelne Ressourcen zu überwachen und zu steuern. Bisher wird diese Tätigkeit manuell vom zuständigen Administrator ausgeführt, doch diese Aufgabe soll künftig durch Policies automatisiert werden. Optimalerweise ist keine Intervention des Administrators mehr erforderlich, um den fehlerfreien Betrieb des VPN-Dienstes zu gewährleisten. Fehler werden durch die technische Infrastruktur selbst erkannt und anhand definierter Policies werden automatisch Lösungsstrategien ausgewählt und ausgeführt. Dieses vollständig automatisierte Management wird von IBM mit dem Begriff *Autonomic Computing* bezeichnet und Policies spielen eine zentrale Rolle bei dieser Thematik.

Im Rahmen der vorliegenden Arbeit soll daher untersucht werden, wie mittels existierender Policy-Standards die Verfügbarkeit des VPN-Dienstes und der damit zusammenhängenden Ressourcen (VPN-Komponenten und Server) sichergestellt werden kann. Für die konkrete Umsetzung der Policies sollen hier die von der IETF spezifizierten Standards bzgl. der Architektur und der Spezifikation von Policies zum Einsatz kommen. Durch die hohe Generizität der IETF-Standards scheint ein solcher Einsatz für höherwertige Dienste auch denkbar. Inwieweit jedoch Anpassungen vorgenommen werden müssen oder generelle Defizite existieren, soll in dieser Arbeit beispielhaft anhand des VPN-Dienstes untersucht werden.

Aus dem beispielhaft verwendeten VPN-Dienst sollen generelle Vorgehensweisen auch für das policy-basierte Management anderer IT-Dienste abgeleitet werden.

1.3 Lösungsansätze und erzielte Ergebnisse

Zum Management des VPN-Dienstes der ATIS soll ein in Information 2 dargestelltes policy-basiertes Managementsystem prototypisch implementiert werden.



Information 2: Lösungsansatz für ein policy-basiertes Managementsystem

Um konkrete Policies für die Ressourcen des VPN-Dienstes formulieren zu können, ist es zunächst notwendig, die an der Diensterbringung beteiligten Ressourcen und ihr Zusammenspiel zu identifizieren. Anhand einer Analyse müssen Ressourcenparameter – die mit der Funktionalität bzw. Qualität des VPN-Dienstes in Zusammenhang stehen – gefunden werden, die durch Policies überwacht und gesteuert werden können. Ressourcenparameter sind Werte, die sich auf den jeweiligen Ressourcen direkt beobachten lassen, wie beispielsweise die Anzahl der aufgebauten VPN-Verbindungen oder die Prozessorauslastung eines Servers. Durch die Identifikation entsprechender Ressourcenparameter können Dienstparameter und deren Wertebereiche definiert werden, die später Teil eines SLA sind, beispielsweise eine Verfügbarkeit von 99%. Policies sollen dafür sorgen, dass der Betrieb innerhalb der im SLA spezifizierten Parameter erfolgt und Fehler im Idealfall automatisch behoben werden oder zumindest eine Benachrichtigung des zuständigen Personals erfolgt. Das Hauptaugenmerk der Arbeit wird dabei jedoch auf die reine Verfügbarkeit des VPN-Dienstes gelegt.

Zunächst sind für den VPN-Dienst konkrete Policies zu erarbeiten und gemäß den Vorgaben des *Policy Core Information Model* (PCIM) [ME+01] der IETF zu formalisieren. Da für die Implementierung des VPN-Dienstes das IPsec Protokoll verwendet wird, ist auch zu prüfen, inwieweit die IPsec Erweiterung [JR+03] und die generelle Erweiterung [Mo03] des PCIM Modells bei der Formalisierung von Policies für den VPN-Dienst genutzt werden können.

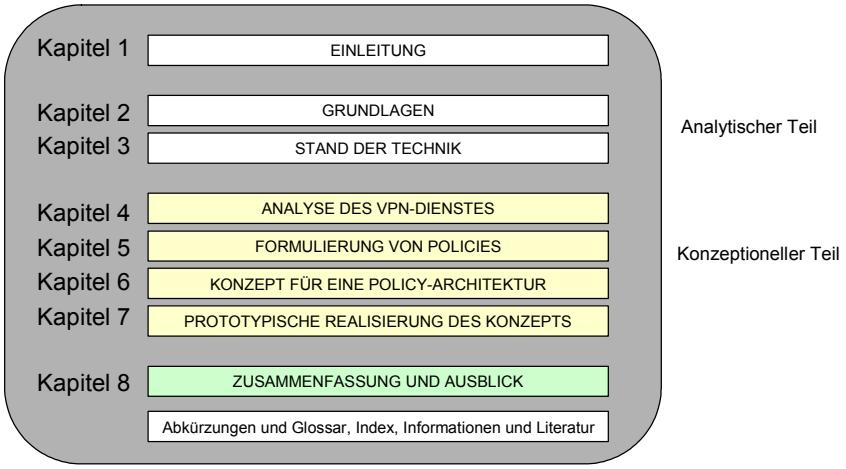
Um die formalisierten Policies einzusetzen ist es notwendig, die Komponenten [YP+00] der IETF Architektur zu realisieren. Ein *Policy Repository* wird benötigt, um alle mit dem *Policy Management Tool* erstellten Policies zu speichern. Der *Policy Decision Point* (PDP) trifft anhand verschiedener Policies eine Entscheidung und der *Policy Enforcement Point* (PEP) realisiert diese Entscheidung auf der entsprechenden Ressource. Die Architektur der IETF muss außerdem um eine Monitorkomponente erweitert werden, damit bestimmte Ressourcenparameter überwacht werden können. Ein Konzept ist auszuarbeiten, wie diese Komponenten konkret für den VPN-Dienst zu realisieren und miteinander zu verschalten sind.

Im letzten Schritt soll anhand der formalisierten Policies und des erarbeiteten Konzepts der Prototyp eines policy-basierten Managementsystems für den VPN-Dienst imple-

mentiert werden. An diesem Prototyp werden verschiedene Sachverhalte diskutiert, um Vorteile und Defizite der IETF-Architektur aufzuzeigen.

1.4 Gliederung der Arbeit

In diesem Kapitel wird der Aufbau der Arbeit in Form einer Kapitelstruktur angegeben.



Information 3: Gliederung der Arbeit

Zunächst wird im Grundlagenkapitel eine Einführung in die Thematik des policy-basierten Managements und in die VPN-Technologie gegeben. Dazu werden Terminologie, Architektur und Spezifikation der IETF eingeführt. In Kapitel 3 werden einige in der Literatur diskutierte Ansätze vorgestellt und daraus die Motivation des vorliegenden Ansatzes durch Aufzeigen der Defizite bzw. Unterschiede abgeleitet.

Kapitel 4 liefert eine detaillierte Analyse des VPN-Dienstes. Anhand dieser Analyse sind Parameter zu identifizieren, die zur Überwachung der Verfügbarkeit des VPN-Dienstes herangezogen werden können.

In Kapitel 5 sollen anhand der im vorigen Kapitel identifizierten Ressourcenparameter dem PCIM-Standard entsprechende Policies formuliert werden. Dazu müssen Wertebereiche der verschiedenen Ressourcenparameter mit Fehlerzuständen in Verbindung gebracht und Aktionen definiert werden, die aus diesen Fehlerzuständen herausführen.

Um die von der IETF spezifizierte Architektur und die erstellten Policies für ein policy-basiertes Managementsystem am Beispiel des VPN-Dienstes einzusetzen, wird in Kapitel 6 ein entsprechendes Konzept erstellt. Dabei werden Fragestellungen geklärt, wie einzelnen Komponenten zu realisieren und zu ergänzen sind.

Mit der Umsetzung des im vorigen Kapitel entwickelten Konzepts befasst sich Kapitel 7. Hier muss geklärt werden, mit welcher Software sich das entwickelte Konzept realisieren lässt und welche Werkzeuge nötig sind, um Messungen vorzunehmen und Konfigurationen anzupassen. Zum Abschluss dieses Kapitels werden einige Tests am Prototyp vorgenommen, um Vor- und Nachteile zu diskutieren.

Eine Zusammenfassung und Einordnung der Ergebnisse der Arbeit wird in Kapitel 8 vorgenommen. Die durch die bearbeiteten Themen aufgeworfenen Fragestellungen werden aufgezeigt und Ausblicke für weitere Untersuchungen gegeben.

2 GRUNDLAGEN

Das Kapitel soll einen grundlegenden Einblick in die Thematik policy-basiertes Management liefern und einen Überblick über die verschiedenen Ansätze und Architekturen der IETF geben. Außerdem wird in diesem Kapitel die grundlegende Funktionsweise eines VPN-Dienstes beschrieben.

In diesem Kapitel wird die von der IETF definierte Terminologie [WS+01] (RFC 3198) eingeführt, um darauf aufbauend PCIM zur Policy-Spezifikation [ME+01] (RFC 3060) und die Policy-Architektur vorzustellen [YP+00] (RFC 2753). Diese Bestandteile bilden die Grundlage, um Dienste – wie beispielsweise den VPN-Dienst – mit Policies überwachen und steuern zu können.

2.1 Policy-Terminologie

Ziel der Terminologie ist es, Begriffe rund um das Thema Policies einheitlich zu definieren, um Informationsasymmetrien zu vermeiden. Nur mit einer solch einheitlichen Begriffsdefinition ist es möglich, projektübergreifende Forschung zu betreiben. Um den Rahmen dieses Abschnitts nicht zu überfrachten, kann die komplette Terminologie in [WS+01] (RFC 3198) nachgeschlagen werden. Begriffe die im weiteren Verlauf der Arbeit von Relevanz sind, werden jedoch im Text bzw. im Glossar näher erläutert.

2.2 Policy-Spezifikation

Damit eine Policy spezifiziert werden kann, wird ein bestimmtes Beschreibungsformat benötigt, eine so genannte Policy-Spezifikation. Dazu existieren bereits verschiedene Ansätze, die sich in zwei Kategorien unterscheiden lassen, die sprachbasierten und die modellbasierten Ansätze. Der Unterschied der beiden Ansätze besteht darin, Policies entweder in einer eigenständigen Sprache zu formulieren (sprachbasierte Ansätze) oder sprachunabhängige Modelle (modellbasierte Ansätze) zu verwenden. Dabei stützen sich die sprachbasierten Ansätze auf bestimmte Beschreibungssprachen, wie beispielsweise *Extensible Markup Language* (XML), während modellbasierte Ansätze keine bestimmte Implementierung wählen, sondern ein allgemeines Informationsmodell spezifizieren. Als Vertreter für die modellbasierten Ansätze wird das *Policy Core Information Model* (PCIM) [ME+01] und dessen Erweiterungen [Mo03] vorgestellt. PCIM ist eine gemeinsame Entwicklung der IETF und der *Distributed Management Task Force* (DMTF) und stellt eine Erweiterung des *Common Information Model* (CIM) dar. CIM wurde als völlig offener Standard zum Management von Systemen und Anwendungen entwickelt. PCIM erweitert diesen Standard gezielt, um Regeln aus Bedingungen und Aktionen formulieren zu können.

Beim Entwurf von PCIM wurde von der IETF zunächst das Augenmerk auf die Formulierung von Policies für die Bereiche QoS (IntServ und DiffServ) und IPsec gelegt. Wird von einem Policy-gesteuerten Netzwerk gesprochen, ist dieses als Zustandsautomat modelliert, in dem Policies dazu verwendet werden, einzelne Policy-gesteuerte Geräte in den zur Zeit gewünschten oder erlaubten Zustand zu versetzen. Dazu werden Bedingungen ausgewertet, um festzustellen, ob Aktionen auszuführen sind. Bedingungen legen genau fest, in welchem Zustand oder unter welchen Voraussetzungen Aktionen durchgeführt werden müssen.

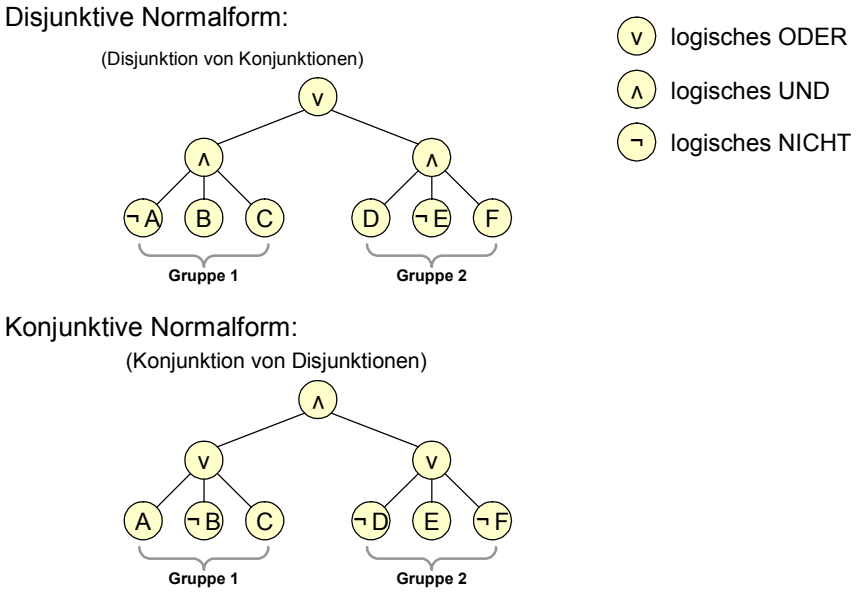
Im Kontext der IT-Infrastruktur werden Policies unter anderem dazu verwendet, (Netzwerk-) Komponenten zu administrieren, zu managen und den Zugriff auf sie zu kontrollieren. PCIM definiert genau, wie eine Policy aufgebaut sein muss, um für die genannten Ziele eingesetzt werden zu können.

2.2.1 Einführung in PCIM

PCIM definiert zwei verschiedene Arten von Objekten: Struktur- und Assoziationsklassen. Strukturklassen definieren instanzierbare Objekte und legen die Attribute fest, die ein Objekt enthalten muss bzw. enthalten kann. Assoziationsklassen legen die Beziehungen zwischen Instanzen der Strukturklassen fest. Die Policy-Klassen und -Assoziationen, die von PCIM definiert werden, sind generisch gehalten und erlauben damit die Formulierung von Policies zu nahezu jedem Sachverhalt. Die einzelnen Klassen können erweitert werden, um speziell benötigte Anforderungen zu erfüllen. Durch PCIM wird also eine universell einsetzbare Basis für die Formulierung von Policies gelegt, die auch für künftige Anwendungen einsetzbar bleibt.

Policies werden durch die Aneinanderreihung einer beliebigen Anzahl von Policy-Regeln definiert. Jede Policy-Regel wiederum besteht aus einer Menge von Bedingungen und einer Menge von Aktionen. Policy-Regeln können zu Policy-Gruppen aggregiert werden und diese Gruppen können selbst wiederum gruppiert werden, um verschachtelte, hierarchische Policies zu bilden. Wenn bei der Verarbeitung eine Bedingung eintritt – die Policy also den Wert „wahr“ annimmt –, werden Aktionen ausgeführt, wodurch die Ressource in einen anderen Zustand gebracht wird. Der konkrete Ablauf der Verarbeitung von Policies wird aber erst durch die Policy-Architektur in Kapitel 2.3 festgelegt. Die Policy-Spezifikation legt fest, wie die einzelnen Policies zu beschreiben sind.

Die Bedingungen einer Policy-Regel werden, wie in Information 4 dargestellt, entweder in konjunktiver (KNF) oder disjunktiver (DNF) Normalform angegeben.



Information 4: Syntaxbedingung an PolicyConditions

Ob Bedingungen in KNF oder DNF angegeben sind, wird durch den Wert des Attributs „ConditionListType“ der Klasse PolicyRule festgelegt. In der Assoziation „Policy-ConditionInPolicyRule“ werden zwei Attribute „GroupNumber“ und „Condition-Negated“ definiert. Mit dieser Assoziation werden Bedingungen an eine PolicyRule gebunden und festgelegt, welcher Gruppe diese Bedingungen angehören und ob sie negiert sind oder nicht.

Für die in einer Policy-Regel auszuführenden Aktionen kann eine Priorisierung angegeben werden; ferner ist es möglich zu spezifizieren, ob diese Reihenfolge für die Ausführung relevant ist oder ob es sich dabei lediglich um eine empfohlene Reihenfolge handelt. Natürlich ist es auch möglich anzugeben, dass die Ausführungsreihenfolge der Aktionen keine Rolle spielt. Auch Policy-Regeln selbst können priorisiert werden, um beispielsweise allgemeingültige Policies mit einzelnen Spezialfällen auszudrücken. Zum Beispiel sollen alle Nutzer des VPN-Dienstes einen Datendurchsatz von 1 Mbit/s erhalten, einzelnen Personen soll aber ein Durchsatz von 10 Mbit/s gewährt werden. Damit die gewünschten Einzelpersonen einen höheren Durchsatz erhalten, muss die Policy, die den höheren Durchsatz gewährleistet, eine höhere Priorität aufweisen als die Policy, die den niedrigen Durchsatz bereitstellt. Die höher priorisierte Policy (Spezialfall) setzt sich also gegen die Policy mit der niedrigeren Priorität (Standardfall) durch.

Policies können entweder einzeln verwendet oder zu Policy-Gruppen aggregiert werden. Einzelne Policies werden Policy-Rules genannt. Um erweiterte Funktionalität zu erhalten, lassen sich entweder Policy-Regeln zu Policy-Gruppen oder auch Policy-Gruppen zu weiteren Policy-Gruppen zusammenfassen. Direktes Aggregieren – die Mischung von Policy-Regeln und Policy-Gruppen innerhalb einer Policy – ist jedoch nicht möglich. Policy-Regeln können dabei einfache QoS-Anfragen – beispielsweise die Abfrage des freien Warteschlangenpuffers eines Switches – sein, während mit Policy-Gruppen komplexe Abläufe – wie zum Beispiel der komplette Anmeldevorgang eines Benutzers an einer bestimmten Ressource – modelliert werden können. Ein Entwurfsziel von PCIM ist es, sowohl einzelne als auch zu Gruppen aggregierte Policies zu unterstützen.

Eine generelle Klassifizierung von Policy-Gruppen kann anhand des beabsichtigten Ziels vorgenommen werden, um den Dienst zu beschreiben und eine künftige Gruppierung neuer Policies zu erleichtern. Von PCIM werden die folgenden Kategorien unterschieden:

Kategorie	**Beschreibung**
Motivierungspolicies	Befassen sich einzig und allein damit, wie ein gewisses Ziel einer Policy erreicht werden kann. Konfigurations- und Benutzungspolicies sind spezielle Motivierungspolicies. Ein anderes Beispiel ist die Zeitplanung eines Dateisicherungsdienstes basierend auf Schreibaktivitäten auf Dateien zwischen 8:00 Uhr und 15:00 Uhr von Montag bis Freitag.

Kategorie	Beschreibung
Konfigurationspolicies	Definieren eine Standardkonfiguration eines Managementobjekts (wie beispielsweise ein Netzwerkdienst). Typische Beispiele für Konfigurationspolicies sind die Einrichtung von Netzweiterleitungsdiensten oder Druckerwarteschlangen.
Installationspolicies	Legen fest, was auf ein System aufgespielt werden darf und was nicht, außerdem wird die Konfiguration der Mechanismen festgelegt, welche die Installation durchführen. Installationspolicies repräsentieren typischerweise administrative Rechte und Abhängigkeiten zwischen verschiedenen Komponenten. (Beispiel: Um Komponente A erfolgreich zu installieren, ist es zunächst notwendig, Komponente B und C zu deinstallieren bzw. zu installieren.)
Fehler- und Ereignispolicies	Beispielsweise die Benachrichtigung des Systemadministrators im Falle eines Ausfalls zwischen 8:00 Uhr und 21:00 Uhr, andernfalls erfolgt eine Benachrichtigung des Help-Desks.
Benutzungspolicies	Kontrollieren die Auswahl und Konfiguration von Komponenten basierend auf bestimmten „Nutzungsdaten“. Konfigurationspolicies können von Benutzungspolicies abgeleitet oder einfach übernommen werden. Ein Beispiel für Benutzungspolicies sind die Rekonfiguration einer Netzweiterleitungskomponente, nachdem ein Nutzer der Dienstgüteklasse “Gold” registriert wurde.
Sicherheitspolicies	Beschäftigen sich mit der Authentifizierung des Klienten, der Vergabe und des Entzugs von Rechten, der Auswahl geeigneter Authentifizierungsmechanismen und führen Accounting der Ressourcennutzung durch.
Dienstpolicies	Dienstpolicies beschreiben und charakterisieren verfügbare Dienste im Netzwerk. Beispiel: Alle Interfaces von Weitverkehrsleitungen sollen einen bestimmten Warteschlangenalgorithmus verwenden.

Tabelle 1: Verschiedene Policy Kategorien von PCIM

Eine Zuordnung von einer Policy zu einer der Kategorien wird in PCIM dadurch erreicht, dass der Wert des Attributs “PolicyKeywords“ der Klasse Policy entsprechend gesetzt wird.

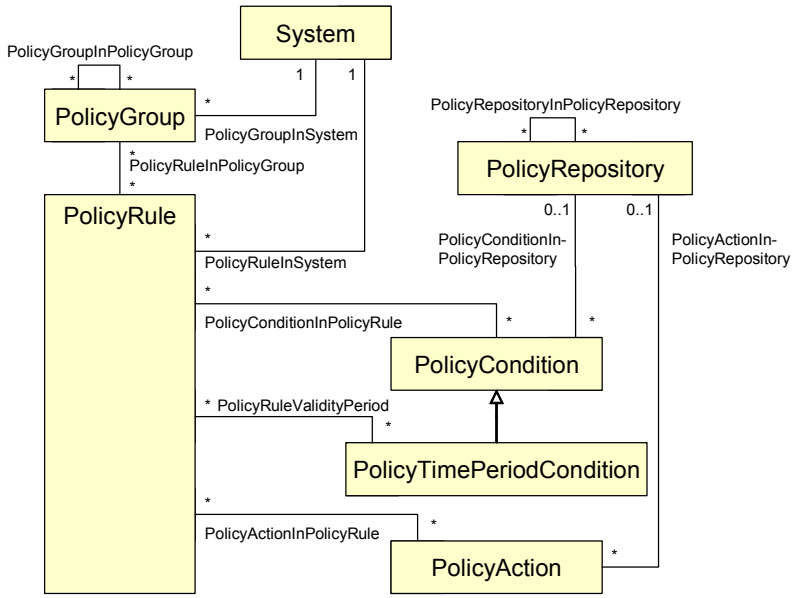
PCIM wurde aus verschiedenen Gründen deklarativ und nicht prozedural modelliert. Es definiert also keine Abfolgen von Schritten, die berücksichtigt werden müssen, um zu einem Ergebnis zu kommen, sondern liefert vielmehr gewisse standardisierte modulare Blöcke von Attributen und Beziehungen, die beim Formulieren einer Policy zu beachten sind. Auf prozedurale Vorgehensweisen wurde absichtlich verzichtet, um die konkrete Verarbeitung von Policies dem Implementierer zu überlassen und ihn nicht weiter einzuschränken. Ziel des deklarativen Modells ist es, eine möglichst einfach ver-

ständliche Beschreibungssprache für Policies zu bieten, die auch gut durch Maschinen verarbeitet werden kann.

Die IETF definiert PCIM auch nicht als vollständig, da bewusst keine Algorithmen zur Verarbeitung der Policies angegeben werden, um eine möglichst freie Implementierung zu ermöglichen. Es wurde bei PCIM versucht, einen Kompromiss zwischen Komplexität und mangelnder Mächtigkeit durch Einsatz logischer Konzepte wie der konjunktiven und disjunktiven Normalform zu finden, die es ermöglichen, zunächst einfach gehaltene Policies zu komplexeren zusammenzufügen.

2.2.2 Das PCIM-Klassenmodell

In Information 5 ist das von PCIM spezifizierte Klassenmodell dargestellt. Es zeigt die wichtigsten Struktur- und Assoziationsklassen.

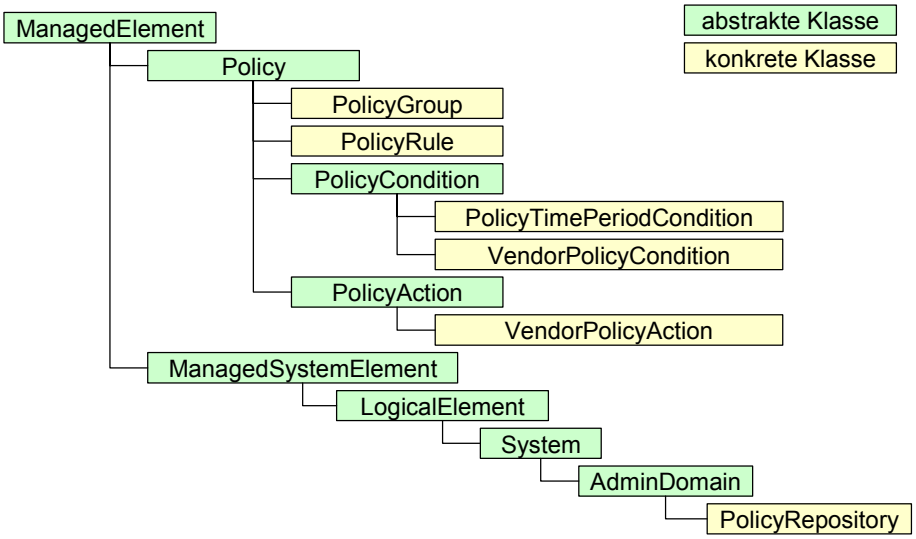


Information 5: Das Klassenmodell von PCIM

Die Kardinalitäten, die für jede Assoziation an den jeweiligen Strukturklassen angegeben sind bezeichnen, wie viele dieser Beziehungen für die angegebene Assoziationsrichtung existieren können. Eine PolicyCondition kann sich also entweder im PolicyRepository befinden (1) oder nicht (0), während das PolicyRepository eine beliebige Anzahl an verschiedenen PolicyConditions enthalten kann (*). Die Beziehung zwischen PolicyCondition und PolicyTimePeriodCondition ist eine Spezialisierung, denn die Klasse PolicyTimePeriodCondition ist lediglich eine ganz bestimmte Ausprägung von Attributen der Klasse PolicyCondition.

Bevor näher auf die einzelnen Struktur- und Assoziationsklassen eingegangen wird, folgt zunächst die Vererbungshierarchie der PCIM-Klassen. In der Vererbungshierarchie (Information 6, Information 7) lässt sich auch gut die Beziehung zwischen PCIM und CIM erkennen. Das von der DMTF entwickelte CIM bietet eine einheitliche Definition von Managementinformationen für Systeme, Netzwerke, Applikationen und Dienste. PCIM erweitert die dort angegebenen Klassen derart, dass es möglich wird, Managementinformationen in Form von Policies zu erstellen. Dazu werden von PCIM,

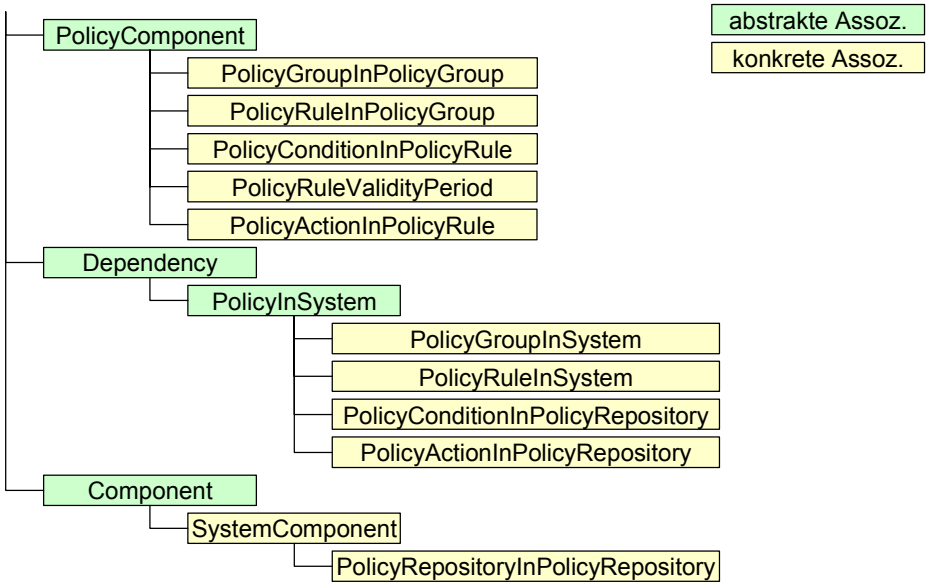
wie im Folgenden dargestellt (Information 6), verschiedene CIM-Klassen um weitere Attribute und Unterklassen ergänzt.



Information 6: Vererbungshierarchie der PCIM-Strukturklassen

ManagedElement, ManagedSystemElement, LogicalElement, System, und AdminDomain werden vom CIM Schema [CIM] definiert und hier nicht näher betrachtet.

In CIM werden Assoziationen als Klassen modelliert. Die Vererbungshierarchie der Assoziationen in PCIM stellt Information 7 dar.



Information 7: Vererbungshierarchie der PCIM-Assoziationsklassen

Die Dependency-, Component- und SystemComponent-Assoziationen sind im CIM-Schema definiert und werden hier nicht näher betrachtet.

Bei Bedarf können auch weitere CIM-Klassen hinzugenommen werden, um zusätzliche Sachverhalte zu modellieren. Beispielsweise kann die CIM-Klasse Statistics hin-

zugezogen werden, um statistische Informationen über die Einsatzhäufigkeit einzelner Policies aufzuzeichnen.

2.2.3 PCIM-Klassen

Im Folgenden werden die PCIM-Klassen näher erläutert, um deren Bedeutung zu verdeutlichen.

Die abstrakte Klasse „Policy“

Diese Klasse dient dazu, gewisse Attribute zu bündeln und den Instanzen der untergeordneten Klassen verfügbar zu machen.

Das Attribut „CommonName“
Dieses Attribut enthält den umgangssprachlichen Namen für ein bestimmtes Objekt.

Das mehrwertige Attribut „PolicyKeywords”
Mit diesem Attribut lässt sich ein Objekt zu einer oder mehreren in Tabelle 1 vorgestellten Kategorien zuordnen. Zusätzlich zu den bereits vorhandenen Kategorien lassen sich auch eigene Kategorien anlegen.

Das Attribut „Caption”
Hier wird eine einzeilige Beschreibung für ein Objekt angegeben.

Das Attribut „Description”
Dieses Attribut wird für eine ausführliche Beschreibung des Objekts verwendet.

Die Klasse „PolicyGroup“

In dieser Klasse können entweder PolicyRules oder PolicyGroups zu einer größeren Einheit zusammengefügt werden. Die Bildung von Schleifen – durch die gegenseitige Aggregation von PolicyGroups oder gar die Aggregation mit sich selbst – ist nicht zulässig. Es können Verschachtelungen mit beliebiger Tiefe erzeugt werden.

Alle benötigten Attribute erbt diese Klasse von der Klasse „Policy“.

Die Klasse „PolicyRule“

Diese Klasse repräsentiert die mit einer Policy assoziierte “Wenn Bedingung, dann Aktion” Semantik. Sie bündelt eine Menge von Bedingungen und Aktionen zu einer PolicyRule.

Die Bedingungen und Aktionen werden über die Aggregationen „PolicyConditionInPolicyRule“ und „PolicyActionInPolicyRule“ mit Instanzen dieser Klassen verbunden. Über die Aggregation „PolicyRuleValidityPeriod“ kann einer Policy ein Zeitplan zugewiesen werden, an dem die Policy aktiv bzw. inaktiv ist.

Das Attribut „Enabled“
Durch diesen Wert wird eine bestimmte Policy aktiviert oder deaktiviert. Dies erleichtert die Handhabung für den Administrator, da er inaktive Policies nicht direkt löschen muss. Zur Fehlerbehebung und zu Testzwecken lassen sich mit diesem Attribut auch Policies in den Zustand versetzen, dass zwar die Bedingungen überprüft werden, aber keine Aktionen bei der Ausführung der Policy stattfinden.

Das Attribut „ConditionListType“
Dieses Attribut gibt an, ob die mit der PolicyRule verknüpften PolicyConditions in KNF oder DNF angegeben sind (s. Information 4).

Das Attribut „RuleUsage“
Das Attribut enthält die Beschreibung, wie eine bestimmte Policy zu benutzen ist.

Das Attribut „Priority“
Dieser nicht negative, ganzzahlige Wert gibt die Priorität einer Policy an. Der Wert „Null“ bezeichnet die niederste Priorität einer Policy. Treten gleichzeitig die Bedingungen von verschiedenen Policies ein, werden nur die mit der höchsten Priorität ausgeführt.

Das Attribut „Mandatory“
Mit diesem Attribut wird angegeben, ob die Verarbeitung der Policy notwendig ist oder nicht. Dieser Mechanismus dient dazu, Policies zu markieren, deren Ausführung selbst bei starker Belastung der Systeme zwingend notwendig ist.

Das Attribut „SequencedActions“
Dieses Attribut gibt an, ob es wichtig ist, die Aktionen in der angegebenen Reihenfolge auszuführen oder nicht. Es gibt die Auswahl zwischen drei möglichen Ausführungsverhalten:

1. Die Aktionen müssen in der angegebenen Reihenfolge ausgeführt werden. Wenn das nicht möglich ist, wird keine der Aktionen ausgeführt.
2. Die Aktionen sollten in der angegebenen Reihenfolge ausgeführt werden. Falls das nicht möglich ist, dürfen die Aktionen auch in einer anderen Reihenfolge ausgeführt werden
3. Die Reihenfolge der Ausführung spielt keine Rolle.

Das mehrwertige Attribut „PolicyRoles“
Rollen sind ein zentraler Aspekt des Policy-Rahmenwerks. Hinter dem Konzept steht die Idee, nicht jeder Ressource direkt Policies zuzuweisen, sondern Ressourcen eine oder mehrere Rollen zuzuordnen. Den verschiedenen Rollen wiederum werden Policies zugeordnet. So ist es möglich, die Konfiguration aller Ressourcen mit derselben Rolle durch Änderung einer einzigen Policy anzupassen. Das Policy-Rahmenwerk sorgt dafür, dass Policies auf all den Ressourcen ausgeführt werden, die der entsprechenden Rolle zugeordnet wurden. Eine Rolle ist also nicht nur ein einfaches Attribut, sondern legt viel mehr das Verhalten einer bestimmten Ressource fest bzw. beschreibt deren Fähigkeiten. Durch die Zuordnung von Rollen oder Rollenkombinationen einer bestimmten Ressource wird festgelegt, welcher Teil einer größeren Gesamtmenge an Policies auf diese Ressource anwendbar ist.

Das Rollenkonzept wird anhand der Konfiguration von Universitätsnetzwerkkarten verdeutlicht. Hierbei wurden drei verschiedene Rollen entworfen. Die Rolle „Ethernet“ enthält die Grundkonfiguration einer Netzwerkkarte, je nachdem ob dieses Interface das Intranet oder Internet verbindet, gibt es zwei weitere Rollen „Intranet“ und „Internet“. Eine Netzwerkkarte kann nun beispielsweise der Rolle „Ethernet“ und „Intranet“ oder der Rolle „Ethernet“ und „Internet“ zugewiesen werden. Ein Policy-Rahmenwerk muss für eine Ressource, die der Rollenkombination „Ethernet“ und „Intranet“ zugeordnet ist,

die Policies für die Rollen „Ethernet“, „Intranet“ und die Rollenkombination „Ethernet+Intranet“ anwenden. Der Policy-Administrator kann bei der Spezifikation angeben, dass die Rolle „Intranet“ und „Internet“ inkompatibel zueinander sind, da eine Kombination dieser beiden Rollen nicht vorgesehen ist. Durch diese Inkompatibilitätsmarkierung treten drei Effekte ein:

1. Wenn einem Interface eine Rollenkombination von „Intranet“ und „Internet“ zugewiesen wird, kann das policy-basierte Managementsystem dies als Fehler markieren.
2. Wenn eine PolicyRule mit einer Rollenkombination von „Intranet“ und „Internet“ assoziiert wird, kann das policy-basierte Managementsystem dies als Fehler markieren.
3. Wenn das policy-basierte Managementsystem zwei PolicyRules erkennt, die einerseits der Rolle „Intranet“ und andererseits der Rolle „Internet“ zugeordnet sind, muss das System nicht nach Konflikten zwischen den beiden Regeln suchen, da sie nicht in Kombination miteinander auftreten können.

Die Eigenschaft PolicyRoles ist mit einer PolicyRule assoziiert und muss in folgender Form angegeben werden:

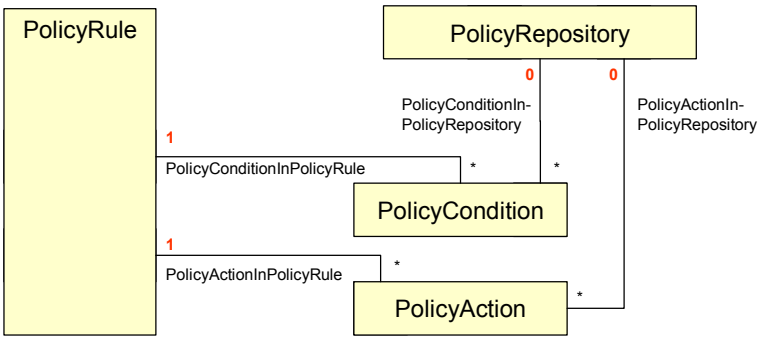
<RoleName>[&&<RoleName>]*

Jeder Wert dieser Eigenschaft kann eine Rollenkombination oder eine einzelne Rolle sein. Rollenkombinationen werden mit logischem UND verknüpft, während die verschiedenen Werte von PolicyRoles mit logischem ODER verknüpft werden. Die Rollennamen einer Rollenkombination sind in alphabetischer Reihenfolge anzugeben, damit Stringvergleiche korrekt ausgeführt werden können. Die Rollennamen selbst werden vom Policy-Administrator nach seinen Anforderungen festgelegt.

Die abstrakte Klasse „PolicyCondition“

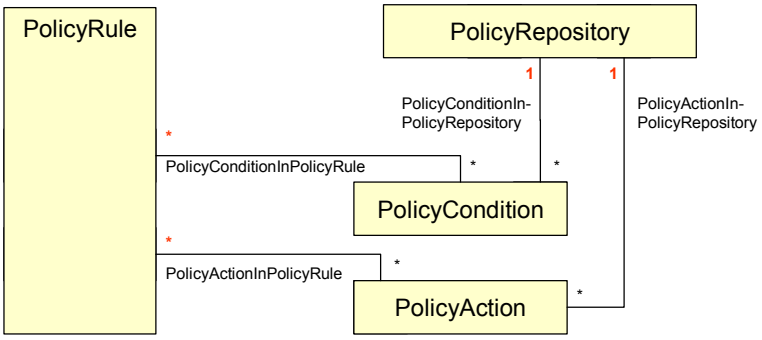
Diese Klasse dient dazu, Bedingungen aufzunehmen, anhand derer dann Entscheidungen über die auszuführenden Aktionen getroffen werden können. Eine einzelne PolicyCondition wird ausgewertet und ist entweder erfüllt oder nicht erfüllt. Um tatsächlich reale Bedingungen formulieren zu können, müssen noch weitere domänenspezifische Attribute aus Unterklassen von PolicyCondition herangezogen werden.

Bedingungen und Aktionen von Policies werden in zwei Gruppen unterschieden: wieder verwendbar oder regelspezifisch. Bei dieser Unterscheidung geht es um die Modellierungsintention des Policy-Administrators. Regelspezifische Bedingungen und Aktionen sind genau einer PolicyRule zugeordnet und werden nicht im PolicyRepository abgelegt.



Information 8: Regelspezifische Aktionen und Bedingungen

Wieder verwendbare Bedingungen und Aktionen müssen im PolicyRepository abgelegt sein und können in keiner, einer oder mehreren PolicyRules verwendet werden.



Information 9: Wieder verwendbare Aktionen und Bedingungen

In PCIM wird diese Bedingung textuell festgehalten mit dem Hinweis, dass eine formale Methode, um komplexe Einschränkungen zu beschreiben, benötigt wird.

Die Unterscheidung in wieder verwendbare und regelspezifische Bedingungen und Aktionen ist wichtig für die Namensgebung der jeweiligen Objekte, denn CIM unterscheidet anhand des Objektnamens diese beiden Klassen.

Die Klasse „PolicyTimePeriodCondition“

Diese Klasse dient dazu, einen Zeitplan für eine PolicyRule zu erstellen, indem diese gelten soll. Die im Folgenden definierten Attribute werden mit logischem UND verknüpft und ergeben gemeinsam den Gültigkeitszeitraum einer PolicyRule.

Das Attribut „TimePeriod“
Der Wert spezifiziert einen Zeitbereich, in dem eine PolicyRule gelten soll. Dabei lässt sich ein genauer Tag und eine genaue Uhrzeit für den Anfang und das Ende der Periode angeben. Zusätzlich gibt es noch die Möglichkeit, den Anfangs- oder Endzeitpunkt nicht zu beschränken.

Das Attribut „MonthOfYearMask“
Mit diesem Wert kann die Gültigkeit einer PolicyRule auf bestimmte Monate beschränkt werden.

Das Attribut „DayOfMonthMask“
Schränkt die Gültigkeit einer PolicyRule auf bestimmte Tage eines Monats ein.

Das Attribut „DayOfWeekMask“
Dieser Wert gibt die Wochentage an, an denen eine PolicyRule angewendet werden darf.

Das Attribut „TimeOfDayMask“
Das Attribut gibt einen Uhrzeitbereich an, zu dem die PolicyRule angewendet werden darf. Ist die angegebene Enduhrzeit größer als die Anfangsuhrzeit, umfasst der Zeitbereich Mitternacht.

Das Attribut „LocalOrUtcTime“
Eine Instanz von PolicyTimePeriodCondition hat fünf Eigenschaften, die Zeiten repräsentieren: TimePeriod, MonthOfYearMask, DayOfMonthMask, DayOfWeekMask und TimeOfDayMask. Dabei werden zwei unterschiedliche Uhrzeittypen unterschieden: Lokalzeit und koordinierte Weltzeit (UTC). Lokalzeit gibt dabei die Uhrzeit des Systems an, in dem die Policy angewendet wird, und UTC bezieht sich auf eine weltweit eindeutige Zeitangabe. Die Eigenschaft LocalOrUtcTime gibt dabei an, auf welchen Zeittyp sich eine Instanz einer PolicyTimePeriodCondition bezieht.

Im *Policy Management Tool*, das zur Eingabe, Bearbeitung und Löschung von Policies dient wird sichergestellt, dass alle für eine Policy erforderlichen PolicyTime-PeriodConditions korrekt erstellt werden. Diese Aufgabe ist teilweise recht komplex, da die lokalen Gegebenheiten von Zeitrechnungen wie Zeitzone, Sommer- und Winterzeit berücksichtigt werden müssen. Durch die Auslagerung der Umrechnung in das *Policy Management Tool*, wird dieser aufwändige Arbeitsschritt nur einmal ausgeführt, und zur Laufzeit im Policy Management System sind keine weiteren Umrechnungen mehr erforderlich.

Die Klasse „VendorPolicyCondition“

Diese Klasse soll es ermöglichen, die Klasse „PolicyCondition“ um andere noch nicht modellierte Eigenschaften zu erweitern.

Das mehrwertige Attribut „Constraint“
Dieser Wert kann eine noch nicht modellierte Eigenschaft aufnehmen.

Das Attribut „ConstraintEncoding“
Das Attribut gibt eine OID der unter „Constraint“ angegebenen Variable(n) an.

Die abstrakte Klasse „PolicyAction“

Das Ziel einer PolicyAction ist es, durch Ausführung einer oder mehrerer Operationen den gewünschten Zustand einer Ressource zu erreichen. Dabei wird mindestens die Konfiguration eines Elementes verändert.

In einer PolicyRule können mehrere PolicyActions zusammengefasst werden. Die Reihenfolge der Aktionen kann durch die Aggregation „PolicyActionInPolicyRule“ festgelegt werden. Durch ein Attribut der Klasse „PolicyRule“ wird angezeigt, ob die Reihenfolge relevant ist (s. o.).

Wie auch für die Klasse „PolicyCondition“ gilt hier die Unterscheidung zwischen wieder verwendbaren und regelspezifischen Aktionen, da für diese beiden Klassen unterschiedliche Namensgebungen verwendet werden.

Die Klasse „VendorPolicyAction“

Durch diese Klasse ist es möglich, noch nicht modellierte PolicyActions durch eigene Erweiterung hinzuzufügen.

Das mehrwertige Attribut „ActionData“
Hier werden die Werte der noch nicht modellierten Eigenschaft aufgenommen.

Das Attribut „ActionEncoding“
ActionEncoding gibt eine OID für die Werte des Attributs „ActionData“ an.

Die Klasse „PolicyRepository“

Diese Klasse repräsentiert den aus administrativer Sicht definierten Container für wieder verwendbare Policy-bezogene Informationen.

2.2.4 Erweiterungen durch PCIM-Extensions (PCIMe)

Um PCIM zu verbessern, hat die IETF im Januar 2003 PCIMe [M003] (RFC 4104) spezifiziert. In dieser Erweiterung wurden viele Verbesserungen erzielt und andere Ansätze wie *Policy QoS Information Model* (QPIM), *IPsec Configuration Policy Model* (ICPM) und *Information Model for Describing Network Device QoS Datapath Mechanisms* (QDDIM) verarbeitet und zu PCIM hinzugefügt.

Änderung an der Klasse PolicyRepository

Um die Verwechslungsgefahr mit der Architekturkomponente *Policy Repository* zu vermeiden, wurde die Klasse PolicyRepository in ReusablePolicyContainer umbenannt. Gleichzeitig wurden verschiedene neue Assoziationen eingeführt, um die Aufnahmefähigkeit des ReusablePolicyContainers zu erweitern.

Zusätzliche Assoziationen und wieder verwendbare Elemente

Eine neue Klasse PolicySet und eine neue Assoziation PolicySetComponent ermöglichen es, größere Teile aus PolicyRules und PolicyGroups zusammenzufügen und im ReusablePolicyContainer abzulegen.

Prioritäten und Entscheidungsstrategien

Die Prioritätseigenschaft wurde aus der Klasse PolicyRule in die Assoziation PolicySetComponent verschoben. Durch diese Modellierung ist es notwendig geworden, dass zu priorisierende Policies entweder in einer gemeinsamen PolicyRule, PolicyGroup oder einem gemeinsamen System zur Verfügung stehen.

Die Beschränkung, dass eine Priorisierung nur im Konfliktfall durchgeführt werden soll, wurde aufgehoben. Eine PolicyDecisionStrategy-Eigenschaft in den Klassen PolicyGroup und PolicyRule wurde eingeführt, um dem Administrator zu ermöglichen, entweder die Aktionen aller PolicyRules die zutreffen (AllMatching) oder nur die Aktionen der PolicyRule mit der höchsten Priorität (FirstMatching) auszuführen.

PolicyRoles

Durch Einführung der Oberklasse PolicySet wird nun auch der Klasse PolicyGroup die Rolleneigenschaft vererbt. Verschachtelte Policies erben automatisch die bereits spezifizierten Rollen, es können aber zusätzlich weitere Rollen für eine derartig verschachtelte Policy angegeben werden.

Da PCIM nicht definiert hat, wie in der Praxis einer Ressource eine Rolle zugeordnet werden kann, wurde die Klasse PolicyRoleCollection eingeführt. Durch diese Klasse lassen sich Anfragen von einer Ressource auch einer Rolle zuweisen, denn zuvor musste diese Zuordnung direkt auf der Ressource vorgenommen werden.

CompoundPolicyConditions und CompoundPolicyActions

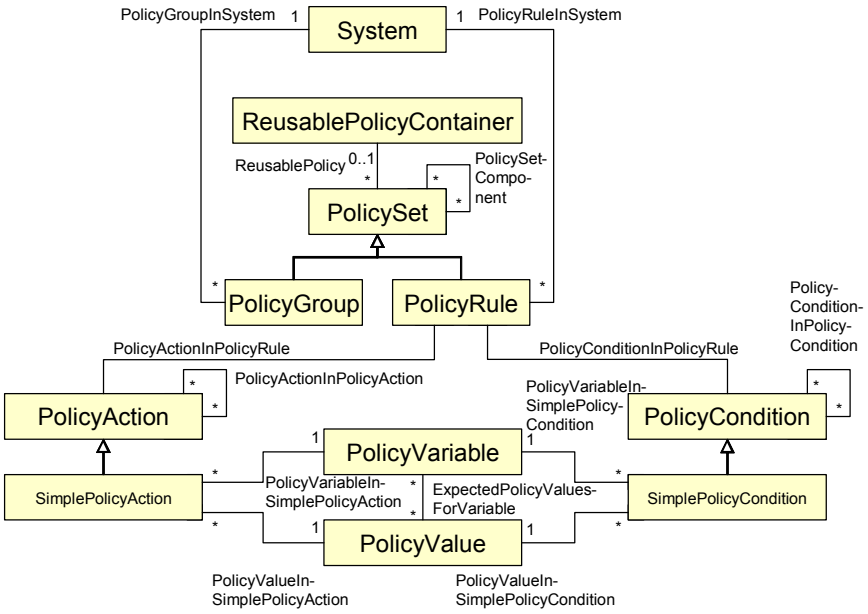
Compound-Klassen wurden eingeführt, um die Speicherung wieder verwendbarer Teile einer Policy im ReusablePolicyContainer zu ermöglichen. Für die Verbünde von PolicyConditions und PolicyActions gelten die von PCIM definierten Regeln bzgl. KNF/DNF und Priorisierung der Ausführungsreihenfolge. Die einzelnen Komponenten können dabei Instanzen der Klassen SimplePolicyCondition oder SimplePolicyAction sein oder aber auch aus anderen Compound-Elementen bestehen.

Variablen und Werte

Die Klassen PolicyVariable und PolicyValue wurden eingeführt. Nun ist es möglich, mit einer SimplePolicyCondition zu prüfen, ob eine bestimmte Variable einen bestimmten Wert hat. Durch SimplePolicyAction können bestimmte Variablen auf einen gewissen Wert gesetzt werden.

PCIMe Klassenmodell

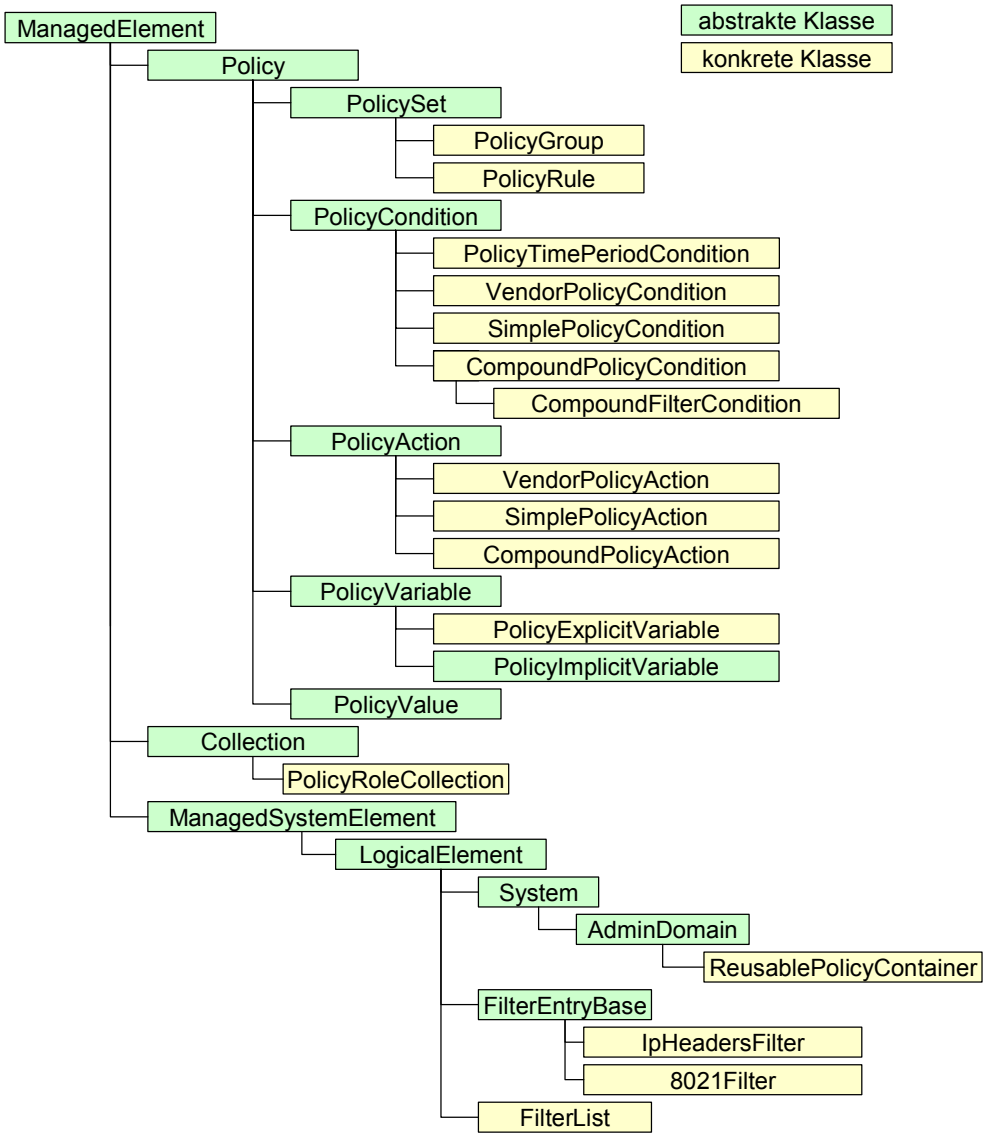
In Information 10 ist das Klassenmodell gemäß PCIMe dargestellt. Die Veränderungen gegenüber PCIM lassen sich hier besonders deutlich erkennen.



Information 10: Klassenmodell von PCIMe

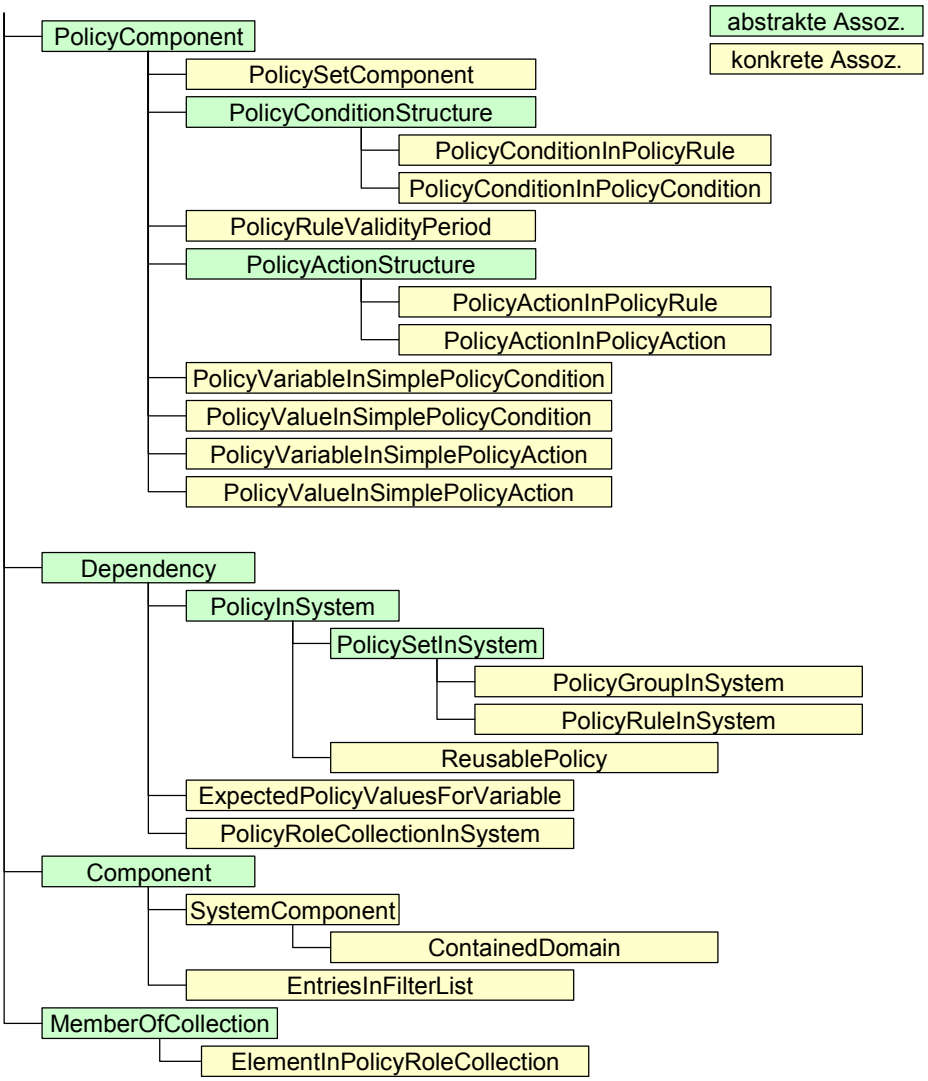
Die wichtigsten Änderungen am Modell sollen an dieser Stelle noch einmal kurz zusammengefasst werden. PolicyGroups und PolicyRules lassen sich nun in der neuen Klasse PolicySet zu größeren Einheiten kombinieren. Das in ReusablePolicyContainer umbenannte PolicyRepository enthält nun PolicySets, statt bisher nur der PolicyActions und PolicyConditions. Um die bisher fehlenden Attribute der Klasse PolicyAction und PolicyCondition zu ergänzen, wurde den beiden neuen Unterklassen SimplePolicy-Action und SimplePolicyCondition die Klassen PolicyVariable und PolicyValue zur

Seite gestellt. Durch diese neuen Klassen ist es nun direkt möglich, die Werte von bestimmten Variablen zu vergleichen oder zu setzen.



Information 11: Vererbungshierarchie der PCIMe-Klassen

Auch die Vererbungshierarchie der PCIMe Klassen zeigt deutlich, dass viele neue Klassen eingefügt wurden. Lediglich die Klasse PolicyRepository wurde entfernt und durch die Klasse ReusablePolicyContainer ersetzt. Die Klassen PolicyRule und PolicyGroup wurden unter die neue Klasse PolicySet verschoben.

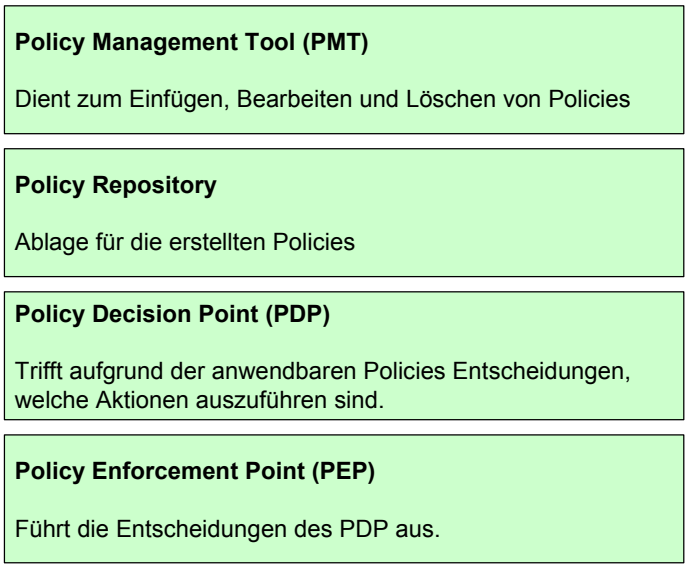


Information 12: Vererbungshierarchie der PCIMe-Assoziationen

Die Anzahl der Assoziationen hat sich auch deutlich vergrößert. Entfernt wurden die Assoziationen PolicyGroupInPolicyGroup, PolicyRuleInPolicyGroup, PolicyCondition-InPolicyRepository, PolicyActionInPolicyRepository und PolicyRepositoryInPolicy-Repository. Die Assoziationen PolicyActionInPolicyRule und PolicyConditionIn-PolicyRule wurden unter die neuen Assoziationen PolicyActionStructure bzw. Policy-ConditionStructure verschoben.

2.3 Policy-Architektur

Nachdem im letzten Kapitel die Beschreibungssprache PCIM detailliert vorgestellt wurde, soll nun die von der IETF definierte Policy-Architektur vorgestellt werden [YP+00] (RFC 2753). Die Architektur bietet ein Rahmenwerk, wie Policies in einer Infrastruktur zur Zugangskontrolle eingesetzt werden können und definiert vier verschiedene Architekturkomponenten sowie deren Beziehungen zueinander. In Information 13 werden diese Komponenten dargestellt.



Information 13: Die vier Komponenten der Policy Architektur

2.3.1 Policy Management Tool – PMT

Das *Policy Management Tool* ist die administrative Komponente des Modells. Mit dieser Komponente können Policies erzeugt, bearbeitet und gelöscht werden. Für den Administrator ist diese Komponente das Werkzeug zur Verwaltung von Policies. Die Komponente übernimmt dabei die Aufgabe zu prüfen, ob durch die Bearbeitung von Policies Inkonsistenzen oder Konflikte entstehen. Eine weitere Aufgabe dieser Komponente ist es, die vom Administrator eingegebenen Gültigkeitsperioden einer Policy in verschiedene PolicyTimePeriodCondition Objekte aufzulösen und diese je nach Wunsch auch in UTC zu übersetzen.

2.3.2 Policy Repository

Zur Ablage der mit dem *Policy Management Tool* erzeugten Policies, wird das *Policy Repository* verwendet. Diese Komponente stellt die tatsächliche physische Datenablage für Policies dar und sollte nicht mit der von PCIM definierten Klasse PolicyRepository verwechselt werden, die lediglich als logisches Konstrukt dient, um PolicyConditions und PolicyActions aufzunehmen. In PCIMe wurde die Klasse aus Gründen der Verwechselbarkeit deshalb in ReusablePolicyContainer umbenannt.

Da im Verhältnis mehr lesende als schreibende Zugriffe auf ein *Policy Repository* erfolgen, übernimmt aus Effizienzgründen in der Praxis meist ein Verzeichnisdienst diese Aufgabe, obwohl auch jede andere Art der Datenspeicherung möglich wäre. Verzeichnisdienste sind für diese Aufgabe prädestiniert, da sie für den Einsatz bei überwiegenden Leseoperationen konzipiert und optimiert wurden.

2.3.3 Policy Decision Point – PDP

Der PDP ist die Komponente, die anhand der vorliegenden Policies eine Entscheidung treffen muss, welche Aktionen durchzuführen sind.

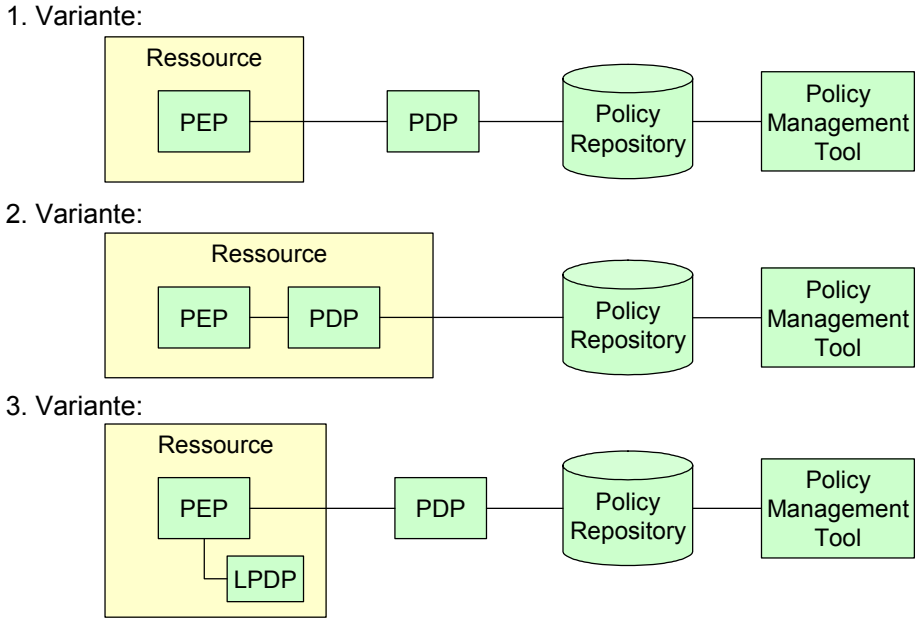
2.3.4 Policy Enforcement Point – PEP

Die Aufgabe des PEP ist es, die vom PDP getroffene Entscheidung auf der Ressource umzusetzen.

2.3.5 Modellierungsvarianten und Kommunikationsablauf

Im Folgenden wird der Ablauf der Kommunikation zwischen den Komponenten PEP und PDP kurz dargestellt. Der PEP erhält eine Nachricht, dass eine Policy-Entscheidung benötigt wird. Wenn eine solche Nachricht den PEP erreicht, formuliert dieser eine Policy-Anfrage und schickt diese an den PDP. Dabei können neben den Daten, die eine Policy-Entscheidung ausgelöst haben, auch mehrere beliebige Policy-Elemente zusätzlich an den PDP übermittelt werden. Die vom PDP getroffene Entscheidung wird – evtl. auch um zusätzliche Policy-Elemente angereichert – an den PEP zur Ausführung übermittelt und entweder mit einer Erfolgs- oder Fehlermeldung bestätigt.

Für die konkrete Lokalisierung von PDP und PEP gibt es verschiedene Modellierungsvarianten, die in Information 14 dargestellt sind.



Information 14: Modellierungsvarianten der Komponenten PDP und PEP

Variante 1 zeigt einen dezentralisierten Ansatz, bei dem es einen zentralen PDP getrennt von den Ressourcen gibt. Dieser Ansatz ist für Szenarien geeignet, bei denen viele Ressourcen existieren, die global verwaltet werden können, ohne dass lokale Unterschiede berücksichtigt werden müssen.

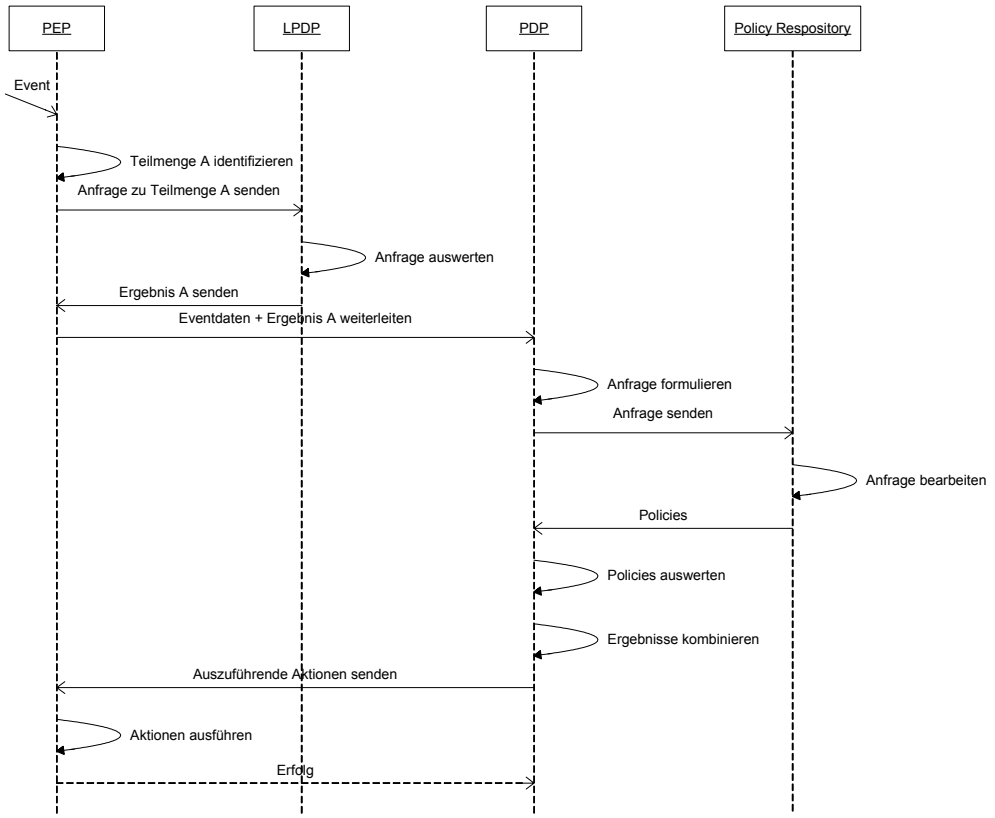
Ein ressourcenzentralisierter Ansatz wird in Variante 2 dargestellt. In diesem Szenario gibt es viele verschiedene Ressourcen, bei denen lokale Gegebenheiten und Anforderungen sehr unterschiedlich und bei einer Policy-Entscheidung zu berücksichtigen sind. Unter diesen Gegebenheiten ist es sinnvoller, den PDP in die jeweiligen Ressourcen zu verlagern, da ein zentralisierter PDP durch die vielen unterschiedlichen Ressourcen nicht effizient arbeiten könnte.

Ein Kompromiss zwischen den ersten beiden Varianten wird von Variante 3 verfolgt. Hier gibt es Ressourcen, die einen lokalen PDP (LPDP) verwenden, der zunächst die Anfrage des PEP bearbeitet. Nachdem der lokale PDP seine Entscheidung formuliert und an den PEP übermittelt hat, wird diese gemeinsam mit der ursprünglichen Anfrage an den zentralen PDP weitergeleitet. Dieser trifft dann seinerseits eine Entscheidung

und übermittelt diese zur Ausführung an den PEP. In der Praxis erweist sich dieser Ansatz als der realistischste, da die Vorteile einer lokalen und einer zentralen Entscheidung kombiniert werden. Einerseits wird der zentrale PDP durch die lokalen PDPs entlastet, andererseits ist es dem zentralen PDP vorbehalten, die endgültige Entscheidung zu fällen.

2.3.6 Ablauf der Kommunikation

Der Ablauf der Kommunikation in Variante 3 erfolgt wie in Information 15 dargestellt.



Information 15: Kommunikationsablauf bei einer Policy-Entscheidung

Dabei teilt sich der Ablauf in 4 Stufen ein:

1. Durch ein lokales Ereignis oder eine Nachricht wird vom PEP eine Policy-Entscheidung gefordert. Dazu formuliert dieser eine Anfrage bestehend aus der Meldung oder dem Ereignis und evtl. zusätzlichen Policy-Elemente.
2. Der PEP kann eine lokale Konfigurationsdatenbank befragen, um eine Menge an Policy-Elementen zu identifizieren (Menge A), die lokal ausgewertet werden müssen. Die lokale Konfiguration legt dabei fest, welche Typen der Policy-Elemente lokal ausgewertet werden müssen. Vom PEP wird die Anfrage mit der Menge A an den lokalen PDP (LPDP) weitergeleitet und das Ergebnis (auch als Teilresultat E(A) bezeichnet) eingesammelt.
3. Nun wird die Anfrage mit allen Policy-Elementen und dem Teilergebnis E(A) vom PEP an den PDP übermittelt. Anhand der Anfrage trifft der PDP auch seinerseits eine Entscheidung (E(Q)) und kombiniert diese mit der Entscheidung E(A) des LPDPs zu einer endgültigen Entscheidung.

4. Vom PDP wird die durch Kombination von E(Q) und E(A) erreichte endgültige Entscheidung an den PEP übermittelt.

Der PEP muss den PDP auch befragen, falls keine Policy-Elemente zur bestehenden Anfrage existieren. Durch diesen Ablauf wird sichergestellt, dass der PDP immer kontaktiert und nicht übergangen wird. Ein Übergehen des PDPs ist nur dann zulässig, wenn die Anfrage selbst schon vom lokalen PDP abgelehnt wird. Allerdings muss der PDP trotzdem von dieser Entscheidung in Kenntnis gesetzt werden. Auch in Kenntnis gesetzt werden muss der PDP davon, falls es dem PEP nicht möglich ist – beispielsweise wegen mangelnder Kapazität – die vom PDP gewünschten Operationen durchzuführen. Dem PDP ist es jederzeit möglich, durch asynchrone Nachrichten an den PEP eine vorhergehende Entscheidung zu ändern oder eine Fehler- oder Warnmeldung zu erzeugen.

2.4 VPN-Grundlagen

Dieses Kapitel liefert eine Einführung in die grundlegenden Komponenten des VPN-Dienstes. Wie in Information 1 zu sehen ist, umfasst der VPN-Dienst im Wesentlichen die drei Komponenten IPsec, *Layer 2 Tunneling Protocol* (L2TP) und *Point-to-Point Protocol* (PPP). In den folgenden Abschnitten werden diese Komponenten vorgestellt.

2.4.1 Aufgaben von IPsec

IPsec wurde im November 1998 von der IETF in RFC 2401 standardisiert, um die dem Internetprotokoll (IP) fehlenden Funktionen zur Authentifikation, Vertraulichkeit und Integrität bereitzustellen. Der Standard liegt mittlerweile in einer am Dezember 2005 überarbeiteten Fassung in RFC 4301 vor. Die wesentlichen drei Komponenten von IPsec sind *Internet Key Exchange* (IKE), *Authentication Header* (AH) und *Encapsulating Security Payload* (ESP). Die Komponenten werden im Folgenden näher beschrieben.

Schlüsselmanagement durch Internet Key Exchange (IKE)

Das IKE Protokoll dient der Schlüsselverwaltung, bevor die eigentlich verschlüsselte Verbindung beginnt. Dazu arbeitet IKE in zwei Phasen:

1. Verhandlung einer *Security Association* (SA) über den *Main Mode* oder den *Agressive Mode*
2. Erzeugung einer SA für IPsec durch den *Quick Mode*

In einer SA wird zwischen den Kommunikationspartnern folgendes vereinbart:

1. Identifizierung (über PSK oder Zertifikat)
2. Wahl des Schlüsselalgorithmus für die IPsec-Verbindung
3. Identifikation des Quellnetzes
4. Identifikation des Zielnetzes
5. Wiederholungszeitraum für die Authentifizierung
6. Erneuerungszeitraum der IPsec-Schlüssel

Der *Main Mode* aus Phase 1 läuft in folgenden Schritten ab:

1. Der Client gibt eine Liste unterstützter Verschlüsselungsalgorithmen an
2. Der Server wählt den sichersten Algorithmus anhand der verfügbaren und unterstützten Algorithmen aus
3. Der Client sendet den öffentlichen Teil seines Diffie-Hellman-Schlüsselpaars und eine Zufallszahl
4. Der Server sendet den öffentlichen Teil seines Diffie-Hellman-Schlüsselpaars und eine Zufallszahl
5. Nun folgt die Authentifizierung durch PSK oder Zertifikat auf der durch Diffie-Hellman-Schlüssel gesicherten Verbindung

Beim *Agressive Mode* erfolgt die Authentifizierung im Klartext und ist deshalb als unsicher einzustufen.

Zur Authentifizierung werden entweder das *Pre Shared Keying* (PSK) oder Zertifikate verwendet. Beim PSK Verfahren wird ein einziges gemeinsames Geheimnis – üblicherweise ein Passwort – verwendet. Werden Zertifikate zum Authentisieren verwendet, wird mittels X.509-Zertifikaten geprüft, ob die eingetragene CA dieses Zertifikat als gültig anerkennt.

Der in Phase 2 verwendete Quick Mode sorgt für die Erzeugung der SA für die IPsec-Verbindung. Hier wird auch sichergestellt, dass eine SA nicht wieder verwendet bzw. aus einer alten SA nicht auf eine neue SA geschlossen werden kann.

Authentizität durch Authentication Header (AH)

Vom AH wird die Authentizität der übertragenen Pakete sichergestellt und gegen Wiederholung geschützt. Lediglich Paketfelder, die auf dem Weg durch das Internet geändert werden müssen, bleiben ungeschützt.

Verschlüsselung durch Encapsulating Security Payload (ESP)

Zum Schutz der Vertraulichkeit, Authentizität und Integrität der übertragenen Daten wird ESP eingesetzt. Im Gegensatz zu AH dient es dazu, nicht den Paketkopf, sondern den Nutzdateninhalt zu schützen.

Übertragungsmodi von IPsec

Von IPsec werden zwei verschiedene Übertragungsmodi spezifiziert. Im Transportmodus wird lediglich das IP-Paket um die Ipsec-Information ergänzt, während im Tunnelmodus das komplette IP-Paket in die Nutzdaten eines IPsec-Pakets aufgenommen wird.

2.4.2 Aufgaben von L2TP

Das L2TP wird als Protokoll bei VPN verwendet und wurde von der IETF in RFC 2661 spezifiziert. L2TP ermöglicht es, Tunnel auf der Sicherungsschicht (Schicht 2) des OSI-Modells aufzubauen. Zur Authentifizierung kann entweder das *Challenge Handshake Authentication Protocol* (CHAP) oder das *Password Authentication Protocol* (PAP) verwendet werden. Da L2TP selbst keine Verschlüsselungsmechanismen verwendet, sollte es aus Sicherheitsgründen in Kombination mit IPsec verwendet werden (RFC 3193).

2.4.3 Aufgaben von PPP

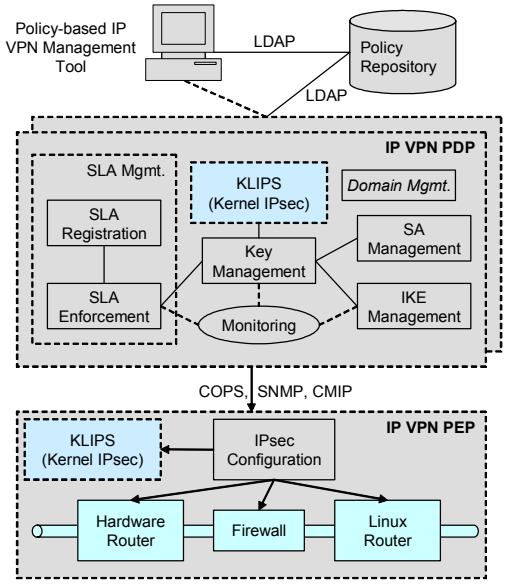
Das *Point-to-Point* Protokoll (PPP) wird zum Verbindungsaufbau über dynamisch aufgebaute Wählleitungen genutzt und ist von der IETF in RFC 1661 definiert. Um den Benutzer zu authentifizieren und ihm eine IP-Adresse zuzuweisen, wird das RADIUS-Protokoll (RFC 2865) verwendet.

3 STAND DER TECHNIK

In diesem Kapitel werden die in den Grundlagen vorgestellten Ansätze betrachtet und bewertet. Dabei werden generelle Defizite der Ansätze aufgezeigt und untersucht.

3.1 Policy-basiertes Netzwerkmanagementsystem für IP VPN [GY+03]

In diesem Ansatz wird versucht, dass von der IETF in RFC2585 spezifizierte *IPsec Configuration Policy Information Model* [JR+03] auf eine höhere Ebene zu heben, um es zu ermöglichen, auch in größeren Netzwerken eine automatische Managementanwendung durch Policies zu realisieren. Information 16 zeigt die in diesem Ansatz verwendete Managementarchitektur.



Information 16: Managementarchitektur für einen VPN-Dienst nach [GY+03]

Der Tragfähigkeitsnachweis dieses Ansatzes wurde in einer Testumgebung vollzogen und hatte die Aufgabe, anhand eines eingehenden Verbindungswunsches die Konfiguration der Tunnelendpunkte vorzunehmen.

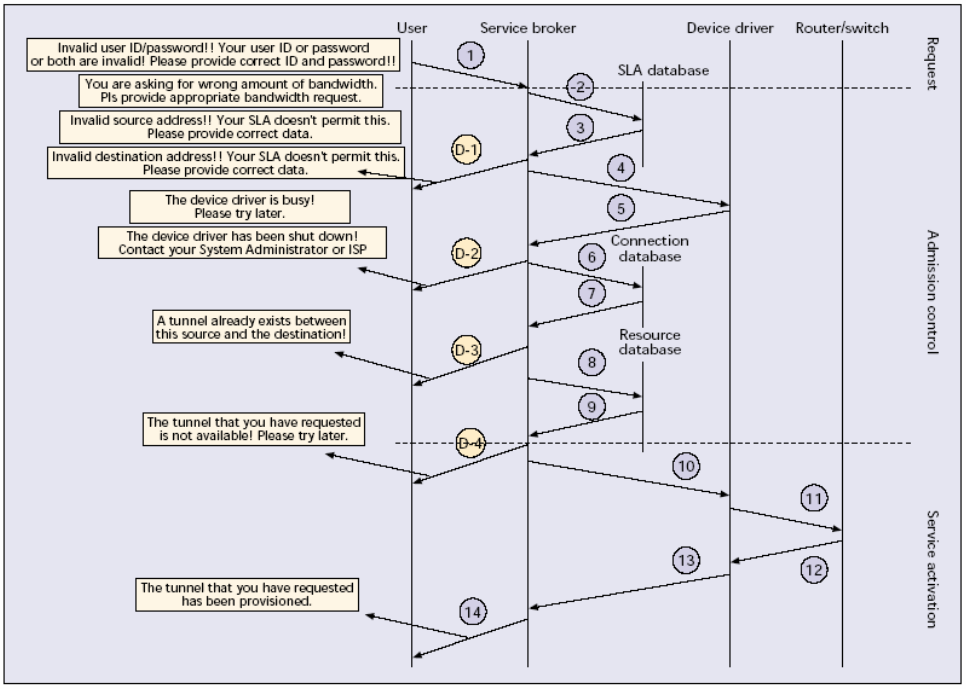
Leider werden sehr wenige Details über die prototypisch implementierte Testumgebung vermittelt. Der Ansatz bezieht sich ausschließlich auf das IPsec-Protokoll – insbesondere auf die *Internet Key Exchange* (IKE) Komponente – und berücksichtigt dabei nicht, dass für die Implementierung eines Einwahl-VPN-Dienstes noch weitere Komponenten – wie das *Layer 2 Tunneling Protocol* (L2TP) und das *Point-to-Point Protocol* (PPP) – nötig sind. Zur Sicherstellung der Verfügbarkeit des VPN-Dienstes ist deshalb das *IPsec Configuration Policy Information Model* [JR+03] nicht geeignet, da es sich – wie dieser Ansatz hier deutlich zeigt – insbesondere auf die IKE-Komponente konzentriert.

3.2 Management von QoS-fähigen VPN-Diensten [BG+01]

Der hier betrachtete Ansatz beschäftigt sich mit dem Management von QoS-fähigen VPN-Diensten unter Verwendung des *Telecommunications Management Network* (TMN) Modells. Auch hier soll das Management von der Netzwerkebene auf ein

höheres Niveau, wie der Dienst- und Geschäftsebene, angehoben werden. Dazu wurde ein *Service Broker* (SB) prototypisch implementiert, der es erlaubt, mittels Policies beim Verbindungsauf- und -abbau Bandbreiten zu reservieren und Abrechnungsinformationen aufzuzeichnen.

Das Sequenzdiagramm in Information 17 stellt den Informationsfluss zwischen den Komponenten dar.



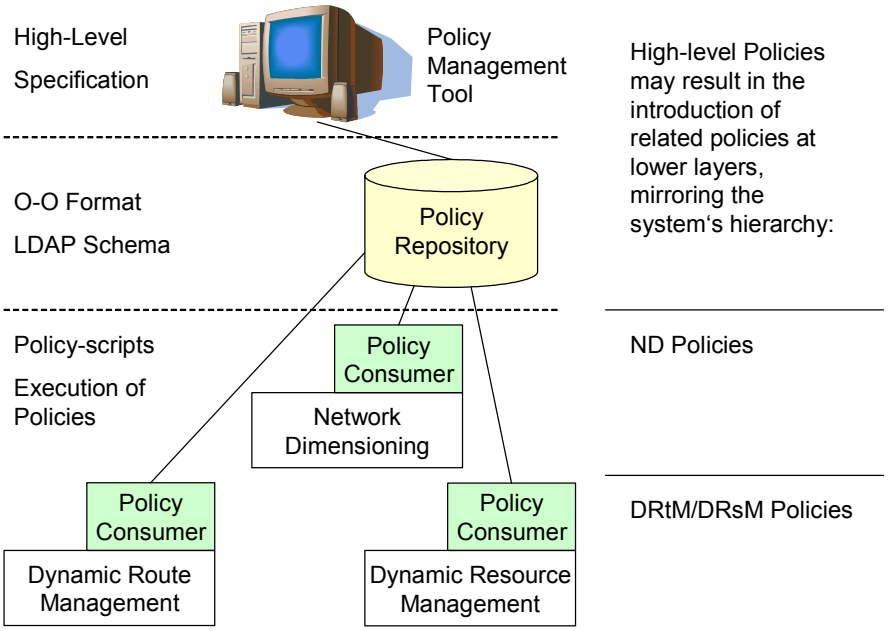
Information 17: VPN-Einwahlprozedur nach [BG+01]

An diesem Ansatz ist zu bemängeln, dass er lediglich Managementfunktionalität zum Zeitpunkt des Verbindungsauf- bzw. -abbaus vorsieht. Es findet keine Überwachung des Dienstes während der Nutzung statt.

3.3 Design und Implementierung einer policy-basierten Ressourcen-Managementarchitektur [FT+03]

Der hier vorgestellte Ansatz entwirft und implementiert eine Policy Architektur, um den Einsatz einer Netzwerkdimensionierungskomponente zu ermöglichen. Dabei geht es vorwiegend darum, Verbindungen gezielt über bestimmte Netzwerkkomponenten aufzubauen und Lastverteilung für das gesamte Netz vorzunehmen. In diesem Ansatz geht es zwar nicht direkt um einen VPN-Dienst, allerdings orientiert sich der Ansatz sehr stark an den Policy-Standards, die von der IETF spezifiziert wurden.

Information 18 zeigt die Architektur, die in diesem Ansatz umgesetzt wurde. Die Komponente „Policy Consumer“ vereint dabei PDP und PEP auf der Ressource und entspricht damit der 2. Modellierungsvariante aus Kapitel 2.3.5.



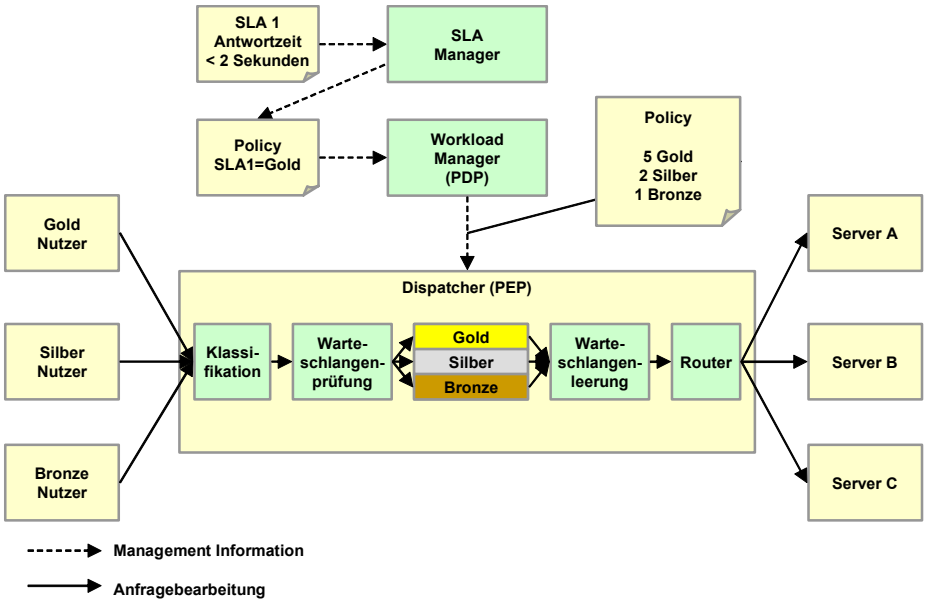
Information 18: Policy-basiertes Ressourcen-Managementsystem

3.4 Defizite der verschiedenen Ansätze auf IETF-Grundlage

Die hier vorgestellten Ansätze betrachten lediglich stark nutzerbezogene Teilaspekte eines VPN-Dienstes und setzen dabei immer die Verfügbarkeit des Dienstes und seiner Ressourcen voraus. Mit dieser Arbeit soll mittels Policies die reine Verfügbarkeit des VPN-Dienstes der ATIS gewährleistet werden, und zwar unabhängig davon, ob ein Nutzer sich anmeldet oder nicht. Probleme einzelner am VPN-Dienst beteiligter Ressourcen sollen erkannt und wenn möglich automatisch behoben werden.

3.4.1 Typisches Einsatzszenario der IETF Policy-Architektur

Ein bisher typisches Einsatzgebiet der IETF Policy-Architektur wird in Information 19 dargestellt.



Information 19: Typisches Einsatzgebiet der IETF Policy-Architektur nach [KL04]

Hier werden Datenpakete von Nutzern klassifiziert und mit entsprechender Priorität bearbeitet, je nachdem, ob der Nutzer eine hohe (Gold) oder niedrige (Bronze) Dienstqualität mit dem Dienstleister vereinbart hat. Um die eingehenden Pakete zu klassifizieren, können beispielsweise die Quell- bzw. Zieladressen der Datenpakete verwendet werden. Ist ein Paket klassifiziert, kann es der zugehörigen Warteschlange zugeordnet werden, damit eine der vereinbarten SLA entsprechende Verarbeitung erfolgen kann.

Policies bilden in diesem Szenario das zentrale Element. Sie enthalten die Bedingungen, welche IP-Adressen einer bestimmten Dienstgütekategorie zuzuordnen sind sowie die zugehörigen Aktionen zur Konfiguration der Warteschlangen und Router. Im Folgenden wird der konkrete Ablauf dieses Vorgangs kurz beschrieben. Trifft beim PEP ein Paket ein, wird die IP-Adresse des Pakets an den PDP weitergeleitet. Der PDP findet zur IP-Adresse passende Policies und übermittelt alle auszuführenden Aktionen an den PEP. Durch den PEP werden diese Aktionen ausgeführt, indem er die Konfiguration der verschiedenen Komponenten vornimmt.

An diesem Beispiel kann man deutlich erkennen, dass Policies dazu verwendet werden, um eine Priorisierung von eingehenden Datenpaketen vorzunehmen. Damit aber eine Überwachung der Ressourcen vorgenommen werden kann, muss eine Monitor-komponente verwendet werden, die in dieser Architektur fehlt.

3.5 Defizite der IETF-Standards

3.5.1 Policy-Spezifikation

Rollenzuordnung von Ressourcen

In PCIM wurde durch das Modell keine Rollenzuweisung von Ressourcen ermöglicht. Es sollte vermieden werden, dass irgendwo innerhalb des Modells eine Zuweisung von sämtlichen Ressourcen zu Policies vorgenommen werden muss. PCIMe macht dies nun

möglich, und es müssen innerhalb des Modells allen Ressourcen Rollen zugeordnet werden.

Fehlende Instanzen der Klasse PolicyVariable und PolicyValue

Um die Verfügbarkeit eines Dienstes sicherzustellen, muss der PCIMe-Standard noch um spezielle Klassen von PolicyVariable und PolicyValue erweitert werden. Nur durch die Entwicklung von spezialisierten Unterklassen kann ein einheitliches Management der Verfügbarkeit von Diensten durchgeführt werden. Die Klassen müssen entsprechend der Anforderungen aus Kapitel 4.3.1 und 4.3.2 definiert werden, um beispielsweise Versionsnummern oder Prozessnamen von Komponenten aufzunehmen.

Hohe Komplexität des Modells

Durch die hohe Generizität von PCIM wird das Modell äußerst mächtig und birgt ein hohes Potential an Komplexität in sich. Mit dem Modell lassen sich komplexe Konstrukte erstellen und miteinander kombinieren, wodurch schnell der Überblick verloren gehen und eine einfache Handhabbarkeit der Policies evtl. nicht mehr gewährleistet werden kann. Es ist deshalb empfehlenswert, die Komplexität des Modells durch Einschränkungen im *Policy Management Tool* für das jeweilige Einsatzszenario zu reduzieren.

3.5.2 IETF-Architektur

Fehlende Architekturkomponenten

Die in Kapitel 2 beschriebene Policy-Architektur bezieht sich auf das Einsatzszenario „ressourcenreservierende Zugangskontrolle“ – wie beispielsweise IntServ unter Verwendung des *Resource reSerVation Protocol* (RSVP) zum Aufbau von Reservierungen – und muss für den Einsatz zur Betriebsüberwachung eines VPN-Dienstes angepasst werden.

Im RSVP-Szenario wird eine Policy-Entscheidung immer dann ausgelöst, falls ein Nutzer den Dienst in Anspruch nehmen will. Für die Verfügbarkeitsprüfung des VPN-Dienstes sollen die Ressourcen aber ständig überwacht werden, dazu wird eine Monitorkomponente benötigt. Werden von der Monitorkomponente ungewöhnliche Werte von Ressourcenparametern gemessen, müssen diese ungewöhnlichen Werte von einer Analysierungskomponente untersucht werden, um herauszufinden, ob es sich um einen Fehler handelt und was diesen Fehler verursacht.

Da die IETF-Architektur bisher nur auf Netzwerkebene eingesetzt wurde, waren diese Komponenten überflüssig, da lediglich einfache Vergleiche von beispielsweise IP-Adressen vorgenommen werden mussten. Bei der Überwachung der Verfügbarkeit des VPN-Dienstes gilt es jedoch, komplexere Parameter zu überwachen und zu analysieren, um festzustellen, ob ein Fehlerzustand eingetreten ist. Wenn der Fehler und die Fehlerquelle gefunden wurde, kann mit der IETF-Architektur die Behebung des Fehlers durch Policies durchgeführt werden.

Völlige Freiheit bei der Implementierung

Eine ungenaue Spezifikation der Komponenten führt zu einem Verschwimmen der Grenzen zwischen den Komponenten. So wird beispielsweise nicht näher darauf eingegangen, wie oder wodurch der PEP eine Nachricht erhalten soll. Im Falle der Überwachung von Ressourcenparametern ist eine Überwachungskomponente erfor-

derlich, die bisherige Überwachung von Datenpaketen im Netzwerk wurde vom PEP übernommen. Die Überwachung der Ressourcenparameter ist eine komplexe Aufgabe und hat mit der eigentlichen Aufgabe des PEP – die vom PDP übermittelten Aktionen auszuführen – nichts zu tun. Für dieses Szenario ist eine Überwachung bzw. Monitorkomponente notwendig, die von der IETF nicht modelliert wird.

Zur Implementierung der beschriebenen Komponenten liefert die IETF keine weiteren Vorgaben. So bleibt es dem Entwickler überlassen, welche Algorithmen er im PDP dazu verwendet, um eine Policy-Entscheidung durchzuführen. Diese Freiheit führt zu vielen unterschiedlichen Lösungen für die in einer Komponente verwendete Logik.

3.5.3 Zusammenfassung der Defizite der IETF-Standards

Insgesamt eignen sich die IETF-Standards hervorragend, um auf niederer Netzwerkebene policy-basiertes Management zu betreiben. Um jedoch policy-basiertes Management auf Serviceebene anzuheben, sind etliche Anpassungen der Architektur nötig. So muss zum Beispiel eine Monitorkomponente entworfen werden, die Ressourcenparameter erfasst.

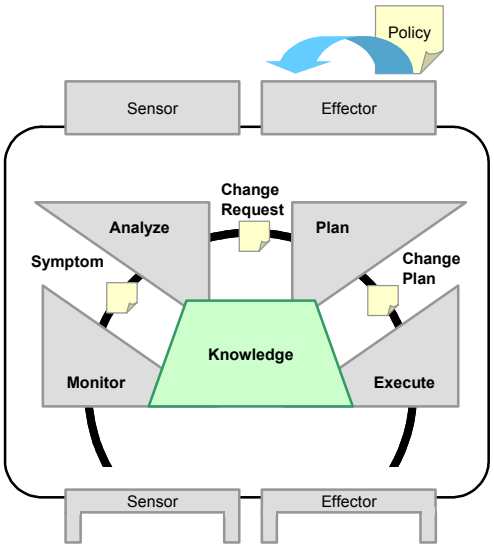
Die hohe Komplexität des gesamten Modells muss zunächst einmal ausgeblendet werden, um schrittweise an die Anforderungen des neuen Einsatzszenarios heranzugehen. Durch die hohe Generizität des Modells lassen sich relativ einfach benötigte Anpassungen vornehmen und die erforderlichen Policies spezifizieren. Ein großes Manko bei der Formulierung von Policies ist das Fehlen entsprechender Standards für Ressourcenparameter, wie sie für die Netzwerkebene bereits existieren.

Abschließend können die IETF-Standards als gute Grundlage bezeichnet werden, um an die Herausforderungen dieses neuen Szenarios heranzugehen. Jedoch muss noch einiges an Forschungsarbeit geleistet werden, bis policy-basiertes Management von Diensten produktiv eingesetzt werden kann, da es in diesem Szenario nicht an komplexen Fragestellungen mangelt.

3.6 Architekturentwurf für Autonomic Computing [IBM04]

IBM beschreibt mit diesem Entwurf eine Architektur, die *Autonomic Computing* ermöglichen soll. Der Begriff *Autonomic Computing* ist von IBM definiert und bedeutet, dass Managementaufgaben an die technische Infrastruktur delegiert werden. Diese so genannten *Self-Managing* Ansätze werden in die Bereiche *Self-configuring*, *Self-healing*, *Self-optimizing* und *Self-protecting* gruppiert (auch als *Self-CHOP* bezeichnet) und fokussieren jeweils verschiedene Managementziele (Konfiguration, Fehlerbehebung, Optimierung, Sicherheit).

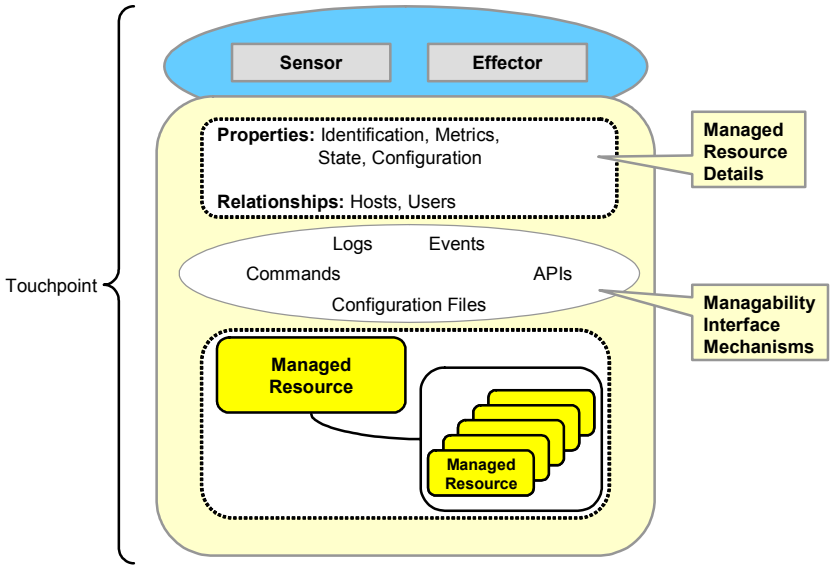
Im Zentrum des Entwurfes steht dabei ein Regelkreis (*Control Loop*), der in einer Komponente namens *Touchpoint Autonomic Manager* (s. Information 20) angesiedelt ist. Zur Kommunikation mit anderen Komponenten gibt es eine einheitliche Sensor- und Effector-Schnittstelle, die zur Abfrage gemessener Werte bzw. zur Änderung des Zustandes dient. Beide Schnittstellen sollen sowohl Push- und Pull-Techniken unterstützen, also sowohl ein gezieltes Abrufen der Schnittstelle (Pull) als auch ein eigenständiges Erzeugen von Meldungen, wenn bestimmte Ereignisse eintreten (Push).



Information 20: Control Loop eines Touchpoint Autonomic Managers

Der Regelkreis des *Autonomic Managers* besteht im Wesentlichen aus 4 Aktionen. Der Monitor erfasst zunächst die erforderlichen Ressourcenparameter und filtert bzw. korreliert diese gegebenenfalls. Stellt der Monitor dabei ein Symptom fest, übermittelt er dieses an die Analysekomponente. In der Analysekomponente wird dieses Symptom analysiert und entschieden, ob der Zustand der Ressource fehlerhaft und eine Änderung erforderlich ist. Die Planungskomponente verarbeitet die von der Analysekomponente geforderte Änderung und ermittelt, welche Schritte auszuführen sind, damit die gewünschte Änderung in Kraft tritt. Von der Ausführungskomponente werden die von der Planungskomponente ermittelten Aktionen dann letztendlich ausgeführt. Die ausgeführten Änderungen äußern sich in Änderungen der Ressourcenparameter, die wiederum von der Monitorkomponente beobachtet werden können, und damit schließt sich der Regelkreis.

Auf der Ressource selbst arbeitet nach dem Architekturentwurf von IBM ein *Managed Resource Touchpoint*. Der Entwurf dieser Komponente wird in Information 21 gezeigt.



Information 21: Architektur eines Managed Resource Touchpoint

Der *Managed Resource Touchpoint* besitzt wie auch der *Autonomic Manager* eine Sensor- und Effector-Schnittstelle, über die sich Ressourcenparameter wie, Zustand, Konfiguration und ähnliches abfragen und steuern lassen. Um diese Werte abzufragen bzw. zu ändern, bedarf es verschiedener ressourcenspezifischer Instrumentierung. Die Instrumentierung legt dabei fest, wie beispielsweise die Auswertung einer Logdatei bzw. die Änderung einer Konfiguration erfolgen muss. Von essentieller Wichtigkeit ist es, dass Ressourcen zwar unterschiedlich instrumentiert werden, aber eine einheitliche Sensor- und Effector-Schnittstelle zur Verfügung stellen. Nur so ist es möglich, auch mit vielen verschiedenen heterogenen Komponenten ein autonomes Management zu realisieren.

Policies spielen in diesem Entwurf eine ganz zentrale Rolle, da durch die verschiedenen Policies das Verhalten des gesamten Managements bestimmt wird. Wunschvorstellung ist es, Geschäftsziele in Form von Policies zu definieren, während sich die technische Infrastruktur vollständig autonom konfiguriert und anpasst, um eben diese Ziele zu erreichen. Um diese Vision zu realisieren, ist jedoch noch einiges an Forschung notwendig. IBM selbst beschreibt den Weg zum *Autonomic Computing* (AC) als evolutorischen Prozess, der sich zunächst von manuell getriebenem Management bis zum vollständig automatischen Management anhand definierter Geschäftsziele entwickelt. Der Entwurf von IBM stellt im Ansatz einen großen Schritt in Richtung des automatischen Managements dar. Doch IBM weist in diesem Entwurf auch deutlich darauf hin, dass bis heute zwar einige Standards existieren, mit denen Teile der Architektur realisiert werden können, aber bisher noch kein einheitlicher Standard für ein vollständig automatisches Management existiert.

3.7 Defizite des IBM-Ansatzes

In dem Entwurf von IBM werden erste wichtige Grundlagen für den Weg zum AC gelegt, jedoch ist es nur ein Architekturentwurf. Das bedeutet, dass viele Sachverhalte nur vage definiert werden und lediglich ein initiales Grundkonzept geliefert wird. Der Architekturentwurf stellt kein Patentrezept zur Lösung des Problems dar, gibt aber wertvolle Hinweise, wie eine mögliche Lösung aussehen könnte.

So fehlen auch hier Standarddefinitionen für Ressourcenparameter von Softwarekomponenten, die Prozessnamen, Versionsnummern oder Konfigurationsdateien enthalten. Der Architekturentwurf ist sehr abstrakt gehalten und vermittelt noch keine verbindlichen Komponenten- bzw. Schnittstellendefinitionen, obwohl bereits erste Tools und Softwareprodukte von IBM erhältlich sind, die Teilfunktionalitäten des *Autonomic Computing* realisieren.

Alles in allem liefert der Entwurf einen guten Leitfaden zur möglichen Lösung des Problems AC. Die grobe Beschreibung des Ansatzes wirft jedoch auch einige Fragen auf, die im Rahmen dieser Arbeit thematisiert werden.

4 ANALYSE DES VPN-DIENSTES

Dieses Kapitel widmet sich der Einführung und Analyse des VPN-Dienstes. Dazu wird zunächst die Architektur des Dienstes analysiert, um beteiligte Ressourcen und Parameter zu identifizieren, für die im folgenden Kapitel dann Policies definiert werden sollen.

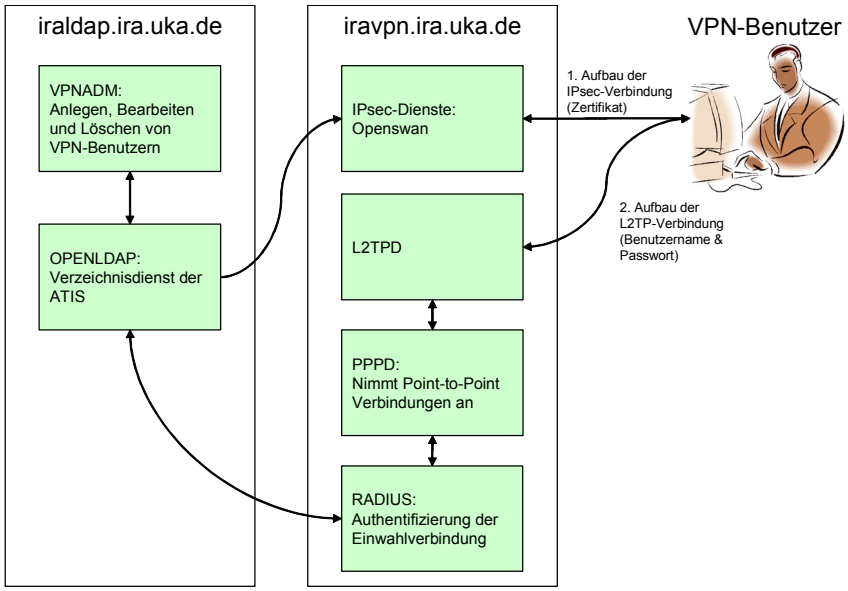
Bei IT-Diensten spielt das Abstraktionsniveau des Beobachters eine wichtige Rolle, denn aus Kundensicht stellt sich ein Dienst nur durch die Eigenschaften dar, die an seinem Dienstzugangspunkt (*Service-Access-Point*, kurz SAP) zu beobachten sind. Aus Betreibersicht resultiert ein IT-Dienst jedoch aus dem Zusammenspiel unterschiedlicher Ressourcen, durch die der Dienst funktional und qualitativ erbracht wird. Damit sich Aussagen darüber treffen lassen, wie stark einzelne Ressourcen die Funktionalität und Qualität eines Dienstes beeinflussen, ist ein Verständnis über die Abhängigkeiten der Ressourcen untereinander sehr wichtig.

In dieser Arbeit werden Policies erstellt, die den Betrieb des VPN-Dienstes der ATIS auf Ressourcenebene sicherstellen. Dazu wurde in Kapitel 2 betrachtet, wie eine Policy PCIM-konform formuliert und wie eine derartig instanzierte Policy in der von der IETF vorgeschlagenen Architektur verarbeitet wird. Eine genaue Betrachtung darüber, wie die Ressourcen des VPN-Dienstes mit dem Dienst selbst in Beziehung stehen, wird nun in diesem Kapitel vorgenommen.

4.1 Architektur des VPN-Dienstes

Der VPN-Dienst der ATIS stellt den Benutzern an ihren entfernten Rechnern über eine gesicherte Tunnelverbindung eine IP-Adresse aus dem Bereich der Fakultät zur Verfügung und ermöglicht es, so zu arbeiten, als würde ein Rechner verwendet, der physikalisch an das Fakultätsnetz angebunden ist.

Zur Realisierung des Dienstes werden verschiedene Komponenten benötigt. Eine schematische Darstellung ausgewählter Komponenten und deren Beziehungen wird in Information 22 gezeigt und stellt eine Verfeinerung von Information 1 dar. Weitere nicht unmittelbar am VPN-Dienst beteiligte Komponenten, wie beispielsweise der DNS-Dienst und das Netzwerk, sind wegen der besseren Übersichtlichkeit hier nicht abgebildet.

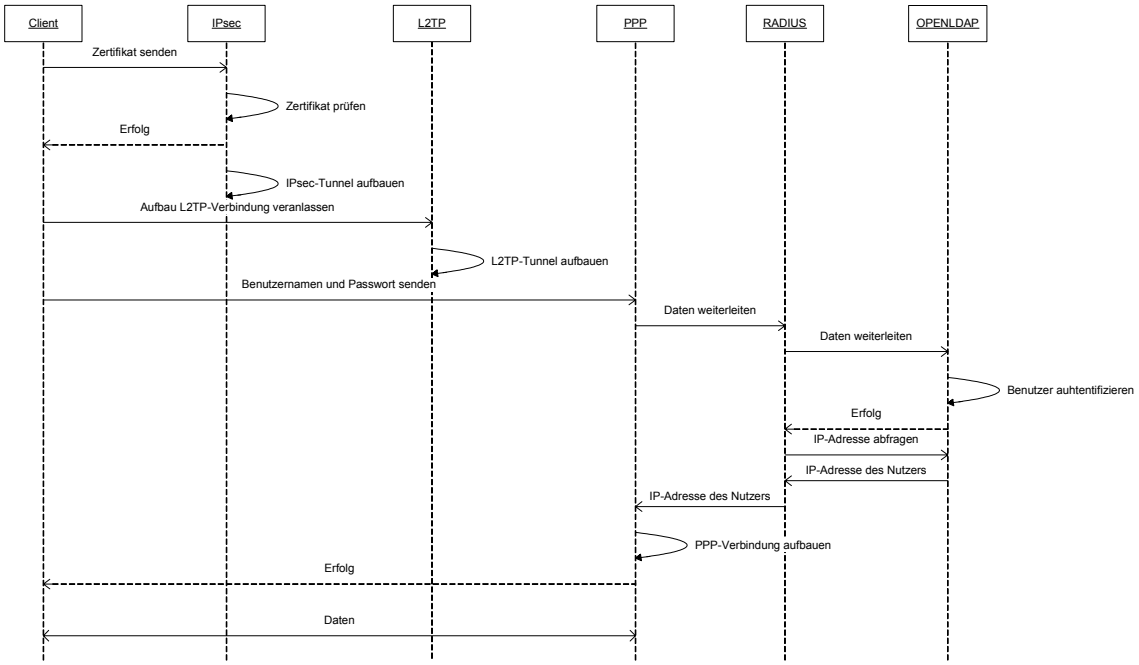


Information 22: Aufbau des VPN-Dienstes der ATIS

Der Verbindungsablauf wird in den nächsten Abschnitten genau erklärt.

4.1.1 Funktionalität der Komponenten

In diesem Abschnitt werden die im Rahmen des VPN-Dienstes der ATIS eingesetzten Software- und Systemkomponenten und der genaue Verbindungsablauf vorgestellt. Beim Verbindungswunsch des Benutzers wird zunächst eine sichere IPsec-Verbindung aufgebaut und der Nutzer anhand seines Zertifikats authentifiziert. Im aufgebauten IPsec-Tunnel wird dann in einem L2TP-Tunnel eine PPP-Verbindung aufgebaut und der Benutzer erhält anhand seines Benutzernamens und Passworts eine IP-Adresse zugewiesen. Eine Darstellung des Verbindungsablaufs folgt in Information 23.



Information 23: Sequenzdiagramm zum VPN-Verbindungsablauf

Die Zertifikate liegen im Verzeichnisdienst der ATIS und werden durch einen Cronjob einmal täglich in die Konfigurationsdateien der IPsec-Implementierung übertragen. Deshalb ist die Verbindung in Information 22 als unidirektional gekennzeichnet. Um eine PPP-Verbindung aufzubauen, wird das RADIUS-Protokoll verwendet, das den vom Nutzer angegebenen Benutzernamen und das zugehöriges Passwort an den Verzeichnisdienst weiterleitet. Der Verzeichnisdienst überprüft, ob Nutzer und Passwort übereinstimmen und liefert entweder einen Fehler oder die dem Nutzer zugeordnete IP-Adresse an das RADIUS-Protokoll zurück. Dadurch wird entweder erfolgreich eine PPP-Verbindung aufgebaut oder der Vorgang abgebrochen.

Im Folgenden wird genauer auf die einzelnen Komponenten eingegangen.

Openswan (IPsec)
Openswan stellt eine Open-Source-Implementierung von IPsec für das Betriebssystem Unix bereit. Das Projekt wurde von einigen FreeS/WAN Entwicklern ins Leben gerufen, die mit den politischen Entscheidungen rund um das FreeS/WAN Projekt nicht zufrieden waren. Das IPsec-Protokoll wird beim VPN-Dienst der ATIS dazu verwendet, einen sicheren Tunnel aufzubauen und das Zertifikat des Nutzers zu überprüfen.

L2TPD
Dieser Daemon ist eine L2TP-Implementierung für den Einsatz von VPN unter Unix- und POSIX-basierten Betriebssystemen. Dieses Projekt ist seit August 2002 nicht mehr aktiv weiterentwickelt worden und liegt aktuell in der Version 0.69 vor. Wenn eine IPsec-Verbindung mit dem VPN-Dienst der ATIS aufgebaut wurde, wird darin ein L2TP-Tunnel aufgebaut. Der Aufbau des Tunnels ist nötig, damit der PPP-Paketkopf aufgenommen werden kann. Außerdem ist L2TP dafür zuständig, die von PPP geforderte Reihenfolge der Pakete zu garantieren, was mit IPsec nicht realisiert werden kann.

PPPD
Der PPPD ist eine von Paul Mackerras entwickelte Implementierung des PPP-Protokolls für das Unix-Betriebssystem. Im bestehenden L2TP-Tunnel wird mittels PPP eine Wählverbindung aufgebaut. PPP wiederum verwendet das RADIUS-Protokoll, um den Benutzer anhand seines Namens und Passworts zu authentifizieren und die dem Benutzer zugewiesene IP-Adresse zurückzuliefern.

OpenLDAP
OpenLDAP ist eine frei verfügbare Implementierung des *Lightweight Directory Access Protocols* (LDAP) und wird von der ATIS als zentrale Benutzerverwaltung verwendet. In diesem Verzeichnis werden die Daten der VPN-Nutzer gespeichert, die zur Nutzung des VPN-Dienstes benötigt werden (Benutzername, Passwort, Zertifikat, IP-Adresse).

RADIUS
Das RADIUS-Protokoll leitet bei einem Verbindungsaufbau direkt den Benutzernamen und das Passwort des Nutzers an den LDAP-Server und dieser meldet dann entweder eine erfolgreiche Authentifizierung oder einen Fehler. Bei einer erfolgreichen Benutzerauthentifizierung liest das RADIUS-Protokoll die im LDAP hinterlegte IP-Adresse des Nutzers aus und übermittelt diese an das PPP-Protokoll.

VPNADM
Der VPNADM ist eine webbasierte Benutzeroberfläche für den LDAP-basierten Verzeichnisdienst und ermöglicht es dem Administrator VPN-Benutzer zu bearbeiten, anzulegen und zu löschen. Durch dieses Werkzeug wird eine komfortable und intuitive Nutzerverwaltung möglich, ohne dass dazu nähere Kenntnisse von LDAP erforderlich sind.

Sonstige Komponenten
Der Cronjob /etc/ipsec.d/scripts/genipsecconf.py auf dem System iravpn holt in periodischen Abständen die Zertifikate der Nutzer aus dem LDAP und schreibt sie in die lokale IPsec-Konfigurationsdatei.

4.2 Anforderungen an den VPN-Dienst

Die Anforderungen an einen Dienst werden vom Kunden und vom Betreiber vertraglich in einem SLA festgehalten. Ein SLA gibt beiden Parteien rechtliche Sicherheit, da der Kunde seine Erwartung und der Betreiber seine Leistung festlegt. Neben der funktionalen und qualitativen Beschreibung des Dienstes in Form von SLA-Parametern enthält ein SLA noch die Namen der Vertragspartner, Preisgestaltung, Gewährleistungen, Vertragsstrafen und weitere organisatorische Regelungen, die für die Fragestellung dieser Arbeit nicht weiter relevant sind.

Welche Ressourcenparameter im Einzelnen zu überwachen sind, wird durch die jeweiligen SLA-Parameter festgelegt, doch nicht alle dort getroffenen Vereinbarungen sind unmittelbar relevant für die technische Infrastruktur. Beispielsweise wird im SLA auch festgelegt, welche Erreichbarkeit der Support haben muss, doch diese Festlegung betrifft nicht unmittelbar die technische Infrastruktur, die den funktionalen Dienst realisiert. Deshalb wird die SLA noch weiter unterteilt in so genannte *Service Level Objectives* (SLOs). Sie sind Teil eines SLA und spezifizieren genaue Ressourcenparameter und operationale Informationen, die notwendig sind, um eine SLA durchzusetzen und zu überwachen. SLOs erfassen also die technischen Teile einer SLA und bestehen aus konkreten Parametern und den zugehörigen Werten. Ein typischer Parameter einer SLO ist beispielsweise die Verfügbarkeit der Infrastruktur eines Dienstes, die meist prozentual mit einem Wert beispielsweise von 99 % angegeben wird.

Tabelle 2 liefert einen Einblick, wie der VPN-Dienst im SLA beschrieben wird. Neben einer reinen Funktionsbeschreibung des Dienstes werden dort auch verschiedene Anforderungen definiert. Zunächst werden die Vertragspartner und der Dienst beschrieben, danach folgt eine Beschreibung des Dienstzugangspunkts selbst. Im Anschluss daran werden die Qualitätsparameter des Dienstes festgelegt.

VPN-Dienst	
Anbieter	ATIS
Kunde	Universitätsmitglied
Dienstbeschreibung	Zugang zum Fakultätsnetz über einen verschlüsselten Tunnel

Dienstzugangspunkt	
Bereitstellungsort	iravpn.ira.uka.de
Benutzername und Passwort	Zur Nutzung ist die Angabe eines Benutzernamens und Passworts nötig
Zertifikat	Zur Nutzung wird ein auf den Nutzer ausgestelltes vom Rechenzentrum signiertes X.509 Zertifikat benötigt
Qualitätsparameter	
Verfügbarkeit der Infrastruktur	≥ 99% (Ausfallzeit ≤ 14,4 Minuten/Tag)
Antwortzeit	≤ 1 Sekunde (Verbindungsaufbau)
Durchsatz	max. 1 Megabyte/Sekunde
…	

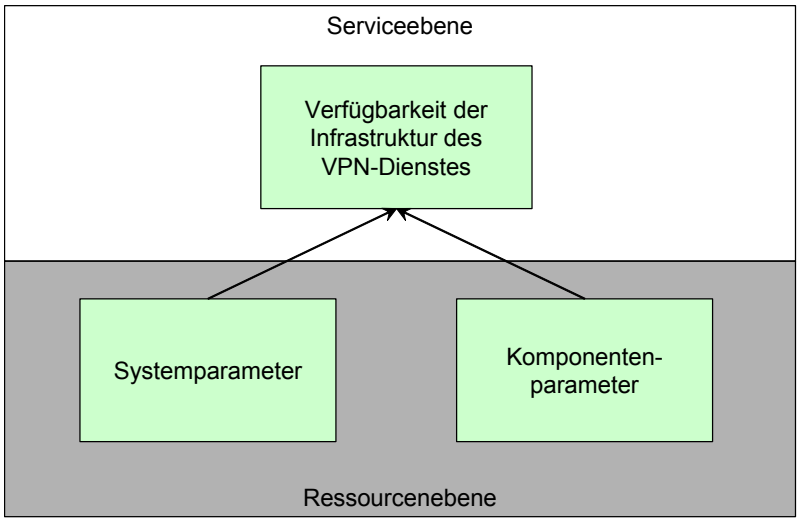
Tabelle 2: Auszug aus dem SLA zum VPN-Dienst

Die in den SLOs definierten Parameter fokussieren die externe Sicht (Kundensicht) auf einen Dienst und befinden sich auf einer sehr hohen Abstraktionsebene. Um ein Management auf Ressourcenebene durchführen zu können, müssen abstrakte Werte, wie die Verfügbarkeit der Infrastruktur auf Ressourcenparameter abgebildet werden.

4.3 Identifizierung von Ressourcenparametern

In dieser Arbeit sollen Policies dazu verwendet werden, die Verfügbarkeit der Infrastruktur des VPN-Dienstes – also der entsprechenden Rechensysteme und Komponenten – zu gewährleisten. Um das zu erreichen, wird ein Bottom-Up-Ansatz verfolgt, der zunächst auf Ressourcenebene bestimmte Parameter überwacht und entdeckte Fehlerzustände durch Policies korrigiert. Hier soll zunächst nur der Teilaspekt der Verfügbarkeit der Infrastruktur betrachtet werden, da dieser bereits genügend Komplexität auf Ressourcenebene bietet und die Grundlage für weitere QoS Fragestellungen darstellt.

In diesem Kapitel wird aufgezeigt, dass bestimmte Rechner- und Systemkomponenten des VPN-Dienstes überwacht werden müssen. Information 24 zeigt, wie die geforderte Verfügbarkeit auf der Serviceebene bestimmte System- und Komponentenparameter auf Ressourcenebene erfordert.



Information 24: Ressourcenparameter liefern Informationen über die Verfügbarkeit

Die gewünschte Verfügbarkeit ist nur dann gewährleistet, wenn sich Systeme und Komponenten in bestimmten Zuständen befinden. Solche Zustände können durch verschiedene Kennzahlen bzw. Ressourcenparameter erfasst werden. Gewünschte Systemzustände werden beispielsweise durch ein gewisses freies Kontingent an Festplattenspeicherplatz und durch eine geringe Systemlast definiert, während erwünschte Komponentenzustände durch laufende Prozesse, fehlerfreie Konfigurations- und Logdateien gekennzeichnet sind.

Die Verfügbarkeit des VPN-Dienstes kann nur dann gewährleistet werden, falls alle zugehörigen Systeme und Komponenten verfügbar sind. Ist ein System oder eine Komponente nicht verfügbar, kann in dieser Zeit auch der VPN-Dienst nicht verfügbar gewesen sein. Der Umkehrschluss gilt natürlich nicht, denn die Verfügbarkeit der Ressourcen bildet zwar die Grundlage für die Verfügbarkeit des Dienstes, gewährleistet diese jedoch nicht.

Eine andere Möglichkeit die Funktionalität des VPN-Dienstes zu überwachen ist das so genannte *Active Probing* [HS+04]. Dieses Konzept folgt einem Top-Down-Ansatz und überprüft den Dienst vollautomatisch und periodisch am Dienstzugangspunkt (SAP) genauso, als würde er von einem Kunden aufgerufen. Durch diese Art der Dienstüberprüfung ist es möglich festzustellen, ob der Dienst in seiner Gesamtheit funktionstüchtig ist. Sie liefert aber im Falle einer nicht erfolgreichen Nutzung keinerlei Hinweise über den Grund des Ausfalls. Ein solches *Active Probing* stellt eine gute Ergänzung zu der Policy-Überwachung auf Ressourcenebene dar und wird im Ausblick auf künftige Fragestellungen wieder aufgegriffen.

4.3.1 Rechensysteme

Maßgeblich verantwortlich für den Betrieb des VPN-Dienstes sind die beiden Rechensysteme „iraldap.ira.uka.de“ und „iravpn.ira.uka.de“. Vom iraldap-System werden der Verzeichnisdienst LDAP bereitgestellt, während vom iravpn-System die Dienste IPsec, L2TP, PPP und RADIUS erbracht werden. Damit der VPN-Dienst erreicht werden kann, sind natürlich auch noch diverse Netzwerkkomponenten (Router, Switches) und der DNS-Server der ATIS, der vom Rechensystem „iraun1.ira.uka.de“ bereitgestellt wird, erforderlich.

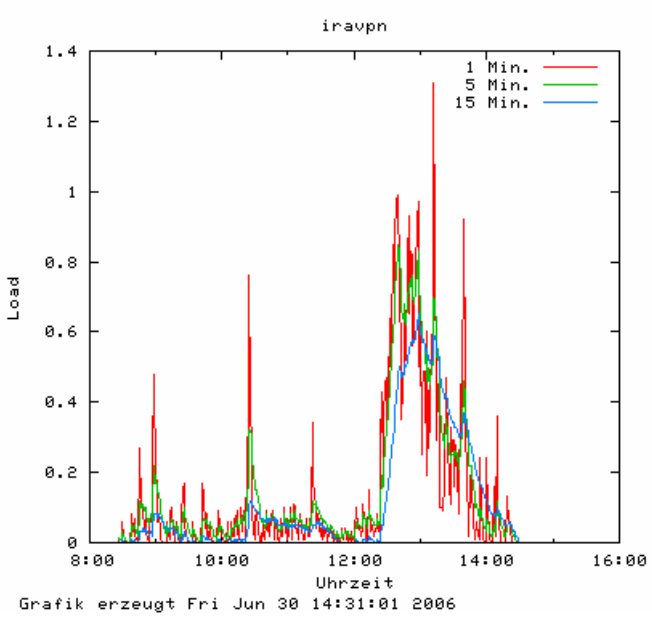
Um die Nutzbarkeit des VPN-Dienstes zu gewährleisten, müssen die beiden genannten Rechensysteme verfügbar sein. Neben der reinen Verfügbarkeit der Systeme – die beispielsweise in Form eines ping-Befehls überprüft werden kann – müssen aber auch noch andere Ressourcenparameter dieser Systeme überprüft werden, um eine fehlerfreie Funktion sicherzustellen. Weitere Ressourcenparameter, die Hinweise auf mögliche Probleme eines Rechensystems liefern können, sind beispielsweise die freie Festplattenkapazität oder die Prozessorlast des Systems.

An dieser Stelle sollen die Systemparameter identifiziert werden, die maßgeblich an der Erbringung des VPN-Dienstes beteiligt sind. Die an dieser Stelle zur Ermittlung des Systemzustandes herangezogenen Parameter erheben nicht den Anspruch auf Vollständigkeit. Trotzdem wird versucht, möglichst viele aussagekräftige Ressourcenparameter zur Zustandsermittlung zu verwenden.

Um zu prüfen, ob die Netzwerkschnittstelle des entsprechenden Systems reagiert, kann mit dem ping-Befehl gearbeitet werden. Der Befehl verwendet das *Internet Control Message Protocol* (ICMP), um die Verfügbarkeit und Reaktionszeit des Zielsystems zu überprüfen. Dazu wird ein ICMP-Echo-Request an das Zielsystem abgesendet und von diesem mit einem ICMP-Echo-Reply beantwortet. Ping misst das Zeitintervall zwischen dem Absenden der Anfrage und dem Eintreffen der Antwort und gibt dieses aus. Sofern die Antwort nicht innerhalb des angegebenen Timeouts eintrifft, wird davon ausgegangen, dass das Zielsystem nicht erreichbar ist. Mit dieser einfachen Methode kann schnell die Verfügbarkeit und Reaktionszeit der Netzanbindung des Zielsystems geprüft werden.

Die freie Festplattenspeicherkapazität bietet sich auch als Parameter an, der Hinweise über die Verfügbarkeit eines Systems geben kann. Laufen beispielsweise Logdateien über und verbrauchen den freien Speicherplatz, reagiert das System zunächst merklich langsamer, bis es letztendlich gar nicht mehr antwortet. Bei diesem Parameter ist es sinnvoll, einen gewissen Schwellenwert festzulegen, den der freie Speicherplatz nicht unterschreiten sollte, um eine korrekte Funktion des Systems gewährleisten zu können.

Ein weiterer Parameter, der Aufschluss über die Verfügbarkeit eines Systems gibt, ist die Systemlast. Bei Rechnern, die mit dem Unix-Betriebssystem arbeiten, bezeichnet dieser Wert üblicherweise den Durchschnitt der gerade ausgeführten Prozesse. Je niedriger dieser Wert ist, desto schneller reagiert das System. Der Durchschnitt für die Systemlast wird in Intervallen von 1, 5 und 15 Minuten angegeben, damit erkenntlich wird, wie lange der Lastzustand schon anhält. Information 25 zeigt die Veränderung der Systemlast des iravpn-Systems in Abhängigkeit von der Tageszeit.



Information 25: Systemlast des iravpn-Systems

Als letzter Parameter zur Messung des Systemzustands soll eine DNS-Abfrage dienen. Diese soll sicherstellen, dass der Rechnername des Zielsystems auch einwandfrei in die zugewiesene IP-Adresse übersetzt werden kann. Kann der Name nicht in die entsprechende IP-Adresse aufgelöst werden, liegt eine Fehlfunktion oder -konfiguration des DNS-Systems vor.

Alle hier beschriebenen Parameter sind in Tabelle 3 zusammengefasst und für das iraldap-System und das iravpn-System zu überprüfen. Bei einem Scheitern der Messungen oder dem Überschreiten der angegebenen Grenzwerte ist der Administrator über das Problem zu informieren.

Parameter	Beschreibung
Pingzeit	Reaktionszeit des Systems auf Netzwerkanfragen.
Freier Festplattenspeicherplatz	Gibt den prozentualen Teil freier Festplattenkapazität an.
Systemlast	Gibt den Wert für die momentane Systemlast an.
Namensauflösung	IP-Adresse des zugehörigen Hostnamens.

Tabelle 3: Beschreibung der Systemparameter

4.3.2 Komponenten

In den folgenden Abschnitten werden Parameter identifiziert, die Aussagen über die Verfügbarkeit der Komponenten des VPN-Dienstes zulassen.

Die einzelnen am VPN-Dienst beteiligten Komponenten werden auf Unix Betriebssystemen durch verschiedene Programme – unter anderem so genannte Daemons – realisiert. Daemons laufen im Hintergrund in Form eines Prozesses ab und warten auf bestimmte Ereignisse – wie beispielsweise das Eintreffen einer Netzwerkanfrage –, um daraufhin aktiv zu werden. Typische Beispiele für Daemons sind Server-Dienste wie zum Beispiel Web-, Mail- oder Datenbankserver.

Wenn die in Kapitel 4.1.1 vorgestellten Komponenten arbeiten, müssen verschiedene Prozesse auf den jeweiligen Rechensystemen gestartet sein. Auf dem iravpn-System müssen beispielsweise die Prozesse „pluto“ (IPsec), „l2tpd“ (L2TP), „pppd“ (PPP) und „radiusd“ (RADIUS) laufen, während auf dem System iraldap der Prozess „slapd“ (OpenLDAP) verfügbar sein muss. Läuft einer dieser Prozesse nicht, kann keine VPN-Verbindung zustande kommen. Im weiteren Sinn sind sogar noch mehr Prozesse betroffen wie beispielsweise der „named“ (BIND) Prozess, der für die Namens-auflösung eines Hostnamens in eine IP-Adresse (DNS) benötigt wird.

Für manche Prozesse – wie beispielsweise ein Webserver – werden mehrere Prozess-instanzen gestartet, damit Anfragen schneller bearbeitet werden können. Deshalb ist nicht nur zu prüfen, ob ein Prozess überhaupt gestartet ist, sondern vielmehr, ob er mit der korrekten Anzahl an Prozessinstanzen gestartet wurde.

Ein weiterer wichtiger Parameter der Komponenten sind ihre Konfigurationsdateien. Nur eine korrekt konfigurierte Komponente funktioniert den Anforderungen ent-sprechend. Die Verfügbarkeit einer Komponente ist also nicht gewährleistet, wenn die entsprechenden Konfigurationsdateien fehlen oder nicht den Vorgaben entsprechen. Deshalb ist es notwendig zu überprüfen, ob die Konfigurationsdateien vorhanden sind, und ob sie den Vorgaben entsprechen und nicht versehentlich durch Fehlkonfiguration oder absichtlich durch einen Angreifer geändert wurden.

Zur Überprüfung der Funktionsfähigkeit einer Komponente werden ihre Logdateien ausgewertet. In Logdateien werden Meldungen über die Tätigkeiten einer Komponente aufgezeichnet. Es werden je nach Konfiguration der Komponente mehr oder weniger Meldungen aufgezeichnet, anhand derer sich eine eventuelle Störung der Komponente identifizieren lässt. Da die Auswertung von Logdateien eine sehr komplexe Aufgabe sein kann, soll hier zunächst nur geprüft werden, ob die entsprechende Logdatei vorhanden ist. Für die genauere Analyse einer Logdatei müsste eine Analysierungs-komponente entwickelt werden, die jedoch nicht Teil dieser Implementierung sein wird. Auf diese Komponente wird jedoch im Ausblick nochmals genauer eingegangen.

Damit der VPN-Dienst einwandfrei betrieben werden kann, müssen bestimmte Versionen von Komponenten installiert sein. Viele Komponenten hängen voneinander ab und erfordern ganz bestimmte Versionsnummern anderer Komponenten, damit deren Interoperabilität gewährleistet ist. Ist eine Komponente nicht installiert oder ist eine falsche Version der Komponente installiert, ist die Funktionsfähigkeit des VPN-Dienstes nicht mehr gegeben. Außerdem sind die Voraussetzungen zu erfassen, die für den Betrieb der unterschiedlichen Komponenten erforderlich sind.

Die Ausprägungen der hier beschriebenen Parameter werden im weiteren Verlauf zur Erkennung von Fehlerzuständen dienen und in Tabelle 4 zusammengefasst.

Parameter	Beschreibung
Prozess	Beschreibt den Namen des Daemons, der in der Prozesstabelle angezeigt wird.
Konfigurationsdatei	Gibt den Pfad zur Konfigurationsdatei an.
Logdatei	Gibt den Pfad zur Logdatei an.
System	Hostname des Systems, auf dem die Komponente ausgeführt wird.
Version	Versionsnummer der Komponente.
Voraussetzung	Voraussetzung, die zur Installation bzw. zum Betrieb der Komponente erfüllt sein muss.

Tabelle 4: Beschreibung der Komponentenparameter

Welche konkreten Werte die Parameter für die jeweiligen Komponenten annehmen, wird in den folgenden Abschnitten spezifiziert.

Openswan (IPsec)
Prozess: pluto
Konfigurationsdatei: /etc/ipsec.conf
Logdateien: /var/log/messages, /var/log/secure
System: iravpn.ira.uka.de
Version: 2.4.4-1
Voraussetzung: Kernel Version 2.4 oder 2.6

L2TPD
Prozess: l2tpd
Konfigurationsdatei: /etc/l2tpd/l2tpd.conf
Logdatei: keine vorhanden
System: iravpn.ira.uka.de
Version: 0.69-13
Voraussetzung: Unix mit pppd

PPPD
Prozess: pppd
Konfigurationsdatei: /etc/ppp/options.l2tpd
Logdatei: /var/log/messages
System: iravpn.ira.uka.de
Version: 2.4.2-6.4
Voraussetzung: Kerner Version 2.2 oder 2.4+ mit ppp Unterstützung

OpenLDAP
Prozesse: slapd, slurpd
Konfigurationsdatei: /etc/slapd.conf
Logdatei: /var/log/slapd.log
System: iraldap.ira.uka.de
Version: 2.2.13-4
Voraussetzung: Unix, OpenSSL 0.9.7+, Cyrus SASL 2.1.18+

RADIUS

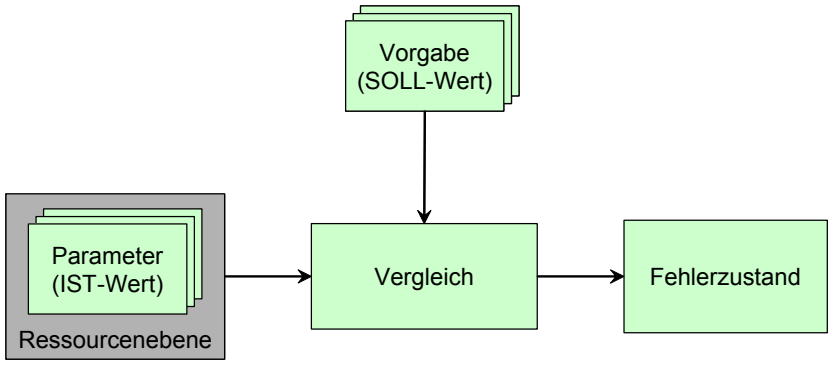
Prozess:	radiusd
Konfigurationsdatei:	/etc/raddb/radiusd.conf und /etc/raddb/iraldap
Logdatei:	/var/log/radius/radius.log
System:	iravpn.ira.uka.de
Version:	1.0.1-3
Voraussetzung:	Unix

VPNADM

Prozess:	httpd
Konfigurationsdatei:	/etc/httpd/conf/httpd.conf
Logdateien:	/var/log/httpd/error_log
System:	iraldap.ira.uka.de
Version:	2.0.52-22
Voraussetzung:	Unix

4.4 Auswertung von Ressourcenparametern

Durch das Überwachen der obigen Parameter können fehlerhafte bzw. nicht erwünschte Zustände festgestellt werden. Wie in Information 26 dargestellt ist, werden die gemessenen Parameter mit den Vorgaben verglichen.



Information 26: Vergleich von Parametern zur Ermittlung von Fehlerzuständen

Der Administrator legt die Vorgaben in Abhängigkeit von der vereinbarten SLA fest. Seine Aufgabe ist es zu ermitteln, in welchen Wertebereichen sich die Parameter befinden müssen, damit die geforderte Qualität des Dienstes eingehalten wird. Entsprechen die Werte dabei nicht den definierten Vorgaben, so wird eine Fehlermeldung generiert. Im folgenden Kapitel geht es darum, mögliche Fehlerzustände und ihre Eintrittsbedingung zu formulieren.

Definition von Fehlerzuständen
Ein System kann folgende Fehlerzustände aufweisen:

- Pingzeit des Systems ist größer als 500 Millisekunden
- Freier Festplattenspeicherplatz sinkt unter 5 % Grenze
- Systemlast hat den Wert 10 überschritten
- DNS-Anfrage liefert falsche oder keine IP-Adresse

Eine Komponente kann folgende Fehlerzustände aufweisen:

- Prozessinstanz nicht gestartet
- Zu wenige Prozessinstanzen gestartet
- Zu viele Prozessinstanzen gestartet
- Konfigurationsdatei fehlt
- Konfigurationsdatei entspricht nicht der Vorgabe
- Logdatei fehlt
- Komponente nicht installiert
- Falsche Version der Komponente installiert
- Voraussetzungen für den Betrieb der Komponente nicht erfüllt

Treten die oben beschriebenen Fehlerzustände ein, müssen Aktionen ausgeführt werden, damit wieder ein normaler Zustand erreicht wird. Im nächsten Kapitel werden zunächst die möglichen Aktionen definiert und dann werden Policies formuliert, die einen Fehlerzustand mit den jeweils auszuführenden Aktionen verknüpfen.

5 FORMULIERUNG VON POLICIES

In diesem Kapitel werden ausgehend von der Analyse des VPN-Dienstes im vorigen Kapitel PCIM-konforme Policies für den VPN-Dienst formuliert.

5.1 Fehlerbehebung

In den letzten Abschnitten wurden Parameter von Systemen und Komponenten erfasst, mit deren Hilfe der Zustand des VPN-Dienstes analysiert werden kann. Ist ein Fehlerzustand eingetreten, müssen Aktionen ausgeführt werden, um den Zustand zu korrigieren. Durch Policies werden die Fehlerzustände mit auszuführenden Aktionen verknüpft. Der Fehlerzustand stellt dabei die Bedingung für die Ausführung der Policy dar. Die Aktionen definieren die notwendigen Maßnahmen, um den entsprechenden Fehler zu beheben. Im Folgenden sollen nun konkret Policies formuliert werden, die bereits definierte Fehlerzustände behandeln und Aktionen definieren, um einen gewünschten Zustand zu erreichen.

5.1.1 Definition von Aktionen

Um von einem Fehlerzustand in einen Normalzustand zu kommen, sind gewisse Aktionen – die hier definiert werden sollen – auszuführen. Als Einschränkung sei hier angemerkt, dass als einzige Aktion bei Fehlerzuständen eines Systems die Benachrichtigung des Administrators vorgesehen wird, da die Behebung eines Systemfehlers in der Regel zu komplex ist, um sie vollständig automatisch durchzuführen zu können.

Dieser Sachverhalt wird am Beispiel des Festplattenspeichers verdeutlicht. Sinkt der freie Plattenspeicher eines Systems unter 5 Prozent, ist es möglich, automatisch alte Logdateien zu komprimieren und ins Datenarchiv zu verschieben. Sollte der freie Speicherplatz nun aber immer noch nicht ausreichen, muss der Administrator darüber entscheiden, ob die Festplattenkapazität des Systems generell erhöht werden muss. Da die Erhöhung der Speicherkapazität – beispielsweise durch den Einbau weiterer Festplatten – in der Regel einen manuellen Eingriff in das System bedeutet, ist an dieser Stelle nur die Benachrichtigung des Administrators durch Policies vorgesehen. Analog zu diesem Beispiel verhält es sich auch für die Parameter Netzverfügbarkeit, Namensauflösung und Systemlast, denn auch bei den Fehlerzuständen dieser Parameter ist im Allgemeinen das Eingreifen eines Administrators nicht zu vermeiden. Natürlich lassen sich auch bei diesen Fehlern gewisse Prozeduren automatisieren, jedoch soll zunächt nur eine Benachrichtigung des Administrators erfolgen. Eine spätere Erweiterung um weitere individuelle Aktionen ist jederzeit möglich.

Damit die Verfügbarkeit von Komponenten gewährleistet werden kann, müssen – wie in Kapitel 4.3.2 beschrieben – gewisse Prozesse auf verschiedenen Systemen gestartet sein. Fällt einer dieser Prozesse aus, muss durch Policies dafür gesorgt werden, dass er neu gestartet wird.

Die vom Administrator hinterlegten Konfigurationen müssen mit den Konfigurationsdateien der einzelnen Komponenten verglichen werden. Treten Unterschiede auf, müssen diese dem Administrator gemeldet werden, damit er die betreffende Konfigurationsdatei untersuchen kann, um den Fehler zu beheben. Versionskonflikte zwischen der installierten und der in der Policy spezifizierten Version müssen dem Administrator auch umgehend gemeldet werden. Hier wäre zwar denkbar eine auto-

matische Installation der gewünschten Version durchzuführen, aber in der Praxis sollte der Administrator eine solche Maßnahme zumindest dann selbst vornehmen, wenn die entsprechende Komponente auch noch für andere Dienste verantwortlich ist. Im Falle des VPN-Dienstes wäre es sinnvoll, die automatische Installation der Komponenten L2TP und Openswan zu ermöglichen, während die anderen Komponenten auch in anderen Diensten Verwendung finden. Evtl. wäre auch eine Meldung an den Administrator denkbar, über die er im Bedarfsfall eine automatische Installation veranlassen kann.

Zusammenfassend seien hier nochmals alle möglichen Aktionen aufgeführt:

- Benachrichtigung des Administrators
- Starten einer Prozessinstanz
- Stoppen einer Prozessinstanz
- Konfiguration von Vorlage wieder herstellen
- Installation einer Komponente
- Deinstallation einer Komponente

5.1.2 Zusammenfassung der Fehler und Aktionen zu Policies

Die bereits definierten Fehler und Aktionen sollen nun zu Policies zusammengefasst werden. Dazu werden in diesem Abschnitt die identifizierten Bedingungen und Aktionen in Tabelle 5 aufgeführt. Jede Zeile stellt dabei eine eigene Policy dar.

Fehlerzustand	**Auszuführende Aktionen**
Pingzeit des Systems ist größer als 500 Millisekunden	Benachrichtigung des Administrators
DNS-Anfrage liefert falsche oder keine IP-Adresse	Benachrichtigung des Administrators
Festplattenspeicherplatz des Systems hat die 5 % Grenze unterschritten	Benachrichtigung des Administrators
Die Systemlast des Systems hat den Wert 10 überschritten	Benachrichtigung des Administrators
Keine Instanz des Prozesses gestartet	Starten des betreffenden Prozesses, Benachrichtigung des Administrators
Zu wenig Instanzen des Prozesses gestartet	Starten einer weiteren Instanz des betreffenden Prozesses, Benachrichtigung des Administrators
Zu viele Instanzen des Prozesses gestartet	Stoppen einer Instanz des betreffenden Prozesses, Benachrichtigung des Administrators
Konfigurationsdatei existiert nicht	Kopieren der Vorlage der Konfigurationsdatei, Benachrichtigung des Administrators
Konfigurationsdatei entspricht nicht der Vorgabe	Benachrichtigung des Administrators
Logdatei existiert nicht	Benachrichtigung des Administrators

Fehlerzustand	Auszuführende Aktionen
Komponente ist nicht installiert	Automatische Installation der entsprechenden Komponente (sofern keine Abhängigkeiten zu anderen Diensten bestehen), Benachrichtigung des Administrators
Installierte Version entspricht nicht der Vorgabe	Deinstallation der falschen Version und automatische Installation der entsprechenden Komponente (sofern keine Abhängigkeiten zu anderen Diensten bestehen), Benachrichtigung des Administrators
Voraussetzungen zum Betrieb sind nicht erfüllt	Benachrichtigung des Administrators

Tabelle 5: Zusammenfassung von Fehlern und Aktionen

5.2 Definition von PCIMe-konformen Policies

Bevor PCIMe-konforme Policies formuliert werden können, müssen in den folgenden Abschnitten zunächst noch spezielle Variablen und Rollen für das VPN-Szenario definiert werden. Erst im Anschluss daran können die jeweiligen Policies erstellt werden.

5.2.1 Definition von Variablenklassen

Von PCIMe werden leider nur einige wenige Variablenklassen, wie beispielsweise Zieladressen und -ports, für das Netzwerkmanagement definiert. Zum Management des VPN-Dienstes sind allerdings andere Parameter – wie beispielsweise Pingzeiten oder Systemlast – von Bedeutung. Deshalb ist es nötig, die Klasse PolicyExplicitVariable zu verwenden, um eigene Variablenklassen zu definieren, die in PCIMe noch nicht existieren.

Um PCIMe mit eigenen Variablenklassen zu erweitern, müssen die Attribute ModelClass und ModelProperty der Klasse PolicyExplicitVariable festgelegt werden. Durch die Belegung dieser Attribute, lässt sich einer Variable eine bestimmte Semantik zuordnen, durch die es erst möglich wird, den ihr zugeordneten Wert zu interpretieren. Diese Definitionen werden sowohl für Variablen der Bedingungen als auch der Aktionen von Policies benötigt.

Als Bedingung einer Policy sollen die bereits definierten Fehlerzustände dienen, als Aktionen werden die ebenfalls definierten Aktionen zur Fehlerbehebung verwendet. Es sind also Variablen nötig, die Fehlerzustände von Systemen bzw. Komponenten – wie Systemüberlast oder Ausfall eines Prozesses – anzeigen und Aktionen zur Fehlerbehebung – wie die Benachrichtigung des Administrators oder den Neustart eines Prozesses – beschreiben.

Die Variablenklasse „Error"

Diese Klasse enthält Variablen, die zur Benachrichtigung über einen Fehlerzustand verwendet werden.

Fehlerzustandsvariable	Wert	Beschreibung
MaximumPingTimeExceeded	<SystemName>	Pingzeit des angegebenen Systems ist > 500 Millisekunden
WrongIPForHostname	<HostName>	DNS-Anfrage zum Hostnamen liefert gar keine oder nicht die korrekte IP-Adresse
NotEnoughFreeStorage	<SystemName>	Festplattenspeicherplatz des Systems hat die 5% Grenze unterschritten
SystemloadTooHigh	<SystemName>	Die Systemlast des Systems hat den Wert 10 überschritten
ProcessNotRunning	<ProcessName>	Keine Instanz des Prozesses gestartet
TooFewProcessesRunning	<ProcessName>	Zu wenig Instanzen des Prozesses gestartet.
TooManyProcessesRunning	<ProcessName>	Zu viele Instanzen des Prozesses gestartet
ConfigfileMissing	<ConfigfileName>	Entsprechende Konfigurationsdatei existiert nicht
ConfigfileMismatch	<ConfigfileName>	Unterschied zwischen hinterlegter und vorhandener Konfigurationsdatei
LogfileMissing	<LogfileName>	Logdatei existiert nicht.
ComponentNotInstalled	<Component>	Komponente ist nicht installiert
VersionMismatch	<InstalledComponent>	Installierte Version entspricht nicht der Vorgabe
RequirementsNotMet	<Component>	Voraussetzungen zum Betrieb der angegebenen Komponente sind nicht erfüllt

Tabelle 6: Variablen für Fehlerzustände

Die Variablenklasse „Component"

Mit dieser Klasse werden Variablen für Aktionen an Komponenten definiert.

Fehlerzustandsvariable	Wert	Beschreibung
StartProcess	<ProcessName>	Startet den angegebenen Prozess
StopProcess	<ProcessName>	Stoppt den angegebenen Prozess

Fehlerzustandsvariable	Wert	Beschreibung
RestoreConfigfile	<ConfigfileName>	Konfigurationsdatei von Vorlage wiederherstellen
InstallComponent	<ComponentName>	Installiert die angegebene Komponente
RemoveComponent	<ComponentName>	Entfernt die angegebene Komponente

Tabelle 7: Variablen für Aktionen auf Komponenten

Die Variablenklasse „Alert"

Diese Klasse enthält eine Variable, die zur Benachrichtigung des Administrators im Fehlerfall verwendet wird.

Fehlerzustandsvariable	Wert	Beschreibung
NotifyAdministrator	<MailAddress>	Sendet eine Benachrichtigung an die angegebenen Mailadressen

Tabelle 8: Variable für die Benachrichtigung des Administrators

5.2.2 Definition von Rollen

Rollen werden benötigt, um die Menge der anwendbaren Policies einzugrenzen. Der PDP wertet bei einer anstehenden Policy-Entscheidung nicht alle Policies im PolicyRepository aus, sondern lässt sich nur die Teilmenge der Policies übermitteln, die für den aufgetretenen Fehlerzustand relevant ist. Um diese relevanten Policies zu erhalten, fragt der PDP alle Policies aus dem Repository ab, deren Rolle mit der Rolle des aufgetretenen Fehlers übereinstimmt.

Die überwachten Fehlerzustände von Systemen und Komponenten des VPN-Dienstes müssen also bestimmten Rollen zugeteilt werden, damit der PDP die zum Fehler passenden Policies identifizieren kann. Dazu werden zunächst drei Hauptkategorien definiert, die den Ort des Fehlers lokalisieren:

- Network (Netzwerkebene)
- System (Systemebene)
- Component (Komponentenebene)

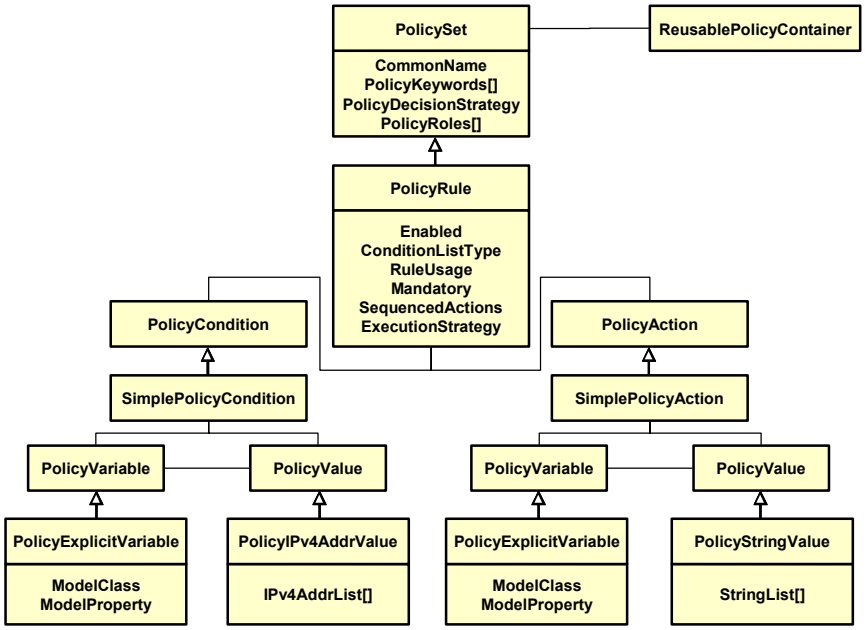
Neben dem Ort des Fehlers ist noch eine Klassifizierung der verschiedenen Arten von Fehlern nötig:

- Availability (Verfügbarkeit)
- DNS (Namensauflösung)
- Storage (Speicherplatz)
- Load (Last)
- Process (Prozessverfügbarkeit)
- Config (Konfigurationsdateien)
- Log (Logdateien)
- Version (Versionsnummer)
- Requirements (Voraussetzungen)

Durch die Kombination der Rolle für den Ort und die Art des Fehlers lassen sich bestimmte Fehlerzustände beschreiben. So kann beispielsweise der Ausfall eines Prozesses der Rollenkombination „Component&&Process“ – wie in Kapitel 2.2.3 unter der Klasse „PolicyRule“ beschrieben – zugeordnet werden.

5.2.3 Vorlage für eine PolicyRule

Nachdem die Policies bereits umgangssprachlich in Tabelle 5 formuliert wurden, können nun unter Verwendung der definierten Variablenklassen und Rollen PCIMe-konforme Policies erstellt werden. Information 27 zeigt, wie eine einfache PCIMe-konforme PolicyRule aufgebaut ist und aus welchen Klassen und Attributen sie besteht.



Information 27: Klassen und Attribute einer PCIM-konformen PolicyRule

Mehrwertige Attribute – also Attribute, die mehr als einen Wert enthalten können – werden durch eine an den Attributnamen angeschlossene eckige Klammer gekenn-zeichnet. Folgende Tabelle liefert eine Beschreibung aller notwendigen Attribute und deren Syntax.

Attribut	Beschreibung	Wert
CommonName (CN)	Umgangssprachlicher Name der Policy	<PolicyName>
PolicyKeywords[]	Kategoriezuordnung der Policy entsprechend der Kategorisierung in Tabelle 1.	"UNKNOWN", "CONFIGURATION", "USAGE", "SECURITY", "SERVICE", "MOTIVATIONAL", "INSTALLATION", "EVENT", "POLICY"
PolicyDecisionStrategy	Gibt an, ob alle Aktionen in einem PolicySet ausgeführt werden müssen oder nur die der ersten zutreffenden PolicyRule.	1 [FirstMatching] default, 2 [AllMatching]
PolicyRoles[]	Rollenzuordnung zu einzelnen und/oder kombinierten Rollen	<RoleName>[&&<RoleName>]*
Enabled	Ermöglicht das Aktivieren und Deaktivieren von Policies. Hier kann auch ein Debugmodus gewählt werden, bei dem nur die Bedingungen der Policy überprüft werden, ohne Aktionen auszuführen.	1 [Enabled] default, 2 [Disabled], 3 [EnabledForDebug]
ConditionListType	Gibt an, ob die angegebenen Bedingungen in KNF oder DNF zu interpretieren sind.	1 [KNF] default, 2 [DNF]
RuleUsage	Beschreibung, wie die Policy genutzt werden soll.	<RuleUsage>
Mandatory	Gibt an, ob die Policy in jedem Fall (Lastsituation) ausgeführt werden muss oder nicht.	"TRUE" default, "FALSE"
SequencedActions	Definiert, ob die Reihenfolge der auszuführenden Aktionen notwendig, gewünscht oder egal ist.	1 [Mandatory], 2 [Recommended], 3 [DontCare] default

Attribut	Beschreibung	Wert
ExecutionStrategy	Gibt an, ob alle Aktionen ausgeführt werden sollen oder nur so lang, bis eine Subaktion erfolgreich oder nicht erfolgreich durchgeführt wurde.	1 [DoUntilSuccess], 2 [DoAll] default, 3 [DoUntilFailure]
ModelClass	Klassenname der zugehörigen Variablen	<ClassName>
ModelProperty	Name der Variablen	<PropertyName>
StringList[]	Liste von Strings	string

Tabelle 9: Beschreibung der Attribute und Syntax einer PolicyRule

Nun gilt es, diese Policy-Vorlage für jede Zeile von Tabelle 5 mit Werten zu belegen und Policies zu formulieren. Um den Rahmen dieses Kapitels nicht zu sprengen, werden hier nur die Policies für Netzverfügbarkeit und Prozessverfügbarkeit dargestellt. Im Anhang findet sich zu jeder Fehlerkategorie eine entsprechende Policy.

5.2.4 Netzverfügbarkeits-Policy

Überschreitet die Pingzeit eines Systems das gewünschte Maximum, muss der Administrator informiert werden.

Attribut	Wert
CommonName (CN)	“alertPing”
PolicyKeywords[]	“EVENT” “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Network&&Availability”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Benachrichtigung des Administrators wenn die maximal zulässige Pingzeit überschritten wurde.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“MaximumPingTimeExceeded”
StringList[]	“iraldap.ira.uka.de” bzw. “iravpn.ira.uka.de”
Aktion	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“netz@atis.uka.de”

Tabelle 10: Policy zur Meldung mangelnder Netzverfügbarkeit

5.2.5 Prozessverfügbarkeits-Policy

Läuft ein Daemon, der für die Erbringung des VPN-Dienst verantwortlich ist nicht, muss dieser wieder gestartet und der Administrator darüber informiert werden.

Attribut	**Wert**
CommonName (CN)	“startPluto” bzw. “startL2tpd” bzw. “startPppd” bzw. “startSlapd” bzw. “startRadiusd”
PolicyKeywords[]	“CONFIGURATION”, “EVENT” “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Component&&Process”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Neustarten des betroffenen Prozesses und Benachrichtigung des Administrators.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“ProcessNotRunning”
StringList[]	“pluto” bzw. “l2tpd” bzw. “pppd” bzw. “slapd” bzw. “radiusd”
Aktion 1	
ModelClass	“Component”
ModelProperty	“StartProcess”
StringList[]	“pluto” bzw. “l2tpd” bzw. “pppd” bzw. “slapd” bzw. “radiusd”
Aktion 2	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

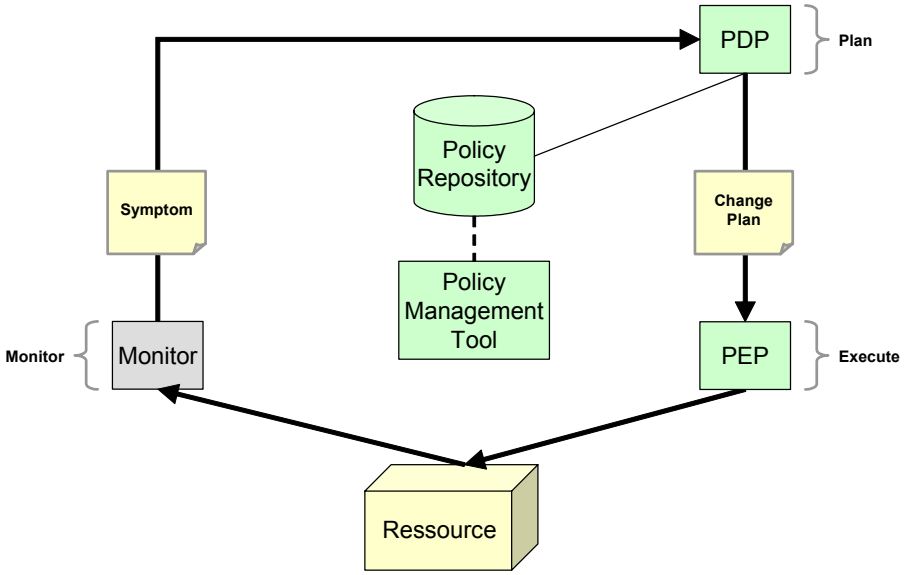
Tabelle 11: Policies zum Neustart von Prozessen

6 KONZEPT FÜR EINE POLICY-ARCHITEKTUR

Hier soll ein Konzept erstellt werden, wie die Komponenten in der von der IETF definierten Architektur realisiert werden können.

6.1 Fehlende Monitorkomponenten für den Einsatz am VPN-Dienst

In Kapitel 2.3 wurde die von der IETF definierte Policy-Architektur zur Zugangskontrolle vorgestellt. An dieser Stelle wird die Erweiterung der IETF Policy-Architektur aufgezeigt, die erforderlich ist, um sie für die Verfügbarkeitsüberwachung des VPN-Dienstes tauglich zu machen. Als Vorlage für diese Anpassungen soll der Architekturentwurf von IBM, der in Kapitel 3.6 vorgestellt wurde, dienen. In Information 28 wird gezeigt, wie die Architektur zu erweitern ist und zeigt die nicht von der IETF definierten Komponenten, den Monitor.



Information 28: Entwurf für eine Policy-Architektur für den VPN-Dienst

Da die Überwachung einer Ressource wesentlich komplexer ist als der Vergleich von Datenfeldern eines Datenpakets im Netzwerk, ist eine Monitorkomponente notwendig, die verschiedene Ressourcenparameter überwacht und eventuell auftretende Symptome an den PDP meldet.

Um den konkreten Überwachungskreislauf zu veranschaulichen, folgt hier ein kleines Beispiel. Die Monitorkomponente überwacht beispielsweise die gestarteten Prozesse auf den für den VPN-Dienst relevanten Systemen und meldet das Nichtvorhandensein eines notwendigen Prozesses an den PDP. Anhand von Policies werden im PDP die nötigen Aktionen ermittelt, um den ausgefallenen Prozess wieder zu starten. Diese notwendigen Aktionen werden zur Ausführung an den PEP übermittelt, der die geforderten Aktionen entsprechend ihrer Reihenfolge ausführt. Wenn der Neustart des Prozesses erfolgreich vom PEP durchgeführt wurde, wird das fehlerhafte Symptom von der Monitorkomponente nicht mehr länger beobachtet. Der Regelkreis ist damit geschlossen.

Dieses Kapitel liefert einen Überblick, wie die Policy-Architektur der IETF technisch umgesetzt werden kann. Bei der Betrachtung wird auch die fehlende Monitorkomponente der IETF-Architektur mit einbezogen, da sie benötigt wird, um eine prototypische Implementierung durchzuführen. Es soll ein Konzept veranschaulicht werden, wie policy-basiertes Management des VPN-Dienstes generell funktionieren kann. Das Konzept soll einen ersten funktionsfähigen Entwurf liefern, der durch spätere Erweiterungen und Optimierungen weiter verbessert werden kann.

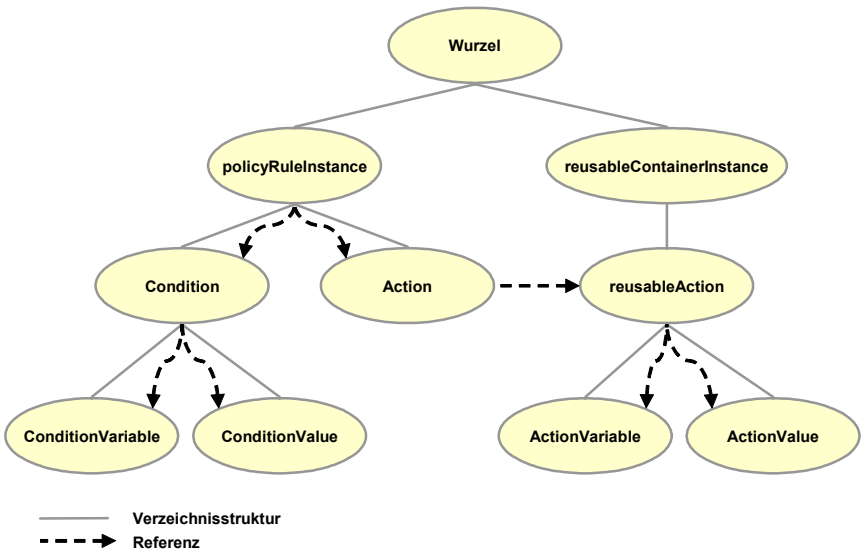
6.2 Policy Repository

Das *Policy Repository* ist die zentrale Ablage für Policies. In dieser Ablage werden sämtliche Policy-Objekte gespeichert, sowohl einfache Regeln, als auch komplexe Verbände von Policies, je nachdem was der Policy-Administrator benötigt. Die Ablage wird durch eine Datenbank realisiert, die von einem zentralen bzw. verschiedenen dezentralen PDPs abfragt werden muss. Dabei ist eine hohe Performanz im Bezug auf die vielen anfallenden Leseoperationen eine wichtige Anforderung an diese Datenbank. Prinzipiell kann zwar ein beliebiges Datenbankmanagementsystem (DBMS) eingesetzt werden. Die geforderte hohe Performanz und die hierarchischen Datenstruktur von LDAP machen jedoch diesen Verzeichnisdienst zu einem exzellenten Kandidaten für ein *Policy Repository*.

6.2.1 Abbildung von PCIM und PCIMe in LDAP

Von der IETF wurde mit [SM+04] (RFC 3703) und [PR+05] (RFC 4104) eine Abbildung von PCIM und PCIMe auf ein LDAP-Verzeichnis definiert. Mit dieser Abbildung lassen sich PCIM und PCIMe-konforme Konstrukte auf LDAP Klassen abbilden.

Eine generelle Herausforderung bei der Abbildung der Spezifikation ins LDAP stellt die Tatsache dar, dass LDAP nur mit Klassen und nicht mit Assoziationen arbeitet. Durch die hierarchische Struktur von LDAP-Objekten ist es jedoch möglich, andere Objekte zu referenzieren und damit denselben Effekt wie mit einer Assoziation zu erreichen. Information 29 zeigt den Aufbau einer Policy im LDAP-Verzeichnis.



Information 29: Aufbau einer PCIMe-konformen Policy im LDAP-Verzeichnis

Hier ist klar ersichtlich, dass keine Assoziationen existieren, lediglich Referenzen von einer Klasse auf eine andere. Jede Instanz eines LDAP-Objekts besitzt einen *Distinguished Name* (DN), der das entsprechende Objekt eindeutig identifiziert. Ein Objekt wird referenziert, indem einem Attribut die DN zum entsprechenden Objekt eingetragen wird. So besitzt eine policyRuleInstanz die Attribute „pcelsActionList“ und „pcelsConditionList“, mit denen die DNs der zugehörigen Bedingungen und Aktionen angegeben werden können. Attribute, die zur Priorisierung und Gruppierung der Bedingungen und Aktionen dienen, werden direkt in den referenzierten Objekten angegeben.

Wieder verwendbare Objekte werden separat in der Verzeichnisstruktur unterhalb von reusableContainer-Klassen angelegt. Im obigen Beispiel referenziert die Aktion der policyRuleInstanz eine wieder verwendbare Aktion, die ihrerseits unterhalb einer reusableContainer-Instanz angeordnet ist. Eine wieder verwendbare Aktion unterscheidet sich von einer normalen Aktion dadurch, dass sie in mehreren Policies verwendet wird. So ist beispielsweise die Benachrichtigung des Administrators eine Aktion, die in vielen Policies benutzt wird. Deshalb macht es Sinn, diese Aktion nur einmal zu erstellen, statt sie in jeder Policy erneut anzulegen.

Es wirft sich die Frage auf, warum die wieder verwendbare Aktion eigentlich nicht direkt von der PolicyRule aus referenziert wird. Diese Art der Referenzierung erlaubt es, der wieder verwendbaren Aktion in jeder PolicyRule eine individuelle Priorität zu geben. Die Aktion direkt an der PolicyRule legt die Priorität der referenzierten wieder verwendbaren Aktion fest. Es macht keinen Sinn, die Priorität der Aktion in der wieder verwendbaren Aktion selbst festzulegen, da es sonst nämlich nicht möglich wäre, die Aktion mit einer individuellen Priorität an eine bestimmte PolicyRule zu knüpfen. So ist es beispielsweise möglich eine PolicyRule zu erstellen, bei der die Benachrichtigung eine hohe Priorität hat, also vor anderen Aktionen mit niederer Priorität ausgeführt wird. Andererseits kann eine PolicyRule mit derselben Benachrichtigungsaktion formuliert werden, bei der die Benachrichtigung eine niedere Priorität besitzt. Ein analoger Mechanismus existiert auch für Bedingungen, aber die in dieser Arbeit definierten Policies verwenden keine wieder verwendbaren Bedingungen.

Wie die in Kapitel 5.2 formulierten Policies im LDAP dargestellt werden, behandelt Kapitel 7.2.

6.3 Policy Management Tool

Das *Policy Management Tool* (PMT) dient als Administrationswerkzeug, um Policies anzulegen, zu bearbeiten und zu löschen. In einem vollständig autonom arbeitenden Managementsystem stellt diese Komponente die einzige Schnittstelle zu einem menschlichen Benutzer dar.

Die Komplexität dieser Komponenten ist relativ hoch, da sie unter anderem für die syntaktische und semantische Konsistenz der Policies im *Policy Repository* verantwortlich ist. Es dürfen also weder Policies existieren, die sich gegenseitig beeinflussen oder aufheben, noch dürfen Policies spezifiziert werden, die Schleifen enthalten.

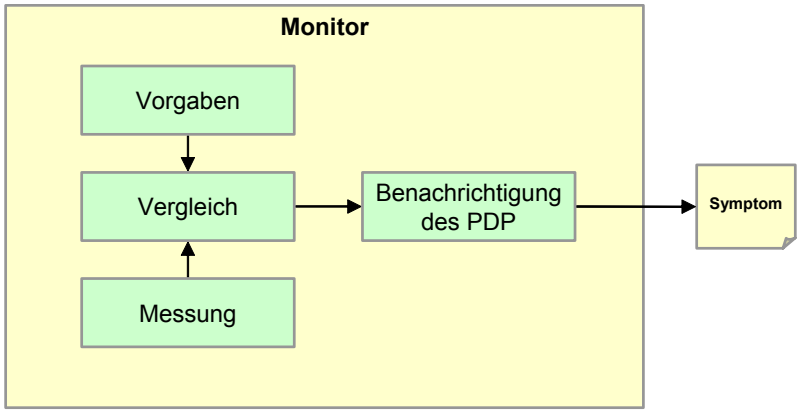
Da die Lösung von Policy-Konflikten [Ke04] ein komplexes eigenständiges Thema ist, wurde hinsichtlich der zu entwickelnden Komponente die Einschränkung getroffen, dass keine Behandlung von Policy-Konflikten erfolgt.

Damit eine gute Bedienbarkeit des PMT gewährleistet ist, sollte der Administrator bei der Erstellung bzw. Bearbeitung von Policies interaktiv durch Assistenten und Hilfestellungen unterstützt werden. Zudem lässt sich durch eine gute Unterstützung des Administrationsprozesses die Fehlerquote enorm senken.

Um eine möglichst hohe Flexibilität bei der Nutzung dieser Anwendung zu erreichen, ist der Einsatz einer Webschnittstelle äußerst sinnvoll. Sie ermöglicht dem Administrator die Benutzung des PMT prinzipiell von jedem Rechner aus, der über einen Browser verfügt, ohne dass die Installation weiterer Software erforderlich ist. Mit der Skriptsprache Perl lassen sich hervorragend dynamische Webseiten gestalten, deren Geschäftslogik vom Webserver ausgeführt wird. Deshalb wird Perl zur Realisierung des PMT eingesetzt.

6.4 Monitor

Die Monitorkomponente ist dafür zuständig, Ressourcenparameter zu überwachen und eventuelle Abweichungen von den Vorgaben an den PDP weiterzuleiten. Der Architekturentwurf für diese Komponente wird in Information 30 dargestellt.



Information 30: Architekturentwurf für die Monitorkomponente

Zunächst werden von der Ressourceninstrumentierung zur Messung von Parametern die verschiedenen Ressourcenparameter wie Anzahl der gestarteten Prozessinstanzen, Pingzeiten und IP-Adressen für gegebene Hostnamen erfasst. Viele System- und Komponentenparameter können nicht über SNMP abgefragt werden. Deshalb muss die Monitorkomponente ganz speziell an die Anforderungen für die Überwachung des VPN-Dienstes angepasst und die verwendeten Ressourcen ganz gezielt instrumentiert werden. Um die Werte für diese Parameter zu erhalten, müssen verschiedene kleine Systemkomponenten zur Instrumentierung der Ressource aufgerufen werden, da es sich um Parameter handelt, die nicht direkt beobachtet werden können. Die Instrumentierung wird im Implementierungskapitel (Kapitel 7) ausführlich behandelt.

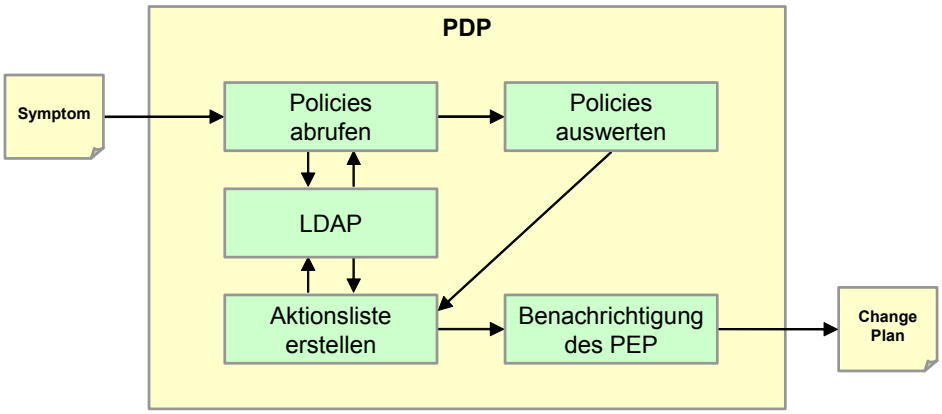
Wurde beispielsweise die Prozesstabelle eines Systems durch die entsprechende Instrumentierung abgefragt, muss die Monitorkomponente als nächstes vergleichen, ob die gemessenen Daten den Vorgaben entsprechen. Liefert der Vergleich Abweichungen zwischen Messungen und Vorgaben, so muss der PDP über diese Abweichungen (die Symptome) informiert werden. Das Monitoring könnte auch ohne weiteres auf Grundlage von Policies erfolgen, da hier lediglich Vergleichsregeln zum Herausfiltern von Fehlersymptomen verwendet werden, die sich auch in Form von Policies definieren

lassen. Zunächst soll das Monitoring aber mit fest verdrahteten Regeln zum Vergleichen entwickelt werden, da es hier um die Veranschaulichung des Gesamtkonzepts gehen soll und eine spätere Optimierung dieser Komponente ohne weiteres möglich ist.

Ähnlich wie der PEP muss auch diese Komponente direkt auf der jeweiligen Ressource lokalisiert sein. Nur so ist eine ressourcenspezifische Instrumentierung möglich, um die gewünschten Parameter abzufragen. Um die Monitoringkomponente zu realisieren, bietet sich die Verwendung der Skriptsprache Perl an, da sie sich besonders gut dazu eignet, die von der Instrumentierung übermittelten Parameter zu verarbeiten und in das gewünschte Format zu überführen.

6.5 PDP

Der *Policy Decision Point* stellt die zentrale Komponente der Policy-Architektur der IETF dar. Wurde ein bestimmter Fehler erkannt, muss der PDP anhand von Policies die nötigen Schritte zur Behebung des Fehlers ermitteln. Information 31 zeigt den Architekturentwurf für den PDP.



Information 31: Architekturentwurf für den PDP

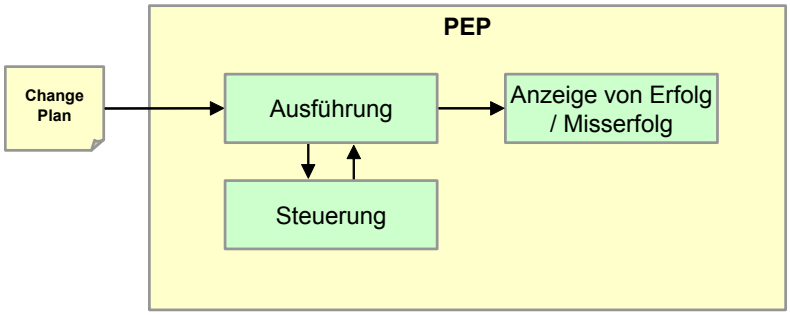
Wurde beispielsweise festgestellt, dass der für IPsec verantwortliche Prozess „pluto“ nicht mehr gestartet ist, wird dem PDP dieses Symptom vom Monitor übermittelt. Zusätzlich zum Symptom sendet der Monitor den Systemnamen, auf dem das Symptom gemessen wurde, und die Rolle der überwachten Komponente. Nun sucht der PDP alle zu der Rolle des Symptoms passenden Policies, indem er eine entsprechende Anfrage an das LDAP *Policy Repository* stellt. Die erhaltenen Policies müssen dann einzeln ausgewertet werden, um zu prüfen, welche der Policies auszuführen sind. Dazu werden die Bedingungen der Policies mit den übermittelten Symptomen verglichen. Sind alle anwendbaren Policies identifiziert, werden sie nacheinander ausgewertet, und es wird eine Liste der auszuführenden Aktionen erzeugt. Diese Aktionsliste wird dann an den PEP übermittelt.

Für das Szenario des VPN-Dienstes ist es sinnvoll, den PDP als zentrale Komponente zu modellieren. Dadurch wird eine Redundanz dieser Komponente auf verschiedenen Ressourcen vermieden und die Pflege der Komponente erleichtert. Die zu treffenden Policy-Entscheidungen erfordern außerdem kaum spezielle Schritte für eine bestimmte Ressource, da beispielsweise Prozesse auf verschiedenen Rechnern mit gleichem Betriebssystem identisch gestartet werden können. Die Verwendung eines lokalen PDPs (LPDPs) auf den einzelnen Ressourcen ist deshalb nicht erforderlich.

Um eine möglichst hohe Plattformunabhängigkeit des PDPs zu gewährleisten, soll die Komponente prototypisch in der Programmiersprache „Java“ implementiert werden. Dadurch ist der PDP sehr flexibel einsetzbar und nicht an ein bestimmtes System gebunden, was eine leichte Skalierung bei sich ändernden Anforderungen ermöglicht.

6.6 PEP

Vom PDP wird dem *Policy Enforcement Point* eine Liste mit den auszuführenden Aktionen übermittelt. Die Aufgabe des PEPs ist es nun, die Ausführungen der Aktionen getreu ihrer Reihenfolge durchzuführen und den PDP über Erfolg oder Misserfolg zu informieren. Der Architekturentwurf für den PEP wird in Information 32 gezeigt.



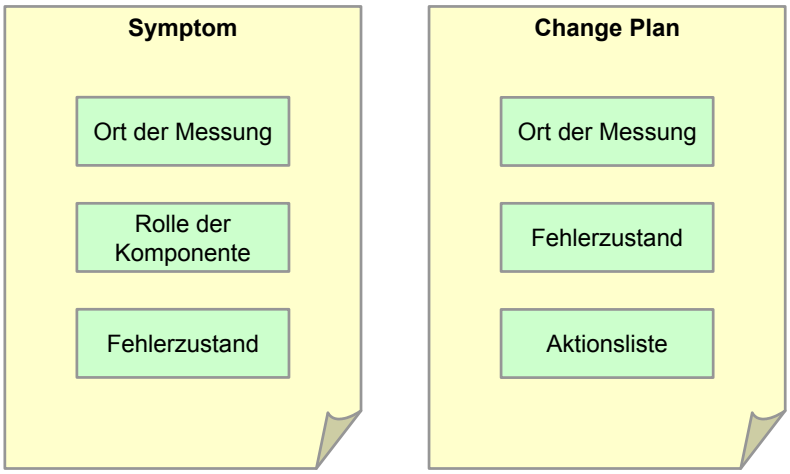
Information 32: Architekturentwurf für den PEP

Die vom PDP übermittelte Liste enthält Aktionen, die vom PEP eine nach der anderen ausgeführt werden müssen. Für jede auszuführende Aktion ist die Instrumentierung der Ressource erforderlich, um den gewünschten Effekt zu erreichen. Wenn beispielsweise der Start eines Prozesses erforderlich ist, wird der Ressourceninstrumentierung diese Aufgabe übermittelt. Die Instrumentierung liefert dem PEP zurück, ob die Aktion erfolgreich ausgeführt werden konnte oder nicht. Sind alle Aktionen abgearbeitet worden, beendet sich der PEP und zeigt dabei an, ob alle Aktionen einwandfrei ausgeführt wurden.

Der PEP ist eine dezentrale Komponente, die auf jedem Server lokal installiert wird und auf den Aufruf durch den PDP wartet. Eine Implementierung des PEPs in Java ist für diese Komponente sehr vorteilhaft, da ein Java-Programm auf nahezu jedem Rechensystem ohne weitere Veränderung ausgeführt werden kann. Zunächst wird aber nur eine Instrumentierung für das Unix-Betriebssystem entwickelt, da alle für den VPN-Dienst relevanten Ressourcen auf Grundlage dieses Betriebssystems arbeiten. In der Funktionalität des PEP sollten alle möglichen Aktionen erfasst sein, damit auf jedem Server die gleiche Komponente installiert werden kann und sichergestellt ist, dass jede Aktion auf jedem System ausführbar ist.

6.7 Nachrichtenelemente

Im Wesentlichen werden zwei Nachrichtenelemente verwendet, das „Symptom“ und der „Change Plan“. Das Symptom wird vom Monitor an den PDP und der Change Plan von PDP an den PEP übermittelt. Welche einzelnen Informationen die Nachrichten enthalten sollen, zeigt Information 33.



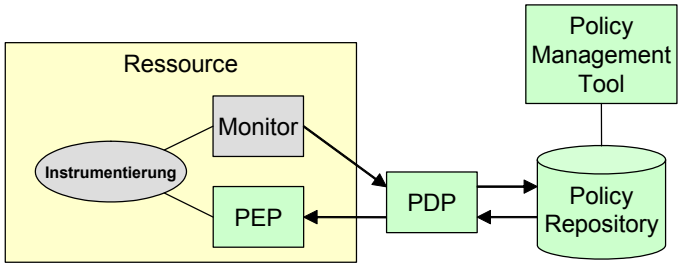
Information 33: Aufbau der einzelnen Nachrichtenelemente

Im Symptom sind neben dem Fehlerzustand Informationen über den Ort der Messung und die Rolle der überwachten Komponente enthalten. Die Information über den Ort der Messung des Fehlerzustands ist wichtig, damit der PDP die auszuführenden Aktionen an den entsprechenden PEP weiterleiten kann. Durch die Übermittlung der Rolle der überwachten Komponente kann der PDP die für diese Rolle relevanten Policies aus dem *Policy Repository* abrufen. Der genaue Fehlerzustand dient dem PDP dazu, die einzelnen Policies auszuwerten und zu prüfen, ob sie ausgeführt werden müssen oder nicht.

Im Change Plan finden sich Informationen über die auszuführenden Aktionen als auch den Ort der Messung und den Fehlerzustand. Der Ort und der Fehlerzustand dienen dem PEP dazu, diese Informationen im Falle einer geforderten Benachrichtigung an den Administrator zu übermitteln. Die Aktionsliste enthält alle Aktionen, die vom PEP der Reihenfolge entsprechend abzuarbeiten sind.

6.8 Lokalisation und Kommunikation

Dieses Kapitel veranschaulicht die Lokalisierung und die Kommunikation zwischen den Komponenten. Information 34 zeigt, dass Monitorkomponente und PEP dezentral auf den Ressourcen des VPN-Dienstes liegen, während PDP, *Policy Repository* und *Policy Management Tool* zentralisiert sind.

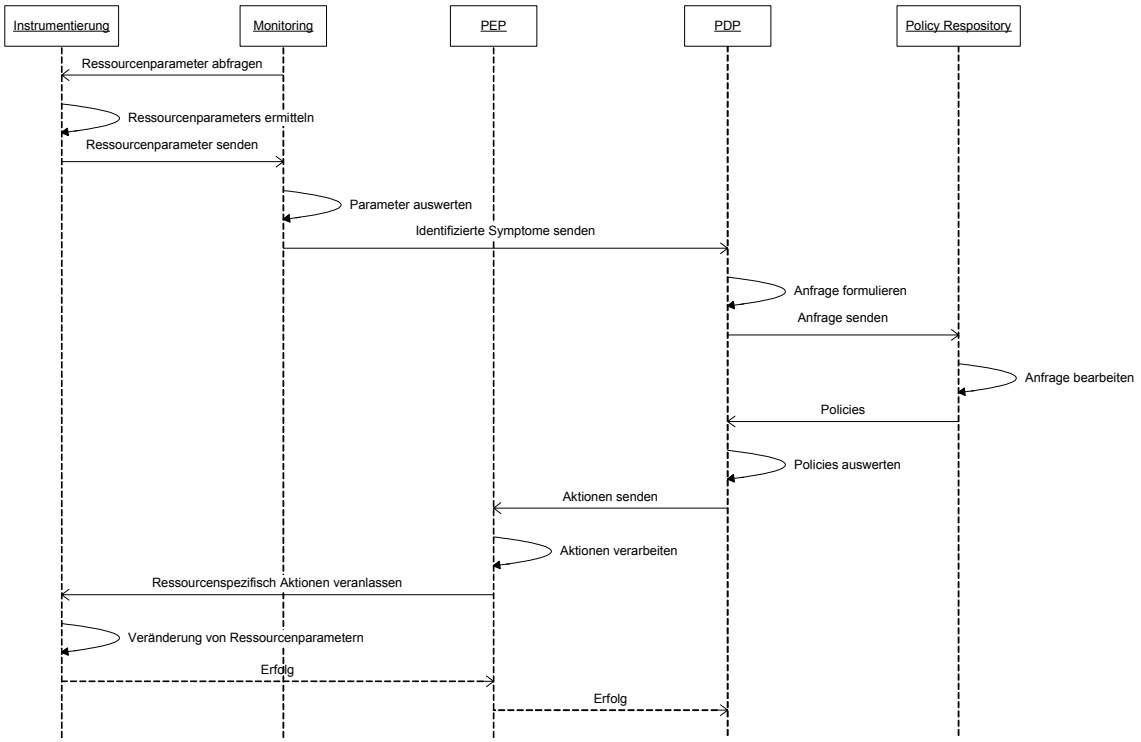


Information 34: Lokalisierung der Komponenten

Den genauen Kommunikationsablauf zeigt Information 35. Zunächst wird durch die Monitorkomponente die Ressourceninstrumentierung aufgerufen, um bestimmte Ressourcenparameter abzufragen. Wurden fehlerhafte Symptome beobachtet, wird das Symptom an den PDP weitergeleitet, um eine Policy-Entscheidung zu treffen. Der PDP

stellt eine Anfrage an das *Policy Repository* und wertet die zurückgelieferten Policies aus. Vom PDP wird eine Aktionsliste erstellt und an den PEP zur Ausführung übertragen. Vom PEP wird die Ressourceninstrumentierung verwendet, um die geforderten Aktionen auszuführen.

Das *Policy Management Tool* ist nicht in den Regelkreislauf eingebunden und dient lediglich dem Administrator als Werkzeug zur Bearbeitung von Policies.



Information 35: Ablauf der Kommunikation

Um die prototypische Implementierung nicht zu überfrachten, wird zunächst ein ganz simpler Mechanismus zur Kommunikation zwischen den Komponenten verwendet. Die Komponenten rufen sich gegenseitig direkt auf und übergeben die gewünschten Nachrichten direkt als Aufrufparameter an die Komponente. Um Missbrauch der Komponenten von Dritten zu verhindern, wird der Aufruf der Komponenten auf entfernten Rechnern in einem sicheren SSH-Tunnel vorgenommen.

7 PROTOTYPISCHE REALISIERUNG DES KONZEPTS

Das im vorhergehenden Kapitel ausgearbeitete Konzept wird hier prototypisch implementiert. Danach werden zusammenfassend die Vorteile und Einschränkungen des entwickelten Prototyps erörtert.

Entsprechend der Konzeption werden in diesem Kapitel die verschiedenen Komponenten umgesetzt. Hier folgt ein grober Überblick der zu implementierenden Komponenten, die in den folgenden Abschnitten genauer beschrieben werden. Ein *Policy Repository* wird auf der Grundlage eines LDAP-Verzeichnisses realisiert, das zur Ablage von Policies dient. Zur administrativen Pflege von Policies wird ein über den Browser bedienbares PMT entwickelt. Um Ressourcenparameter zu überwachen, wird die entworfene Monitorkomponente implementiert. Der hier zu implementierende PDP soll Policy-Entscheidungen treffen, die dann durch den realisierten PEP ausgeführt werden.

Alle hier zu entwickelnden Komponenten bilden gemeinsam ein policy-basiertes Managementsystem für den VPN-Dienst. Dieses Managementsystem soll in der Lage sein, Fehlerzustände von Systemen und Komponenten zu erkennen und diese anhand entsprechender Policies zu beheben bzw. den Administrator zu benachrichtigen.

An dieser Stelle ist es jedoch nicht wichtig, ein absolut vollständiges Managementsystem zu entwickeln. Es geht vielmehr um die Vermittlung eines Konzepts, wie policy-basiertes Management für den VPN-Dienst generell funktionieren kann. Die Behandlung von weiteren Fehlerzuständen lässt sich durch entsprechende Erweiterung der Komponenten erreichen. Anhand der in dieser Arbeit definierten Policies lässt sich das entwickelte Konzept sehr gut verdeutlichen.

7.1 Verwendete Software

Um den Prototyp zu implementieren, wurde bestimmte Software verwendet, die an dieser Stelle kurz beschrieben wird.

Der LDAP-Verzeichnisdienst wird durch die Software „OpenLDAP“ realisiert und in der ATIS bereits für das zentrale *Identity Management* verwendet. Von der IETF wird für die *Policy Repository* Komponenten auch ein LDAP-Verzeichnis vorgesehen. Für die Ablage von Policies soll der bestehende LDAP-Verzeichnisdienst der ATIS verwendet werden.

Zur Implementierung des Monitors und des *Policy Management Tools* wird die Skriptsprache Perl in der Version 5.8 verwendet. Sie eignet sich hervorragend zur Verarbeitung von Ausgaben der Ressourceninstrumentierung und kann durch komplexe Suchmuster den gewünschten Parameter herausfiltern. Außerdem eignet sich Perl auch sehr gut für eine Webanwendungen wie das *Policy Management Tool*, da die entsprechenden Skripte direkt vom Webserver ausgeführt werden können.

Der Einsatz von Java ist für die Implementierung von PDP und PEP vorgesehen. Da der PEP direkt auf der Ressource eingesetzt wird, bietet eine Implementierung in Java für diese Komponente die größtmögliche Plattformunabhängigkeit, auch wenn die Instrumentierung an die jeweilige Plattform angepasst werden muss. Es ist jedoch

denkbar, die Instrumentierung für verschiedene Betriebssysteme in den PEP zu integrieren, so dass eine einheitliche Komponente auf allen Systemen installiert werden kann. Zur Implementierung der Java-Komponenten wird JDK Version 5.0 und die Programmierumgebung (IDE) „eclipse“ verwendet.

7.2 Realisierung des Policy Repositories in OpenLDAP

Um PCIM-konforme Policies im LDAP speichern können, sind von der IETF die LDAP-Abbildungen [SM+04] (RFC 3703) und [PR+05] (RFC 4104) entworfen worden. Neben diesen beiden Abbildungen wird auch noch die von der DMTF entworfene LDAP-Abbildung von CIM [CIM-LDAP] (DSP0123) benötigt, da PCIM auf verschiedenen CIM-Klassen aufgebaut ist. Um diese Abbildungen in OpenLDAP nutzbar zu machen, mussten die Definitionen in OpenLDAP-Schemas überführt werden. Information 36 zeigt einen Auszug aus dem OpenLDAP-Schema für PCIMe.

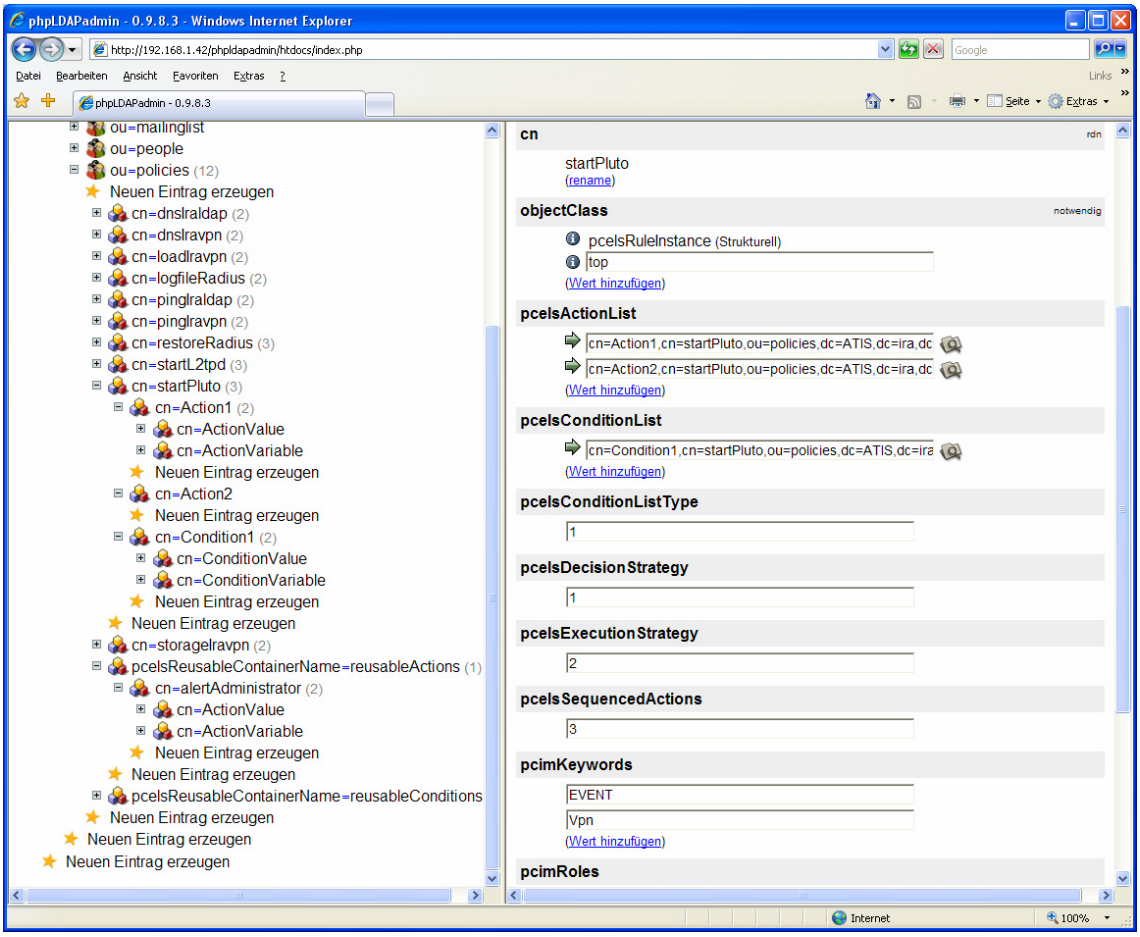
```
objectclass ( 1.3.6.1.1.9.1.6
    NAME 'pcelsRule'
    DESC 'Base class for representing a policy rule'
    SUP pcelsPolicySet
    ABSTRACT
    MAY ( pcimRuleName
       $ pcimRuleEnabled
       $ pcimRuleUsage
       $ pcimRuleMandatory
       $ pcelsRuleValidityPeriodList
       $ pcelsConditionListType
       $ pcelsConditionList
       $ pcelsActionList
       $ pcelsSequencedActions
       $ pcelsExecutionStrategy )
  )

attributetype ( 1.3.6.1.1.9.2.6
    NAME 'pcelsConditionListType'
    DESC 'Indicates the type of condition aggregation'
    EQUALITY integerMatch
    ORDERING integerOrderingMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
    SINGLE-VALUE
  )

attributetype ( 1.3.6.1.1.9.2.7
    NAME 'pcelsConditionList'
    DESC 'Unordered set of DNs of pcelsConditionAssociation entries'
    EQUALITY distinguishedNameMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  )
```

Information 36: Auszug aus dem OpenLDAP-Schema für PCIMe

Es ist gut zu erkennen, dass in Schemata für Openldap Objektklassen und Attribute definiert werden. In der obigen Information 36 wird beispielsweise die Objektklasse „pcelsRule“ und die zwei Attribute „pcelsConditionListType“ und „pcelsConditionList“ gezeigt.

Einem Objekt in OpenLDAP werden Objektklassen zugeordnet, während in Objektklassen notwendige (MUST) und optionale (MAY) Attribute definiert sind. Wird einem Objekt eine Objektklasse zugewiesen, müssen die in der Objektklasse als notwendig definierten Attribute mit angegeben werden, während optionale Attribute angegeben werden können. Die Attributdefinition enthält die Syntaxvorschriften, von welchem Typ die Werte des Attributs sein dürfen. Außerdem wird definiert, wie sich die Attribute bei Suchanfragen verhalten und wie sie sortiert werden. Hier soll die Definition von LDAP-Schemata allerdings nicht weiter ausgeführt werden. Genaueres kann in RFC 2252 nachgelesen werden.

Nachdem OpenLDAP mit den entsprechenden Schemata nun dafür vorbereitet wurde PCIM-konforme Policies aufzunehmen, wurden mit dem Tool „phpLDAPadmin“ die ersten Policies entsprechend der Struktur aus Information 29 angelegt. Um die hierarchische Struktur einer Policy zu verdeutlichen, zeigt Information 37 die Baumstruktur des Objekts, wie sie im phpLDAPadmin dargestellt wird.



Information 37: Darstellung einer Policy im LDAP

Aus der obigen Information 37 ist gut ersichtlich, dass Aktionen und Bedingungen in der LDAP-Baumstruktur direkt unter einer Policy angeordnet sind. Unterhalb der jeweiligen Bedingungen und Aktionen sind die zugehörigen Variablen und Werte angeheftet.

Im Folgenden wird ein Ausschnitt der „startPluto“ Policy im LDIF (*Ldap Data Interchange Format*) gezeigt. LDIF ist ein Datenformat zum Im- und Export von Daten eines LDAP-Verzeichnisses. Die Policy „startPluto“ dient dazu, bei einem Ausfall des Daemons „pluto“, dessen Neustart zu veranlassen und den Administrator zu benachrichtigen. In der LDIF-Darstellung erkennt man gut die Belegung der einzelnen Attribute. Eine vollständige Liste aller Policies im Format LDIF findet sich im Anhang dieser Arbeit.

```
# LDIF Export von:
# cn=startPluto,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=startPluto,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
…
pcelsActionList: cn=Action1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsActionList: cn=Action2,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=Action1,cn=startPluto,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
…
pcelsValueDN: cn=ActionValue,cn=Action1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ActionVariable,cn=Action1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=ActionValue,cn=Action1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
…
pcelsStringList: pluto

dn: cn=ActionVariable,cn=Action1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
…
pcelsVariableModelClass: Component
pcelsVariableModelProperty: ProcessRestart

dn: cn=Action2,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
…
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
…
```

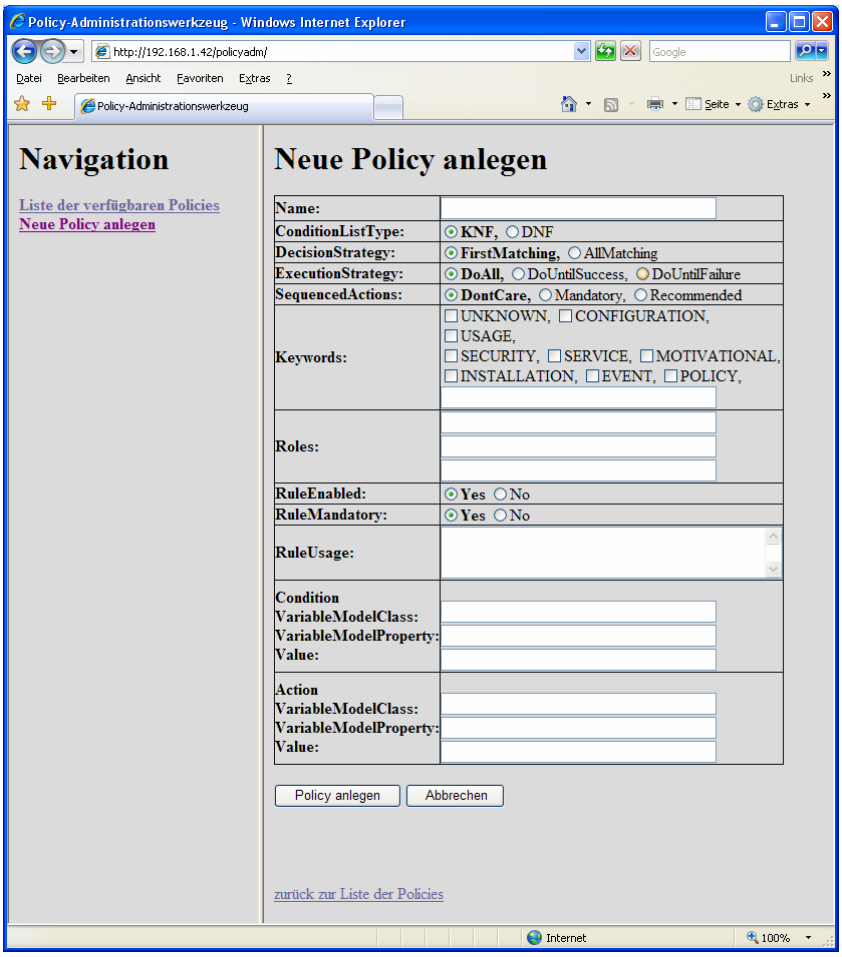
Diese Policy besteht aus einer Bedingung (Condition) und zwei Aktionen (Action). Wenn die Bedingung (Condition1) – der Daemon „pluto“ ist nicht gestartet – eintritt, soll zunächst der Prozess neu gestartet (Action1) und danach der Administrator benachrichtigt (Action2) werden. Da der Administrator bei vielen Fehlern benachrichtigt werden muss, ist diese Aktion als wieder verwendbare Aktion modelliert und kann dadurch auch von anderen Policies referenziert werden (s. Information 29). Innerhalb der LDAP-Baumstruktur liegt diese Aktion deshalb unterhalb eines ReusableContainers und wird von der zweiten Aktion der Policy über den *Distinguished Name* (DN) referenziert.

Ein DN ist ein jedem Objekt eindeutig zugeordneter Wert und fungiert als Adresse, unter der das jeweilige Objekt gefunden werden kann. In der LDIF-Darstellung der Policy steht deshalb auch zu Beginn von jedem Absatz der DN des entsprechenden Objekts. In den Attributen „pcelsConditionList“ und „pcelsActionList“ werden die der Policy zugehörigen DNs von Bedingungen und Aktionen aufgelistet. Die Aktionen und Bedingungen enthalten Attribute zur Priorisierung bzw. Gruppierung und verweisen entweder direkt auf die DNs der entsprechenden Variablen und Werte oder – wie hier zum Beispiel Action2 – auf den DN einer wieder verwendbaren Aktion bzw. Bedingung.
Für die Objekte von Variablen und Werten wurde für diesen Prototyp die folgende Konvention getroffen. Der *Common Name* (CN) einer Variablen die einer Bedingung zugeordnet ist, lautet „ConditionVariable“; der CN einer Aktion zugeordneten Variablen lautet „ActionVariable“. Analog gelten für Werte von Bedingungen und Aktionen die jeweiligen CNs „ConditionValue“ und „ActionValue“.

7.3 Realisierung des Policy Management Tools

Da die Benutzung von phpLDAPadmin recht komplex ist und ein grundlegendes Verständnis über die Funktionsweise von LDAP erfordert, wurde als einfaches administratives Werkzeug das *Policy Management Tool* (PMT) entwickelt. Die Anforderungen an das PMT sind eine einfache Bedienung über einen Browser und die Möglichkeit zum Anlegen und Löschen von Policies. Zur Implementierung des PMT wird die Skriptsprache Perl verwendet, die eine einfache Ausführung auf einem Webserver ermöglicht und die Benutzung des PMT von einem Browser aus gewährleistet.

Mit diesem Tool kann der Administrator Policies anlegen, ansehen und löschen. Information 38 zeigt die Maske des PMT, um eine neue Policy anzulegen.

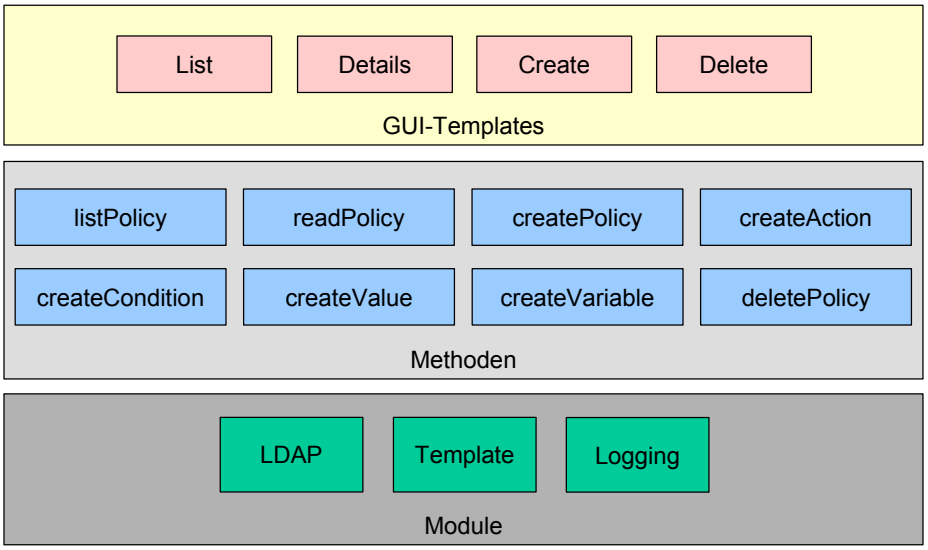


Information 38: Maske zum Anlegen einer neuen Policy

Zur Implementierung des PMT werden HTML-Templates verwendet, die es ermöglichen, Geschäftslogik und Präsentationslogik (GUI) vollständig voneinander zu trennen. Die HTML-Templates sind Dateien, die den Aufbau einer HTML-Seite enthalten. In den Templates sind bestimmte Variablen angegeben, die durch die Geschäftslogik des Perlskripts mit Werten belegt werden.

Außerdem wird eine Loggingkomponente eingesetzt, mit der sich Ausgaben verschiedener Detailstufen am Bildschirm oder in Logdateien ausgeben lassen. Der gewünschte

Loglevel und die Ausgabeart können bequem über eine Konfigurationsdatei angepasst werden. Information 39 zeigt die Architektur des *Policy Management Tools*.



Information 39: Architektur des Policy Management Tools

In der obigen Information 39 sind sowohl die verschiedenen Bausteine der Präsentationslogik zu sehen als auch die für die Geschäftslogik wesentlichen funktionalen Methoden. Die dargestellten Methoden zeigen die Hauptfunktionalität der Komponente. Das LDAP-Modul regelt die Verbindung mit dem LDAP-Verzeichnis.

Es existieren Methoden, um eine Liste der Policies zu erzeugen (listPolicy), die Details einer einzelnen Policy auszulesen (readPolicy), eine Policy zu erzeugen (createPolicy) und zu löschen (deletePolicy). Um eine Policy zu erzeugen, werden Methoden zur Erzeugung von Bedingungen (createCondition), Aktionen (createAction), Variablen (createVariable) und Werten (createValue) benutzt.

Um die Komplexität des PMT für eine rasche Implementierung möglichst gering zu halten, wurden folgende Einschränkungen getroffen:

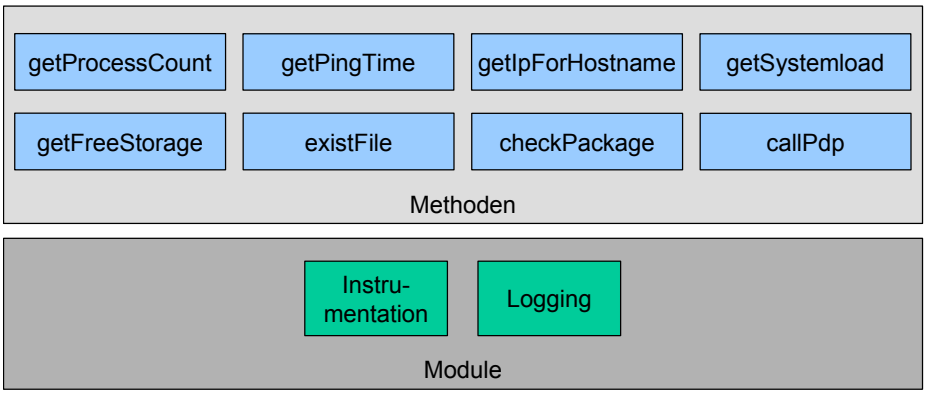
- Anlegen nur mit einer Bedingung und Aktion
- Werte auf Typ „pcelsStringList“ festgelegt
- Modifikation nicht möglich
- Assistenten bzw. Hilfestellung nicht implementiert

Trotz dieser Einschränkungen liefert dieses PMT einen guten ersten Entwurf für ein Policy-Administrationswerkzeug. Die vorhandenen Einschränkungen lassen sich durch Weiterentwicklung des Werkzeugs aufheben. Diese Erweiterungen auszuführen hätte aber den begrenzten zeitlichen und inhaltlichen Rahmen dieser Arbeit gesprengt. Vor allem die Verhinderung von möglichen Policy-Konflikten stellt die größte Herausforderung an ein PMT dar und bedarf noch weiterer Forschungsarbeit.

7.4 Realisierung des Monitoring

Zur Implementierung der Monitorkomponente kommt ebenfalls die Skriptsprache Perl zum Einsatz, da sie sich sehr gut dazu eignet, Ressourcenparameter aus der Ausgabe

von Ressourceninstrumentierungen heraus zu filtern. Die Architektur der Monitorkomponente wird in Information 40 veranschaulicht.



Information 40: Architektur des Monitors

Im Monitor kommt dieselbe Loggingkomponente zum Einsatz, die auch im PMT verwendet wird. Das Logging erfolgt in eine Logdatei (/tmp/monitor.log). So ist jederzeit nachvollziehbar, wann die Monitorkomponente gestartet wurde und welche Symptome von ihr identifiziert wurden. Die Instrumentierung zur Ermittlung von Ressourcenparametern wird von einem Instrumentierungsmodul bereitgestellt, das sich leicht anpassen und erweitern lässt. Beispielhaft wird für den VPN-Dienst ein Rechner mit dem Betriebssystem „Unix“ instrumentiert. Die prototypische Implementierung des Monitors ist in der Lage, die Anzahl gestarteter Prozessinstanzen, die Netzverfügbarkeit, die Namensauflösung, die Systemlast, den freien Speicherplatz und die installierten Pakete zu überwachen.

Die Vorgaben zu jeder Art von Überwachung werden am Anfang des Perlskripts festgelegt. Werden die Vorgaben nicht eingehalten, übermittelt die Methode „callPdp“ die Überschreitung an den PDP, damit dieser eine Policy-Entscheidung treffen kann. Dazu werden die gemessenen Symptome innerhalb einer Überwachungskategorie – wie beispielsweise der Prozessüberwachung – zusammengefasst und mit der Rolle der überwachten Ressource – im Beispiel also „Component&&Process“ – an den PDP gesendet. Dem PDP werden der Systemname des Monitors, die Rolle der Überwachung und die Symptome als Aufrufparameter übergeben.

Die Anzahl der aktuell gestarteten Instanzen eines bestimmten Prozesses wird von der Methode „getProcessCount“ ermittelt. Dabei wird verglichen, ob die aktuell gestartete Prozesszahl mit den Vorgaben übereinstimmt. Um die Prozesszahl zu ermitteln, wird der Unix-Befehl „ps“ verwendet, der eine Tabelle der gestarteten Prozesse anzeigt. Diese Tabelle wird entsprechend formatiert und ausgewertet, um die entsprechenden Informationen zu erhalten.

Auch die Netzverfügbarkeit der Systeme „iravpn“ und „iraldap“ wird vom Monitor überwacht. Mit der Methode „getPingTime“ wird die durchschnittliche Pingzeit von drei Pings der Systeme in Millisekunden ermittelt. Das erlaubte Maximum sind 500 Millisekunden Pingzeit, bevor ein Symptom festgestellt wird. Zur Ermittlung der Pingzeiten wird das Kommando „ping“ verwendet und dessen Ausgabe entsprechend verarbeitet.

Der Monitor überwacht außerdem die korrekte Namensauflösung von Hostnamen zu IP-Adressen für die Systeme „iravpn“ und „iraldap“. Die Methode „getIpForHostname“ ermittelt die IP-Adresse, zu der der gegebene Hostname aufgelöst wird. Entspricht diese IP-Adresse nicht der im Skript definierten Vorgabe, wird ein Symptom aufgezeichnet. Um den Hostnamen in eine IP-Adresse aufzulösen, wird das Kommando „nslookup“ aufgerufen und das Ergebnis abgefragt.

Die Methode „getSystemload“ überprüft die Systemlast des Systems, auf dem der Monitor ausgeführt wird. Als maximal zulässige Systemlast ist ein Wert von 10 vorgegeben. Die mittlere Systemlast innerhalb einer Minute, die durch das Kommando „uptime“ ausgegeben wird, dient hier als Auswertungsgrundlage.

Von der Methode „getFreeStorage“ wird der geringste freie Speicherplatz des Systems, auf dem der Monitor ausgeführt wird, zurückgegeben. Dazu werden sämtliche eingehängten Datenträger mit dem Befehl „df“ abgefragt und die geringste freie Kapazität ermittelt. Das Minimum für freien Speicherplatz beträgt 5 % der Gesamtkapazität des betreffenden Datenträgers.

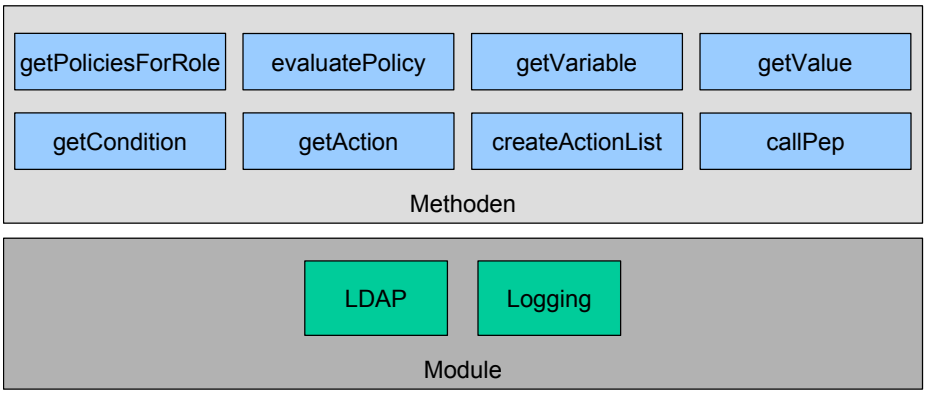
Zur Überprüfung, ob Log- und Konfigurationsdateien existieren, dient die Methode „existFile“. Die Methode prüft mit dem „ls“ Kommando, ob das angegebene File existiert oder nicht. Im Monitor sind die zu prüfenden Log- und Konfigurationsdateien angegeben.

Mit der Methode „checkPackage“ kann der Monitor prüfen, ob bestimmte Komponenten installiert sind. Dazu fragt diese Methode mit dem Paketmanager „rpm“ – der standardmäßig auf Redhat basierten Unixsystemen eingesetzt wird – alle installierten Pakete ab und prüft, ob das geforderte Paket vorhanden – also installiert – ist.

Methoden zur Überwachung der Voraussetzungen zum Betrieb von Komponenten wurden aus zeitlichen Gründen nicht mehr implementiert.

7.5 Realisierung des PDP

Für die Umsetzung des PDP wurde die Programmiersprache „Java“ von Sun Microsystems verwendet. Durch diese Entscheidung ist der PDP auf nahezu jeder Plattform einsetzbar, was eine einfache Skalierbarkeit dieser zentralen Komponente ermöglicht. Information 41 zeigt die Architektur des PDPs.



Information 41: Architektur des Policy Decision Points

Auch der PDP verwendet eine Loggingkomponente, damit die Ausführung des PDP in eine Logdatei (/tmp/pdp.log) protokolliert werden kann. Außerdem benötigt der PDP noch eine LDAP-Komponente, damit eine Kommunikation mit dem LDAP-Verzeichnis – dem PolicyRepository – überhaupt erst möglich wird. Die Java-Plattform bietet dazu die Komponente JNDI an, die einen einheitlichen Zugriff auf verschiedenste Namens- und Verzeichnisdienste bietet.

Aufgabe des PDPs ist es, anhand des vom Monitor übermittelten Symptoms eine Aktionsliste zu erzeugen und an den PEP zur Ausführung zu übermitteln. Um diese Aufgabe auszuführen, werden im PDP diverse Methoden definiert. Der genau Ablauf und der Aufruf der einzelnen Methoden soll in den folgenden Abschnitten verdeutlicht werden.

Der PDP wird vom Monitor über Fehlersymptome – und die Rolle der Komponente auf denen diese gemessen wurden – informiert und muss zunächst mit der Methode „getPoliciesForRole“ alle der Rolle zugeordneten aktivierten Policies aus dem Policy-Repository abrufen. Danach wird diese Liste von Policies eine nach der anderen mit der Methode „evaluatePolicy“ ausgewertet, um zu prüfen, ob die Bedingungen der jeweiligen Policy zutreffen und entsprechend die in dieser Policy definierten Aktionen auszuführen sind. Nun ist der PDP im Besitz einer Liste aller auszuführenden Policies. Mit Hilfe der Methode „createActionList“ wird für jede Policy eine Liste von Aktionen entsprechend ihrer Reihenfolge erzeugt. Diese Listen werden nacheinander mit der Methode „callPep“ zur Ausführung an den PEP übermittelt. Zusätzlich zu den reinen Aktionen wird dem PEP auch das Symptom und der Messort mitgeteilt, damit der PEP im Falle einer Benachrichtigung auch diese Informationen weiterleiten kann.

In der Methode „evaluatePolicy“ steckt die höchste Komplexität des PDPs verborgen, denn dort erfolgt die Auswertung einer Policy entsprechend der IETF-Vorgaben. So wird zunächst durch die Methode „getCondition“ die Variable, der Wert, die Gruppennummer und die Negationsangabe zu jeder Bedingung einer Policy abgerufen. Danach werden die Policies entsprechend der Gruppennummern sortiert und der Wahrheitswert jeder Gruppe ausgewertet. Ist die Policy in KNF angegeben, werden innerhalb einer Gruppe die Bedingungen mit logischem ODER verknüpft und die verschiedenen Gruppen dann mit logischem UND. Genau umgekehrt verhält es sich, wenn die Policy in DNF angegeben ist. So erhält die gesamte Policy einen Wahrheitswert, entweder trifft sie zu und ihre Aktionen müssen ausgeführt werden oder nicht. Policies lassen sich

auch zu Testzwecken aktivieren, dann können sie zwar wahr werden, aber die enthaltenen Aktionen werden nicht ausgeführt.

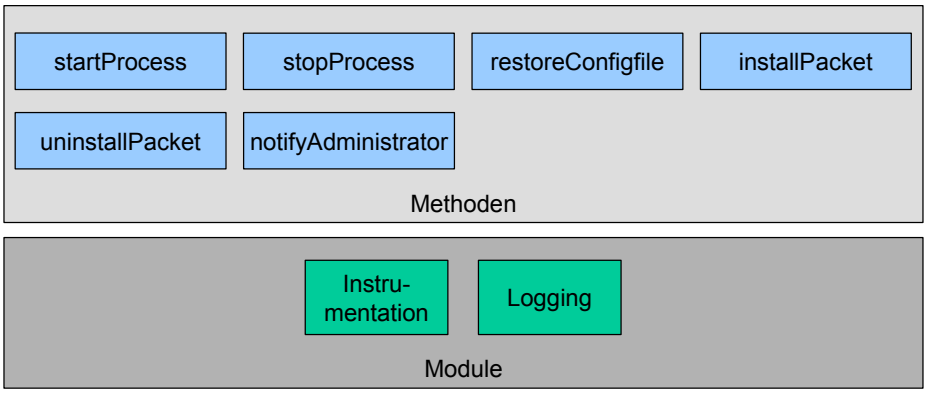
Wenn die zur Ausführung bestimmten Policies ermittelt sind, erstellt die Methode „createActionList“ die Liste von Aktionen entsprechend ihrer Reihenfolge zu jeder Policy. Dazu werden die Variablen, die Werte und die Reihenfolgewerte der Aktionen einer Policy mit der Methode „getAction“ ausgelesen. Danach werden sie nach den Reihefolgewerten sortiert und mit der Methode „callPep“ an den PEP übermittelt.

Die Methoden „getVariable“ und „getValue“ dienen den Methoden „getAction“ und „getCondition“ als Hilfe, um Variable und den Wert aus Bedingung bzw. Aktion auszulesen. Auch wieder verwendbare Aktionen und Bedingungen können problemlos von den Methoden „getAction“ und „getCondition“ ausgelesen werden.

Lediglich die Auswertung von CompoundPolicyActions, CompoundPolicyConditions, PolicyGroups und PolicySets wurde bisher noch nicht im PDP implementiert. Allerdings wurden diese Elemente auch nicht in den entsprechenden Policies für den VPN-Dienst verwendet. Der hier implementierte PDP bietet jedoch alle grundlegenden Funktionen, um PolicyRules auszuwerten und korrekt priorisierte Aktionslisten zu erzeugen. Mit dem PDP lassen sich sogar komplexe Policies auswerten, die verschiedene – auch wieder verwendbare – Bedingungen enthalten können.

7.6 Realisierung des PEP

Ebenso wie für den PDP kommt zur Implementierung des PEP die Programmiersprache Java zum Einsatz. Durch den Einsatz von Java ist ein Plattformunabhängiger Einsatz des PEP möglich. Zunächst ist jedoch nur eine Implementierung für das Bestriebssystem Unix vorgesehen. In Information 42 wird die Architektur des PEP dargestellt.



Information 42: Architektur des Policy Enforcement Points

Um die Protokollierung der Aktionen des PEPs zu ermöglichen, wird – wie die anderen Komponenten auch – eine Loggingkomponente verwendet, mit deren Hilfe eine Logdatei (/tmp/pep.log) erzeugt wird. Um die Aktionen auf der Ressource ausführen zu können, benötigt der PEP eine entsprechende Ressourceninstrumentierung.

Mit den Methoden „startProcess“ und „stopProcess“ können nach Bedarf Prozesse gestartet oder gestoppt werden. Das Starten und Stoppen wird mit den Startskripten im Verzeichnis „/etc/init.d“ vorgenommen, die üblicherweise im Unix-Umfeld zum Einsatz kommen.

Die Methode „restoreConfigfile“ stellt eine angegebene Konfigurationsdatei anhand einer Vorlage wieder her. Dazu wird mit dem „cp“ Befehl die Vorlage der Konfiguration aus einem Vorlagenverzeichnis an die entsprechende Stelle kopiert.

Installation und Deinstallation von Softwarekomponenten werden von den Methoden „installPacket“ und „uninstallPacket“ durchgeführt. Die Methoden verwenden dazu den RPM-Paketmanager des Radhat-Betriebssystems.

Um den Administrator zu benachrichtigen, wird die Methode „notifyAdministrator“ verwendet. Sie teilt dem angegebenen Administrator per E-Mail den aufgetauchten Fehler, die bisher ausgeführten Aktionen und deren Ausführungsstatus mit. Um die E-Mail zu versenden, wird der Unixbefehl „mail“ verwendet.

Der PEP beendet sich selbst mit einem so genannten Exit-Code, um dem den PEP aufrufenden Programm anzuzeigen, ob alle Aktionen erfolgreich ausgeführt wurden. Hat der Code den Wert „0“, sind alle Aktionen erfolgreich ausgeführt worden. Ist der Wert größer als „0“, zeigt der Code die Anzahl der aufgetretenen Fehler an. Durch diesen Mechanismus kann der PDP entscheiden, was zu tun ist, falls der PEP die geforderten Aktionen nicht ausführen konnte.

7.7 Zusammenfassung

Der Prototyp zeigt, wie sich das policy-basierte Management nach Vorgaben der IETF durch Erweiterung der Achitektur um eine Monitorkomponente auch auf Ressourcen-ebene durchführen lässt. Mit der IETF-Architektur wurden bisher im Wesentlichen Szenarien auf Netzwerkebene behandelt. Die grundlegenden Konzepte lassen sich aber auch auf Ressourcen- bzw. Dienstebene verwenden und müssen nicht neu entwickelt werden. Der Prototyp hat gezeigt, dass durch Erweiterung des vorhandenen Konzepts ein breiteres Einsatzgebiet für policy-basiertes Management erschlossen werden kann, denn analog zum exemplarisch verwendeten VPN-Dienst lassen sich auch etliche andere Dienste mit dem hier entwickelten Managementsystem überwachen und steuern.

Sofern keine neuen Fehlerzustände und Aktionen definiert werden sollen, lässt sich das Managementsystem flexibel über die Eingabe, Änderung oder Löschung von Policies konfigurieren. Für den Administrator ist die Bedienung des Systems recht einfach, da die einzige Benutzerschnittstelle zur Administration das PMT ist. Die Monitor-komponente muss lediglich durch einen zyklischen Mechanismus – wie beispielsweise der „crontab“ unter Unix – automatisch in den gewünschten Zeitabständen gestartet werden. Die restliche Kommunikation zwischen den Komponenten verläuft vollständig automatisch.

Die modulare Architektur erleichtert die künftige Erweiterung der einzelnen Komponenten enorm, falls neue Fehlerzustände oder Aktionen aufgenommen werden sollen oder die Vorgaben des Monitors anzupassen sind. Neue Funktionalität kann leicht integriert werden, indem neue Methoden eingeführt und in den Ablauf der Komponente eingebunden werden.

Durch den Einsatz der Programmiersprache Java und der Skriptsprache Perl ist der Einsatz des Prototyps auf nahezu jeder Plattform möglich. Der Instrumentierung wurde speziell für das Unix Betriebssystem entwickelt und ist deshalb nicht ohne weiteres auf anderen Plattformen verwendbar. Jedoch nur die Paketinstallation und -deinstallation

des PEP hängt von dem RPM-Paketmanager, der in der Distribution der Firma Redhat zum Einsatz kommt, ab. Ansonsten sollten sich die entwickelten Komponenten auf jedem beliebigen Unixsystem verwenden lassen.

Auch wenn von diesem Managementsystem noch nicht alle Fehlerzustände automatisch behoben werden können, so hilft das System dem Administrator enorm, das Auftreten eines Fehlers festzustellen. Der Administrator erhält im Falle eines Fehlers eine Benachrichtigung, was im Vergleich zur ständigen manuellen Suche nach möglichen Fehlern wesentlich zeitsparender ist. Außerdem hilft ihm die Benachrichtigung auch dabei, den Fehler schon im Vorfeld einzugrenzen und gezielt nach einer Lösung für die Ursache des Problems zu suchen. Insgesamt verringert sich die Zeit eines Ausfalls durch den Einsatz des Managementsystems um ein Vielfaches.

7.7.1 Einschränkungen

Um komplexere Policies zu erstellen – also Policies, die mehr als eine Bedingung und Aktion enthalten – muss das PMT erweitert werden. Mit der bisherigen Implementierung lassen sich jedoch gute Vorlagen für Policies erzeugen, die mit dem Werkzeug phpLDAPadmin um weitere Bedingungen und Aktionen ergänzt werden können.

Policy-Konflikte werden derzeit nicht vom PMT erkannt und behandelt. Es ist also prinzipiell möglich, dass Policy-Konflikte entstehen können. Die Lösung solcher Konflikte [Ke04] ist allerdings eine sehr komplexe Aufgabe, die jedoch in das vorhandene PMT integriert werden kann.

Die Auswertung von PolicySets, PolicyGroups und CompoundPolicy-Objekten ist nicht im PDP integriert. Einzelnen PolicyRules mit beliebig vielen Bedingungen und Aktionen lassen aber bereits auch die Erzeugung recht komplexer Policy-Objekte zu.

Das entwickelte Managementsystem verwendet keine Gültigkeitsperioden für Policies. Es werden also weder Gültigkeitsperioden erzeugt, noch werden diese ausgewertet. Die entsprechende Funktionalität kann durch Weiterentwicklung nachgerüstet werden, war aber zur Veranschaulichung des Konzepts nicht relevant.

Momentan kommunizieren die Komponenten durch direkte Aufrufe. Damit die Kommunikation der Komponenten auf entfernten Rechnern nicht durch SSH-Verbindungen getunnelt werden muss, wäre der Einsatz eines Kommunikationsprotokolls wünschenswert. Unter anderem muss dieses Protokoll dann auch die Sicherheit der kommunizierenden Komponenten gewährleisten. So ist es notwendig, dass die Komponenten sich gegenseitig authentifizieren, damit nicht ein Angreifer beispielsweise einen PEP aufrufen kann, um schadhafte Aktionen auszuführen. Außerdem müsste die Kommunikation zwischen den Komponenten verschlüsselt sein, um Abhören und das wiederholte Senden von abgefangenen Paketen zu unterbinden.

7.8 Durchgeführte Tests

Auf dem virtuellen VMware-Testsystem „iraldap-test“ konnten alle Komponenten des Prototyps ausführlich getestet werden, ohne den produktiven Betrieb des VPN-Dienstes zu beeinflussen. Die Tests haben verifiziert, dass die Komponenten in der Praxis so arbeiten, wie sie konzipiert wurden.

Ein Durchlauf des Monitors auf dem Testsystem dauert ca. 6,3 Sekunden, wenn alle Überwachungsarten ausgeführt und keine Symptome festgestellt werden. Wurde beispielsweise der Ausfall eines Prozesses bemerkt und der Neustart und eine Benachrichtigung veranlasst, dauert der gesamte Vorgang ca. 2,3 Sekunden länger. Information 43 zeigt die vorgenommenen Messungen am Prototyp.

Keine Symptome:

real	0m6.293s
user	0m0.261s
sys	0m1.921s

Neustarten von Testprozess A:

real	0m8.518s
user	0m0.425s
sys	0m3.638s

Neustarten von Testprozess B:

real	0m8.681s
user	0m0.498s
sys	0m3.713s

Neustarten von Testprozess A+B:

real	0m9.692s
user	0m0.544s
sys	0m4.505s

Information 43: Laufzeitmessung am Prototyp

Die Messungen wurden mit dem Unixbefehl „time“ durchgeführt. Dabei wird die Laufzeit des aufgerufenen Programms in drei verschiedene Kategorien unterteilt. Die Kategorie „real“ misst, wie lange die Ausführung des aufgerufenen Programms – also vom Start bis zu dessen Beendigung – tatsächlich gedauert hat. Wartezeiten auf Ein- oder Ausgaben werden in der Kategorie „user“ erfasst. In der Kategorie „sys“ wird die Zeit aufgenommen, die das System an dem Programm gearbeitet hat.

Die Laufzeit des Prototyps hängt natürlich sehr davon ab, wie lange der Neustart des neu zu startenden Prozesses generell dauert. Für die Messungen wurden zwei Testprozesse herangezogen, die sich sehr schnell starten lassen (ca. 0,5 Sekunden). Bei einem neu zu startenden Prozess verlängert sich die reine Systemzeit des Managementsystems um 1,7 Sekunden gegenüber dem Durchlauf ohne fehlerhafte Symptome (wenn 5 Policies auszuwerten sind). Müssen sogar zwei Prozesse neu gestartet werden, steigt die Systemzeit um weitere 0,9 Sekunden.

Treten innerhalb einer Überwachungskategorie mehrere Symptome auf, werden diese etwas schneller gelöst, da keine erneute Anfrage an das *Policy Repository* gestellt werden muss. In der obigen Messung benötigt deshalb der Neustart eines weiteren Prozesses 0,9 statt 1,7 Sekunden, was einer Beschleunigung des Vorgangs um 0,8 Sekunden entspricht. Insgesamt lässt sich das in etwa lineare Laufzeitverhalten des Prototyps als sehr gut bezeichnen, da es aufgetretene Symptome innerhalb weniger Sekunden erkennt und beseitigt. Eine schnelle Lösung des Problems garantiert kurze Ausfallzeiten und geringe Ausfallkosten und macht das hier entwickelte Managementsystem zu einem wertvollen Helfer.

7.9 Installation und Betrieb

Um den Prototyp auf einem System zu installieren, muss es folgende Anforderungen erfüllen:

- Betriebssystem Unix (zur Nutzung der automatischen Installation wird der RPM-Paketmanager benötigt)
- Webserver (z.B. Apache)
- OpenLDAP Server Version 2 (inkl. Schemas “cim.schema”, “pcim.schema”, “pcels.schema”)
- Perl Version 5.8 (inkl. Module “Log::Log4perl”, “Net::LDAP”, “HTML::Template”)
- Java Runtime Environment Version 5.0

Auf dem System ist nun ein Ordner „/opt/policy“ anzulegen, der später die einzelnen Komponenten enthalten soll. In diesem Verzeichnis muss ein Ordner „rpms“ und ein Ordner „configfiles“ erzeugt werden. Diese Ordner enthalten die Vorlagen für wiederherzustellende Konfigurationsdateien und zu installierende Pakete. Die folgenden Abschnitte beschreiben, wie die einzelnen Komponenten genau zu installieren sind.

7.9.1 PMT

Um das PMT über den Webserver verfügbar zu machen, muss es in die so genannte DocumentRoot des Webservers gelegt werden. Unter dem Betriebssystem Redhat ist dies üblicherweise der Ordner „var/www/html“. Zudem muss die Konfigurationsdatei des Webservers (httpd.conf im Falle des Apache Webservers Version 2) angepasst werden (durch hinzufügen der Option „ExecCGI“), damit die Skripte vom Webserver ausgeführt werden, statt sie dem Browserbenutzer anzuzeigen.

Um die benötigten Module für Perl (s. o.) zu installieren, kann einfach der Befehl „perl –MCPAN -e 'install <Kategorie>::<Paket>'“ eingegeben werden. Die erforderlichen Pakete werden dann automatisch heruntergeladen, kompiliert und in die Perlinstallation integriert.

Im LDAP-Modul des PMTs (modules/Ldap.pm) müssen nun die Daten des LDAP-Servers, wie der Hostname, die LDAP-Wurzel und der Benutzername und das Passwort des Managers eingetragen werden.

Neben der Installation des PMTs wird auch der Einsatz des Werkzeugs phpLDAPadmin empfohlen. Auch dieses Werkzeug muss in die DocumentRoot des Webservers integriert werden, und die Daten des LDAP-Servers sind zu konfigurieren. Durch dieses Werkzeug lassen sich Policies, die mit dem PMT angelegt wurden, einfach um weitere Aktionen bzw. Bedingungen erweitern.

7.9.2 Monitor

Der Monitor wird im Verzeichnis „/opt/policy“ installiert, dazu wird einfach das entsprechende Archiv in diesem Verzeichnis entpackt. Bei dieser Komponente sind lediglich die eingetragenen Vorgaben zu prüfen bzw. zu ergänzen, um beispielsweise weitere Komponenten in die Überwachung aufzunehmen.

Ist der Monitor installiert, muss der Administrator festlegen, wie oft eine Überwachung durchgeführt werden soll, und den Monitor entsprechend in den „crontab“-Mechanismus zur automatischen Ausführung integrieren.

7.9.3 PDP

Auch das Archiv des PDPs ist im Verzeichnis „/opt/policy“ zu entpacken. Hier sind ebenfalls der LDAP-Server, Adresse des Wurzelverzeichnisses und Benutzername und Passwort des Managers zu konfigurieren.

Weitere Anpassungen des PDP sind nicht notwendig.

7.9.4 PEP

Das Archiv des PEPs ist ebenfalls im Verzeichnis „/opt/policy“ zu entpacken. Im PEP sind die Vorgaben zum Verzeichnis mit den Vorlagen der Konfigurationsdateien (/opt/policy/configfiles) und der installierbaren Pakete (/opt/policy/rpms) enthalten. Im Bedarfsfall können diese Verzeichnisvorgaben auch geändert werden.

7.9.5 Betrieb der Komponenten

Zum Betrieb der Komponenten muss lediglich gewährleistet sein, dass der Monitor in regelmäßigen Abständen gestartet wird, um eine Überprüfung der Ressourcenparameter vorzunehmen.

Im Idealfall muss der Administrator nur das PMT bedienen, um Policies einzutragen oder zu löschen, ansonsten läuft das Managementsystem vollständig autonom. Falls zusätzliche Überwachungen gewünscht sind oder sich die Vorgaben ändern, muss auch der Monitor entsprechend angepasst werden. Ebenso ist der PEP anzupassen, falls neue Aktionen spezifiziert werden.

Alle Komponenten schreiben Logdateien. Durch diese Logdateien ist es möglich, ein eventuell fehlerhaftes Verhalten des Managementsystems zu überprüfen und die Fehlerquelle zu identifizieren. Mit den Logdateien können auch diverse Statistiken erstellt und Auswertungen über Häufigkeiten von Fehlern untersucht werden. Durch solche Auswertungen können Berichte über die Verfügbarkeit der Infrastruktur erstellt und im Bedarfsfall an den Kunden übermittelt werden.

8 ZUSAMMENFASSUNG UND AUSBLICK

Hier werden die Ergebnisse der Arbeit ausgewertet, um weitere Forschungsgebiete und künftige Entwicklungen auf dem Gebiet des policy-basierten Managements aufzuzeigen. Es wird hier auch weiterführende Literatur zu diesem Thema angegeben.

In dieser Arbeit wurden erstmals die Policy-Standards der IETF um die Monitorkomponente des IBM-Architekturentwurfs für *Autonomic Computing* (AC) erweitert, um ein policy-basiertes Managementsystem für einen VPN-Dienst der ATIS zu realisieren. Es wurden etablierte Standards zum Management der Netzwerkebene verwendet und erweitert, um dem visionären Gedanken von AC ein Stückchen näher zu kommen und Management auf Dienstebene zu betreiben.

AC im Sinne einer sich selbst konfigurierenden, heilenden, optimierenden und schützenden technischen Infrastruktur ist immer noch Zukunftsmusik. Es existieren zwar viele Ideen, Entwürfe und Produkte, aus wissenschaftlicher Sicht aber fehlen einheitliche Standards, um die existierenden Konzepte zu vereinen. Meist werden in wissenschaftlichen Arbeiten nur kleine Teilaspekte der Fragestellung beleuchtet und Insellösungen geschaffen, ohne bestehende Standards und Konzepte entsprechend zu berücksichtigen und in das eigene Konzept einzubeziehen.

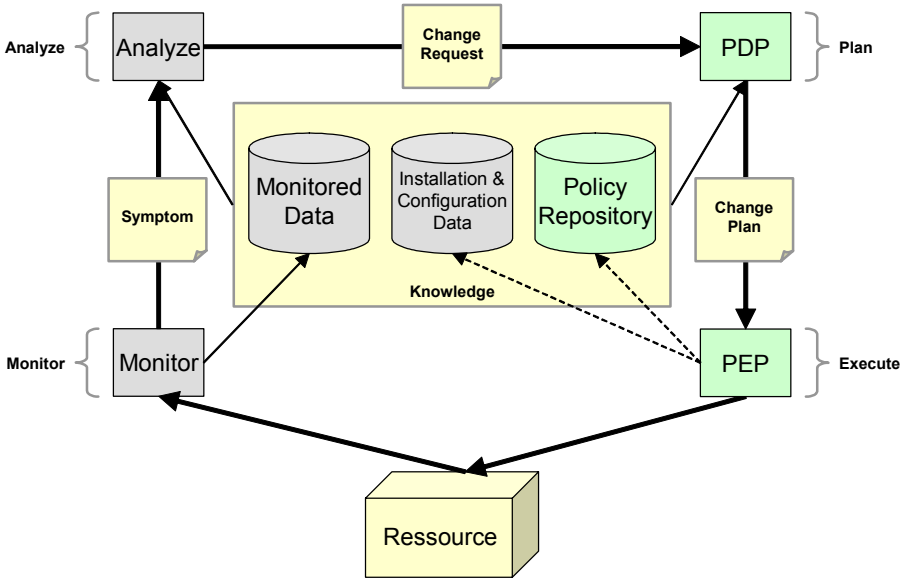
Der hier angesprochenen Problematik wird in dieser Arbeit Rechnung getragen, indem keine Insellösung entwickelt, sondern bisheriger Standards verwendet und erweitert wurden. Um die Fragestellung zu konkretisieren, wurde der VPN-Dienst als exemplarisches Beispiel für einen Dienst gewählt. Zunächst wurde der Dienst genau untersucht, um die zu überwachenden Komponenten und Systeme zu identifizieren. Eine Analyse dieser Ressourcen zeigte, welche Parameter es zu überwachen gilt, um die Verfügbarkeit der technischen Infrastruktur des Dienstes zu gewährleisten. Anhand der Standards der IETF wurden nun die Komponenten und Policies entworfen, die für ein Managementsystem realisiert werden müssen. Die Unzulänglichkeiten dieser Standards im Bezug auf die Überwachung von Ressourcenparameter wurden durch das Hinzufügen einer Monitorkomponente entsprechend des IBM-Architekturentwurfs ausgeglichen. Abschließend wurde erfolgreich ein Prototyp des neu konzipierten Managementsystems implementiert und getestet.

Das hier entwickelte Konzept stellt in Sachen policy-basiertem Managementsystem einen ersten Ansatz dar, um die bestehenden Konzepte in diesem Bereich zu vereinen. So bleiben noch viele weitere Fragestellungen rund um das Thema unbeantwortet, die bis dato noch nicht geklärt wurden.

An dieser Stelle sollen einige Fragestellungen aufgezeigt werden, die sich durch den hier entstandenen Prototyp nun aufwerfen. Die Lösungen der folgenden Fragestellungen sind notwendig, um den Weg hin zum AC weiterzuschreiten.

8.1 Erweiterung der IETF durch den IBM-Ansatz

Der Architekturentwurf von IBM [IBM04] beschäftigt sich mit dem Szenario AC und vermittelt Konzepte, welche Problematiken auftauchen und wie diese angegangen werden können. Information 44 zeigt, wie der hier entwickelte Prototyp künftig erweitert werden müsste.



Information 44: Erweiterte IETF Policy-Architektur

Die Monitorkomponente wurde bereits in den Prototyp integriert. Es zeigt sich aber recht schnell, dass lediglich die Überwachung von Ressourcenparametern nicht ausreicht, um einen Dienst zu managen. Ab einer gewissen Komplexität müssen die auf einer Ressource überwachten Parameter analysiert werden, um festzustellen, ob und wo ein Fehler vorliegt.

Folgendes Beispiel soll den Sachverhalt verdeutlichen: Wenn die Temperaturmessung einer Ressource einen gewissen Schwellenwert überschreitet, übermittelt die Monitorkomponente dieses Symptom an die Analysierungskomponente und schreibt das Messergebnis zusätzlich in eine Datenbank. Die Aufgabe der Analysierungskomponente ist nun, anhand der Mess- und Systemkonfigurationsdaten herauszufinden, ob die entsprechenden Lüfter und die Klimaanlage korrekt arbeiten oder ob ein Fehler im Kühlsystem vorliegt. Wird ermittelt, dass die Klimaanlage defekt ist, wird dieser Fehler an der PDP übermittelt, der anhand bestimmter Policies das Vorgehen zur Behebung des Problems bestimmt und an den PEP zur Ausführung weiterleitet.

Weder die Monitor- noch die Analysekomponente werden von der IETF definiert. Die Monitorkomponente wurde jedoch bereits im hier entstandenen Prototyp integriert, um Fehlerzustände zu identifizieren, die sich direkt an Ressourcenparametern ablesen lassen. Für die Analyse von komplexeren Fehlern muss der Prototyp jedoch um eine Analysekomponente erweitert werden.

Außerdem bleiben von der IETF auch die Datenbanken zur Erfassung von Mess- und Konfigurationsdaten unerwähnt, die wesentlich zur Problemfeststellung und -lösung beitragen. Um den Prototyp weiterzuentwickeln, müssten die vom Monitor gemessenen Daten in eine so genannte „Problem Determination Knowledge“ Datenbank aufgezeichnet werden, damit diese Daten jederzeit von anderen Komponenten – wie der Analysekomponente – ausgewertet werden können. Zusätzlich werden aber auch Daten zur Problemlösung benötigt, die so genannte „Problem Solution Knowledge“. Diese Datenbank enthält Informationen, die zur Lösung eines Problems verwendet werden können. Dazu zählen Daten, die Informationen über die Konfiguration, Installation,

Topologie, usw. enthalten. Dabei gilt es auch zu hinterfragen, ob dieses Wissen nicht schon teilweise in existierenden Datenbanken wie beispielsweise einer *Configuration Management Database* (CMDB) vorhandenen ist. Eine Zusammenfassung solcher Datenbestände zu einem Gesamtwissen, das sich aus Policies, Problemfeststellungs- und -lösungswissen zusammensetzt, muss vorgenommen werden. Um dieses Ziel zu erreichen, ist noch einige Forschungsarbeit notwendig, damit Standards und Konzepte für eine solche zentrale Wissensdatenbank erstellt werden. Die Standardisierung der gespeicherten Daten und deren Struktur ist deshalb so wichtig, damit die Interoperabilität zwischen Produkten verschiedener Hersteller gewährleistet werden kann.

Um den Regelkreis des AC zu schließen, müsste der PEP so erweitert werden, dass Aktionen zum Erstellen, Ändern und Löschen von Wissensdaten ausgeführt werden können. Entdeckt der PDP also, dass eine Policy geändert werden muss, so delegiert er diese Aufgabe zur Ausführung an den PEP. Nur durch einen solchen Mechanismus ist ein Management möglich, das sich selbst adaptiv an sich ändernde Anforderungen anpasst, also vollständig autonom arbeitet.

8.2 Policies zur Steuerung anderer Komponenten

In dieser Arbeit wurde der Prototyp eines Monitors umgesetzt, der die Vorgaben für die Vergleichsoperationen in der Komponente selbst speichert. Um Vorgaben zu ändern, muss der Monitor entsprechend umkonfiguriert werden. Durch Policies könnte dieses Problem gelöst werden.

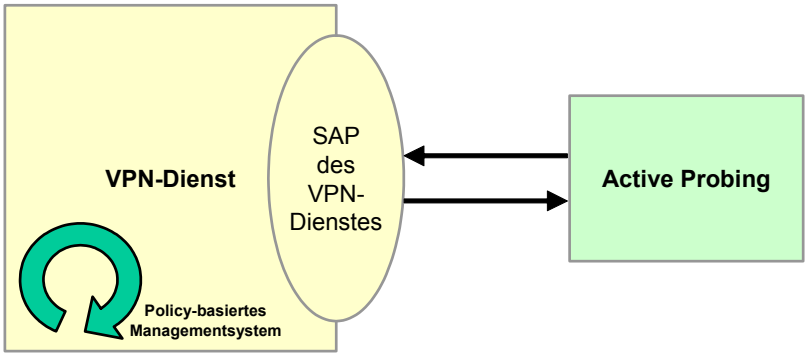
Die Vorgaben für den Monitor müssten als spezielle Policies für diese Komponente gespeichert werden. Die Zuordnung von solch speziellen Monitorpolicies lässt sich leicht über das Rollenattribut realisieren. Jeder Monitor stellt dann, genauso wie der PDP auch, eine Anfrage an das *Policy Repository*. Dazu übermittelt er die Rolle „Monitor&& <Systemname des Monitors>“ und erhält dann alle Policies (also die Vorgaben) für den entsprechenden Monitor aus dem Repository. Nun kann der Monitor die gemessenen Ressourcenparameter mit den übermittelten Policies vergleichen, um festzustellen, ob Policies zutreffen und somit Symptome erzeugen und an den PDP geleitet werden müssen.

Lediglich zur Erweiterung der überwachbaren Ressourcenparameter wäre die Monitorkomponente dann noch selbst anzupassen. Anpassungen der Vorgaben könnten beliebig durch Hinzufügen, Ändern oder Löschen von Policies vorgenommen werden.

Analoges gilt für die Konzeption einer Analysekomponente. Sie kann die Analyse der Symptome auch anhand von Policies vornehmen. Dazu ist das Rollenattribut dieser Policies entsprechend zu belegen.

8.3 Ergänzende Überwachung durch Active Probing

Das in dieser Arbeit entwickelte policy-basierte Managementsystem gewährleistet die Verfügbarkeit der internen technischen Ressourcen des VPN-Dienstes. Die Verfügbarkeit aller technischen Ressourcen ist zwar die Voraussetzung, aber keine Garantie für die Funktionsfähigkeit des VPN-Dienstes im Gesamten. Um den VPN-Dienst als Ganzes auf Verfügbarkeit zu prüfen, muss der Dienst – wie ihn auch der Kunde nutzt – am Dienstzugangspunkt überwacht werden. Information 45 stellt policy-basierte Managementsystem – die Überwachung von „innen“ – und *Active Probing* [HS+04] – die Überwachung von „außen“ – gegenüber.



Information 45: Active Probing prüft den Dienst von „außen“

Eine Messung am Dienstzugangspunkt (SAP) gibt schnell Aufschluss über die Verfügbarkeit des Dienstes. Sollte der Dienst verfügbar sein, können am Dienstzugangspunkt auch andere Parameter wie beispielsweise Durchsatz und Antwortzeit gemessen werden. Ist der Dienst jedoch nicht am Dienstzugangspunkt verfügbar, kann das *Active Probing* kaum Hinweise dafür liefern, welche internen Komponenten diesen Fehler verursachen. Die internen Komponenten des VPN-Dienstes bleiben für das *Active Probing* hinter dem Dienstzugangspunkt verborgen.

Andererseits lassen sich durch das *Active Probing* beispielsweise andere Symptome – wie beispielsweise das Ansteigen der Antwortzeit – erkennen und an den PDP übermitteln. So kann das *Active Probing* auch als alternative Monitorkomponente für das policy-basierte Managementsystem verwendet werden.

Durch diese unterschiedlichen Eigenschaften der Dienstüberwachung ergänzen sich policy-basiertes Managementsystem und *Active Probing* sehr gut. Zusammen eingesetzt ist jederzeit nachvollziehbar, ob der VPN-Dienst funktioniert bzw. welche Ressourcen im Falle eines Ausfalles betroffen sind.

ANHÄNGE

Index

Informationen

Literatur

[BG+01] Torsten Braun, Manuel Guenter, Ibrahim Khalil: „Management of Quality of Dervice Enabled VPNs“, IEEE Communications Magazine:90-98, Mai 2001

[CIM] Distributed Management Task Force, Inc.: „Common Information Model (CIM) Standards”, http://www.dmtf.org/standards/cim

[CIM-LDAP] CIM Core Model V2.5 LDAP Mapping Specification, DSP0123, DMTF, April 2002

[FT+03] Paris Flegkas, Panos Trimintzios, George Pavlou, Antonio Liotta: „Design and Implementation of a Policy-based Resource Management Architecture“, IEEE Communications Magazine:215-229, März 2003

[GY+03] Xin Guo, Kun Yang, Alex Galis, Xiaochun Cheng, Bo Yang, Dayou Liu: „A Policy-based Network Management System for IP VPN”, Proc. of the IEEE International Conference on Communication Technology (ICCT 2003): 1630-1633, April 2003

[HS+04] Andreas Hanemann, Martin Sailer, David Schmitz: „Assured Service Quality by Improved Fault Management”, November 2004

[IBM04] IBM, Autonomic Computing, White Paper: „An architectural blueprint for autonomic computing“, http://www.ibm.com/autonomic, Oktober 2004

[JR+03] J. Jason, L. Rafalow, E. Vyncke: „IPsec Configuration Policy Information Model”, RFC3585, August 2003

[Ke04] Bernhard Kempter: „Konfliktbehandlung im policy–basierten Management mittels a priori Modellierung“, Dissertation, Ludwig-Maximilians-Universität München, Juli 2004

[KL04] Alexander Keller, Heiko Ludwig: „Policy-basiertes Management: State-of-the-Art und zukünftige Fragestellungen”, Praxis der Informationsverarbeitung und Kommunikation 27(2):93-102, Februar 2004

[KS05] S. Kent, K. Seo: „Security Architecture for the Internet Protocol”, RFC4301, Dezember 2005

[ME+01] B. Moore, E. Ellesson, J. Strassner, A. Westerinen: „Policy Core Information Model -- Version 1 Specification”, RFC3060, Februar 2001

[Mo03] B. Moore, Ed.: „Policy Core Information Model (PCIM) Extensions”, RFC3460, Januar 2003

[PR+05] M. Pana, Ed., A. Reyes, A. Barba, D. Moron, M. Brunner: „Policy Core Extension Lightweight Directory Access Protocol Schema (PCELS)", RFC41014, Juni 2005

[SM+04] J. Strassner, B. Moore, R. Moats, E. Ellesson: „Policy Core Lightweight Directory Access Protocol (LDAP) Schema", RFC3703, Februar 2004

[WS+01] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser: „Terminology for Policy-Based Management", RFC3198, November 2001

[YP+00] R. Yavatkar, D. Pendarakis, R. Guerin: „A Framework for Policy-based Admission Control", RFC2753, Januar 2000

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
Active Probing	Bezeichnet nach [HS+04] die Überwachungskomponente, die einen Dienst an dessen SAP überwacht.
ATIS	Abteilung Technische Infrastruktur Einrichtung der Fakultät für Informatik, die für den Betrieb der technischen Infrastruktur (beispielsweise Netzwerk, Mailsystem, etc.) verantwortlich ist.
Authentifizierung	Authentifizierung bezeichnet den Vorgang, die Identität eines Subjekts mit Hilfe von bestimmten Merkmalen zu überprüfen. Dies geschieht oft durch Passwörter, digitale Signaturen, Fingerabdrücke oder einem beliebigen anderen Berechtigungsnachweis. Bei einer Identitätsüberprüfung gibt es immer einen Teilnehmer, der sich authentisiert und einen, der diesen authentifiziert. C&M: Identity Management
Authentisierung	Die Authentisierung bezeichnet den Vorgang, seine eigene Identität nachzuweisen. Dies geschieht oft durch Passwörter, digitale Signaturen, Fingerabdrücke oder einem beliebigen anderen Berechtigungsnachweis geschehen. Bei einer Identitätsüberprüfung gibt es immer einen Teilnehmer, der sich authentisiert und einen, der diesen authentifiziert. C&M: Identity Management
Availability	Ability of a component or service to perform its required function at a stated instant or over a stated period of time. It is usually expressed as the availability ratio, i.e. the proportion of time that the service is actually available for use by customers within the agreed service hours. C&M: Service Management
Business Process	A group of business activities undertaken by an organization in pursuit of a common goal. Typical business processes include receiving orders, marketing services, selling products, delivering services, distributing products, invoicing for services, accounting for money received. A business process usually depends upon several business functions for support, e.g., IT, personnel, and accommodation. A business process rarely operates in isolation, i.e., other business processes will depend on it and it will depend on other processes. [OGC-ITIL00]

Geschäftsprozess
Ein Geschäftsprozess ist eine Abfolge von Aktivitäten, die der Erzeugung eines Produktes oder einer Dienstleistung dienen. Er wird durch ein oder mehrere Ereignisse gestartet und durch ein oder mehrere Ereignisse abgeschlossen. Es liegt eine Organisationsstruktur zu Grunde.

C&M: General

CIM

Common Information Model
The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment. A fundamental taxonomy of objects is defined - both with respect to classification and association, and with respect to a basic set of classes intended to establish a common framework [DMTF-CIM2.4].

DMTF-Standard zur Beschreibung von Management-informationen.

C&M: Application Management

Control Loop

For a system component to be self-managing, it must have an automated method to collect the details it needs from the system; to analyze those details to determine if something needs to change; to create a plan, or sequence of actions, that specifies the necessary changes; and to perform those actions. When these functions can be automated, an intelligent control loop is formed.

C&M

Cooperation & Management
Name of an academic research group at the University of Karlsruhe (TH).

Name der an der Universität Karlsruhe (TH) angesiedelten Forschungsgruppe.

C&M: General

Daemon

Als *Daemon* bzw. Dämon bezeichnet man unter Unix und seinen Derivaten ein Programm, das im Hintergrund abläuft und bestimmte Dienste zur Verfügung stellt.

DiffServ

Differentiated Services (RFC 2474, RFC 2475) ist ein Quality of Service (QoS) Verfahren zur Priorisierung von IP-Datenpaketen.

Ein herkömmliches IP-Netzwerk unterscheidet nicht zwischen verschiedenen Anwendungen, die im Netzwerk unterwegs sind. Dem Risiko von Engpässen wurde bislang deshalb durch Bereitstellung großer (und teurer) Kapazitäten begegnet. Durch das neue Verfahren wird jedes IP-Paket zur Feststellung der Paketwichtigkeit geprüft.

Im Gegensatz zu anderen QoS Verfahren (wie IntServ mit RSVP) wird die Priorität eines IP-Paketes bereits vom Sender bestimmt. Die Router auf dem Weg zum Empfänger entscheiden allein anhand dieser Angabe über die bevorzugte Weiterleitung zum Empfänger.

Directory

A directory is a repository of information about objects that is used to provide directory services. The directory may be distributed and the information in the repository may be accessible via LDAP or other protocols. There are fundamental differences in the data models of directories and (relational) databases. Using relational databases, objects have a complex relationship to one another. Queries can be based on complex relationship between objects. This cannot be provided by a directory. On the other hand objects are essentially independent in the directory, and linked into a hierarchy. There is a fixed core schema for naming basic types of objects and managing them in a hierarchy. This common core is the key for the internal linking in a directory. Directories are optimized for read and search operations, they do not focus on transactions and other database specific features.

C&M: Identity Management

DMTF

Distributed Management Task Force
An open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications.
The DMTF is an industry organization leading the development of management standards and integration technology for enterprise and Internet environments. One of the standards developed by the DMTF is the Common Information Model (CIM).

C&M: Application Management

DN

Distinguished Name
A Distinguished Name is a unique name that unambiguously identifies a single entry. DNs are made up of a sequence of relative distinguished names [TE+04].

C&M: Identity Management

Eclipse	Eclipse ist ein Open-Source-Framework zur Entwicklung von Rich-Client-Applikationen. Die bekannteste Verwendung ist die Nutzung als Entwicklungsumgebung (IDE), diese umfasst unter der Bezeichnung Eclipse SDK die Eclipse Platform, Werkzeuge zur Java-Entwicklung (Java Development Tools JDT) und die Umgebung zur Entwicklung von Eclipse-Plugins (Plug-in Development Environment PDE).
GUI	*Graphical User Interface* Grafische Benutzeroberfläche C&M: General
HTML	*HyperText Markup Language* Vom W3C standardisierte Auszeichnungssprache, die die Struktur und die Darstellung eines Dokuments beschreibt. C&M: General (K&D)
HTTP	*HyperText Transfer Protocol* Application protocol used in the World Wide Web. Internet-Anwendungsprotokoll, das innerhalb des World Wide Web benutzt wird. C&M: General (K&D), Application Management
IBM	*International Business Machines* (IBM) ist eines der ältesten IT-Unternehmen, das anfangs mit Lochkartensortiermaschinen und später mit Großrechnern, z. B. der Serie 360, eine Markt beherrschende Stellung einnahm. Der Firmensitz befindet sich in Armonk bei North Castle im US-Bundesstaat New York.
IDE	*Integrated Development Environment* C&M: Application Management
IEEE	*Institute of Electrical and Electronic Engineers* Standardisierungsorganisation C&M: General (K&D)
IETF	*Internet Engineering Task Force* Organisation, die maßgeblich am Standardisierungsprozess der Internet-Standards beteiligt ist. C&M: General (K&D)
Instrumentation	Additional interface of an application of application component to access their management information and operations

C&M: Application Management

IntServ — *Integrated Services* ist ein feingranulares Quality of Service (QoS) Verfahren zur Parametrisierung von IP-Datenpaketen. Im Gegensatz zum DiffServ-Verfahren werden die Ressourcen für einzelne Verbindungen und nicht Verkehrsklassen angefordert.

Als Protokoll für den Aufbau der Ressourcenreservierung wird das Resource Reservation Protocol (RSVP) verwendet. Ein Router kann über seine verfügbaren Kapazitäten Buch führen und auf eine Anfrage nach QoS-Zusicherung mit einer positiven oder negativen Antwort reagieren. Im Falle einer positiven Antwort garantiert der Router eine Einhaltung der geforderten Merkmale für die gesamte Sitzungsdauer. Einmal gemachte Zusicherungen dürfen nicht beschnitten werden.

IP — *Internet Protocol*
Verbindungsloses Protokoll der Vermittlungsschicht.

C&M: General (K&D)

IT — *Information Technology*
Informationstechnik

C&M: General

IT Infrastructure — The sum of an organisation's IT related hardware, software, data telecommunication facilities, procedures and documentation.
[ITIL-Glossar, 16.11.2004,
www.serview.de/content/itsm/glossar]

IT-Infrastruktur
Die Menge der innerhalb einer Organisation, mit IT zusammenhängenden Hardware, Software, Daten, Kommunikationsstrukturen, Abläufe und Dokumentationen

C&M: Service Management

IT Service — A described set of facilities, IT and non-IT, supported by the IT Service Provider that fulfills one or more needs of the customer and that is perceived by the customer as a coherent whole.
[itsmwatch.com]

IT-Dienst
Ein IT-Dienst ist ein immaterielles Produkt eines IT-Dienstleisters, das entweder auf IT basiert oder im Zusammenhang mit IT steht und dem Zweck dient, den Zustand des IT-Dienstnehmers unmittelbar zu verbessern. Der Zustand

	des Dienstnehmers bezieht sich dabei auf seinen Informationsstand, sein Wissen, seine wirtschaftliche Situation usw. Ein IT-Dienst zeichnet sich durch seine Funktionalität bzw. Leistung sowie durch eine Menge von Dienstmerkmalen aus. C&M: Service Management
IT Service Provider	The role of an IT Service Provider is performed by any organizational unit, whether internal or external, that delivers and supports IT services to a customer. [itsmwatch.com] IT-Dienstleister Die Rolle des IT-Dienstleisters wird von jeder (internen oder externen) Organisationseinheit besetzt, die IT-Dienste für einen Kunden bereitstellt und betreibt. C&M: Service Management
IT Quality of Service	An agreed or contracted level of service between a service customer and a Service Provider. [ITIL-Glossar, 16.11.2004, www.serview.de/content/itsm/glossar] IT Dienstqualität Die Zusammenfassung der einzelnen Qualitäten der Merkmale eines IT-Dienstes welche zwischen Kunde und IT-Dienstleister vereinbart wird. C&M: Service Management
ITSM	*IT Service Management* Concepts for the professionalization of IT organizations transforming to become IT service providers. Tasks of ITSM therefore focus on planning, developing, monitoring and controlling the provision of IT services. IT-Dienstmanagement Das *IT Service Management* umfasst Methoden und Werkzeuge, die den qualitätsgesicherten Betrieb und die Verwaltung von IT-Diensten ermöglichen. Hierbei betrachtet ITSM ein breites Spektrum an Teilaufgaben. Eine dieser Aufgaben ist das *Service Level Management*. C&M: Service Management
Java	Java ist eine objektorientierte Programmiersprache und als solche ein eingetragenes Warenzeichen der Firma Sun Microsystems. Sie ist eine Komponente der Java-Technologie.

	Java-Programme werden in Bytecode übersetzt und dann in einer speziellen Umgebung ausgeführt, die als Java-Laufzeitumgebung oder Java-Plattform bezeichnet wird. Deren wichtigster Bestandteil ist die Java Virtual Machine (Java-VM), die die Programme ausführt, indem sie den Bytecode interpretiert. Java-Programme laufen in aller Regel ohne weitere Anpassungen auf verschiedenen Computern und Betriebssystemen für die eine Java-VM existiert. Sun bietet neben dem eigenen UNIX-Derivat Solaris auch Java-VMs für Linux und Windows an. Andere Hersteller lassen ihre Java-VM für ihre Plattform zertifizieren, zum Beispiel die Firma Apple für Mac OS X.
JDK	*Java Development Toolkit* Menge von Entwicklungswerkzeugen (Interpreter, Compiler) und Klassenbibliotheken. C&M: General
JNDI	The *Java Naming and Directory Interface* (JNDI) is part of the Java platform, providing applications based on Java technology with a unified interface to multiple naming and directory services. [http://java.sun.com/products/jndi/] C&M: Application Integration
LDAP	*Lightweight Directory Access Protocol* LDAP is designed to provide access to directories supporting the X.500 models, while not incurring the resource requirements of the X.500 Directory Access Protocol (DAP). This protocol is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. When used with a directory supporting the X.500 protocols, it is intended to be a complement to the X.500 DAP [IETF-RFC2251]. C&M: Identity Management
LDIF	*LDAP Data Interchange Format* The file format, known as LDIF, for LDAP Data Interchange Format, is typically used to import and export directory information between LDAP-based directory servers, or to describe a set of changes which are to be applied to a directory [IETF-RFC2849]. C&M: Identity Management
Managed Resource Touchpoint	The interface to an instance of a managed resource, such as an operating system or a server. A touchpoint implements sensor

	and effector behavior for the managed resource, and maps the sensor and effector interfaces to existing interfaces.
Monitoring	Unter Monitoring versteht man alle Arten der Erfassung von Zuständen, eines Vorgangs oder Prozesses mittels technischer Hilfsmittel oder anderer Beobachtungssysteme. C&M: General
OpenLDAP	OpenLDAP ist eine Implementation des LDAP-Protokolls als freie Software. OpenLDAP ist Bestandteil der meisten aktuellen Linux-Distributionen und läuft auch unter verschiedenen Unix-Varianten, MAC OS X und verschiedenen Windows-Versionen.
PDP	*Policy Decision Point* PDPs store security policy and make policy decisions at runtime at the request of policy enforcement points (PEPs); PDPs are sometimes collocated with PEPs due to product packaging. [BG-GLOS] C&M: Identity Management
PEP	*Policy Enforcement Point* PEPs enforce security policies at runtime by referring to policy decision points (PDPs). PEPs may be combined with PDPs for performance reasons, or as a result of product packaging [BG-GLOS] C&M: Identity Management
Perl	Perl ist eine freie, plattformunabhängige und interpretierte Programmiersprache. Der Linguist Larry Wall entwarf sie 1987 als Synthese aus C, den UNIX-Befehlen und anderen Einflüssen. Ursprünglich als Werkzeug zur System- und Netzwerkadministration vorgesehen, hat Perl auch bei der Entwicklung von Webanwendungen und in der Bioinformatik weite Verbreitung gefunden. Hauptziele sind eine schnelle Problemlösung und größtmögliche Freiheit für Programmierer. Der Umgang mit Texten und viele frei verfügbare Module sind Stärken der Sprache.
PMT	*Policy Management Tool* Eine Komponente, mit der ein Administrator Policies definiert und validiert, sowie dem PDP und dem *Policy Repository* (s. u.) bekannt gibt. Ferner sollte ein *Policy Management Tool* die Verfeinerung von Policies unterstützen sowie prüfen, ob eine Menge von Policies widerspruchsfrei ist. *Policy Management Tools* können als eigenständige Komponenten implementiert oder in eine Managementplattform integriert sein.

Policy Repository	Policy Repository can be defined from three perspectives: 1. A specific data store that holds policy rules, their conditions and actions, and related policy data. A database or directory would be an example of such a store. 2. A logical container representing the administrative scope and naming of policy rules, their conditions and actions, and related policy data. A "QoS policy" domain would be an example of such a container. 3. In RFC 3060, a more restrictive definition than the prior one exists. A PolicyRepository is a model abstraction representing an administratively defined, logical container for reusable policy elements.
QoS	*Quality of Service* siehe IT-Dienstqualität C&M: General
Radius	*Remote Authentication Dial-In User Service* Protokoll zur Authentifizierung von Nutzern bei Einwahlverbindungen, wird beispielsweise im Zusammenhang mit PPP und VPN verwendet.
Regelkreis	Siehe Control Loop
Ressource	Betriebsmittel wie Personal, Maschinen, Räume, Finanzen. Quelle: K&D C&M: General
Ressourcenparameter	Bezeichnet Parameter einer Ressource, wie beispielsweise Pingzeiten, Prozesslisten oder Temperaturen einer Ressource. Sie dienen der Feststellung von Symptomen.
RFC	*Request for Comments* Bezeichnung für Dokumente, die zum Standardisierungsverfahren der IETF eingereicht worden sind.
RSVP	Das Resource reSerVation Protocol ist eines der wichtigsten Signalisierungsprotokolle im Internet Protocol-Stack. Es erlaubt Empfängern außerhalb einer Multicast-Gruppe, deren Dienstanforderungen festzulegen. Damit können für bestimmte Anwendungen, z. B. für die Übertragung von Video-Streams bestimmte Bandbreiten für einzelne Verbindungen reserviert werden. In der Version 4 des Internet-Protokolls sind solche Garantien eigentlich nicht vorgesehen, was im Beispiel der Video-Streams zu ruckelnden Bildern führen kann.
SAP	*Service Access Point* (auch Dienstzugangspunkt) ist ein Fachbegriff aus der Telekommunikation. Man bezeichnet damit

	die Schnittstelle zur Interaktion mit einer Kommunikations-Schicht an der oberen Grenze selbiger Schicht.
SLA	*Service Level Agreement* Written agreement between a Service Provider and the customer(s) that documents agreed Service Levels for a service. [itsmwatch.com] Dienstleistungsvereinbarung Vertragliche Vereinbarung zwischen einem IT-Dienstleister und einem IT-Dienstnehmer. Dabei werden die Funktionalität bzw. Leistung, die geforderte Qualität sowie die Kosten der gewünschten und bereitzustellenden IT-Dienste festlegt. C&M: Service Management
SNMP	*Simple Network Management Protocol* Internet-Anwendungsprotokoll, das im Internet-Management zur Kommunikation von Managementinformation genutzt wird. C&M: General
Symptom	The monitor function collects the details from the managed resources and organizes them into symptoms that need to be analyzed. The details can include topology information, metrics, configuration property settings and so on. This data includes information about managed resource configuration, status, offered capacity and throughput. Some of the data is static or changes slowly, whereas other data is dynamic, changing continuously through time. The monitor function aggregates, correlates and filters these details until it determines a symptom that needs to be analyzed.
System	Eine Menge von Elementen, die untereinander in einer wohl definierten Beziehung stehen. C&M: General
Touchpoint Autonomic Manager	Autonomic managers implement intelligent control loops that automate combinations of the tasks found in IT processes. Touchpoint autonomic managers are those that work directly with the managed resources through their touchpoints. These autonomic managers can perform various self-management tasks, so they embody different intelligent control loops. Most autonomic managers use policies that provide the goals or objectives for the intelligent control loops. Touchpoint autonomic managers use these policies to determine what actions should be taken for the managed resources that they manage.

Verfügbarkeit	Siehe Availability
Verzeichnis	Siehe Directory
VPN	*Virtual Private Network* Ein entfernter Rechner erhält über einen (i. d. R. verschlüsselten) Tunnel Zugriff in ein internes bzw. privates Netzwerk und wird mit den gleichen Rechten ausgestattet, wie ein lokaler Rechner.
W3C	*World Wide Web Consortium* An organization that standardizes web-related technologies and concepts. Organisation, durch die mit dem Web verbundene Technologien und Konzepte standardisiert werden. C&M: General
WWW	*World Wide Web* Wichtigste Anwendung des Internet. C&M: General
X.509	X.509 ist ein ITU-T-Standard für eine Public- Key-Infrastruktur und derzeit der wichtigste Standard für digitale Zertifikate. Die aktuelle Version ist X.509v3.
XML	eXtensible Markup Language W3C recommendation for creating special-purpose markup languages; it is a simplified subset of SGML, capable of describing many different kinds of data. Vom W3C standardisierte Auszeichnungssprache, durch die sich die Struktur eines Dokuments beschreiben lässt. C&M: General, Application Management

PCIMe-konforme Policies

Netzverfügbarkeits-Policy

Überschreitet die Pingzeit eines Systems das gewünschte Maximum, muss der Administrator informiert werden.

Attribut	**Wert**
CommonName (CN)	“pingIravpn” bzw. “pingIraldap”
PolicyKeywords[]	“EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Network&&Availability”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Benachrichtigung des Administrators wenn die maximal zulässige Pingzeit überschritten wurde.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“MaximumPingTimeExceeded”
StringList[]	“iraldap.ira.uka.de” bzw. “iravpn.ira.uka.de”
Aktion	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 12: Policy zur Meldung mangelnder Netzverfügbarkeit

Prozessverfügbarkeits-Policy

Läuft ein Daemon, der für die Erbringung des VPN-Dienst verantwortlich ist nicht, muss dieser wieder gestartet und der Administrator darüber informiert werden.

Attribut	**Wert**
CommonName (CN)	“startPluto” bzw. “startL2tpd” bzw. “startPppd” bzw. “startSlapd” bzw. “startRadiusd”
PolicyKeywords[]	“EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Component&&Process”

Attribut	Wert
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Neustarten des betroffenen Prozesses und Benachrichtigung des Administrators.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“ProcessNotRunning”
StringList[]	“pluto” bzw. “l2tpd” bzw. “pppd” bzw. “slapd” bzw. “radiusd”
Aktion 1	
ModelClass	“Component”
ModelProperty	“StartProcess”
StringList[]	“pluto” bzw. “l2tpd” bzw. “pppd” bzw. “slapd” bzw. “radiusd”
Aktion 2	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 13: Policies zum Neustart von Prozessen

Namensauflösungs-Policy

Wird der Hostnamen eines Systems falsch aufgelöst, ist ebenfalls eine Benachrichtigung des Administrators notwendig.

Attribut	Wert
CommonName (CN)	“dnsIravpn” bzw. “dnsIraldap”
PolicyKeywords[]	“EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Network&&DNS”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Benachrichtigung des Administrators bei fehlerhafter DNS-Namensauflösung.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]

Attribut	Wert
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“WrongIPForHostname”
StringList[]	“iraldap.ira.uka.de” bzw. “iravpn.ira.uka.de”
Aktion	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 14: Policy zur Benachrichtigung bei falscher Namensauflösung

Speicherplatz-Policy

Wenn der freie Speicherplatz eines Systems eine bestimmte Grenze unterschritten hat, muss der Administrator davon in Kenntnis gesetzt werden.

Attribut	Wert
CommonName (CN)	“storageIravpn” bzw. “storageIraldap”
PolicyKeywords[]	“EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“System&&Storage”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Benachrichtigung des Administrators bei unzureichend freiem Speicherplatz.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“NotEnoughFreeStorage”
StringList[]	“iraldap.ira.uka.de” bzw. “iravpn.ira.uka.de”
Aktion	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 15: Policy zur Meldung von geringem Speicherplatz

Systemlast-Policy

Überschreitet die Systemlast eines Rechners die festgelegte Grenze, ist die Benachrichtigung des Administrators notwendig.

Attribut	Wert
CommonName (CN)	“loadIravpn” bzw. “loadIraldap”
PolicyKeywords[]	“EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“System&&Load”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Benachrichtigung des Administrators bei zu hoher Systemlast.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“SystemloadTooHigh”
StringList[]	“iraldap.ira.uka.de” bzw. “iravpn.ira.uka.de”
Aktion	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 16: Policy zur Meldung von Überlastungen von Systemen

Config-Policies

Fehlt die Konfigurationsdatei einer Komponente, wird eine entsprechende Konfigurationsdatei von einer Vorlage erzeugt und der Administrator informiert.

Attribut	Wert
CommonName (CN)	“restoreIpsec” bzw. “restoreL2tpd” bzw. “restoreOptions” bzw. “restoreSlapd” bzw. “restoreRadius” bzw. “restoreIraldap” bzw. “restoreHttpd”
PolicyKeywords[]	“CONFIGURATION”, “EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Component&&Config”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Kopiert die Vorlage und benachrichtigt den Administrator, wenn die Konfigurationsdatei fehlt.”

Attribut	Wert
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“ConfigfileMissing”
StringList[]	“/etc/ipsec.conf” bzw. “/etc/l2tpd/l2tpd.conf” bzw. “/etc/ppp/options.l2tpd” bzw. “/etc/slapd.conf” bzw. “/etc/raddb/radiusd.conf” bzw. “/etc/raddb/iraldap” bzw. “/etc/httpd/conf/httpd.conf”
Aktion 1	
ModelClass	“Component”
ModelProperty	“RestoreConfigfile”
StringList[]	“/etc/ipsec.conf” bzw. “/etc/l2tpd/l2tpd.conf” bzw. “/etc/ppp/options.l2tpd” bzw. “/etc/slapd.conf” bzw. “/etc/raddb/radiusd.conf” bzw. “/etc/raddb/iraldap” bzw. “/etc/httpd/conf/httpd.conf”
Aktion 2	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 17: Policies zur Wiederherstellung von Konfigurationsdateien

Log-Policies

Sind Logdateien nicht vorhanden, wird der Administrator darüber in Kenntnis gesetzt.

Attribut	Wert
CommonName (CN)	“logfileMessages” bzw. “logfileSecure” bzw. „logfileSlapd“ bzw. „logfileRadius“
PolicyKeywords[]	“EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Component&&Log”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Benachrichtigung des Administrators, falls Logdateien fehlen.”
Mandatory	“TRUE”

Attribut	Wert
SequencedActions	1 [Mandatory]
ExecutionStrategy	2 [DoAll]
Bedingung	
ModelClass	“Error”
ModelProperty	“LogfileMissing”
StringList[]	“/var/log/messages” bzw. “/var/log/secure” bzw. “/var/log/slapd.log“ bzw. “/var/log/radius/radius.log“
Aktion 1	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 18: Policies zu Behandlung von fehlenden Logdateien

Version-Policy

Falls eine bestimmte Komponente nicht installiert ist, muss diese installiert und der Administrator über diesen Vorgang informiert werden.

Attribut	Wert
CommonName (CN)	“installIPsec” bzw. “installL2tp” bzw. “installPpp” bzw. “installLdap” bzw. “installRadius”
PolicyKeywords[]	“INSTALLATION”, “EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Component&&Version”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Falls die entsprechende Komponente nicht installiert ist, installieren und Administrator benachrichtigen.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung 1	
ModelClass	“Error”
ModelProperty	“ComponentNotInstalled”
BooleanValue	<Component>
Aktion 1	
ModelClass	“Component”
ModelProperty	“InstallComponent”
StringList[]	<RequiredComponent>

Attribut	Wert
Aktion 2	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 19: Policy zur Installation einer Komponente

Requirement-Policy

Falls die Voraussetzungen zum Betrieb einer Komponente nicht erfüllt sind, muss der Administrator darüber informiert werden.

Attribut	Wert
CommonName (CN)	“requirementsIPsec” bzw. “requirementsL2tp” bzw. “requirementsPpp” bzw. “requirementsLdap” bzw. “requirementsRadius”
PolicyKeywords[]	“INSTALLATION”, “EVENT”, “Vpn”
PolicyDecisionStrategy	1 [FirstMatching]
PolicyRoles[]	“Component&&Requirements”
Enabled	1 [Enabled]
ConditionListType	1 [CNF]
RuleUsage	“Falls die Voraussetzungen zum Betrieb der entsprechenden Komponente nicht erfüllt sind, muss der Administrator benachrichtigt werden.”
Mandatory	“TRUE”
SequencedActions	3 [DontCare]
ExecutionStrategy	2 [DoAll]
Bedingung 1	
ModelClass	“Error”
ModelProperty	“RequirementsNotMet”
BooleanValue	<Component>
Aktion 1	
ModelClass	“Alert”
ModelProperty	“NotifyAdministrator”
StringList[]	“it-dienste@atis.uka.de”

Tabelle 20: Policy zur Betriebsvoraussetzungsprüfung einer Komponente

Policies im LDIF-Format

Netzverfügbarkeis-Policy „pingIravpn“

```
# LDIF Export von:
# cn=pingIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=pingIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: pingIravpn
pcelsActionList: cn=Action1,cn=pingIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=pingIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: Network&&Availability
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
pcimRuleUsage: Benachrichtigung, falls maximale Pingzeit
 ueberschritten wurde
objectClass: top
objectClass: pcelsRuleInstance

dn: cn=Action1,cn=pingIravpn,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 1

dn: cn=Condition1,cn=pingIravpn,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=pingIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=pingIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass

dn: cn=ConditionValue,cn=Condition1,cn=pingIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: iravpn.ira.uka.de
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsStringValueAuxClass
```

```
dn: cn=ConditionVariable,cn=Condition1,cn=pingIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: MaximumPingTimeExceeded
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Prozessverfügbarkeits-Policy „startPluto“

```
# LDIF Export von:
# cn=startPluto,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=startPluto,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: startPluto
pcimKeywords: EVENT
pcimKeywords: Vpn
pcelsDecisionStrategy: 1
pcimRoles: Component&&Process
pcimRuleEnabled: 1
pcelsConditionListType: 1
pcimRuleUsage: Neustarten des Pluto Prozesses und
 Benachrichtigung des Administrators
pcimRuleMandatory: TRUE
pcelsSequencedActions: 3
pcelsExecutionStrategy: 2
objectClass: pcelsRuleInstance
objectClass: top
pcelsActionList: cn=Action1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsActionList: cn=Action2,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=Action1,cn=startPluto,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
objectClass: pcelsSimpleActionAuxClass
pcimActionOrder: 1
pcelsValueDN: cn=ActionValue,cn=Action1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ActionVariable,cn=Action1,
 cn=startPluto, ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=ActionValue,cn=Action1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionValue
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsStringValueAuxClass
pcelsStringList: pluto
```

```
dn: cn=ActionVariable,cn=Action1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionVariable
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsExplicitVariableAuxClass
pcelsVariableModelClass: Component
pcelsVariableModelProperty: StartProcess

dn: cn=Action2,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action2
pcimActionOrder: 2

dn: cn=Condition1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
objectClass: pcelsConditionAssociation
objectClass: top
objectClass: pcelsSimpleConditionAuxClass
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
pcelsValueDN:
cn=ConditionValue,cn=Condition1,cn=startPluto,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN:
cn=ConditionVariable,cn=Condition1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=ConditionValue,cn=Condition1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsStringValueAuxClass
pcelsStringList: pluto

dn: cn=ConditionVariable,cn=Condition1,cn=startPluto,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsExplicitVariableAuxClass
pcelsVariableModelClass: Error
pcelsVariableModelProperty: ProcessNotRunning
```

Namensauflösungs-Policy „dnsIravpn“

```
# LDIF Export von:
# cn=dnsIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=dnsIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
```

```
cn: dnsIravpn
pcelsActionList: cn=Action1,cn=dnsIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=dnsIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: Network&&DNS
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
objectClass: top
objectClass: pcelsRuleInstance
pcimRuleUsage: Benachrichtigt den Administrator, falls der
 Hostname falsch aufgeloest wird

dn: cn=Action1,cn=dnsIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 1

dn: cn=Condition1,cn=dnsIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=dnsIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=dnsIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass

dn: cn=ConditionValue,cn=Condition1,cn=dnsIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: iravpn.ira.uka.de
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsStringValueAuxClass

dn: cn=ConditionVariable,cn=Condition1,cn=dnsIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: WrongIpForHostname
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Speicherplatz-Policy „storageIravpn“

```
# LDIF Export von:
# cn=storageIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=storageIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,
 dc=de
cn: storageIravpn
pcelsActionList: cn=Action1,cn=storageIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=storageIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: System&&Storage
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
objectClass: top
objectClass: pcelsRuleInstance
pcimRuleUsage: Benachrichtigung, falls der minimale freie
 Speicherplatz unterschritten wird

dn: cn=Action1,cn=storageIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 1

dn: cn=Condition1,cn=storageIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=storageIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=storageIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass

dn: cn=ConditionValue,cn=Condition1,cn=storageIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: iravpn.ira.uka.de
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsStringValueAuxClass

dn: cn=ConditionVariable,cn=Condition1,cn=storageIravpn,
```

```
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: NotEnoughFreeStorage
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Systemlast-Policy „loadIravpn“

```
# LDIF Export von:
# cn=loadIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=loadIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,
 dc=de
cn: loadIravpn
pcelsActionList: cn=Action1,cn=loadIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=loadIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: System&&Load
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
objectClass: top
objectClass: pcelsRuleInstance
pcimRuleUsage: Benachrichtigung, falls die maximale
 Systemlast ueberschritten wird

dn: cn=Action1,cn=loadIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 1

dn: cn=Condition1,cn=loadIravpn,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=loadIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=loadIravpn,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass
```

```
dn: cn=ConditionValue,cn=Condition1,cn=loadIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: iravpn.ira.uka.de
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsStringValueAuxClass

dn: cn=ConditionVariable,cn=Condition1,cn=loadIravpn,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: SystemloadTooHigh
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Config-Policy „restoreRadius“

```
# LDIF Export von:
# cn=restoreRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=restoreRadius,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: restoreRadius
pcimKeywords: CONFIGURATION
pcimKeywords: EVENT
pcimKeywords: Vpn
pcelsDecisionStrategy: 1
pcimRoles: Component&&Process
pcimRuleEnabled: 1
pcelsConditionListType: 1
pcimRuleUsage: Wiederherstellen der angegebenen
 Konfiguration und Benachrichtigung des Administrators
pcimRuleMandatory: TRUE
pcelsSequencedActions: 3
pcelsExecutionStrategy: 2
objectClass: pcelsRuleInstance
objectClass: top
pcelsActionList: cn=Action1,cn=restoreRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsActionList: cn=Action2,cn=restoreRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=restoreRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=Action1,cn=restoreRadius,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
objectClass: pcelsSimpleActionAuxClass
pcimActionOrder: 1
pcelsValueDN: cn=ActionValue,cn=Action1,cn=restoreRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ActionVariable,cn=Action1,
 cn=restoreRadius, ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
```

```
dn: cn=ActionValue,cn=Action1,cn=restoreRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionValue
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsStringValueAuxClass
pcelsStringList: /etc/raddb/radiusd.conf

dn:
cn=ActionVariable,cn=Action1,cn=restoreRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionVariable
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsExplicitVariableAuxClass
pcelsVariableModelClass: Component
pcelsVariableModelProperty: RestoreConfigfile

dn: cn=Action2,cn=restoreRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action2
pcimActionOrder: 2

dn: cn=Condition1,cn=restoreRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
objectClass: pcelsConditionAssociation
objectClass: top
objectClass: pcelsSimpleConditionAuxClass
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
pcelsValueDN:
cn=ConditionValue,cn=Condition1,cn=resotreRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN:
cn=ConditionVariable,cn=Condition1,cn=restoreRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=ConditionValue,cn=Condition1,cn=restoreRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsStringValueAuxClass
pcelsStringList: /etc/raddb/radiusd.conf

dn: cn=ConditionVariable,cn=Condition1,cn=restoreRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsExplicitVariableAuxClass
pcelsVariableModelClass: Error
```

```
pcelsVariableModelProperty: ConfigfileMissing
```

Log-Policy „logfileRadius“

```
# LDIF Export von:
# cn=logfileRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=logfileRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,
 dc=de
cn: logfileRadius
pcelsActionList: cn=Action1,cn=logfileRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=logfileRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: Component&&Log
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
objectClass: top
objectClass: pcelsRuleInstance
pcimRuleUsage: Benachrichtigung, falls das angegebene
 Logfile nicht exisitiert

dn: cn=Action1,cn=logfileRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 1

dn: cn=Condition1,cn=logfileRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=logfileRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=logfileRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass

dn: cn=ConditionValue,cn=Condition1,cn=logfileRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: /var/log/radius/radius.log
objectClass: top
objectClass: pcimPolicyInstance
```

```
objectClass: pcelsStringValueAuxClass

dn: cn=ConditionVariable,cn=Condition1,cn=logfileRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: LogfileMissing
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Version-Policy „installRadius“

```
# LDIF Export von:
# cn=installRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=installRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,
 dc=de
cn: installRadius
pcelsActionList: cn=Action1,cn=installRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsActionList: cn=Action2,cn=installRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=installRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: INSTALLATION
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: Component&&Version
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
objectClass: top
objectClass: pcelsRuleInstance
pcimRuleUsage: Installation der Komponente und
Benachrichtigung, falls diese nicht installiert ist

dn: cn=Action1,cn=installRadius,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
objectClass: pcelsSimpleActionAuxClass
pcimActionOrder: 1
pcelsValueDN: cn=ActionValue,cn=Action1,cn=installRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ActionVariable,cn=Action1,
 cn=installRadius, ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=ActionValue,cn=Action1,cn=installRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionValue
objectClass: pcimPolicyInstance
objectClass: top
```

```
objectClass: pcelsStringValueAuxClass
pcelsStringList: freeradius-1.0.1-3.RHEL4

dn:
cn=ActionVariable,cn=Action1,cn=installRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionVariable
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsExplicitVariableAuxClass
pcelsVariableModelClass: Component
pcelsVariableModelProperty: InstallComponent

dn: cn=Action2,cn=installRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Action2
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 2

dn: cn=Condition1,cn=installRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=installRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=installRadius,ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass

dn: cn=ConditionValue,cn=Condition1,cn=installRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: freeradius-1.0.1-3.RHEL4
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsStringValueAuxClass

dn: cn=ConditionVariable,cn=Condition1,cn=installRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: ComponentNotInstalled
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Requirement-Policy „requirementRadius“

```
# LDIF Export von:
# cn=requirementsRadius,ou=policies,dc=ATIS,
```

```
# dc=ira,dc=uka,dc=de

dn: cn=requirementsRadius,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: requirementsRadius
pcelsActionList: cn=Action1,cn=requirementsRadius,
 ou=policies, dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionList: cn=Condition1,cn=requirementsRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsConditionListType: 1
pcelsDecisionStrategy: 1
pcelsExecutionStrategy: 2
pcelsSequencedActions: 3
pcimKeywords: INSTALLATION
pcimKeywords: EVENT
pcimKeywords: Vpn
pcimRoles: Component&&Requirements
pcimRuleEnabled: 1
pcimRuleMandatory: TRUE
pcimRuleUsage: Benachrichtigung, falls die Voraussetzungen
 zum Betrieb nicht erfuellt sind
objectClass: top
objectClass: pcelsRuleInstance

dn: cn=Action1,cn=requirementsRadius,ou=policies,dc=ATIS,
 dc=ira,dc=uka,dc=de
cn: Action1
objectClass: pcelsActionAssociation
objectClass: top
pcimActionDN: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimActionOrder: 1

dn: cn=Condition1,cn=requirementsRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: Condition1
pcelsValueDN: cn=ConditionValue,cn=Condition1,
 cn=requirementsRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ConditionVariable,cn=Condition1,
 cn=requirementsRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcimConditionGroupNumber: 0
pcimConditionNegated: FALSE
objectClass: top
objectClass: pcelsConditionAssociation
objectClass: pcelsSimpleConditionAuxClass

dn: cn=ConditionValue,cn=Condition1,cn=requirementsRadius,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionValue
pcelsStringList: freeradius-1.0.1-3.RHEL4
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsStringValueAuxClass

dn: cn=ConditionVariable,cn=Condition1,
```

```
 cn=requirementsRadius,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ConditionVariable
pcelsVariableModelClass: Error
pcelsVariableModelProperty: RequirementsNotMet
objectClass: top
objectClass: pcimPolicyInstance
objectClass: pcelsExplicitVariableAuxClass
```

Wiederverwendbare Aktion „alertAdministrator“

```
# LDIF Export von:
# pcelsReusableContainerName=reusableActions,
# ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de

dn: pcelsReusableContainerName=reusableActions,
 ou=policies,dc=ATIS,dc=ira,dc=uka,dc=de
pcelsReusableContainerName: reusableActions
objectClass: pcelsReusableContainerInstance
objectClass: top

dn: cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: alertAdministrator
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsSimpleActionAuxClass
pcelsValueDN: cn=ActionValue,cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
pcelsVariableDN: cn=ActionVariable,cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de

dn: cn=ActionValue,cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionValue
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsStringValueAuxClass
pcelsStringList: it-dienste@atis.uka.de

dn: cn=ActionVariable,cn=alertAdministrator,
 pcelsReusableContainerName=reusableActions,ou=policies,
 dc=ATIS,dc=ira,dc=uka,dc=de
cn: ActionVariable
objectClass: pcimPolicyInstance
objectClass: top
objectClass: pcelsExplicitVariableAuxClass
pcelsVariableModelClass: Alert
pcelsVariableModelProperty: NotifyAdministrator
```