INSTITUT FÜR INFORMATIK
DER LUDWIG MAXIMILIAN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Design und prototypische Implementierung eines integrierten Dienstmanagements für Nomadische Systeme in Intranets

Bearbeiter: Simone Demmel
Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Stephen Heilbronner

INSTITUT FÜR INFORMATIK
DER LUDWIG MAXIMILIAN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Design und prototypische Implementierung eines integrierten Dienstmanagements für Nomadische Systeme in Intranets

Bearbeiter:	Simone Demmel
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Stephen Heilbronner
Abgabetermin:	1. November 1999

Zusammenfassung

zusammenfassung.tex

Ein Nomadisches System ist eine Komponente, die ihren räumlichen Aufenthaltsort innerhalb eines Netzwerkes verändert. Aufgrund dieses Standortwechsels benötigt sie für die Nutzung anderer Dienste spezielle, lokal verschiedene Zusatzinformationen. Diese sollen dem System automatisch zugewiesen werden können.

Außerdem soll die Nutzung von Diensten (z.B. Drucker, Fax) automatisch im Nomadischen System konfiguriert werden (jede Abteilung hat einen lokalen Drucker, dieser muß vermittelt werden). Dieser Vorgang wird als Dienstvermittlung bezeichnet.

Damit eine gewisse Sicherheit gewährleistet ist, muß zusätzlich ein Mechanismus entwickelt werden, mit dem man Regeln verfassen kann, die zum Beispiel besagen, welche Services ein bestimmtes Nomadisches System nutzen darf. Solche Regeln werden Policies genannt.

Die vorliegende Arbeit untersucht zunächst eine Reihe bereits existierender Protokolle und Plattformen, die in sich bereits eine Dienstvermittlung anbieten. Dabei wird untersucht, in wie weit Ansätze zur Integration von Policies vorhanden sind, sowie auf welcher Basis diese Plattformen arbeiten.

Ziel ist es, ein Konzept für ein Managementsystem zu entwickeln, welches nach außen hin alle dienstvermittelnden Protokolle und Plattformen sowie Policies unterstützt. Dabei soll lediglich der Bereich der Dienstvermittlung ausgearbeitet werden.

Dazu wird im ersten Teil der Arbeit ein Kriterienkatalog erstellt, der die Anforderungen an eine solche Managementplattform beschreibt. Inwieweit die verschiedenen Protokolle diese Anforderungen bereits unterstützen, was sie nicht können, was zusätzlich vorgesehen ist und inwiefern bereits Implementierungen vorhanden sind, ist Hauptbestandteil dieser Arbeit.

Im zweiten Teil der Arbeit wird eine Managementplattform entwickelt, die alle Protokolle (sowie zukünftige Protokolle) miteinander verbindet. Ziel ist es, Nomadische Systeme in jeden Fall mit den benötigten Informationen zu versorgen, egal, welches Protokoll oder welche Managementplattform sie verwenden.

Der dritte Abschnitt befaßt sich mit einer Implementierungsmöglichkeit eines solchen policybasierten Managements. Sofern möglich soll eine prototypische Implementierung im Rahmen des bereits vorhanden Projektes Masa durchgeführt werden. Diese Implementierung umfaßt die Erweiterung eines bereits vorhandenen Policy-Agenten, sowie die Erstellung eines DHCP-Agenten, der mit Hilfe des Policy-Agenten entsprechend IP-Adressen über DHCP vergibt.

Diese Implementierung konnte größtenteils nicht mehr in die Praxis umgesetzt werden, da das Masa-System umgestellt wurde und somit nicht für Testzwecke zur Verfügung stand.

Entwickelt werden konnte der Basis-DHCP-Agent, sowie eine Beispiel-Policy zur Verwendung vom Policy-Agenten.

Nicht mehr verwirklicht werdenkonnte die konkrete Umsetzung des Policy-Agenten, da dieser zu kurzfristig fertig umgestellt war.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

einleitung.tex

In der heutigen Zeit gewinnen zwei Dinge zunehmend an Bedeutung: der Einsatz von Technik (insbesondere Computern) und eine extreme Mobilität.

In fast allen modernen Berufen kommen Computer zum Einsatz. Ein Leben ohne sie ist nicht mehr denkbar. In vielen Firmen wird die Korrespondenz mit Kunden und Geschäftspartnern über Computer teilweise sogar automatisch (Rechnungen, Mahnungen) abgewickelt. Somit gehören in den meisten Firmen ein lokales Netzwerk mit Drucker, Faxzugängen und zum Teil auch Internetzugängen zum Alltag.

Dadurch, daß viele Firmen sich auf den internationalen Markt ausweiten, entsteht eine höhere Mobilität der Mitarbeiter: Manager treffen sich in den USA, um firmenpolitische Entscheidungen zu diskutieren, Mitarbeiter aus allen weltweiten Tochter- und Zweigstellen besuchen gemeinsam eine Schulung.

1.1 Motivation

anwendungsgebiete.tex

Was ist aber nun, wenn ein Mitarbeiter nicht an seinem gewohnten Arbeitsplatz in der Firma arbeiten kann? Oder was macht ein Student, wenn er die Literatur aus der Bibliothek nicht entfernen darf (Präsenzbibliothek)?

Ein moderner Mitarbeiter ist mehrmals im Jahr unterwegs: auf Kongressen, Fortbildungsmaßnahmen oder einfach bei auswärtigen Kunden vor Ort. Dabei kann er nicht auf moderne Hilfmittel verzichten.

Der Mitarbeiter, der tagsüber an einem Kongress im Ausland teilnimmt, möchte am Abend oder in den Pausen seine Email im Heimatbüro lesen und bearbeiten. Haben sich interessante Bekanntschaften auf diesem Kongress ergeben, kann es nützlich sein, diesen Kontakt sofort in die Firma weiterzuvermitteln. So kann sich sofort jemand mit dem zukünftigen Kunden in Verbindung setzen und die Kontaktaufnahme verschiebt sich nicht auf den Zeitpunkt, zu dem der Mitarbeiter vom Kongress zurück ist und seine Notizen so weit aufgearbeitet hat.

Auch ein Student, der viel mit Literatur in Bibliotheken arbeitet, ist oft darauf angewiesen, Querverweise in der Literatur zu verfolgen. Leider gehen diese Verweise in aktueller Literatur immer häufiger auf Webseiten im Internet, so daß ein paralleler Internetzugang fast unerläßlich für ein ausreichendes Studium der Quellen wird.

Manchmal werden öffentlich zugängliche Systeme für diesen Zweck zur Verfügung gestellt. Diese sind meist aber nicht in ausreichender Menge vorhanden oder es sind spezielle Betrachtungsprogramme nicht installiert.

In allen Fällen ist eine schnelle und unkomplizierte Anbindung mittels eines eigenen Rechners oder Notebooks an ein Netzwerk wünschenswert, sowie eine eventuelle Nutzung lokaler Dienste (Drucker, Internet-Zugang, Fax).

Bisher sah dies in der Praxis wie folgt aus:

Will jemand das lokale Netz nutzen, muß er sich eine IP-Adresse beim lokalen Systemadministrator holen; neue Gateways, Nameserver, Proxies auf seinem Notebook konfigurieren oder die MAC-Adresse seiner Netzwerk-Karte abgeben, damit ein Teil der benötigten Dienste automatisch auf dem Notebook konfiguriert werden können.

Die meisten Handgriffe für eine derartige Umkonfiguration des Notebook oder eines anderen mobilen Rechners sind für jemanden, der den Rechner als Werkzeug und nicht als Lebenszweck betrachtet, böhmische Dörfer. Wer weiß schon, wie er die MAC-Adresse seiner Ethernetkarte abfragen kann?

Viel angenehmer und einfacher wäre folgendes Vorgehen:

Bibliothek

In einer Bibliothek:

Der Student kommt in den Lesesaal der Bibliothek. Dort findet er an den Tischen Strom- und Netzwerkanschlüsse vor. Hier kann er sein Notebook anschließen. Ist er in der Bibliothek seiner eigenen Universität, kann er sich direkt mit seiner normalen Kennung anmelden. Er erhält dann seine gewohnte Arbeitsumgebung und Services zur Verfügung gestellt.

Ist er Gast in dieser Bibliothek, kann er die allgemein bekannte Gastkennung verwenden und ebenfalls auf einige Dienste, wie zum Beispiel Internetzugang zugreifen.

Hotel

In einem Hotel:

Der Kunde kommt in sein Hotelzimmer. Beim Einchecken hat er bereits eine Kennung und ein Passwort mitgeteilt bekommen. Unter den üblichen Informationen zum Hotel findet er in seinem Zimmer auch Informationen zum Netzwerkservice, sowie Hinweise über Standardkosten und Dienste, die über das Netzwerk vom Hotel zur Verfügung gestellt werden. Er verbindet sein Notebook mit einem vorhandenen Ethernet-Anschluß, startet ein Programm, welches von ihm die Kennung abfragt und wenige Augenblicke später kann er wie gewohnt drucken, Internetdienste nutzen und normal arbeiten. Er kann den Fax-Server seiner entfernten Firma nutzen und den Drucker am Ende des Ganges von seinem Hotelzimmer.

Seine gewohnte Arbeitsumgebung wird lediglich um die lokal angebotenen Dienste erweitert. Damit stehen ihm jetzt zusätzliche lokale Services zur Verfügung: ein lokaler Proxy zum Websurfen, ein normaler Schwarz-Weiß-Drucker, ein Folien-Drucker, ein Fax-Server vor Ort, auf dem er auch Faxe empfangen kann.

Abrechnung von Service-Leistungen

Service-Leistungen sind nie umsonst. Entweder man bezahlt sie als Pauschale bereits mit seinem Semesterbeitrag oder dem Zimmerpreis oder sie werden gesondert erhoben und abgerechnet.

Ist die Gebühr zur Nutzung verschiedener Services im Preis noch nicht enthalten, muß sie gesondert erhoben und verrechnet werden. In einem Hotel bietet es sich an, diese Beträge sofort auf die jeweilige Zimmerrechnung zu buchen und am Ende mit allen angefallenen Kosten zusammen abzurechnen und vom Kunden zahlen zu lassen.

In einer Bibliothek ist es allerdings sehr schwierig bis unmöglich den Nutzer der Gastkennung, der gerade 10 Seiten ausgedruckt hat, aufzufinden. Deshalb könnte man in solchen Bereichen zusätzlich auf anonyme Zahlungsmethoden zurückgreifen.

Dabei könnte an jedem Arbeitsplatz (oder in jedem Hotelzimmer) ein Geldkarten-Gerät stehen. Will man einen kostenpflichtigen Service nutzen, muß man dort eine Geldkarte eingeben. Der Computer teilt mit, wieviel er für die entsprechende Service-Nutzung von der Karte abbuchen wird (und wieviel Geld noch auf der Karte ist). Der Nutzer muß dieses bestätigen und erst dann bekommt er den Service 'geliefert' und der Betrag wird direkt von seiner Karte abgebucht.

Alternativ kann man sich dieses mit Kreditkarten und EC-Karten vorstellen. Auch ist eine direkte Abbuchung vom Konto des Nutzers denkbar, hier muß dann keine Karte eingefügt, sondern eine Kontonummer angegeben werden.

bisherige Realisierung

Bisher ist diese Form der alles umfassenden Dienstleistung in Deutschland eher unbekannt.

Zur Realisierung dieser Vorstellung gibt es seit einiger Zeit weltweite Bemühungen. Dazu wurden inzwischen unter anderem Protokolle und Plattformen zur automatischen Vergabe einer IP-Adresse, Dienstvermittlung und zum zentralen Service-Management entwickelt.

Ein momentan sehr aktuelles Gebiet ist die Dienstvermittlung an sich; also die Frage: Wie erfährt das neu hinzugekommene System, welche Dienste ihm zur Verfügung stehen und wie es sie ansprechen kann (z.B. den Namen der Netzwerk-Drucker und ihre Beschreibung). Die Beschreibung ist für den Nutzer, damit der sich aus den möglichen angebotenen Druckern den von ihm Bevorzugten auswählen kann.

Dabei muß die Frage der Berechtigung gestellt werden: Darf dieser Nutzer auf diese Daten zugreifen und wenn ja, muß eine weitere Aktion (Geld von einer Karte abbuchen, auf die Hotelrechnung verbuchen) eingeleitet werden? Muß der Nutzer sich vielleicht noch einmal extra authentisieren (also Name und Passwort angeben)? Was ist im zu nutzenden Dienst eine Abrechnungseinheit und wie viel soll sie kosten?

Damit wird ein weiteres Problem sichtbar: Das allgemeine Management aller im Netzwerk angebotener Dienste zusammen. Schließlich macht es keinen Sinn, wenn der Drucker A die Diensterbringung verweigert und Drucker B sie erbringt, lediglich, weil vergessen wurde, Drucker B 'mitzuteilen', daß dieser Nutzer nicht berechtigt ist, die Dienstleistung in Anspruch zu nehmen. Damit solche Kommunikationspannen nicht passieren ist ein zentrales Management wichtig. Dies läßt sich zusätzlich vereinfachen, indem man nicht jeden Service separat komplett konfiguriert, sondern nur die Teile, die für diesen Service spezifisch sind. Alle anderen Informationen 'holt sich' der Service bei Bedarf von einer zentralen Stelle.

Somit fragen zum Beispiel Drucker A und Drucker B einen Service, ob der Nutzer X bei ihnen drucken darf. Beide bekommen ein 'Nein' und verweigern die Dienstleistung einheitlich.

Es wird also ein System benötigt, welches

- ein zentrales Management aller Services erlaubt
- einem neuen Computer, welcher temporär im Netz arbeitet (*Nomadisches System* genannt) alle nötigen Zugangsdaten mitteilt (Policy, Authentisierung..)
- zentrale Dienste anbietet, welche Zugriffsberechtigungen entscheiden
- Abrechnung von Service-Nutzungen durchführen lassen kann
- allgemein Dienste an ein Nomadisches System vermitteln kann

Protokolle, die einen Aufbau solcher Systeme erlauben, wurden in den letzten Jahren verschiedentlich entwickelt und veröffentlicht. Aktuelle Beispiele hierfür sind:

DHCP (Dynamic Host Configuration Protocol) Vergabe einer IP-Adresse, sowie allgemeine Systeminformationen

SLP (Service Location Protocol) Vermittlung von Diensten

CORBA Trading Service Vermittlung von Diensten

JINI (Java Intelligent Network Infrastructure) Vermittlung und Management von Diensten

SNMP (Simple Network Management Protocol) Management von Diensten.

Diese Protokolle sollen jetzt in einer Managementplattform umgesetzt werden. Dabei sollten die jeweiligen Vorzüge der einzelnen Protokolle genutzt werden. Da dieses beim momentanen Stand der Technik die Verknüpfung mindestens dreier in sich nicht untereinander kompatibler Systeme bedeuten würde, ist der nächste Schritt:

Ein Konzept zu entwickeln, welches erweiterbar alle nötigen Protokolle unterstützen kann und somit zumindest für den Nutzer, der dieses Netzwerk nutzen möchte (also für das Nomadische System), völlig transparent in jedem Fall den gewünschten Service - nämlich Zugang zum Netz und Vermittlung benötigter Dienste - anbieten kann. Und dabei soll das vom Nomadischen System verwendete Protokoll (sofern es sich um ein Standard-Protokoll handelt), für das Funktionieren oder Nichtfunktionieren des Konzeptes belanglos sein.

1.2 Aufgabenstellung

aufgabenstellung.tex

Die verschiedenen Protokolle und bereits vorhandenen großen Ansätze zum Dienstmanagement sollen analysiert und nach Kommunikationsmodell, Funktionsmodell, Informationsmodell und Organisationsmodell gegliedert beschreiben werden.

Dabei soll evaluiert werden, inwieweit sich diese unterschiedlichen Protokolle in einem Managementsystem zusammen verwenden und integrieren lassen können.

Um Nomadische Systeme und Dienstnutzung mit einander zu verbinden, werden Policies entwickelt, nach denen Zugriffsrechte und Verteilung zwischen Komponenten der gemanageten Dienste untereinander, sowie im Zusammenspiel mit Nomadischen Systemen als externe Partner, verwaltet und durchgesetzt werden können.

Dazu sollen Policy-Regeln entworfen werden, die möglichst viele der Informationen, welche bei den betrachteten Protokollen und Lösungsansätzen vermittelt werden können, ebenfalls vermitteln können. Zu diesem Zweck soll kein neues Protokoll entwickelt werden, sondern ein System, welches mittels aller Protokolle ansprechbar ist. Das heisst, es soll eine Umsetzung erstellt werden, die es erlaubt alle Protokolle und protokollähnlichen Ansätze zu vereinen.

Als praktischer Teil der Arbeit soll die Entwurfsidee der Policy-Regeln in einem Policy-Agenten für das Masa-System und einem DHCP-Agenten umgesetzt und implementiert werden.

1.3 Ergebnisse

ergebnis.tex

Es wurden die Protokolle DHCP und SLP, sowie die Plattformen der CORBA Trading services, Jini untersucht.

Außerdem wurde der Vorschlag einer Policy-Beschreibung von Strassner betrachtet.

Anhand dieser Voraussetzungen, sowie selbst entwickelter Einsatz-Szenarien, wurden Policy-Regeln entwickelt.

Im weiteren wurden die allgemeine Funktionen eines Managementsystem im Hinblick auf Dienstvermittlung, sowie die dazugehörigen Abläufe dargestellt.

Daraus wurde eine eigene Policy-Darstellung entwickelt, die auf die Bedürfnisse innerhalb des betrachteten Managementsystems abgestimmt wurden.

In einem letzten Schritt wurde ein Konzept entwickelt, wie man die bereits vorhandenen Plattformen und Protokolle unter einem einzigen Managementsystem vereinen kann. dies bedeutet, daß das Nomadische System eines der möglichen Protokolle nutzen kann, ohne Kenntnis vom dahinterliegenden Managementsystem zu haben.

Im praktischen Teil der Arbeit sollte ein DHCP-Agent entwickelt werden, der vom Managementsystem beeinflußbar ist. Außerdem soll er anhand der entwickelten Policies agieren. Dieser Agent wurde in seinem Basis-Gerüst implementiert.

Zum Management dieses Agenten sollte der bereits vorhandene Policy-Agent des Masa-Systemes erweitert werden. Dies konnte leider nicht mehr durchgeführt werden, da im entsprechenden Zeitraum das Masa-System, sowie der Policy-Agent auf verschlüsselte Kommunikation erweitert wurde. Als Folge davon standen die benötigten Komponenten des

Masa-Basis-Systems, sowie des Policy-Agenten nicht ausreichend lange zur Verfügung um die Erweiterungen vorzunehmen, sowie den DHCP-Agenten zu testen. Allerdings konnte herausgearbeitet werden inwieweit diese Erweiterung stattfinden kann, nur die praktische Implementierung konnte nicht mehr erfolgen.

1.4 Struktur der vorliegenden Arbeit

da-struktur.tex

Ein Nebenziel dieser Arbeit ist es, einen Überblick über die momentan aktuellen Protokolle und Ansätze zur Dienstvermittlung und zum Service-Management zu bieten. Daher sind diese Kapitel vom Aufbau her so, daß die Möglichkeit des Nachschlagens und Vergleichens erreicht wird.

Es wird ein aktuelles Beschreibungssystem verwendet. Dieses unterteilt ein Protokoll oder eine Plattform in verschiedene Teilmodelle und erlaubt somit unterschiedliche Blickwinkel auf ein und dasselbe Projekt, ohne dabei durch Informationen aus einer anderen Ebene vom eigentlichen Betrachtungsziel abzulenken.

Es gibt vier solche Teilmodelle: Informationsmodell, Kommunikationsmodell, Organisationsmodell, Funktionsmodell

Jedes Kapitel wird mit einem kurzen Kapitelüberblick eingeleitet. Dieser faßt stichpunktartig zusammen, welche Aspekte besprochen werden und was das Ziel des Kapitels ist.

Die verschiedenen vorgestellten Protokolle, haben einen allgemeinen Abschnitt, ein einleitendes Stück und werden dann in ihre verschiedenen Teilmodelle unterteilt und dargestellt.

Da von Seiten der Spezifikationen oft unterschiedliche Begriffe verwendet werden, werden diese zusätzlich am Anfang jedes Abschnittes kurz geklärt. Dies soll dem leichteren Lesen dienen, da aus Gründen der Einheitlichkeit zu den jeweiligen offiziellen Drafts und RFCs die jeweilige Grundterminologie hier in der Arbeit nach Möglichkeit beibehalten wurde.

Das jeweils vorhandene einleitende Kapitel wurde mit 'Designziel' bezeichnet. In ihm wird die Grundidee zum Einsatz und zur Funktionsweise des Protokolls knapp erläutert, um einen leichteren Überblick zu schaffen, unter welchen Gesichtspunkten das jeweilige Protokoll entwickelt wurde, damit es für sein jeweiliges Einsatzgebiet optimal ist.

Diagramme

Bei den Diagrammen wurde sich weitestgehendst an die Darstellung in UML bzw. CIM ([CIM99]) gehalten. CIM ist als eine Unterklasse von UML zu betrachten.

Teilweise wurden Diagramme direkt aus den offiziellen Dokumentationen übernommen oder ihnen nachempfunden. Diese sind dann in der jeweiligen Darstellungsform ihres Originals belassen und mit erklärendem Text versehen.

Kapitel 2

Grundlagen und Standards

grund+stand.tex

Kapitelüberblick:

- Das vorliegende Kapitel beschreibt die in der Arbeit verwendeten Modelle, mit denen Protokolle beschrieben werden, führt die später als Beispiele verwendeten Szenarien ein und behandelt allgemeine Themen wie Policies, Sicherheitsmaßnahmen und Dienstmanagement.
- Dabei werden an vielen Stellen Punkte angesprochen, die im Verlauf der Arbeit detaillierter ausgeführt werden.
- Hauptziel dieses Kapitels ist es, die in der Arbeit besprochenen Aspekte vorbereitend darzustellen und das Endziel der Arbeit vorzustellen.

2.1 Modelle

modelle.tex

Die Modellsichtweise soll das Beschreiben und Verstehen von Protokollen erleichtern. Dazu werden für jedes Protokoll Modelle entwickelt, die nur einen bestimmten Bereich der Betrachtungsmöglichkeiten des jeweiligen Protokolls darstellen. Auf diese Art und Weise ist es möglich, sich über Teilaspekte eines Protokolls zu informieren, ohne gleichzeitig andere, im Augenblick uninteressante Aspekte betrachten zu müssen. Die folgenden Beschreibungen sind verhältnismäßig kurz und knapp gehalten. Eine ausführliche Beschreibung der Modelle findet man in [HAN99]

2.1.1 Informationsmodell

Im Informationsmodell wird beschrieben, welche Informationen ausgetauscht werden. Dabei wird darauf eingegangen, welchen Inhalt diese Informationen haben und welche Variationen es geben kann. Mit diesen Angaben kann auch die Manipulationsmöglichkeit und das Zusammenspiel mit anderen Empfängern oder Sendern von Informationen dargestellt.

2.1.2 Kommunikationsmodell

Im Kommunikationsmodell werden die miteinander kommunizierenden Partner sowie Konzepte zum Austausch der Informationen der kommunizierenden Partner untereinander dargestellt. Im Kommunikationsmodell werden ebenfalls die verwendeten Übertragungsprotokolle angegeben.

Viele der in dieser Arbeit betrachteten Ansätze verwenden feste Protokolle, deren Interna nicht weiter ausgeführt sind. Es werden Protokolle wie CORBA verwendet, die in sich schon komplex sind und wieder auf anderen Standardprotokollen aufsetzen. Trotzdem werden diese Produkte, die eher als bereits umfangreiche Plattformen zu bezeichnen wären im Kapitel Protokolle vorgestellt.

2.1.3 Funktionsmodell

Im Funktionsmodell werden die einzelnen Funktionen, die ein Objekt innerhalb des Protokolls oder der Plattform ausführen kann, dargestellt und beschrieben. Hier finden sich Flußdiagramme, Statediagramme und andere tiefergehende Protokolldetails, wie zum Beispiel auch Fehlermanagement.

2.1.4 Organisationsmodell

Im Organisationsmodell werden die miteinander kommunizierenden Partner und ihr Verhältnis zueinander dargestellt. Dabei kommen Informationen bezüglich Gruppenbildung, Hierarchien innerhalb von Gruppen und die verschiedenen Rollen der einzelnen Teilnehmer zum Ausdruck.

2.2 Szenarien

szenarien.tex

Um einen leichteren Überblick über die Aufgaben eines Managementsystems zu bekommen, ist es sinnvoll, typische Einsatz-Szenarien als Hintergrund zu verwenden. Mit Hilfe dieser Szenarien soll verhindert werden, daß eine Betrachtung wichtiger Aspekte verloren geht.

Auch kann man anhand eines Szenarios aufgestellte Behauptungen und Theorien leicht nachvollziehen und ggf. widerlegen oder als sinnvoll betrachten.

2.2.1 Bibliotheks-Szenario

bibliotheksszenario.tex

Als erstes wurde ein typisches Anwendungsbeispiel aus dem universitären Umfeld angenommen (bildliche Darstellung in Abbildung 2.1).

Beispiel

Die Universitätsbibliothek ist ein öffentlich zugänglicher Bereich, in dem vorwiegend Präsenz-Literatur liegt. Das bedeutet, man kann sich die Bücher nicht ausleihen und zu Hause lesen, sondern muß den Lesesaal nutzen.

Fachliteratur im Bereich der Informatik, ist leider als reine Leselektüre oft ungeeignet und die angegebenen Beispiele im Buch sollten besser sofort ausprobiert werden. Dazu ist in den meisten Fällen ein normaler Notebook ausreichend, immer häufiger jedoch ist ein Internet-Zugang wünschenswert.

Gründe hierfür wären zum Beispiel:

- Es ist nicht möglich, den Computer zum Buch zu transportieren, weil er in einem der Rechneräume steht, die Programmierliteratur aber in der Bibliothek ausliegt und dort nicht entfernt werden darf. Ein Zugang zum lokalen Intranet ist also nötig.
- Neuere Fachbücher verweisen oft auf Webseiten im Internet. Dies liegt daran, daß viele Projekte schon an die Öffentlichkeit getragen werden und dort implementiert und getestet werden, bevor sie richtig abgeschlossen sind. Somit geben Fachbücher (oder Artikel) oft Webseiten zu den jeweiligen Standards an, die der Leser bei Bedarf per Internet einsehen muß.
- Dadurch, daß viel Dokumentation im Internet frei zugänglich ist, beschränken sich Autoren von Fachbüchern auch gern auf die Angabe der URL, statt, wie früher, den Standard zumindest auszugsweise abzudrucken.

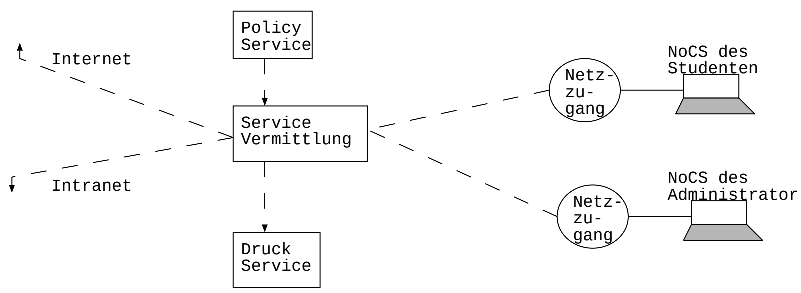

Abbildung 2.1: Szenario: Bibliothek

Will man in einer Bibliothek einen solchen Zugang anbieten, muß man verschiedene Anforderungen erfüllen:

Der Konfigurationsaufwand, den die Nutzer beim Anschluß ihrer Geräte an das lokale Netz haben, muß minimal sein. Dies bedeutet, daß eine automatische Konfiguration aller erlaubten Dienste angestrebt ist.

Dabei fallen zwei grundlegende Benutzeranforderungen ins Auge:

1. Die Nutzer wollen transparent auf 'fremde' Netze zugreifen, also zum Beispiel auf das Netzwerk der Physik oder das der Informatik an der TU-München oder das der Informatik an der LMU-München. Dabei kann es zu Überschneidungen in der Namenskonvention geben, z.B. drucker3 kann der Drucker an der TU genauso heißen, wie der an der LMU. Es muß also automatisch erkannt werden, welcher Drucker gemeint ist, bzw. nachgefragt werden.

2. Die Nutzer wollen Dienste nutzen. Solche Dienste können sein:
   - Internet-Zugang
   - Email
   - News
   - Drucker

2.2.2 Hotel-Szenario

hotelszenario.tex

Das zweite Szenario stammt aus dem kommerziellen Umfeld und wird im späteren Verlauf der Arbeit noch einmal aufgegriffen werden.

Mitarbeiter in Firmen sind heutzutage nicht nur häufiger unterwegs, sie nehmen auch noch ihre Computer mit und wollen auf ein Mindestmaß an alltäglichem Arbeitskomfort nicht mehr verzichten.

An diesem Punkt setzt dieses Szenario ein. Es beleuchtet die Situation, wenn ein Mitarbeiter von einem entfernten Hotel aus arbeiten möchte.

Er benötigt einen schnellen und unkomplizierten Zugang zum Netz (und will nicht alle drei Tage, wenn er im nächsten Hotel ist, seinen Rechner umkonfigurieren). Außerdem wünscht er Zugriff auf die lokalen Ressourcen seiner Firma, wobei sich lokal auf den Standort der Firma bezieht. Also: der Drucker seiner Abteilung, das Faxgerät, welches automatische ein Deckblatt generiert mit dem entsprechenden Firmenlayout. Eine Umleitung seiner Arbeitsplatz-Telefonnummer auf ein mobiles IP-Telefon wäre ebenfalls denkbar. Schließlich heißt 'auf Kongress sein' nicht, daß die Arbeit inzwischen von einem Kollegen erledigt wird, sondern, daß sie liegen bleibt.

In dieser Arbeit wurde sich neben dem Bibliothekszenario für die Hotel-Variante entschieden, da hier zusätzliche Anforderungen aus dem Bereich Sicherheit gestellt werden: Kunden sollen sich nicht gegenseitig abhören können, die Verbindung zwischen Kunde und seiner Firma soll ebenfalls sicher sein (nicht abhörbar, Daten nicht veränderbar).

Außerdem erscheint das Hotel-Problem als das komplexere in Hinsicht auf Sicherheitsmaßnahmen (Hotelgäste zueinander, genauso wie Hotelgäste zu hotelinternen Daten wie Reservierungen, Abrechnungsdaten etc.). Alternativ wäre auch ein Szenario in einer Firma denkbar, mit Außenmitarbeitern, Konferenzen und Heimarbeitsplätzen. Diese Idee wurde aber zugunsten der Hotelvariante mit Kongress verworfen. Die Ursachen sind mehrschichtig. Zum Einen hat man in einer Firma weniger das Problem, daß ein Mitarbeiter seine gewohnte Arbeitsumgebung wünscht, aber nicht direkt auf die nötigen Dienste zugreifen kann. Im Hotelszenario wird in diesem Punkt das Thema Tunneling gewonnen. Weiterhin wäre in einer Firma alles deutlich vertrauter, die Umgebung ist für Sicherheitsbetrachtungen zwar geeignet, aber häufig unrealistisch - das Netzwerk innerhalb einer Firma ist meist sicherheitsmäßig nicht dermaßen paranoid aufgebaut, wie es in einer Umgebung wie dem Hotel nötig ist.

In einem Hotel findet man in der Regel nur einen Typ Nutzer, nämlich den Kunden. Bei Administratoren wird in diesem Fall davon ausgegangen, daß sie die entsprechenden Mechanismen nicht nutzen müssen. Trotzdem ergeben sich im Hotel eine ganze Reihe von unterschiedlichen Service-Bereichen, die in diesem Ausmaß in einer Bibliothek ungewöhnlich wären (siehe Abb. 2.2).

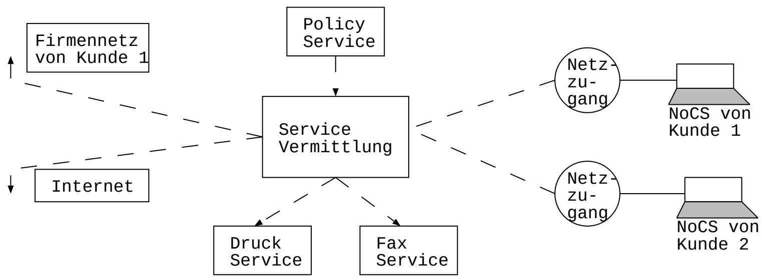

Abbildung 2.2: Szenario: Hotel

Der Hotelgast möchte vom Hotel aus direkt auf das interne Netzwerk seiner Firma zugreifen und es wie gewohnt nutzen, genauso wie die daran angeschlossenen Ressourcen, zum Beispiel Drucker.

Auch das Hotel stellt seine sonstigen Ressourcen nicht kostenfrei zur Verfügung. Zur Abrechnung im Hotel, benötigt man noch verschiedene Attribute, die ein Quality of Service (QOS) bezeichnen. Zum Beispiel ist es denkbar, daß mit einem Kunden ein Frei-Kontingent von 20 Seiten Papier auf dem Laserdrucker X vereinbart wurde oder ein Kunde eine allein ihm zur Verfügung stehende Bandbreite nach Außen zugesichert bekommen möchte (weil er z.B. per Video Konferenz an einer wichtigen Tagung teilnehmen muß) Genauso, nur umgekehrt, soll natürlich auch eine 'Deckelung' möglich sein, also eine Angabe der Art: es dürfen nicht mehr als 20 Seiten Papier auf dem Drucker X gedruckt werden.

2.3 Sicherheit

security.tex

2.3.1 Allgemeines

Mit dem Begriff 'Sicherheit' werden im Allgemeinen zwei Bereiche verbunden.

Zum Einen handelt es sich um *Ausfallsicherheit*. Darunter versteht man Maßnahmen, die verhindern sollen, daß ein Dienst nicht erbracht werden kann, weil zum Beispiel gerade der Strom ausgefallen ist. Gängige Maßnahmen zur Ausfallsicherheit sind: redundante Hardware (ein Ersatzsystem kann die Arbeit eines anderen Systems automatisch übernehmen, wenn dieses ausfällt), unterbrechungsfreie Stromversorgungen (Batterien, die für einen gewissen Zeitraum den Rechner mit Strom versorgen können um eine korrekte Verarbeitung von Diensten zu gewährleisten), Notstromaggregate (Geräte, die selbständig Strom erzeugen und ins Netz einspeisen können, falls der öffentliche Strom länger ausfällt) und verschiedene Service-Verträge mit externen Dienstleistern (innerhalb von x Stunden wird ein fehlerhaftes Gerät ausgetauscht).

Um diese Art Sicherheit soll es in der vorliegenden Arbeit nicht gehen. Vielmehr soll es um den Bereich *Datensicherheit* gehen und wie man verhindert, daß jemand das System abhört, sensible Daten herausfiltert oder den kompletten Ablauf im Managementsystem stört.

Unter Datensicherheit versteht man die Sicherheit der Datenübertragung, sowie Datenspeicherung. Dabei handelt es sich um den Verlust oder das Verfälschen von Daten und somit Informationen. Dies kann lokal auf dem Speichermedium eintreten oder aber auch während der Datenübertragung im Netz. Letzteres Problem, also der Verlust oder das Verfälschen von Daten auf ihrem Übertragungsweg soll in diesem Kapitel beleuchtet werden.

Sicherheit ist in der heutigen Zeit ein ernst zu nehmendes Thema geworden. Wenn vertrauliche Verträge von Managern verschiedener Unternehmen per Email vorbesprochen werden, ist es unumgänglich jegliche Zugriffe Dritter auf diese Daten zu unterbinden.

Klassische Möglichkeiten sind:

- Zugriffskontrolle auf den Rechnern - Passwort-Zugang
- Verschlüsselte Speicherung der relevanten Daten
- Daten 'verstecken' (Steganografie)

Datensicherung gegen Verlust durch Löschen (Backupstrategien, Spiegelung von Daten) soll nicht Gegenstand dieser Arbeit sein und wird deshalb außer Acht gelassen.

Sicherheits-Strategien und -Risiken finden sich quer durch alle Schichten (hier am Beispiel der OSI-Schichten):

Layer	Name	Sicherungs- / Sabotage-Möglichkeiten
1	Physical	Kabel durchschneiden
2	Data Link (MAC)	Switch manipulieren auf MAC-Adress-Ebene
3	Network	IP-Dienste nach IP-Adressen filtern, (nicht geroutete Adressen verwenden...)
4	Transport	Protokolle verbieten (z.B. UDP)
5	Session	Sockets sperren
6	Presentation	verschlüsselndes Protokoll entwerfen (z.B. verschlüsseltes FTP)
7	Application	Verschlüsselung (PGP), Passwortschutz

In einem Hotel geht man in erster Linie davon aus, daß die Benutzer sich gegenseitig 'feindlich' gesinnt sind und somit klar in allen Aktivitäten voneinander getrennt werden müssen. Dies bedeutet, daß Kunden nicht in der Lage sein dürfen, die Informationen, die ein anderer Kunde bekommt, ebenfalls zu erhalten. Ob dies zufällig oder absichtlich geschieht, spielt keine Rolle. Schließlich kann man nie wissen, wer das Nachbarzimmer gebucht hat.

Im Idealfall sollte es allgemein nicht möglich sein Zugang zum Computer oder zu den Übertragungen, die einen anderen Kunden betreffen, zu erhalten. Dies beinhaltet nicht nur Informationen, die der Kunde selbst verschickt oder von jemandem empfängt, sondern auch Informationen, die einzelne Komponenten des Managementsystems untereinander austauschen und die diesen Kunden betreffen.

In der allgemeinen Praxis werden solche Probleme mit Verschlüsselung der Informationen gehandhabt, sowie mit weiteren kryptografischen Mitteln, die sicherstellen, daß die übertragenen Daten nicht verfälscht wurden. Das bedeutet, eine verschlüsselte Kommunikation sowohl intern, als auch nach außen, wird angestrebt.

Ob man eine Kommunikation von Kunden untereinander (der Nutzer des Notebooks in Zimmer 240 redet mit dem in Zimmer 107 über das lokale Netzwerk) zulassen möchte, sollte im Einzelfall regelbar sein.

security-hotel.tex Sicherheit im Hotel

In einem Hotel soll der Nutzer zum Beispiel eine Verbindung und einen Zugang zum Firmen-Netz von außen bekommen. Hierbei muß man damit rechnen, daß die Verbindung zum Firmen-Netz grundsätzlich unsicher ist und abgehört werden kann. Außerdem ist es möglich, daß auch Fremde versuchen, sich über diesen Weg an ein Firmen-Netz anzubinden. Beiden Punkten muß man begegnen. Abhörmaßnahmen lassen sich durch konsequente Verschlüsselung der transportierten Daten unterbinden. Eine 'Einwahl' durch nichtautorisierte Personen kann von einer einfachen Passwort-Abfrage, über Einmal-Passworte, bis hin zu fest vorgeschriebenen Absenderadressen oder biologischen Erkennungsmerkmalen (Fingerbdruck) verhindert werden.

In vielen Unternehmen wird der Zugriff auf Ressourcen anhand der IP-Adresse des Zugreifenden geregelt. Deshalb muß es eine Möglichkeit geben, den Rechner im Hotel so aussehen zu lassen, als wäre er mit seiner normalen IP-Adresse und seinem eigenen Domainnamen im Hausnetz angebunden (alternativ kann man auch versuchen, die IP-Adressen, die das Hotel intern an Kunden vergibt freizuschalten, was aber einen erhöhten - und somit unerwünschten - Administrationsaufwand darstellt. Der andere Weg, daß der Nutzer grundsätzlich seine eigene IP-Adresse behält, egal wo er ist, ist aus Internet-technischer Sicht (Routing) oft nicht realisierbar.

Ein weiterer Punkt ist eine transparente Kommunikation. Das bedeutet: der Hotelkunde kann 'über den Hotelanschluß in einem anderen Netz' arbeiten, wobei sich für den Kunden kein Unterschied in der Handhabung seiner Funktionen ergibt, ob er jetzt vor Ort ist (z.B. an seinem Arbeitsplatz) oder weit entfernt (z.B. in einem Hotelzimmer). Er sollte in beiden Fällen zum Beispiel sein Laufwerk K mit dem Rechner superserver.firma.domain verbinden können (Windows-Terminologie). Daß dazu auch entsprechende Mechanismen auf der Gegenseite (aus Hotelsicht also beim Arbeitgeber des Gastes) vorhanden sein müssen, ist Voraussetzung.

Verbindungen, die leitungstechnisch das Haus und somit den direkten Einflußbereich verlassen, müssen unbedingt durch eine Verschlüsselung gesichert werden können.

2.3.2 Sicherheit innerhalb des Management-Systems

Es ist nicht ausreichend, nur an die Verbindung der Nutzer mit anderen Teilnehmern zu denken. Ein Managementsystem besteht aus einer Reihe interner Programme, die miteinander kommunizieren. Dabei muß sich jedes Programm darauf verlassen können, daß die Informationen, die es von einem vermeintlichen Hilfsprogramm bekommt, auch echt sind. Das heißt, daß dieses Hilfsprogramm tatsaechlich zum Managementsystem gehört und sich nicht nur als dazugehörig ausgibt. Es wird also ein Authentifizierungsmechanismus für die Komponenten untereinander benötigt. Ebenfalls sollte es nicht möglich sein, die Daten, die die Komponenten untereinander austauschen, von außen abzuhören. Deswegen ist eine verschlüsselte Kommunikation auch in diesem Punkt erstrebenswert. Dieses Problem wird in der Arbeit von Harald Roelle [R9̈9] ausführlich behandelt.

2.3.3 Weitere Sicherheits-Aspekte in gemanageten Systemen

Mit Verschlüsselung allein ist keine absolute Sicherheit zu erreichen. Es bleiben zusätzlich noch eine Reihe anderer Möglichkeiten um Zugriff auf die gespeicherten Daten zu erhalten. Als Stichwort kann man hier zum Beispiel 'Social Hacking' nennen.

Unter 'Social Hacking' versteht man den Einbruch über soziale Strukturen. Dabei werden Freunde, die direkt mit den gewünschten Daten in Kontakt kommen (z.B. Schwestern und Patientendaten) gebeten die entsprechenden Daten zu besorgen. Der Angriff erfolgt dann quasi von innen aus den eigenen Reihen heraus.

Auch ein direkter Zugriff auf die Hardware kann oft erfolgversprechend sein. Das bedeutet, daß der 'Einbrecher' sich direkt physikalisch zum Rechner begeben muß. Er kann dann zum Beispiel die Festplatte mit den benötigten Daten ausbauen und diese zu Hause in Ruhe entschlüsseln und auswerten.

Eine vollkommene Sicherheit kann man nicht garantieren, man kann es einem 'Einbrecher' höchstens besonders schwer und teuer gestalten, an wichtige Daten zu gelangen. Man muß im Vorfeld abschätzen, wie wertvoll die betreffenden Daten wirklich sind und dementsprechende Sicherheitsmaßnahmen ergreifen - denn Sicherheitsmaßnahmen können sehr teuer sein.

2.3.4 Sicherheit am Beispiel des Bibliotheks-Szenarios

security-bibliothek.tex

In einer Bibliothek sind die folgenden Sicherheitsfragen zu klären:

Geht man davon aus, daß die Bibliothek und ihr Service-Angebot auch von Fremden (also nicht ausgewiesenen Studenten) genutzt wird, muß man alle Zugänge und Verbindungen zu nichtöffentlichen Bereichen besonders sichern.

Ansonsten sollten kostenpflichtige Services wie Drucken vor Fremdnutzung geschützt sein. Besonders trifft dies Verbindungsaufbau und Authentifizierung beim Druck-Service. Hier sollte ein Mißbrauch durch wiederholtes Senden der Authentifizierungs-Bytes nicht möglich sein.

Ganz allgemein kann man folgende Regeln festlegen:

- niemals Authentisierungssequenzen im Klartext über das Netz senden
- das Netz an sich auf physikalischer Ebene soweit wie möglich abhörsicher gestalten (Switching)
- Verbindungen in denen vertrauliche Daten versendet werden, grundsätzlich verschlüsseln

Außerdem kann man gezielte Zusatzmaßnahmen ergreifen:

- physikalischer Zugang zu Servern sollte grundsätzlich nur Administratoren vorbehalten sein
- Passworte entsprechend den Regeln wählen
- physikalischen Zugang auch zu den Servicegeräten (Drucker) kontrollieren und ggf. erschweren
- bestimmte Verbindungen grundsätzlich verschlüsseln oder komplett ein Protokoll mit Verschlüsselung verwenden

Diese Regeln sind mit den normalen, zur Verfügung stehenden Mitteln umsetzbar. Sie benötigen keine zusätzliche teure Hard- oder Software. Dafür bieten sie auch keine extrem hohe Sicherheit.

2.4 Policies

policies.tex

2.4.1 Begriffsklärung

policies-begriffe.tex

Policies sind Regeln, die die Abläufe beeinflussen. Solche Regeln können sowohl einschränkender Natur sein, als auch erweiternder Natur. Das bedeutet, eine Policy kann zum Beispiel eine Dienstnutzung unterbinden, sie kann aber auch Maßnahmen innerhalb eines Systems anstoßen, die die Dienstnutzung beschleunigen oder vereinfachen könnten. Ein klassisches Beispiel hierfür wäre es, eine zusätzliche Außenleitung aufzubauen, sobald die bestehende Leitung ausgelastet ist. Dies würde sich dann auf die Nutzung der Außenanbindung beziehen.

Um solche Policy-Regeln und den Ablauf, wie eine Policy umgesetzt werden kann, zu formalisieren, wurden verschiedene 'Zwischenstationen' eingeführt.

Die grundlegenden Begriffe dazu werden im Internet Draft [ESM99c] für Policy-basiertes Management definiert.

Abbildung 2.3 verdeutlicht die Beziehung der einzelnen 'Stationen' grafisch.

SLA (Service Level Agreement) Unter SLA versteht man einen Dienstleistungsvertrag zwischen dem Kunden und einem Dienstleister, der die vereinbarte Dienstleistung spezifiziert.

PDP (Policy Decision Point) Der PDP ist die Instanz, in der Policy Entscheidungen getroffen werden.

PEP (Policy Enforcement Point) Der PEP ist die Instanz, der die Implementierung zur Umsetzung der vom PDP getroffenen Policy-Entscheidung enthält.

PIN (Policy Ignorant Node) Ein PIN ist eine Instanz, die nicht der Kontrolle durch die Policy unterliegt, bzw. eine solche Kontrollmöglichkeit nicht unterstützt.

2.4.2 Anforderungen an Policies

policies-anforderungsanalyse

Policies sind Regeln. Dabei werden die einzelnen Aussagen von Reglements als Policy-Regel bezeichnet, das gesamte Regelwerk kurz als 'Die Policy'. Policies sind zum Beispiel:

- die Gabel liegt links vom Teller und das Messer rechts
- zwischen 18:00 Uhr und 9:00 Uhr haben die Eingänge zum Gebäude verschlossen zu sein
- ab 9:00 Uhr morgens ist mindestens ein Mitarbeiter im Büro
- die Rechnungen für Kunde A in München gehen grundsätzlich nicht an den Kunden selbst, sondern an seine Mutterfirma in Berlin
- die abzuschickende Post wird in die blaue Kiste mit der Aufschrift 'Postausgang' gelegt
- alle Mitarbeiter der Abteilung Technik haben einen Schlüssel für den Technikraum
- wenn jemand auf diesem Gerät von der Nummer 1234 aus anruft, bekommt er einen login-prompt, ansonsten wird das Gespräch nicht angenommen (Zugangsbeschränkung auf Geräten)

Man sieht, daß Regeln in den verschiedensten Teilen des Lebens vorkommen und nicht unbedingt technischer Natur sein müssen. Fast alle sind der Art: 'Wenn dieses und jenes gegeben ist, dann veranlasse oder verweigere etwas'.

Policy-Regeln können nicht nur konkrete Daten betreffen, wie zum Beispiel eine IP-Adress-Angabe, sondern auch Abrechnungsmodalitäten, Freieinheiten, Sperrung von Services und sonstige Besonderheiten:

- der Nutzer A darf den Drucker D nutzen, aber nicht den Drucker P
- wenn das Nomadische System X Zugang bekommt, darf den Nomadischen Systemen Y und Z nicht gleichzeitig Zugang gewährt werden
- der Service S darf nur von 5 Nutzern gleichzeitig verwendet werden
- alle Accountingdaten bezüglich Drucker werden von Service B gesammelt.
- wenn die Außenanbindung eine Auslastung von 90% hat, wird eine zweite Leitung dazugeschaltet

Im Fall von Managementsystemen im Computer sind Policies Gruppen von Regeln, die man festlegen kann und an denen sich alle teilnehmenden Komponenten soweit wie möglich orientieren müssen. Solche Komponenten entsprechen dann einem PEP. Sie setzen die Regeln durch. Ausgenommen sind Komponenten, die nicht policyfähig sind (weil sie so alt sind, daß ihre Implementierung die Integration in ein Managementsystem und die Verwendung von Policies nicht vorsieht). Sie werden PIN genannt. Würden sich die Aktionen, die aus einer solchen Gruppe von Regeln resultieren, gegenseitig widersprechen, spricht man von einem Policy-Konflikt. Solche Konflikte sollten vermieden werden. Treten sie trotzdem auf, müssen sie speziell behandelt werden, d.h. alle betroffenen PEPs müssen miteinander koordiniert werden. Policy-Konflikte kann man erkennen und bereits im PDP entsprechend behandeln.

Um zu ermitteln, welche Arten von Policies beim Management Nomadischer Systeme benötigt werden, wurden die beiden bereits bekannten Grundszenarien um mögliche Policy-Regeln erweitert. Bevor man Policy-Regeln in Klassen oder Typen unterteilen kann, benötigt man Beispiel-Regeln. Zur Findung expliziter Regeln wird angenommen, daß:

alle Services des Systems nicht frei zugänglich sind Will der Nutzer einen Service nutzen, muß er sich authentifizieren. Dies passiert im klassischen Fall durch einen Benutzernamen und ein Passwort. Man kann einen Nutzer aber auch anhand der eindeutigen MAC-Adresse seiner Ethernet-Karte identifizieren oder anhand seines Daumenabdruckes (biometrische Verfahren).

sämtliche genutzten Ressourcen finanziell verrechnet werden müssen Dienstleistungen verursachen in der Regel Kosten. Diese können materieller Art sein, wie Druckerpapier oder virtueller Art sein, wie Rechenzeit. Alle diese Kosten kann man in Geldkosten umrechnen (z.B. eine Seite Druckerpapier kostet 10 Pfennige und 20 Minuten Rechenzeit 20 Pfennige). Um diese Kosten auch vom Benutzer einfordern zu können, muß man erfassen, wieviel Service welcher Nutzer genutzt hat. Wenn zum Beispiel gedruckt wird, wird Druckerpapier verbraucht, wer Web-Surft, verbraucht IP-Volumen und wer elektronische Videos sammelt, verbraucht Plattenplatz. Weiß man, wer wie viel verbraucht hat, kann man dem Nutzer dies in Rechnung stellen.

Eine solche Buchführung hat noch einen weiteren positiven Nebeneffekt: unliebsame Überraschungen wie: 'das Druckerpapier ist alle' oder 'der Drucker benötigt eine neue Tonerkasette, aber es ist noch keine nachbestellt worden', kann man vermeiden indem man diese Daten auswertet und rechtzeitig automatisch auf einen niedrigen Bestand der Ressource hinweist.

ein Quality of Service angeboten werden soll 'Quality of Service' (QOS) ist die Möglichkeit, einen bestimmten Nutzer bevorzugt zu behandelt. So könnte man beispielsweise mit dem Nutzer vereinbaren: Wenn er einen Druckjob absendet - also den Druckservice nutzen möchte - wird der als nächstes behandelt, egal wie viele andere Druckjobs bereits auf Abarbeitung warten. Auch eine garantierte Bandbreite ist ein Quality-of-Service-Merkmal. Solche bevorzugte Behandlung bedeutet aber andererseits wieder zusätzlichen Aufwand dadurch, daß eventuell weitere Hardware gekauft werden muß, um diesen Zusatzservice anbieten zu können. Somit kostet QOS Geld, welches in der Regel auf den Verursacher, also denjenigen, der bevorzugt behandelt wird, umgelegt wird.

Ein QOS, Abrechnungen von verbrauchten Ressourcen und Zugangsbeschränkungen kann man als Regel und somit als Policy betrachten. Damit kann man einem Benutzer Policy-Regeln zuordnen in denen seine Quality-of-Service-Merkmale stehen oder sein Benutzername mit Passwort erfasst sind.

Policies kann man aber auch einem Service zuordnen. Diese Zuordnung kann von der Art sein: 'wenn der Nutzer X (der durchaus wieder ein Service sein darf) den betreffenden Service nutzen will, wird er vorrangig behandelt (oder die Nutzung verweigert)'.

Policies kann man auch einem kompletten Managementsystem zuordnen und festlegen: 'täglich um 22:00 Uhr müssen alle Geräte mit Festplatten, ein Backup ihrer Platten machen' oder 'alle Services, die externe Geräte (Drucker) ansteuern sind zwischen 22:30 und 6:00 Uhr nicht verfügbar, es sei denn, in den Policies des Nutzers steht etwas Gegenteiliges'.

Um Policies innerhalb eines Systems möglichst effizient umsetzen zu können, verwendet man ein Client-Server-System. Dabei übernimmt der Service, welcher einem Nutzer eine Dienstleistung erbringen soll, die Rolle des Clients und fragt bei einem Policy-Service nach, welche Regelungen er für diesen Nutzer anwenden soll. Der Policy-Service wertet die Regeln aus und teilt das Ergebnis dem anfragenden Service mit. Der Policy-Service wird in diesem Fall als PDP bezeichnet. Der anfragende Service bekommt vom PDP eine Art Anweisung, was er zu tun hat und setzt diese Anweisung um. Er übernimmt die Rolle des PEP.

Beispiel: Bibliothek

policy-bibliothek.tex

In Bibliotheken kann man Nutzer in Nutzergruppen unterteilen und ihnen so einheitliche Policy-Regeln zuordnen. Alternativ kann man auch Services und Policy-Regeln in Gruppen ordnen und einem Nutzer jeweils eine solche Service-Gruppe und Policy-Gruppe zuordnen.

Ein normaler Student darf zum Beispiel nur innerhalb seines vorausbezahlten Kontingents den Drucker nutzen, wohingegen ein Administrator diesbezüglich keine Begrenzungen hat. Ein Gast könnte zum Beispiel Zugriff auf die lokale Datenbank der Bibliothek erhalten, aber keinen Internetzugang oder Zugang zu anderen Teilnetzen der Universität.

Hierbei kristallisieren sich vier Grundtypen von Benutzern, bzw. Benutzerkreise heraus, denen einheitliche Service-Policies zugewiesen werden könnten:

1. Professoren bzw. Assistenten und wissenschaftliche Mitarbeiter, die Internet nutzen wollen und zusätzlich auf u.U. geschützte Bereiche ihrer Lehrstühle zugreifen dürfen
2. Studenten, die Zugang zu den Daten ihres Fachbereiches bekommen wollen, sowie Internetdienste (z.B. WWW) nutzen möchten.
3. Administratoren, die uneingeschränkten Zugang auf alle Systeme ihres Aufgabenbereiches benötigen
4. Gäste, die sehr eingeschränkte Privilegien haben dürfen, z.B. Drucker der Bibliothek nutzen, kein Internet, kein Email...

Dabei fallen zwei grundlegende Benutzer-Anforderungen ins Auge:

1. Die Nutzer wollen transparent auf 'fremde' Netze zugreifen, also zum Beispiel auf das Netzwerk der Physik oder das der Informatik an der TU-München, oder das der Informatik an der LMU-München. Dabei kann es zu Überschneidungen in der Namenskonvention geben, z.B. drucker3 kann der Drucker an der TU genauso heißen, wie der an der LMU. Es muß also automatisch erkannt werden, welcher Drucker gemeint ist, bzw. nachgefragt werden.

2. Die Nutzer dürfen nicht alle Dienste unkontrolliert nutzen. Es muß zum Beispiel eine unerlaubte Nutzung des Internets durch einen Gast unterbunden werden.

Beispiel: Hotel

policy-hotel.tex

Im Hotel hat man vorwiegend einen Typ: Kunde. Allerdings kann man den ebenfalls nach Kategorien sortieren: No-Service, Standard-Service, Premium-Service, Customized.

Dabei bedeutet die Kategorie No-Service, daß der Kunde keinerlei Netz-Zugang erhalten soll.

Standard- und Premium-Service sind Kategorien, in denen mehr oder weniger Service geleistet wird. So könnte sich Premium-Service dadurch auszeichnen, daß ein solcher Kunde bei der Diensterbringung vor einem Standard-Service-Kunden immer Vorrang hat.

Customized ist eine 'Kategorie' die besagt, daß dieser Kunde ganz individuelle Serviceleistungen ausgehandelt hat, die nicht in eine der anderen drei Kategorien hineinpassen.

Weiterhin muß jeder Kunde eindeutig über seinen Usernamen und das zugehörige Passwort identifizierbar sein.

Alle Service-Gruppen müssen mit den ihnen angehörigen Services beschrieben sein. Hierbei wird der Einfachheit halber ein 'Service' gleich als konkreter Server aufgefaßt.

2.4.3 Benötigte Policy-Regeln beider Szenarien zusammen

policies-szenarien-anforderungen.tex

Aus den oberen beiden Beispielszenarien kann man eine Reihe allgemeiner Anforderungen ableiten. Ein Teil dieser Anforderungen läßt sich mit einfachen technischen Mitteln umsetzen. Andere Anforderungen benötigen umfangreichere Zusatzimplementierungen.

Die gefundenen Policy-Anforderungen kann man in drei Gruppen gliedern: Management, Infrastruktur und Abrechnung. Als zusätzliche Gruppe ist hier noch Diverses angeführt. Sie enthält alle Anforderungen, die sich nicht in eine der drei anderen Gruppen einordnen lassen.

Management

- Gruppenbildung kann man zu verschiedenen Zwecken einsetzen. Sie kann zum Beispiel im Hotelszenario das Eintragen (den checkin) eines neuen Kunden erleichtern, indem man den Kunden einer allgemeinen Service-Gruppe zuordnet. In dieser Servicegruppe ist festgelegt, welche Dienste er nutzen darf, sowie weitere Parameter wie QOS. Ein Kunde kann zum Beispiel der Servicegruppe Standard oder der Servicegruppe Premium zugeordnet sein. Beide Gruppen haben unterschiedliche Privilegien und Nutzungsbedingungen. *(Randnotiz: Gruppenbildung von Services)*

  Gruppenbildung von Personen kann wichtig werden, wenn es um Organisatorisches geht. Von einer Gruppe Kunden seien zum Beispiel folgende Parameter bekannt: *(Randnotiz: Gruppenbildung von Personen)*

- alle Kunden kommen am gleichen Tag
- alle Kunden fahren am gleichen Tag wieder ab
- alle Kunden haben einen Service gebucht, in dem ihnen immer eine bestimmte Mindestkapazität ins Internet zur Verfügung steht
- es sind 20 Kunden, die alle zur gleichen Konferenz wollen

Aus diesen Informationen kann ein Managementsystem unter anderem folgende Anforderungen ableiten:

- es wird eine Spitzenzeit geben, zu der alle 20 Teilnehmer auf ihre Internet-Verbindungen angewiesen sind (Konferenzschaltungen)
- alle Teilnehmer werden etwa zur gleichen Zeit aufstehen, duschen und frühstücken wollen (und damit eine entsprechende Geräuschkulisse verursachen)
- es werden wahrscheinlich mehrere Teilnehmer gleichzeitig einen Drucker benötigen (Handouts für den Vortrag am nächsten Tag ausdrucken), sowie ein Kopiergerät
- es werden auf einmal 20 zusätzliche Zimmer mit Zimmerservice zu versorgen sein

Das Zimmerbuchungs-System kann jetzt zum Beispiel aufgrund dieser Informationen folgende Maßnahmen ergreifen, bzw. ergreifen lassen:

- alle 20 Teilnehmer in ein und dem selben Teil des Gebäudes unterbringen um so die morgendliche Lärmbelästigung für andere Gäste zu minimieren
- die Außenbandbreite des Hotels temporär aufstocken
- einen weiteren Drucker auf diesem Stockwerk zur Verfügung stellen
- die Unterbringung der Gäste ablehnen, da für so viele Personen nicht genug Ressourcen zur Verfügung stehen

Auch im Bibliotheks-Szenario gibt es den Punkt der Gruppenbildung. Zum Beispiel kann ein Nutzer der Gruppe Student angehören und hat somit ein Druckkontingent von 100 Seiten pro Semester. Ein anderer Nutzer kann der Gruppe Administrator angehören und hat freien und unbegrenzten Zugang zu allen Ressourcen im System.

Gruppenbildung von Policies

Weist man einem Nutzer eine Service-Gruppe zu, so liegt es nahe, ihm auch eine Policy-Gruppe zuzuweisen. Mit einigen wenigen Ausnahmen kann man davon ausgehen, daß jeder Nutzer, der einer Nutzergruppe oder einer Servicegruppe zugeordnet ist, den gleichen Policy-Regeln unterliegt, wie ein anderer Nutzer. Ein Hotelkunde zum Beispiel, der einen Standard-Service gebucht hat, unterscheidet sich policy-seitig lediglich durch seine individuellen Einstellungen wie Benutzername und Passwort von einem anderen Hotelkunden, der den selben Standard-Service gebucht hat.

Eine Policy-Gruppe kann man nicht nur Nutzern zuweisen sondern auch Services.

Viele Einzel-Services werden ähnlich behandelt, z.B. ein Druckservice, der Papier ausdruckt und einer, der Farbfolien ausdruckt.

Vorteil von Gruppenbildung

Man kann allgemein feststellen, daß Gruppenbildungen immer dann sinnvoll sind, wenn man einheitliche Einstellungen benötigt. Sie erleichtern es dem Administrator den Überblick über die möglichen Varianten zu behalten und vermindern Fehler, die entstehen können, wenn bei der Konfiguration eines neuen Nutzers eine einzelne Regel vergessen wurde einzutragen.

Zeitangaben

- Zeitangaben sind ein wichtiges Mittel um Konfigurationen und Gültigkeiten von Angaben klarer herausstellen und detaillierter zu bestimmen. So kann ein Kunde ein Zimmer für drei Tage buchen und danach wird aus einem unbekannten Grund der Eintrag des Kunden in der Policy-Datei nicht gelöscht. Es wäre denkbar, daß ein anderer Kunde zufällig die Zugangsdaten des bereits nicht mehr im Hotel weilenden Kunden erfährt und somit auf Kosten des ersten Kunden Dienstleistungen des Hotels nutzen. Auch gibt es pauschale Hotelreservierungen, bei denen zu Beginn des Jahres gleich auf ein Jahr im Voraus regelmäßig Zimmer vorgebucht werden (an jedem Monatsanfang drei Zimmer für drei Tage und ein Tagungsraum). Auch hier ist in der Regel nicht erwünscht, daß Dienstleistungen außerhalb der gebuchten Zeiten genutzt werden.

Die Gruppenbildung und Angabemöglichkeit von Zeiten und periodischen Wiederholungen muß vom PDP und der Policy-Syntax unterstützt werden.

Infrastruktur

Tunneling

- Unter Tunneling versteht man das transparente Weiterreichen von Paketen über ein anderes logisches Netz. Dabei werden die zu transportierenden Daten am Anfang des Tunnels gekapselt und am anderen Ende wieder 'ausgepackt'. Dies kann hilfreich sein, wenn man transparent über ein Drittnetz auf ein anderes Netzwerk zugreifen möchte - zum Beispiel der Kunde, der über Internet auf die Daten des entfernten Firmennetzes zugreifen muß. Tunneling und Verschlüsselung werden in der Praxis meist gleichzeitig eingesetzt.

  Tunneling läßt sich durch den Einsatz entsprechender Hardware realisieren. Dazu ist es allerdings nötig automatisch und unter Beachtung von Policy-Regeln diesen Tunnel aufzubauen und offen zu halten. Dazu muß 'das andere Ende des Tunnels' informiert sein und diese Funktion ebenfalls unterstützen, da das Tunnelende sich meist nicht mehr im direkten Einflußbereich des lokalen Management-Systems befindet.

QOS

- QOS (Quality Of Service) wird derzeit in der Öffentlichkeit fast noch nirgends garantiert. Ein Trend in diese Richtung ist aber vorhanden. Das Fehlen von QOS ist oft auf das Fehlen passender Protokolle zurückzuführen, sowie auf die Tatsache, daß die bisherige Infrastruktur meist nur ein grundlegendes Service-Angebot erlaubt hat. Ursache hierfür war, daß Netzwerkdienste nur von einer sehr kleinen Gruppe Menschen genutzt wurde. Durch die Verbreitung der neuen Kommunikationsmöglichkeiten wie Email und WWW hat sich dieses Bild grundlegend geändert.

Notebooks sind auf Reisen keine Seltenheit mehr und ihre Besitzer möchten auf den tägliche Komfort Email und Web nicht mehr verzichten. Mit der steigenden Verbreitung steigen auch die Ansprüche der Kunden an den jeweiligen Dienstleister vor Ort.

Zur Durchsetzung von QOS muß man verschiedene Mechanismen heranziehen. Zum Einen kann man Leitungs-Qualitäten (Stichwort: Bandbreitenreservierung) mit Hilfe moderner Netzwerkprotokolle wie ATM umsetzen. Eine bevorzugte Behandlung des Kunden, wenn er zum Beispiel einen Drucker nutzen möchte, kann man im Druckservice einbinden. In vielen Fällen ist allerdings noch eine Erweiterung vorhandener Protokolle nötig.

Abrechnung

- Abrechnung von Diensten ist besonders im kommerziellen Bereich gewünscht. Das Hauptproblem sind in diesem Punkt diejenigen Agenten oder Systemprogramme, die Daten sammeln müssen um sie zur finanziellen Abrechnung oder zur Einhaltung von Policies (Freikontingente) zur weiteren Verarbeitung zur Verfügung zu stellen. Eine detaillierte Abrechnung kann ein Management-System deutlich belasten, weswegen hier eine Lastverteilung hilfreich sein kann. Diese kann man zum Beispiel erreichen, indem man verschiedene Programme hat, die jeweils unterschiedliche Typen von Abrechnungsdaten sammeln, zum Beispiel verbrauchtes IP-Volumen oder verbrauchtes Emailvolumen oder Druckerpapier. Dienst-Abrechnung

Diverses

- Verschlüsselung ist eine grundlegende Sicherheitsmaßnahme. Allerdings kann diese meist global stattfinden und muß nicht zwangsläufig auf den einzelnen Kunden angepaßt werden. Verschlüsselung

  Verschlüsselung ist bereits in vielen Netzwerkprodukten wie Routern eingebaut und muß lediglich aktiviert werden.

2.4.4 Ein allgemeines Policy-Modell

policies-modell.tex

Die Notwendigkeit von Policies im Allgemeinen ist bereits bekannt. Derzeit arbeiten einige Gruppen an konkreten Vorschlägen für ein allgemeines Policy-Modell. Einer dieser Vorschläge kommt von Elleson, Ed, Strassner, Weiss und Westerinen . Die Arbeit lag zum Zeitpunkt der Erstellung dieses Kapitels als Internet Draft vor.

Die hier beschriebene Version dieses Vorschlages beruht auf dem Draft vom März 1999 (strassner-policy-terms-01) inzwischen ist diese Version nicht mehr verfügbar.

Das allgemeine Policy-Modell ist in Abb. 2.3 dargestellt. Es ist nach dem Draft [ESM99c] erstellt.

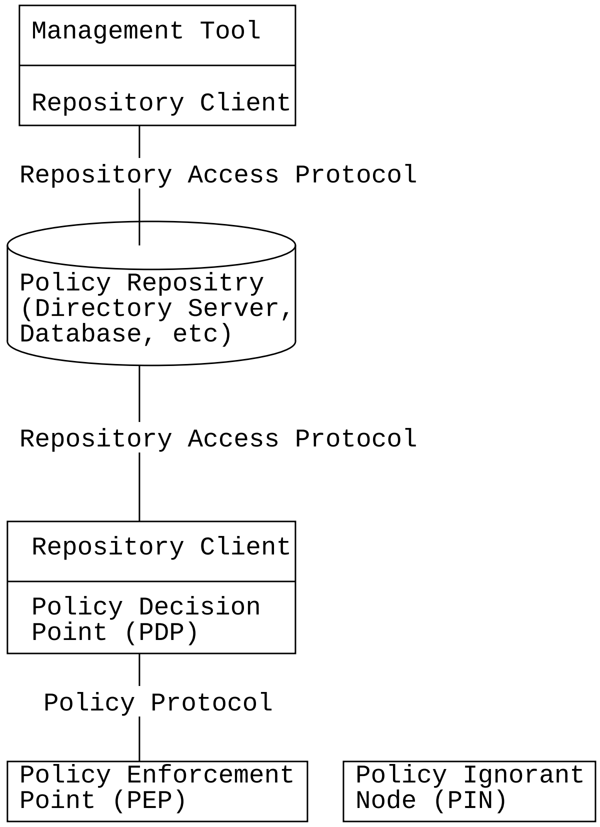

Abbildung 2.3: Allgemeines Policy-Modell

Demnach gibt es ein *Management-Tool*, welches einen *Repository Client* besitzt. Dieser greift mittels eines *Repository Access Protocols* (dabei kann es sich um ein proprietäres Protokoll handeln) auf einen Datenbestand zu, der auf diesem Wege verwaltet wird.

Auf der Abbildung 2.3 unten rechts befinden sich *Policy Ignorant Nodes (PIN)*. Dabei handelt es sich um Programme und Services, die nicht mit Policies zu beeinflussen sind, da sie Policies generell nicht unterstützen oder ein inkompatibles Policy-System unterstützen.

Policy Enforcement Points (PEP), sind Programme, die mit Policies beeinflußt werden können und Policies umsetzen können. Sie kommunizieren mit einem *Policy Decision Point (PDP)*, der seinerseits einen Repository Client hat, mit dem er auf den vom Managementsystem gepflegten Datenbestand zugreifen kann. Dabei ist vorgesehen, daß das Repository Access Protocol, welches zwischen Repository Client und Policy Repository, sowie zwischen Policy Repository und Management Tool verwendet wird, jeweils identisch sind. Es muß aber nicht mit dem Policy Protocol, welches zwischen PEP und PDP verwendet wird, identisch sein.

Im Policy Decision Point werden ausschließlich Entscheidungen getroffen. Das Ergebnis wird dann dem Policy Enforcement Point mitgeteilt. Nur dieser besitzt die Implementierung zur Durchsetzung der Policy-Entscheidung.

Das allgemeine Vorgehen bei einer Policy-Entscheidung zeigt Abb 2.4. Ein PEP erhält ein Ereignis, welches eine policy-basierte Handlung erfordert. Der PEP stellt an den PDP eine konkrete Anfrage, was er tun soll, bzw. wie die dazugehörige Policy-Entscheidung aussieht.

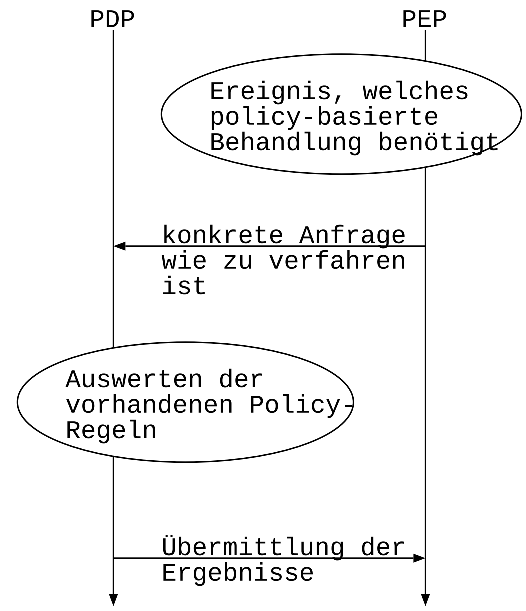

Abbildung 2.4: Policy-Nutzung

Der PDP wertet die ihm vorliegenden Policies aus und trifft danach eine Entscheidung. Diese übermittelt er an den PEP, der sie umsetzen kann.

Das Policy-Objekt-Modell ist in Abb 2.5 dargestellt. Es handelt sich um ein komplettes Objektmodell einer Policy. Dieses Modell beinhaltet sechs Hauptklassen.

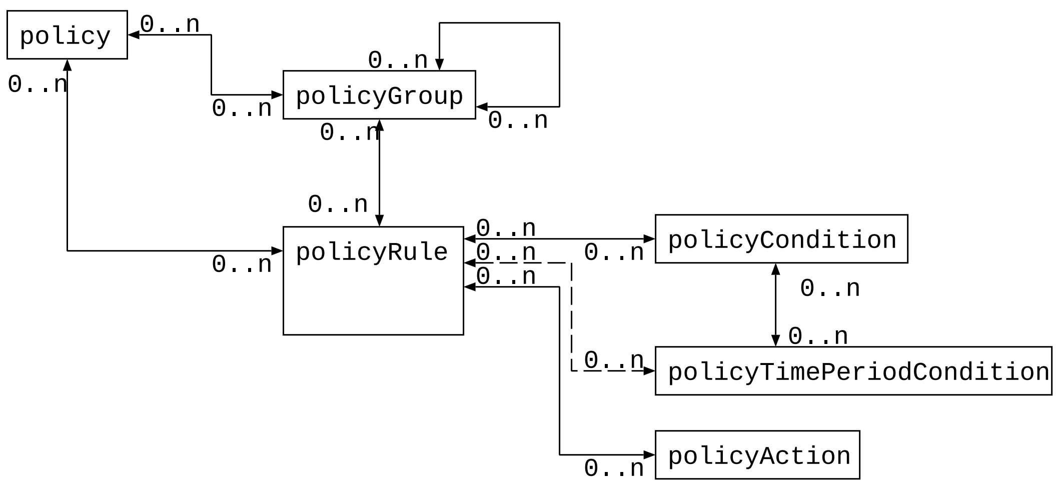

Abbildung 2.5: policy-general-schema.eps

Die Klasse Policy enthält vier Attribute: commonName (cn), caption, description und policyKeywords. (siehe Abb 2.6) policy

cn ist ein benutzerlesbarer Name. Dabei handelt es sich lediglich um eine menschenlesbare Bezeichnung, welche nicht eindeutig sein muß. Sie ist typischer Weise lediglich in einem begrenzten Bereich eindeutig.

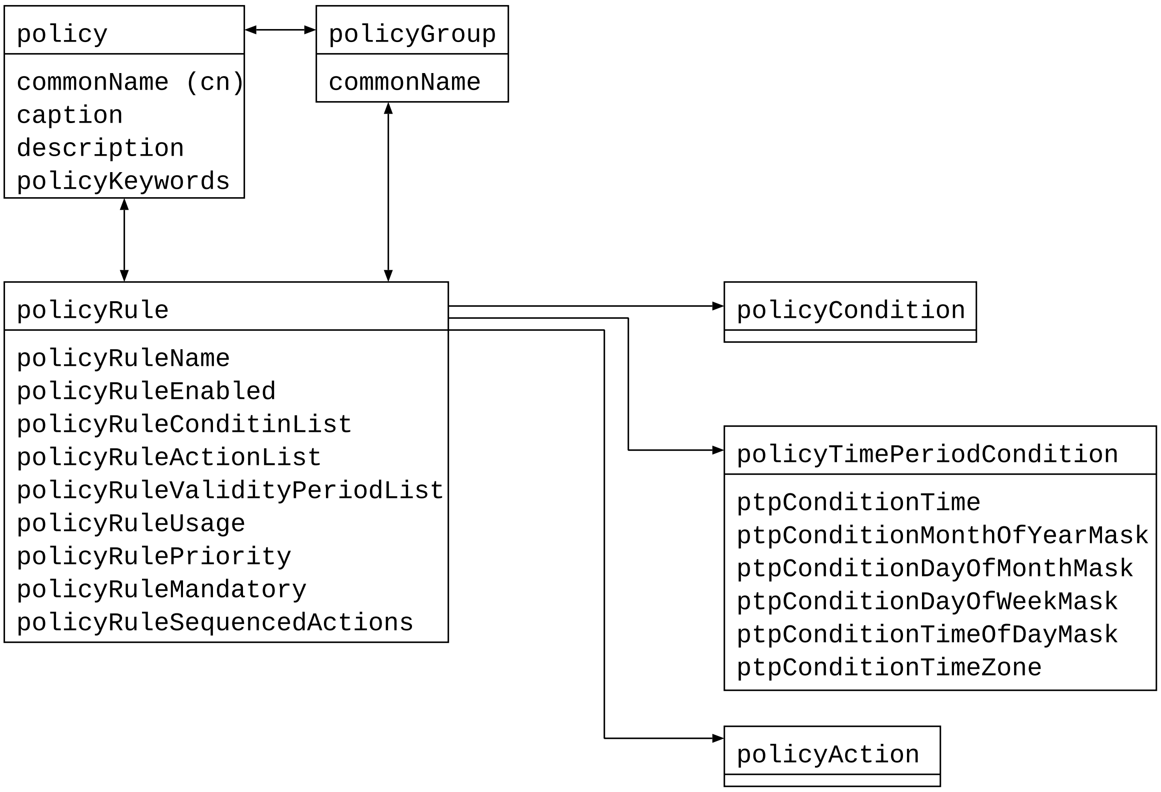

Abbildung 2.6: Policy-Klassen Modell

caption wird verwendet, um eine einzeilige Beschreibung des entsprechenden Objektes vorzuhalten.

description enthält dann die ausführliche Beschreibung des Objekts.

policyKeywords enthält eine Reihe von Schlüsselworten, die vom Administrator gegeben werden, um eine einfachere und genauere Zuweisung des Objekts an einen client zu unterstützen. Beispiele für solche Schlüsselworte sind: 'Engineering', 'Billing', 'Review in December 1999', 'UNKNOWN', 'CONFIGURATION', 'USAGE', 'SECURITY', 'SERVICE', 'MOTIVATIONAL', 'INSTALLATION', und 'EVENT'.

Die Klasse **Policy** hat im Original-Modell zwei direkte Unterklassen: **PolicyRule** und **PolicyGroup**. Im oben beschriebenen Management-Modell wäre es allerdings denkbar die Klasse **PolicyGroup** anstatt der Klasse **Group** zu verwenden, bzw. umgekehrt. **Group** ist eine Unterklasse der Klasse **Verwaltung**.

Die Klasse **PolicyGroup** enthält im Modell Instanzen der Klasse **PolicyGroup** oder **PolicyRules**. Sie ist als reine Container-Klasse gedacht (siehe Abb. 2.7. Dabei ist zu beachten, daß die Klasse **PolicyGroup** entweder **PolicyRules** oder **PolicyGroups** enthalten darf, aber nicht beide. policyGroup

Diese Klasse hat als Attribut einen Namen, der wie cn der Klasse **Policy** lediglich zur leichteren Benennung durch Menschen zu verwenden ist.

Die Klasse **policyRule** enthält drei Unterklassen: **PolicyCondition**, PolicyTimePeriod und PolicyAction.

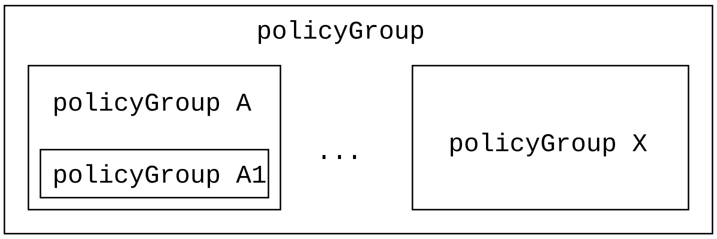

Abbildung 2.7: policy-PolicyGroup.eps

Eine policRuleCondition ist in der Regel entweder ein Set, welches mit 'OR' verbunden ist und jede Einzelregel innerhalb dieses Sets mit 'AND' verbunden ist (Disjunktiv Nornmalform (DNF)) oder ein Set, welches mit 'AND' verbunden ist und jede Einzelregel innerhalb dieses Sets mit 'OR' verbunden ist (Konjunktiv Normalform (KNF)) dargestellt.

Einzelne Regeln können zusätzlich negiert sein. Eine **policyRule** darf dann und nur dann ausgeführt werden, wenn ihre Auswertung TRUE ergibt - egal ob es sich um KNF oder DNF handelt.

Die genaue Regel, wann etwas auszuführen ist, wird in der Unterklasse **policyCondition** abgelegt. In der Unterklasse **policyRule** ist festgelegt, was im Falle einer erfüllten **policyCondition** zu tun ist.

Die Unterklasse **policyTimePeriodCondition** beinhaltet eine Angabe über den Zeitraum, an dem diese spezielle **policyRule** gültig ist. policyRule

Die Klasse **policyRule** hat folgende Attribute:

policyRuleName enthält einen benutzerfreundlichen Namen.

policyRuleEnabled zeigt an, ob dieses spezielle policyRule gerade aktiv ist oder nicht (d.h. ob es ausgewertet werden soll oder nicht.) Die Idee dahinter ist die, daß man einzelne Conditions temporär entfernen kann ohne sie wirklich aus dem Regelset herauszunehmen und später wieder einzugeben.

policyRuleConditionList ist eine Liste, in der die policyConditions, die mit diesem policyRule verbunden sind, aufgelistet sind.

policyRuleConditionList beinhaltet eine ungeordnete Liste von Pointern, die ein Set von Policy Conditions, die mit dieser Policy Regel assoziiert werden enthält.

policyRuleActionList enthält eine unsortierte Liste mit Strings der Form: 'n:DN' (DN = distinguished name; entspricht einer Instanz in einem Directory), welches eine **policyAction** beschreibt, die mit diesem **policyRule** assoziiert ist. mit 'n' wird eine Priorität der einzelnen Regeln zueinander angegeben, je niedriger der Wert von 'n' desto niedriger die Priorität der Regel. Diese Angabe ist wichtig, wenn es zu Konflikten kommt, bei denen sich die Ausführung einzelner Aktionen gegenseitig widersprechen oder gar unmöglich machen würden.

policyRuleValidityPeriodList enthält eine ungeordnete Liste mit Pointern auf eine oder mehrere **policyTimePeriodConditions**, die anzeigen, wann die **policyRule** aktiv ist und wann nicht.

policyRuleUsage ist ein 'free-form' String, der angibt, wie diese Policy zu verwenden ist.

policyRulePriority enthält eine nichtnegative Integer Zahl, die angibt wie diese **policyRule** gegenüber anderen policyRules zu priorisieren ist.

policyRuleMandatory enthält, ob eine Auswertung (und das Durchführen möglicher Aktionen) dieses **policyRule** 'mandatory' ist oder nicht.

policyRuleSequencedActions gibt dem Policy Administrator die Möglichkeit anzugeben, wie die Reihenfolge der Aktionen, die mit dieser **policyRule** assoziiert sind, zu interpretieren ist. Drei Werte sind standardmäßig unterstützt: *mandatory* (Aktionen in der angegebenen Reihenfolge durchführen oder gar keine Aktionen durchführen), *recommended* (Aktionen nach Möglichkeit in der angegebenen Reihenfolge durchführen oder, wenn dies nicht möglich ist, in einer anderen Reihenfolge abarbeiten), *dontCare* (Aktionen durchführen, egal in welcher Reihenfolge)

Der Sinn der Klasse **policyCondition** ist es, festzulegen wann das Set **policyActions** auszuführen oder nicht auszuführen ist. policyCondition

Die Klasse hat ein einzelnes Attribut **policyConditionName**, welches lediglich einen benutzerfreundlichen Namen enthält.

Die Klasse **policyTimePeriodCondition** liefert eine Möglichkeit anzugeben, in welchen Zeitabschnitten diese spezielle **policyRule** aktiv ist und somit auf das System wirken kann. Ist in einem **policyRule** keine **policyTimePeriodCondition** angegeben, wird davon ausgegangen, daß diese **policyRule** immer aktiv sein soll.

policyTimePeriodCondition ist als direkte Unterklasse der Klasse **policyCondition** definiert. Sie enthält bis zu fünf Attribute, die Zeitperioden definieren. Ist ein Attribut nicht explizit angegeben, wird auch hier wieder davon ausgegangen, dass es die Wirkungszeit nicht einschränkt, also immer gültig ist. policyTimePeriodCondition

ptpConditionTime gibt die Zeitdauer als Datumsfolge an: 'Startdatum und Zeit':'Enddatum und Zeit', das erste Datum gibt den Beginn der Zeitperiode an, das zweite das Ende (das zweite Datum muß später sein als das erste). Das Datum wird dabei wie folgt ausgedrückt: yyyymmddhhmmss. 19990901121500:19990902000000 bezeichnet also den Zeitraum vom 01. September 1999 12:15:00 Uhr bis zum 02. September 1999 00:00:00 Uhr.

ptpConditionMonthOfYearMask gibt an, zu welchen Monaten innerhalb des angegebenen Zeitraumes der ptpConditionTime die Regel gültig sein soll. Sie wird durch 12 Stellen dargestellt, die entweder 0 oder 1 sind, wobei 0 bedeutet, daß die Regeln nicht gelten und 1 bedeutet, daß das Regelset aktiv sein soll: 110011100011 entspricht zum Beispiel der Zeit im Jahr, an der an der LMU München Vorlesungen gehalten werden.

ptpConditionDayOfMonthMask gibt den genauen Tag innerhalb eines Monats an. Der String enthält 31 Felder, die mit 0 und 1 bezeichnet werden. 1110000000000000000000000000000 bezeichnet eine Regel, die nur an den ersten drei Tagen eines Monats aktiv ist.

ptpConditionDayOfWeekMask wird zum genauen Benennen des Wochentages verwendet. Begonnen wird an Montag und bis Sonntag gezählt: 0010011 entspricht einer Gültigkeit am Mittwoch und am Wochenende.

ptpConditionTimeOfDayMask ist zur genaueren Angabe von Zeitperioden im Stundenbereich. Es wird eine Start- und eine Endzeit angegeben, welche mit ':' voneinander getrennt sind: 121500:140000 entspricht einer regelmäßigen Periode von 12:15:00 Uhr bis 14:00:00 Uhr.

ptpConditionTimeZone gibt an, welche Zeitzone zugrunde gelegt werden soll, ist nichts angegeben, wird die lokale Zeit angenommen.

Die Spezifikation [ESM99c] beinhaltet noch einige zusätzliche Klassen, welche hier nicht erwähnt sind. Diese dienen der zusätzlichen Gruppierung einzelner policyAction und policyCondition-Klassen. Die daraus entstehende Komplexität ist für den hier vorgesehenen Verwendungszweck allerdings nicht nötig.

2.5 Dienst-Management

stand-management.tex

2.5.1 Allgemeines

Bisher wurden Teilaspekte der allgemeinen Beschreibungsmöglichkeiten von Protokollen, sowie Einsatzszenarien von Nomadischen Systemen besprochen. Außerdem wurden die daraus resultierenden Bereiche Sicherheit und Policy kurz behandelt. Als letztes soll der Aspekt des Managements erläutert werden.

Unter Management versteht man einen Mechanismus, der die Wartung und Installation eines größeren Netzes mit verschiedenen Services und Servern, sowie wechselnden Clients erleichtert. Bisher war es häufig so, daß einer oder eine Gruppe von Administratoren über jeden Dienst im Netz Bescheid wußten. Sie wußten, auf welchem Rechner dieser Dienst läuft und wie er konfiguriert war. Jeder einzelne Dienst wurde per Hand individuell administriert. Meist hat jeder Administrator seine Spezialgebiete. Als Folge daraus gibt es meist Probleme wenn einer der Administratoren krank ist und ein anderer seinen Aufgabenbereich übernehmen muß.

Ein Managementsystem soll diesen Vorgang kompakter gestalten. Der Administrator hat eine zentrale Stelle, an der er einen Überblick über alle im System laufenden Services und ihre Konfigurationen erhalten kann. Von hier aus kann er einzelne Dienste starten, beenden oder umkonfigurieren.

In einem weiteren Schritt wäre es sogar denkbar, daß Services direkt vom Managementsystem aus automatisch unter Berücksichtigung der allgemeinen Policy-Regeln konfiguriert und gestartet werden würden.

Damit diese Vorstellung einer Administrationszentrale umgesetzt werden kann, benötigt man einige Voraussetzungen:

- Die benötigten Services und Server müssen Kontakt mit dem Managementsystem aufnehmen können und umgekehrt.

- Die benötigten Services und Server müssen Konfigurationsanweisungen vom Managementsystem akzeptieren und umsetzen.
- Es muß eine Administrationsumgebung zum Zweck des Managements geschaffen werden, welche eine Oberfläche und einheitliche Kommunikationsprotokolle zur Kommunikation mit den externen Komponenten beinhaltet.

2.5.2 Umsetzungs-Idee

In einem komplexen Umfeld fallen naturgemäß eine große Anzahl von Einträgen und Konfigurationen an. Diese betreffen oft nicht nur einzelne Dienste, sondern das gesamte System, mit all seinen Diensten.

Damit dieses einfach zu warten ist, benötigt man eine Möglichkeit schnell und menschenlesbar größere Mengen an Regeln und sonstigen Angaben niederschreiben zu können.

Solche Regeln umfassen die Eintragung neuer Nutzer des Systems (Nomadische Systeme) genauso, wie Dienste und deren Teilkomponenten, den Proxy für den Dienst Web (HTTP) und den Server, der Emails mit dem Protokoll POP3 ausliefert.

Will man diese Idee praktisch umsetzen, werden eine Reihe von Fragen aufgeworfen.

Einem Nutzer muß mindestens eine IP-Adresse zugewiesen werden können. Soll diese statisch (also immer die gleiche Adresse) sein oder soll sie dynamisch bei jedem Anmelden im Netz aus einem größeren Adresspool zugewiesen werden?

Ein Service muß erfragen können, welche Nutzer seine Dienste nutzen dürfen. Hat ein spezieller Nutzer besondere Vorrechte?

Wer einen Service nutzt, verursacht zwangsläufig Kosten, welche abgerechnet werden müssen. Soll diese Abrechnung nach jeder Nutzung erfolgen? Was, wenn der Nutzer schon so viel von diesem Dienst genutzt hat, daß er ihn jetzt nicht mehr nutzen darf? Soll vor jeder Erbringung des Services abgerechnet werden, sobald die zu erwartenden Kosten bekannt sind oder nach der Nutzung? Wie sieht eine zu berechnende Einheit aus: Pro angefangenen Meter Papier? Pro benötigte Zeit? Pro Auftrag?

Was passiert, wenn im System ein Sicherheitsproblem auftaucht oder ein Ereignis, welches einen weiteren reibungslosen Ablauf verhindert?

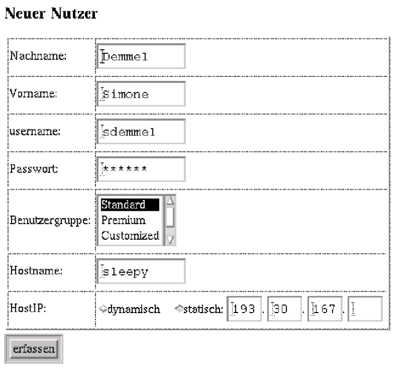

Abbildung 2.8: GUI zum Erfassen eines neuen Nutzers

Lösungen zu all diesen Fragen muß man individuell dem Managementsystem vermitteln können. Dazu benötigt man eine einfach zu handhabende Beschreibungsmöglichkeit.

Wie eine solche Eingabemöglichkeit aussehen kann, ist in Abbildung 2.8 zu sehen.

Diese Art der Eingabemöglichkeit über ein fertiges grafisches Formular, sieht sehr gut aus und ist für einen Ungeübten leicht bedienbar. Leider ist diese Form der Datenerfassung sehr behindernd, wenn eine größere Menge Nutzer auf einmal erfasst werden müssen, wie zum Beispiel die Teilnehmer eines Kongresses, die

- alle ähnliche Serviceleistungen benötigen
- alle gleiche Rahmendaten wie An- und Abreise haben
- alle auf einmal eintreffen
- in ihrer Gesamtzahl auch einige hundert Personen umfassen können

Hinderlich an dieser Art Formular ist die Tatsache, daß man nicht mit wenigen Handgriffen einen bereits konfigurierten Nutzer duplizieren kann. Häufig pendelt man zwischen Tastatur und Maus hin und her. Dabei verliert man wertvolle Zeit und die Eintragung der Daten dauert unnötig lange.

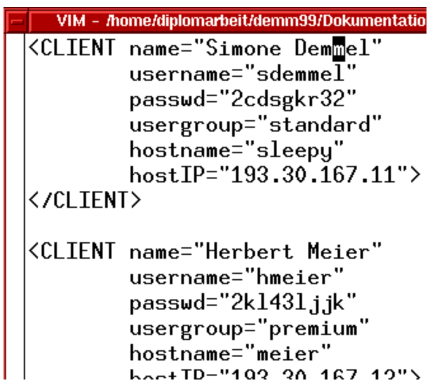

Abbildung 2.9: ASCII-File zum Erfassen eines neuen Nutzers

Abbildung 2.9 zeigt einen anderen, rein ASCII-basierten Ansatz für genau die gleiche Problemstellung. Dabei wird ein normaler Editor zum Erfassen der Daten verwendet. Vorteil dieser Vorgehensweise ist, daß man die Techniken, die ein Editor zur Verfügung stellt (cut and paste) leichter nutzen kann, als in einem GUI-basierten Interface.

Wenn man zusätzlich eine Eingabesprache wählt, die vom Konzept her bereits bekannt und einfach verständlich ist, kann auch ein Ungeübter sich in diese Technik schnell einarbeiten.

2.5.3 Beispiel

policy-dtd-bsp.tex

Wenn man Policies als allgemeine Regeln betrachtet, die beschreiben, wie etwas zu handhaben ist und welche Informationen zu einem Service oder einem Kunden vorhanden sein müssen, kann man alle Angaben, die das System konfigurieren als Policies betrachten. Trotzdem werden zusätzlich Regeln der Art: 'Wenn dieses und jenes eintritt, dann führe X aus' benötigt. Diese werden dann explizit als 'Policy' gekennzeichnet.

Die verschiedenen Anforderungen muß man in maschinenlesbarer Form vermitteln können.

Um die maschinelle Verarbeitung der Beschreibung verschiedener Regeln so effizient wie möglich zu gestalten, muß eine Form gewählt werden, die sich leicht in direkte Maschinensprache umsetzen läßt, dabei aber trotzdem noch für Menschen lesbar bleibt. Eine solche Form ist die Beschreibungssprache XML. XML ist inzwischen weit verbreitet und es gibt eine Reihe Parser, die Angaben in dieser Sprache in Objekte umsetzen können und sie somit für Anwendungen aller Art verfügbar machen. Auf Grund ihrer Ähnlichkeit mit HTML ist die Grundidee der Syntax vertraut. Dies erleichtert das Erlernen der Sprache, da es - aus Sicht des Anwenders - lediglich ein HTML mit anderen Tags ist.

Ein kleines Beispiel soll illustrieren, wie die Beschreibung eines solchen Management-Systems und seinen Klassen und Nutzern in XML aussehen könnte.

Das Beispiel könnte so oder so ähnlich im Hotelszenario benötigt werden. In der vorliegenden Arbeit wurde aus Platzgründen lediglich ein kleiner Auszug abgedruckt.

Dieses Hotel bietet seinen Kunden verschiedene Services, bzw. Geräte zur Nutzung an:

- Laserdrucker schwarz/weiss, mit Freieinheiten (10 Seiten pro Gast)
- Farblaserdrucker für Folien
- Fax (Computerfax)
- Mail-Service
- News-Service
- Internetzugang
- Jeder Kunde, bis auf eine Ausnahme, bekommt seine IP-Adressen dynamisch per DHCP zugewiesen

Weiterhin existieren zwei Service-Gruppen die die folgenden Services beinhalten:

Multimedia Mail, News, Internet

Standard Laser schwarz/weiss, Fax

Es seien drei Gäste konfiguriert, von denen einer die Service-Gruppe Standard und zwei die Service-Gruppe Multimedia gebucht haben.

Der Gast, der Standard gebucht hat, bekommt außerdem eine feste IP-Adresse zugewiesen.

Die XML

Im Folgenden werden Teile der diesem Beispiel entsprechenden XML-Eintragungen vorgestellt und beschrieben.

Die Grundidee ist, daß man primär Kunden und Services benennen muß. Diese werden dann um spezielle Policy-Regeln erweitert oder in Gruppen zusammengefaßt.

Beispielkonfiguration für Services:

```
<service name="drucker-papier"
system="testhotel"
type="printer"
protocol="lp"
area="network"
server="lpr1"
domain="hotel.de"
alt="lokaler Printserver fuer
Papierdrucke">

<INFO>Postscript</INFO>
<INFO>3. Stock</INFO>
<INFO>Zimmer 310</INFO>
</service>
```

Dies ist der einfache Papierdrucker. Er steht im Hotel, welches mit `"testhotel"` bezeichnet wurde, gehört in die Kategorie `"printer"` und ist über das lineprotocol (lp) ansprechbar. Der Drucker ist ein Netzwerkdrucker, dessen Name `lpr1` ist mit der Domain `hotel.de`. Es handelt sich um einen lokalen Druckservice für Papierdrucke. Als Zusatzinformationen zu diesem Service werden angeboten:

- es handelt sich um einen Postscript-Drucker
- er steht im 3. Stock
- er steht in Zimmer 310

```
<service name="fax"
system="testhotel"
type="fax"
area="network"
server="fax"
domain="hotel.de"
alt="Allgemeiner Fax-Service">

</service>
```

Das Fax ist ebenfalls ein netzweiter Dienst. Er ist abfragbar über den Server fax.hotel.de. Eine Nummer, unter der man diesem Fax eine Nachricht senden kann, könnte wahlweise als `<INFO>` oder im Bereich für die Option gw, incomingip oder outgoingip angegeben werden. Ein `<INFO>`-Eintrag wäre die sinnvollste Möglichkeit, da es denkbar ist, daß mehrere Telefonnummern auf diesen Faxserver umgeleitet werden, aber nur ein Eintrag in den alternativ verwendbaren Attributen vorgesehen ist.

```
<service name="mail"
system="testhotel"
type="email"
protocol="POP3"
area="network"
server="mail"
domain="hotel.de"
alt="lokaler Mailserver">
</service>
```

Der Email-Service wird analog den anderen Services definiert.

Bei der Beschreibung der Services wurde das allgemeine Klassenmodell etwas vereinfacht. In der Service-Beschreibung wird direkt die Server-Beschreibung integriert. Diese Maßnahme vereinfacht das Erstellen des Konfigurationsfiles und erleichtert somit die Lesbarkeit.

Eine Zuordnung Service zu Server ist weiterhin möglich. Allerdings nur indirekt über die Auswertung des Attributes type.

Die folgende Tabelle kommentiert Beispiele für Service-Gruppen. Zu Gruppen kann man nur Services zusammenfassen, welche definiert wurden.

```
<group name="standard">
<insert name="drucker-papier"
type="service">
<insert name="fax" type="service">
</group>
```

Dies ist die Service-Gruppe Standard, ihr werden die bereits definierten Services Fax und Papierdruck zugeordnet. Ein Nutzer, dem diese Gruppe zugeordnet wird, kann die hier angegebenen Services nutzen.

```
<group name="multimedia">
<insert name="mail" type="service">
<insert name="news" type="service">
<insert name="internet"
type="service">
</group>
```

Der Gruppe Multimedia werden die Services mail, news und internet zugeordnet.

In <INSERT> muß in diesem Fall type="service" angegeben werden. Dies zeigt an, welcher Art diese Gruppe ist. Ebenfalls zulässige Gruppierungen können <CLIENT> betreffen, sowie in sich wieder <GROUP> und <POLICY>.

An dieser Stelle könnte man zusätzliche Policy-Definitionen den einzelnen Services zuweisen. Eine solche Policy-Anweisung könnte beispielsweise wie folgt aussehen:

```
<POLICY name="nacht"
alt="testpolicy für Nachtbetrieb"
descr="Verbot zur Nutzung des
Druckservices zwischen
22:30 Uhr und 6:00 Uhr"
keywords="USAGE">

<PR name="nachtruhe"
ena="TRUE">

<PC value="between 22:30 and 6:00">

<PA value="deny drucker-papier">
</PR>
</POLICY>
```

Dabei ist die Syntax der Attribute value in <PC> und <PA> frei erfunden. Eine Verwendung von <TIME> könnte die Gültigkeit dieser Regel weiter einschränken, z.B. ausschließlich auf Wochenenden.

Die letzte Tabelle zeigt Beispiele für Kundenkonfigurationen. Dabei wird ein Kunde als Client aufgefaßt, der keine eigenen Services anbietet.

```
<client username="u1"
passwd="02oawyZdjhhpg"
hostname="kunde1"
hostIP="dynamic"
group=""
alt="User Eins vom Testhotel">

<time
time="19990101000000:19991231235959"
moym="110011100011"
dowm="1111100"
todm="090000:180000"
alt="">

<qos sname="#drucker-papier"
free="10">

<insert name="#multimedia"
type="group">

</client>
```

Ein Kunde benötigt allgemein seine Zugangsinformationen wie Username und Passwort. Weiterhin kann er einen festen Hostnamen zugewiesen bekommen. Die Angabe in hostIP gibt an, ob er seine IP-Adresse dynamisch zugewiesen bekommt. Ist dies nicht der Fall steht die zu vergebende Adresse an dieser Stelle. Die Gültigkeitsdauer dieser Nutzer-Definition wird mit time angegeben. Hat der Nutzer spezielle Service-Vereinbarungen, wird dies in `<QOS>` vermittelt. Erlaubte Services können entweder direkt als

```
<INSERT name="#Servicename"
type="service">
```

angegeben werden oder als Service-Gruppe mit:

```
<INSERT name="#Service-Gruppe"
type="group">
```

stand-management.tex Mitte

```
<client username="u3"
hostname="kunde3"
hostIP="192.168.1.100"
group="Konferenz"
alt="User drei vom Testhotel">

<time time="19991101:20000229"
moym="110000000011"
alt="">

<insert name="#standard"
type="group">

</client>
```

Nebenstehend das Beispiel für einen Nutzer, der eine feste IP-Adresse erhält.

Auf diese Art und Weise kann man alle wichtigen Informationen für das Managementsystem codieren. Der Vorteil besteht darin, daß diese Form der Codierung menschen- und maschinenlesbar ist. Der Administrator ist nicht auf grafische Anzeige- und Erfassungs-Programme angewiesen und die Sprache ist verhältnismäßig einfach zu erlernen.

Die Korrektheit der Eintragungen kann man von Programmen - sogenannten Validators - automatisch prüfen lassen.

Soll diese Technik in einem nicht-technischen Umfeld eingesetzt werden, kann man zusätzlich ein grafisches Interface mit vorgefertigten Eingabefeldern und automatischer Prüfung auf Korrektheit der Eingaben anbieten. Dies würde dann auch sicherstellen, daß die Eingaben korrekt sind und eine zusätzliche Überprüfung könnte entfallen.

Kapitel 3

Protokolle mit Bezug zur Dienstvermittlung

protokolle.tex

Kapitelüberblick:

- Das vorliegende Kapitel faßt allgemeine Anforderungen an die verwendeten Protokolle zusammen. Danach werden Protokolle und Plattformen zur Dienstvermittlung, die bereits in anderen Projekten entwickelt wurden, vorgestellt.
- Die vorgestellten Protokolle und Plattformen werden möglichst tiefgehend beschrieben.
- In diesem Kapitel soll vermittelt werden, welche Fähigkeiten bereits vorhandene Protokolle und Plattformen zur Dienstvermittlung haben. Diese Anforderungen werden die im weiteren Verlauf der Arbeit entstehenden Eigenentwicklungen erfüllen müssen.

3.1 Allgemeines

protokolle-allgemeines.tex

Client Als Client wird im Folgenden das zu konfigurierende oder zu überwachende Nomadische System bezeichnet.

Server Als Server werden Dienstanbietende Rechner bezeichnet. Es handelt sich dabei in vielen Fällen um sogenannte Agenten, die ein spezielles Service-Angebot (z.B. Mail, DHCP) überwachen und managen.

Service - Dienst Ein Dienst ist die deutsche Übersetzung von Service. Beide Begriffe werden synonym verwendet.

Netzwerk Unter Netzwerk oder kurz 'Netz' wird in dieser Arbeit ein IP-Netz verstanden.

IP-Netz Ein IP-Netz ist ein Computernetz, auf welchem Daten mit dem IP-Protokoll übertragen werden; RFC 781 [Su81] (sowie RFC 815 [Cla82], RFC 1356 [MRU92], RFC 894 [Hor84] und RFC 895 [Pos84]).

3.2 Anforderungen an die Infrastruktur

protokolooe-anforderungen-umfeld.tex

Ein funktionierendes Management-System hängt auch nicht zuletzt von der richtigen Auswahl und Konfiguration der verwendeten Hardware ab. Moderne Hardware übernimmt oft einen großen Teil von Filterfunktionen und Zwischenberechnungen, sowie transparente Umsetzungen von einem Protokoll in ein anderes. Dies kann bei (für die Hardware-Komponente) unbekannten Protokollen oder bei falscher Konfiguration zum Nichtfunktionieren des gesamten Netzes oder nur einiger Teile führen. Solche Hardware-Komponenten sind zum Beispiel Router.

Router können bei der Verwendung von Tunneling-Mechanismen sinnvoll sein. Auch Firewalls kommen gelegentlich als separate Zusatzhardware (black box) und können durch ihre eigenen restriktiven Regeln und Filtermechanismen die Kommunikation innerhalb eines Netzwerks blockieren.

Um später eventuelle Umstrukturierungen problemlos durchführen zu können, ist es hilfreich, wenn die einzelnen Netzsegmente physikalisch voneinander abtrennbar sind. Das bedeutet, daß das Office-Netz vom Kundennetz physikalisch trennbar sein muß um es gegebenenfalls später unabhängig von anderen Netzsegmenten betreiben zu können. Dies läßt sich zum Beispiel durch die Verwendung von Routern oder 'Managed Switches' realisieren. Dort kann jedes einzelne Segment physikalisch angeschlossen und wieder abgetrennt werden. Auch die Kommunikation der Netze untereinander wird direkt in das richtige Netzsegment vermittelt.

All diese Überlegungen führen zu der folgenden Liste von Anforderungen, die man allgemein in einem zu managenden Netz erwartet und deren Realisierung auch Komponenten in niederen Schichten (OSI-Schicht 1,2 und 3) treffen können. Man kann ein und das selbe Ziel nicht immer nur auf ein und dem selben Weg erreichen. Oft gibt es verschiedene Wege um eine Anforderung umzusetzen. Man sollte sich aber immer bemühen, den einfachsten Weg zu wählen.

Verschlüsselung Es muß möglich sein, Informationen oder eventuell komplette Datenpakete verschlüsselt zu übertragen. Für diesesn Zweck gibt es als neues Transportprotokoll IPSec. Damit ist eine verschlüsselte Übertragung transparent möglich. Dieses Protokoll darf von den Routern und Switches (und allgemein Firewalls) des Netzes nicht herausgefiltert werden, da in diesem Fall eine Datenübertragung nicht möglich wäre.

Erreichbarkeit aller Netzteilnehmer Dieses Kriterium zielt auf Routing und Filter innerhalb des lokalen Netzes ab. Hauptaugenmerk liegt in diesem Fall auf verwendete Pakettypen, die zwecks Entlastung eines Netzes in der Praxis oft gefiltert werden. Allerdings werden sie bei der automatischen Vermittlung von Diensten als grundlegendes Transportprotokoll verwendet. Besonders betroffen ist von dieser Regelung 'Broadcast', welches vom DHCP zur Vermittlung einer IP-Adresse verwendet wird. Das DHCP-Protokoll sieht auch keinerlei alternative Kommunikationswege vor.

Tunneling Um einen entfernten Rechner so aussehen zu lassen, als würde er zu einem anderen, lokalen Netz gehören, verwendet man Tunneling.

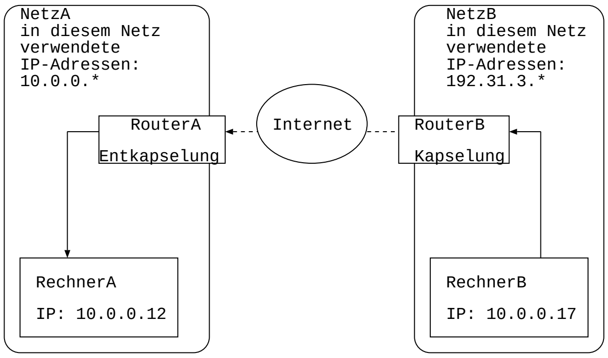

Abbildung 3.1: Tunneling zwischen zwei Netzen

Dies ist eine Möglichkeit, bei der zwischen dem entfernten Rechner (oder dem nächsten Router zu diesem Rechner) ein sogenannter Tunnel zum Router im anderen Netz generiert wird. In diesem Tunnel werden die Pakete des sendenden Rechners in Pakete eines anderen Protokolls gekapselt und über das Internet übertragen. Der Router am anderen Ende des Tunnels entfernt diese Kapselung und leitet das Originalpaket weiter (siehe Abb. 3.1).

3.3 Anforderungen an die Protokolle

protokolle-anforderungen-protokolle.tex

Nicht alle Bedingungen, die ein Managementsystem erfüllen soll, kann man immer auf niederen Ebenen durch einfache Manipulation der physikalischen Datentransportwege und Aufsetzen passender Filter erreichen. Manchmal ist dies nicht möglich oder die Anforderungen sind zu kompliziert. Für solche Fälle muß man in höheren Schichten (meist Schicht 7) passende Maßnahmen ergreifen. Diese dürfen die Anforderungen an die Infrastruktur ergänzen, oder auch ersetzen. Ist zum Beispiel kein Tunneling über eine Blackbox-Lösung möglich, so müssen die dazugehörigen Voraussetzungen künstlich geschaffen werden und eine andere (Software)Komponente diese Aufgabe übernehmen.

Als Softwarekomponente kommt im Fall von Dienstvermittlung nur die Implementierung der Protokolle, bzw. Plattformen für diesen Zweck in Frage. Man kann also eine Reihe Anforderungen stellen, welche man an ein Managementsystem im Gesamten stellt, die aber von Softwarekomponenten realisiert werden können oder beachtet werden müssen.

Verschlüsselung Einzelne Komponenten des Managementsystems müssen sicher und 'ungestört' miteinander kommunizieren können. Um dies gewährleisten zu können, wird die managementsysteminterne Kommunikation verschlüsselt und signiert. Dadurch kann man gewährleisten, daß die Daten 'unterwegs' nicht verändert wurden und daß sie von keiner gefälschten Komponente gesendet wurden. Es kann zum Beispiel der Teil des IP-Paketes, der konkrete Nutzdaten (außer Absender- und Zieladresse) enthält verschlüsselt werden. Dieses Problem kann man auf Ebene der Hardware und Infrastruktur lösen, oder aber auch, in dem das Programm selbst bereits verschlüsselte Informationen sendet. Mögliche Zusatzprotokolle sind: *IPSec* und *SSL (Secure Socket Layer)*.

Proxyfähigkeit Ein Proxy ist eine Art Strohmann, der Anfragen eines Clients an den Server weiterleitet, die Antwort vom Server empfängt und sie an den Client zurücksendet. Solche Mechanismen sind nötig, wenn man Protokolle in andere Protokolle umsetzen muß, weil dieser Typ Protokoll zum Beispiel nicht weiter unterstützt wird, man aber die Clientsoftware deswegen nicht verändern kann oder will. Proxies findet man, wenn es darum geht, Daten zu cachen. Häufige Anwendungsgebiete sind Web (Proxy/Cache) und DHCP (Proxy). Allgemein sollten für jeden Service des Managementsystems Proxy-Mechanismen vorgesehen sein. Oft ist das Netzwerk nicht entsprechend anpassbar, so daß man für viele Dienste auf Proxies angewiesen ist (zum Beispiel beim Einsatz einer Firewall).

Bei der Verwendung von Proxies wird ein Teil der Kommunikation zwischen Client und Server leicht modifiziert.

Bei Web-Proxies ist es die Client-Seite, auf der sich das Protokoll leicht verändert. Der Client spricht den Proxy gezielt an.

Bei DHCP ist es die Strecke zwischen DHCP-Relay (so nennt man den DHCP-Proxy) und Server, die entsprechend abgewandelt wird. Für den Client ist die Verwendung des Relays transparent.

An- und Abmeldung bei einem Directory Service Manche Ansätze für Dienstvermittlung in einem Computernetz gehen davon aus, daß ein Service, der seine Dienste im Netz zur Verfügung stellt und an Clients vermittelt werden soll, sich aktiv bei einem zentralen Vermittlungs-Service anmelden muß. Nur dann wird er auch an anfragende Clients vermittelt. Diese Information kann man beispielsweise auch nutzen um Ausfälle von Services rechtzeitig zu registrieren und automatisch eingreifen zu können. Jeder Service sollte sich außerdem explizit abmelden können um anzuzeigen, daß es sich um einen beabsichtigten 'Dienstausfall' handelt.

Policy-Fähigkeit Ausnahmen bestätigen die Regel. Deshalb ist es möglich jeder Instanz in einem Managementsystem eigene, individuelle Policy-Regeln zu geben. Diese Regeln können unter Umständen globalere Regeln verletzen. Allerdings möchte man nicht immer und überall mit allen eventuell benötigten Regeln arbeiten, sondern das gerade aktuelle Set an Policies dynamisch erweitern können. Deshalb ist es von Vorteil, wenn der sich anmeldende Service eigene Policy-Regeln mit einbringen darf. Diese könnten der Art sein, daß zum Beispiel das allgemeine Nacht-Nutzungsverbot bei diesem sich anmeldenden Drucker-Service nicht gilt, da der hier verwendete Drucker von einer neuen, geräuschlosen Drucker-Generation ist.

Kommunikation mit einem Abrechnungs-Service Nicht nur im Kommerziellen Umfeld besteht die Forderung, daß genutzte Dienstleistungen abgerechnet werden können müssen. Dazu muß vom Service, der die Leistung erbringt, mitgeteilt werden, wer gerade wieviel Service-Gut verbraucht hat. Es gibt zwei grundlegende Strategien: entweder der Service selbst meldet einem externen Dienst seine abzurechnenden Beträge oder der externe Dienst befragt in regelmäßigen Abständen den Service. Beide Ansätze sind bereits verbreitet: Um Accountingdaten bei Internet-Providern zu erheben, werden üblicher Weise in regelmäßigen Abständen die Logfiles der Außenrouter abgerufen und ausgewertet. Beim Telephonieren erzeugt die Vermittlungsstelle sogenannte *CDR*s (Call Data Record). Diese werden dann an einen Accountingserver geschickt (siehe dazu auch [Hei99]).

Flexible Reaktion auf neue Protokolle Das zugrunde liegende Protokoll bzw. die zugrunde liegende Managementplattform darf nicht auf einige bekannte Protokolle beschränkt sein, vielmehr muß es möglich sein, neue Protokolle zu integrieren. Daher ist die Entwicklung eines modularen Ansatzes nötig, der es erlaubt neue Protokolle beliebig zu integrieren. Dies ist möglich, indem man eine zusätzliche Einheit baut, die 'nach außen hin das fremde Protokoll 'spricht' und nach innen, zum Management-System, das interne Protokoll (siehe Abb. 3.2)

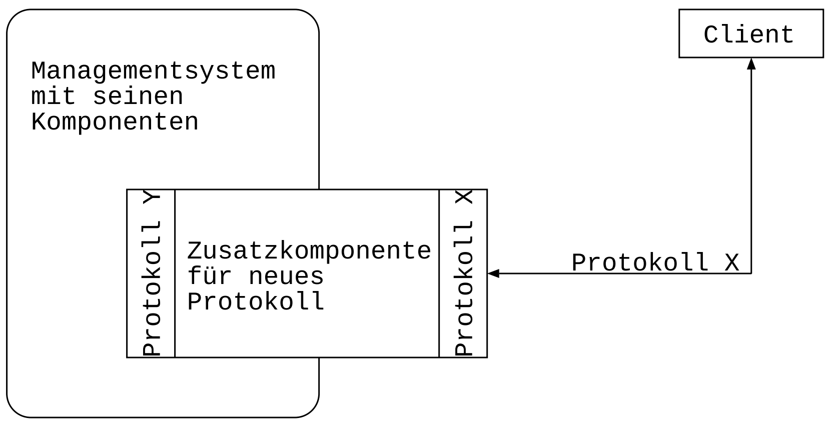

Abbildung 3.2: modulare Komponente im Managementsystem

3.4 Zur Verfügung stehende Protokolle

Im Folgenden werden Protokolle und bereits entwickelte Plattformen, die Dienstvermittlung in der ein oder anderen Form unterstützen, vorgestellt. Als Protokolle sind hier DHCP und SLP vorhanden. Beide sind bis in das Aussehen einzelner IP-Pakete und die darin befindlichen Bits und deren Inhalt in den dazugehörigen RFCs beschrieben. Jini und der Corba Trading Service kann man bereits als Plattformen bezeichnen. Ihre Beschreibung ist ist auf Objektebene (OSI-Schicht 7) angesetzt, während die der Protokolle auf Transportprotokollebene beschrieben liegt. Somit sind Trading Service und Jini insofern protokollunabhängig, daß sie eine Klassenbeschreibung beinhalten, die nicht vom Transportprotokoll abhängt. Ihre bereits vorhandenen Umsetzungen sind auf Basis von CORBA (Trading Services) und somit als Teil eines umfangreichen Service-Paketes, welches auf Basis von CORBA kommuniziert, sowie auf SUNs Entwickeltes Hochsprache Java und der dort implementierten API als Schnittstelle zum Netzwerk (OSI-Schicht 3) realisiert.

3.4.1 DHCP

DHCP.tex

Allgemeines

Client Als Client wird im folgenden das zu konfigurierende Nomadische System bezeichnet.

Server Als Server werden Systeme bezeichnet, die einem Nomadischen System die benötigten Daten mit dem richtigen Protokoll zur Verfügung stellen können. Es ist möglich, daß mehrere voneinander unabhängige Server gleichzeitig zur Verfügung stehen.

Bei der Protokollbeschreibung wird das Zusammenspiel zwischen einem Client und mehreren Servern betrachtet.

Das DHCP-Protokoll ist in den RFCs 1497 [Ste93], 1533 [AD93], 1534 [Dro93], **2131 [Dro97]**, 2132 [AD97] und 2241 [Pro97] (DHCP-Erweiterungen für Novell-Netzwerke) beschrieben.

Designziel

DHCP (Dynamic Host Configuration Protocol) wurde entwickelt, um Rechnern in lokalen IP-Netzen automatisch/dynamisch IP-Adressen zuzuweisen. Zielsetzung bei der Protokoll-Entwicklung ist es, einen Mechanismus zu schaffen, mit dem ein Client-Rechner ohne manuelle Hilfe eines Administrators für das lokale Netzwerk konfiguriert werden kann. Besonders wichtig ist in diesem Punkt die Zuordnung einer freien IP-Adresse. Als Ergänzung können weitere Informationen wie Gateway, Druckerserver etc. übertragen werden.

Der zu konfigurierende Client schickt ein Broadcast-Paket. Alle DHCP-Server im Netz, die dieses Paket empfangen und diesem Client Zugangsdaten geben dürfen, senden ein Angebot zurück, in dem enthalten ist, welche Daten (in der Regel ist die IP-Adresse wichtig) der Client erhalten würde.

Von allen Angeboten, die den Client innerhalb einer angemessenen Zeitspanne erreichen, wählt der Client eines aus. Er erteilt dem ausgewählten Server eine direkte Anfrage und bekommt das Angebot als konkrete Daten zurück.

Diese Daten umfassen neben der IP-Adresse und eventuellen weiteren Konfigurationsinformationen auch eine *Leased Time*.

Leased Time

Dies ist eine festgelegte Zeitspanne, innerhalb der der Client dem Server eine Nachricht zukommen lassen muß, um die Gültigkeit seiner exklusiv genutzten Ressourcen (IP-Adresse) zu verlängern.

Bleibt diese Rückmeldung des Clients aus, nimmt der Server an, daß der Client sich nicht mehr im Netz befindet oder die Daten nicht mehr benötigt. Alle vom Client exklusiv genutzten Informationen werden intern im DHCP-Server freigegeben und können an den nächsten anfragenden Client vergeben werden.

Es ist erlaubt, daß ein Client seine verwendeten Daten selbst explizit wieder frei gibt (Release).

Dazu sendet der Client eine entsprechende Nachricht an den Server. Der Server verfährt dann genauso, als würde es sich um eine Überschreitung der Leased Time handeln.

Das DHCP-Protokoll ist weit verbreitet, besonders in großen Firmen und im Schulungsbereich.

DHCP-modelle.tex

Organisationsmodell

DHCP-OM-new.tex

Im DHCP-Protokoll werden einem Client von einem DHCP-Server Daten übermittelt.

Wird ein neuer Rechner im Netz aktiviert, so bezeichnet man ihn aus DHCP-Sicht als Client. Dieser Client benötigt in erster Linie eine Adresse, sowie möglicher Weise eine Reihe weiterer Daten.

Diese Informationen kann er von einer zentralen Stelle, dem DHCP-Server, zugewiesen bekommen. Dabei ist es vorgesehen, daß in einem Netzwerk mehrere DHCP-Server aktiv sind.

Das Protokoll bietet Mechanismen, die es dem Client ermöglichen, diese Server anzusprechen und einen auszuwählen. Der ausgewählte Server wird dann die von diesem Client exklusiv genutzten Daten (z.B. IP-Adresse) zuweisen und verwalten.

Informationsmodell

DHCP-IM-new.tex

Das DHCP-Protokoll hat zwei Aufgaben.

1. Vergabe von IP-Adressen.
2. Übermittlung allgemeiner Netzkonfigurationsdaten.

Vergabe von IP-Adressen Hauptziel des Protokolls ist es, einem neuen System im Netzwerk automatisch eine IP-Adresse zuzuweisen. Da in einem Netzwerk in der Regel nicht nur ein Rechner seine Adresse per DHCP zugewiesen bekommen soll, sondern meist auch alle anderen Rechner auch, fällt als Nebenaufgabe noch die interne Verwaltung dieser Adressen an.

Das Bedeutet: Jeder DHCP-Server hat eine gewisse Menge IP-Adressen, die er vergeben kann. Dabei muß er darauf achten, daß

- er jede IP-Adresse nur einmal gleichzeitig vergibt
- wenn ein Client seine IP-Adresse nicht verlängert, er sie automatisch freigibt
- wenn ein Client seine IP-Adresse wieder zurückgibt, er sie als wieder zu vergeben markiert

Übermittlung allgemeiner Netzkonfigurationsdaten Manche Systeme in einem Netzwerk benötigen nicht nur eine IP-Adresse, sondern allgemeine Konfigurationsdaten des Netzes, wie zum Beispiel Druckserver, DNS-Server, Netzmasken oder Boot-Files.

Diese Informationen sollen ebenfalls auf Anfrage an die Clients weitergegeben werden können.

In beide Fällen ist es wichtig, daß der Client auch individuell Informationen zugewiesen bekommt. Dazu muß der Server den Client erkennen können (zum Beispiel an der eindeutigen Adresse seiner Netzwerkkarte) und ihm so spezielle Daten zukommen lassen (z.B. eine spezielle IP-Adresse, die im Nameserver auf den Namen des Clients eingetragen ist).

Funktionsmodell

DHCP-FM-neu.tex

Das DHCP-Protokoll sieht neben der Vermittlung einer IP-Adresse noch die Übertragung weiterer Informationen vor, wie zum Beispiel die Angabe von Nameserver, Gateway, Mailserver, Druckserver. Dies ist unabhängig von der Vergabe einer IP-Adresse möglich.

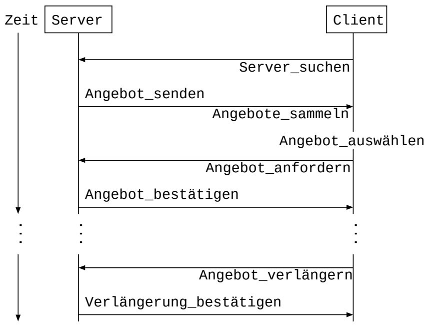

Abbildung 3.3: DHCP: Funktionen

Um diese Aufgabe durchführen zu können, müssen einige Funktionen implementiert werden. Abbildung 3.3 stellt die benötigten Funktionen der IP-Adressvergabe dar. Diese wurden des einfacheren Verständnisses halber schon dem im Organisationsmodell beschriebenen Client und Server zugeordnet. Außerdem wurde die Darstellung insofern vereinfacht, daß es genau einen Server darstellt. Dieser eine dargestellte Server steht symbolisch für eventuell mehrere Server.

Im Folgenden werden unter 'Daten' zugewiesene Daten verstanden, die vom Server exklusiv genutzte Informationen enthalten können, wie zum Beispiel eine IP-Adresse.

SERVER_SUCHEN Der Client hat zu diesem Zeitpunkt keinerlei Informationen ob und wenn ja wie viele Server im Netz vorhanden sind. Deshalb muß er einen Server suchen. Er schickt dazu eine Nachricht in das Netz und wartet, ob er Antworten bekommt. Diese Nachricht ist so beschaffen, daß sie alle im Netz aktiven DHCP-Server bekommen und darauf reagieren können.

ANGEBOT_SENDEN Empfängt ein DHCP-Server eine Such-Nachricht, wie sie beim SERVER_SUCHEN versendet wird, muß er dem Client ein 'Angebot' senden. Dieses Angebot enthält die Daten, die der Server dem Client zusenden würde, wenn der Client eine konkrete Anforderung an diesen Server stellen würde. Zusätzlich ist das Angebot abhängig von den Regeln, die der DHCP-Server zu diesem Client kennt. So ist es möglich, daß der Server diesem Client ein anderes Angebot machen würde, als einem anderen Client.

ANGEBOTE_SAMMELN Der Client sammelt über einen gewissen Zeitraum alle Angebote von DHCP-Servern, die als Antwort auf seine Anfrage gesendet wurden.

ANGEBOT_AUSWÄHLEN Der Client wählt ein einzelnes Angebot eines Servers aus den erhaltenen Angeboten aus.

ANGEBOT_ANFORDERN Der Client sendet eine entsprechende Nachricht an den Server, dessen Angebot er ausgewählt hat. Damit fordert er die vom Server angebotenen Daten explizit an.

ANGEBOT_BESTÄTIGEN Der Server bestätigt sein Angebot und weist somit dem Client eine IP-Adresse zu. Diese darf er so lange keinem anderen Client zuweisen, bis sie wieder frei wird. Einmal vergebene Daten können wieder 'frei' werden, indem der Client eine Nachricht schickt, daß er sie zurück gibt (ANGEBOT_FREIGEBEN) oder wenn die Gültigkeit der Daten abgelaufen ist.

Angebot_ablehnen Der Server darf das Angebot wieder 'zurückziehen'. Dies kann zum Beispiel passieren, wenn inzwischen so viel Zeit vergangen ist, daß die angebotenen Daten anderweitig vergeben wurden.

Angebot_verlängern Der Client muß seine erhaltenen Daten nach einer gewissen Zeit (Leased Time) 'verlängern'. Geschieht dies nicht, werden die Daten frei und können wieder anderweitig vergeben werden.

Verlängerung_bestätigen Die Verlängerung muß vom Server bestätigt werden.

Angebot_freigeben Der Client sendet eine Nachricht an den Server, die besagt, daß er die von diesem Server vergebenen, von ihm exklusiv genutzten Daten (z.B. IP-Adresse) nicht mehr benötigt und somit zur Wiedervergabe frei gibt.

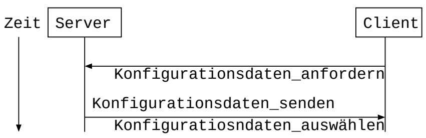

Abbildung 3.4: DHCP: Vermittlung von Konfigurationsdaten

Abbildung 3.4 stellt eine weitere Funktion des Protokolls das: die Vermittlung von Konfigurationsdaten. Diese Vermittlung kann unabhängig von der Zuweisung einer IP-Adresse ausgehandelt werden.

Konfigurationsdaten_anfordern Der Client kann einen Server suchen, der ihm allgemeine Konfigurationsdaten des Netzes sendet. Er schickt dazu eine Nachricht in das Netz und wartet, ob er Antworten bekommt. Diese Nachricht ist so beschaffen, daß sie alle im Netz aktiven DHCP-Server bekommen und darauf reagieren können.

Konfigurationsdaten_senden Alle Server, die vom Client die Anforderung einer Konfiguration bekommen haben, senden die ihnen bekannten Konfigurationsparameter zurück. Der Server kann zusätzlich zum Client individuelle Konfigurationsinformationen senden.

Konfiguration_auswählen Der Client wählt aus den erhaltenen Konfigurationsparametern diejenigen aus, die er benötigt.

Abbildung 3.5 stellt die verschiedenen Aktionen und Stati des Clients dar. Die Darstellung ist nach einer Skizze in RFC 2131 angefertigt ([Dro97]). Dabei sind folgende Änderungen zu beachten:

in der Abbildung verwendete Funktionsbezeichnung	in der Arbeit verwendete Funktionsbezeichnung
DHCPDISCOVER	Sender_suchen
DHCPOFFER	Angebot_senden
DHCPREQUEST	Angebot_anfordern, Angebot_verlängern
DHCPACK	Angebot_bestätigen, Verlängerung_bestätigen, Konfigurationsdaten_senden
DHCPNACK	Angebot_ablehnen, Verlängerung_ablehnen
DHCPRELEASE	Angebot_freigeben
DHCPINFORM	Konfigurationsdaten_anfordern

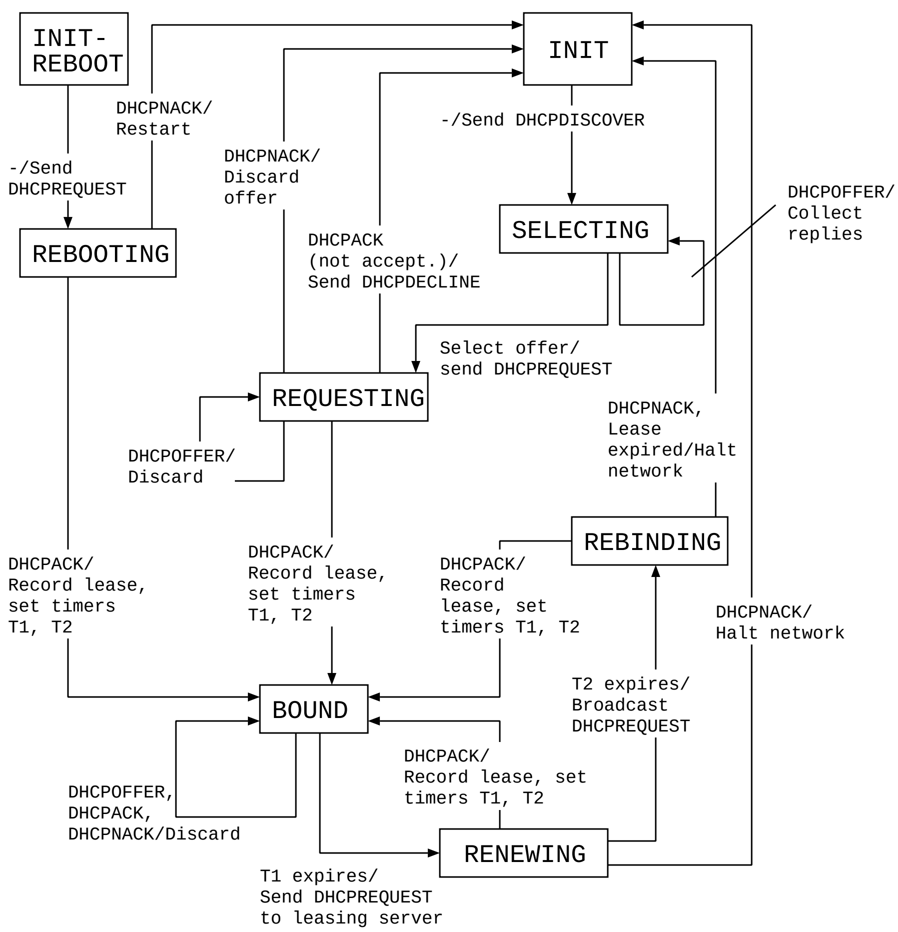

Abbildung 3.5: DHCP: State-Diagramm DHCP-Client nach [Dro97]

Kommunikationsmodell

DHCP-KM-neu.tex

Wenn ein Client neu in ein Netz hineinkommt und den DHCP-Service nutzen soll um sich eine IP-Adresse zuweisen zu lassen, hat er in der Regel noch keine IP-Adresse. Eine solche Adresse ist aber normaler Weise nötig, um in einem IP-Netzwerk zu kommunizieren.

Aus diesem Grund muß eine Übertragungsart gewählt werden, die jedes System im Netz empfangen kann, auch wenn es noch keine eigene IP-Adresse hat: BROADCAST.

Broadcast-Pakete sind Pakete, welche jeder in diesem Netz aktive Rechner empfangen kann. Sie sind an die sogenannte Broadcast-Adresse des Netzes adressiert.

Auf diese Art und Weise werden die meisten Daten im DHCP-Protokoll ausgetauscht.

Um Daten per Broadcast zu übermitteln, hat jedes Paket eine spezielle Form, je nachdem, welchen Typ von Daten es enthält. So sieht die Anfrage nach einer Konfiguration anders aus, als die Bestätigung einer IP-Address-Zuweisung.

Es werden also alle Informationen in einem einzigen Nachrichtenpaket untergebracht und versendet. In einem solchen Fall würde ein kompletter Verbindungsauf- und Verbindungsabbau, wie man ihn bei TCP/IP hat, einen übermäßigen Protokoll-Overhead produzieren. Deshalb werden alle Informationen über UDP übertragen.

Dabei muß man in Kauf nehmen, daß bei UDP Bei der Verwendung von UDP muß man in Kauf nehmen, daß die Möglichkeit besteht, daß einzelne Nachrichtenpakete verloren gehen und ihren Empfänger nicht erreichen. Diesem Problem kann man im Protokolldesign und in der programmtechnischen Umsetzung entgegen wirken.

So kann man zum Beispiel den Empfang von Daten extra bestätigen lassen:

1. Schritt: Absenden der Daten
2. Schritt: Der Empfänger bestätigt den Empfang der Daten
3. Schritt: der Sender bestätigt den Empfang der Bestätigung oder schickt die nächsten Daten

Bei einem solchen Vorgehen ist es auch erlaubt, daß die Empfangsbestätigung der Empfangsbestätigung verloren geht und das Protokoll deshalb nicht abgebrochen werden muß. Der Verlust anderer Pakete fällt ebenfalls auf.

Die untenstehende Tabelle gibt einen Überblick über die verschiedenen, im DHCP-Protokoll verwendeten Pakettypen, sowie über die Auswirkungen, wenn eines der Pakete verloren geht. Der leichteren Zuordbarkeit ist die Funktion des jeweiligen Paketes mit angegeben.

Paketbezeichnung	Funktion	Folge bei Paketverlust
DHCPDISCOVER	Aufforderung an alle DHCP-Server im Netz, ein Angebot zu schicken, welche Daten der Client vom jeweiligen Server erhalten könnte.	Der Client bekommt auf dieses verlorene Paket keine Antwort von einem DHCP-Server, da die Anfrage nach seiner Existenz verloren gegangen ist. Im schlimmsten Fall bekommt der Client als Folge keine IP-Adresse.
DHCPOFFER	Das Angebot der jeweiligen Server, welche Daten sie dem Client geben können.	Der Client bekommt vom Server das Angebot nicht, kann es also nicht auswählen. Im schlimmsten Fall verfällt das Angebot serverseitig und der Client bekommt mangels IP-Adresse keinen Netzzugang.
DHCPREQUEST	Paket des Clients an die Server. Es enthält die Daten des ausgewählten Angebotes.	Der Server bekommt die explizite Anfrage nicht, kann somit keine endgültige Vergabe der Ressource durchführen und dem Client nicht auf seine Anfrage antworten. Der Client bekommt entweder keinen Netzzugang oder arbeitet das Anmeldeprotokoll von vorne nochmals ab, da er diesmal keine Antwort bekommen hat.
DHCPACK/ DHCPNACK	Endgültige Bestätigung/Ablehnung des Servers an den Client, die Daten aus dem DHCPREQUEST-Paket für diesen Client zu verwalten.	Die endgültige Bestätigung der Ressourcenvergabe geht verloren, der Client ist in einem Schwebezustand und der Server hat die Ressource auf diesen Client bereits festgelegt. Sollte sich daran nichts ändern, bekommt der Client keinen Netzzugang und nach Ablauf der Leased Time wird im Server die vergebene Adresse wieder freigegeben (wegen Zeitüberschreitung). Handelt es sich um ein ACK auf ein DHCPINFORM, kann der Client die benötigten Daten nicht verwenden, da sie im ACK-Paket enthalten sind.

Paketbezeichnung	Funktion	Folge bei Paketverlust
DHCPRELEASE	Explizite Aufforderung des Clients an den Server, die diesem Client zugeordneten Daten wieder frei zu geben.	Der Server kann die vom Client freigegebenen Ressourcen nicht freigeben, da er die vorzeitige Freigabe nicht bekommt. Die Ressourcen werden nach Ablauf der Leased Time verspätet freigegeben.
DHCPINFORM	Anfrage des Clients an DHCP-Server, nach den allgemeinen Konfigurationsparametern des Netzes. (z.B. Gateway, Netmask, aber auch Drucker und andere Services)	Der Client bekommt vom Server keine Antwort, weil dieser die Anfrage nach den Konfigurationsdaten nicht bekommen hat.

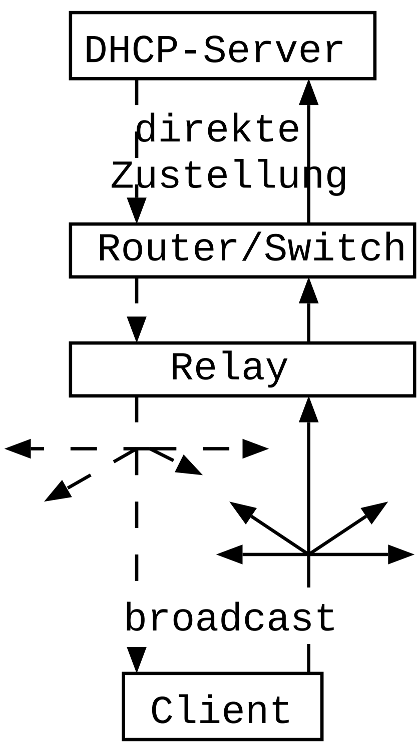

Abbildung 3.7: DHCP: Relay über Netzgrenzen hinweg

Den Ablauf des DHCP-Protokolls stellt Abbildung 3.6 dar. Dabei wird von zwei aktiven DHCP-Servern im Netz ausgegangen.

In der Darstellung fehlt der Ablauf bei der Verwendung von DHCPINFORM. Dieser sieht vor, daß die mit DHCPINFORM angeforderten Konfigurationsdaten in einem DHCPACK-Paket versendet werden.

DHCP-Relay

In manchen Netzen werden Broadcast-Pakete von aktiven Komponenten (z.B. Router) gefiltert und nicht weitergleitet. Dies verhindert die Verwendung von DHCP. Um DHCP trotzdem verwenden zu können, wurde ein DHCP-Relay entwickelt. Dabei handelt es sich um eine Art Proxy für DHCP.

Dabei werden die Broadcast-Pakete von einer Komponente (DHCP-Relay) empfangen, in ein anderes IP-Paket umgesetzt und direkt an den DHCP-Server gesendet. Dieser sendet dann seine Antworten an den DHCP-Relay zurück, welcher sie in ein Broadcast-Paket umsetzt und im richtigen Netzsegment wieder weiterleitet. Abbildung 3.7 stellt diesen Vorgang dar.

Stand der Technik

Es sind zahlreiche Implementierungen für Client und Server vorhanden. Dies gilt sowohl für kommerzielle Lösungen, als auch für nichtkommerzielle Lösungen. Es sind für alle gängigen Betriebssysteme lauffähige Implementierungen für Client- und Serverseite verfügbar.

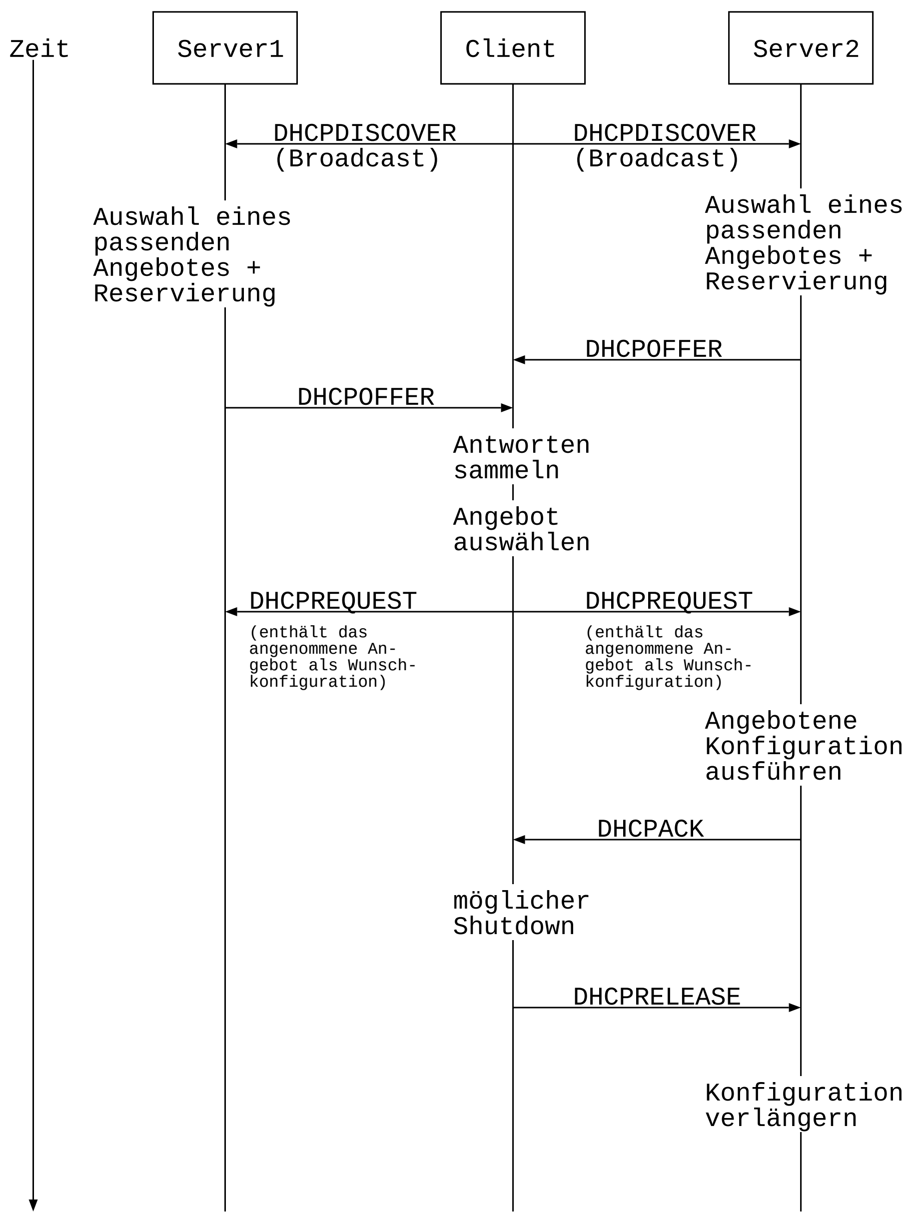

Abbildung 3.6: DHCP: Ausführliches Protokoll-Fluß-Bild

In der Regel wird allerdings von Seite des Clients aus lediglich das Aushandeln einer IP-Adresse verwendet. Die anderen Funktionen werden weniger häufig verwendet.

In vielen Bereichen, in denen DHCP eingesetzt wird, wechseln die einzelnen Clients nur sehr selten ihren Standort (Schulungsräume, Arbeitsplätze). Hat ein Unternehmen aber sehr viele solche Clients im Einsatz, ist die Gesamtzahl der Geräte, die transportiert werden, wieder erheblich und rechtfertigt den Einsatz automatischer Konfiguration.

Damit in diesen beiden Fällen das Netzwerk nicht unnötig von DHCP-Paketen belastet wird, sind hier Lease-Zeiten von mehreren Tagen üblich.

In Schulungsräumen werden die Rechner regelmäßig in einen definierten Zustand zurückversetzt. Dabei handelt es sich meist um eine Neuinstallation oder um ein komplettes Neubeschreiben einer Festplatte von einem Zentralserver aus. Dieser beinhaltet dann eine einheitliche Installation. Die Clients unterscheiden sich letztendlich nur noch in ihren individuellen IP-Adressen. Dieser letzte Unterschied kann dann mit DHCP automatisch beim Einschalten des Rechners durchgeführt werden.

Anmerkungen/Probleme

Proxyfähigkeit: Damit der Service auch in größeren Netzen verwendbar ist, gibt es die Idee des Relayers. Dieser habt die Aufgabe, Anfragen von Clients in ein anderes Netzsegment weiterzureichen (zum Beispiel weil die Netze untereinander keine Broadcast-Pakete austauschen können). Dabei wird das weiterzureichende Paket direkt an einen DHCP-Server zugestellt. Der Relay muß also bereits wissen, welche DHCP-Server aktiv sind. Weiterhin wird das Paket derart erweitert, daß der DHCP-Server erkennen kann, daß es sich um eine Anfrage handelt, die von einem Relayer weitergereicht wurde. Eine Antwort wird dann direkt an den Relayer geschickt, der sie entsprechend im anderen Netzsegment weiterleitet.

Erreichbarkeit: Die Kommunikation zwischen Client und DHCP-Server erfolgt über Broadcast. Werden diese Pakete nicht gefiltert, kann der Dienst (DHCP) von jedem im Netz angesprochen und genutzt werden. Die Verwendung von DHCP kann somit unter Umständen ein Sicherheitsrisiko sein.

Anmeldung: Ein DHCP-Server beantwortet öffentlich (via Broadcast) Anfragen, er meldet sich aber nicht selbständig bei Aktivierung im Netz bei einem zentralen Service (z.B. Directory-Service) an.

Policy-Fähigkeit: DHCP-Server haben sehr einfache Konfigurationsmöglichkeiten, so ist es möglich, abhängig der *MAC-Adresse* (Hardware-Adresse einer Ethernet-Karte) eines Systems festzulegen, welche Daten dieses System erhält. Umfangreiche Gruppenbildungen sind in den vorhandenen Implementierungen nicht vorgesehen.

Kommunikation mit anderen Programmen: Kommunikation mit einer Abrechnungseinheit (z.B. Buchhaltungs-Agent) ist im Protokoll nicht vorgesehen und auch nicht in gängigen Implementierungen. DHCP-Server werden derzeit als Stand-alone-Dienste betrachtet. Allerdings verwenden alle Server - schon aus Gründen der Debuggbarkeit - Logfiles, die man für Accountingzwecke auswerten kann.

Modularität: Flexible Reaktion auf neue Protokolle ist insofern möglich, daß Änderungen unterhalb des UDP-Protokolls erlaubt sind. Das DHCP-Protokoll selbst ist starr.

3.4.2 CORBA - Trading Services

CORBA.tex

Der Trading-Service ist ein Service aus dem Corba-Programm von *OMG (Object Management Group)* und somit komplett in *CORBA (Common Object Request Broker Architecture)* integriert. (http://www.corba.org/)

Client Ein Client ist das Endsystem des Nutzers. Es kann sich dabei um ein Nomadisches System handeln. Oft wird aber auch ein Service in die Rolle eines Clients gedrängt, zum Beispiel, wenn dieser Service zur Erfüllung seiner Aufgabe Zusatzinformationen benötigt, die er bei einem anderen Service erfragen muß. Dann ist er aus Sicht des anderen Services der Client.

Server/Service Ein Service ist eine Dienstleistung. Diese kann von mehreren Servern erbracht werden. Dies bedeutet, daß im Netzwerk zu ein und dem selben Service mehrere Programme existieren, die Anfragen, diesen Service zu nutzen, bearbeiten. Im Folgenden wird häufig von einem Service gesprochen, wenn eigentlich ein beliebiges solches Server-Programm gemeint ist, welches einen Service der entsprechenden Art erbringt.

Designziel

CORBA Trading Services ist zur automatischen Vermittlung verfügbarer Dienste entwickelt worden.

Services (Server) sollen sich bei einem sogenannten *Trader* registrieren können. Dieser Vorgang wird als *Import* bezeichnet. Die Registrierung soll auch in großen Netzen, wie dem Internet funktionieren. Service-Suchende (Clients) erhalten bei dem entsprechenden Trader dann auf Anfrage die nötigen Informationen vermittelt, um den Service zu kontaktieren. Dies wird als *Export* bezeichnet.

Trader sammeln also Informationen, welche Services im Netz zu einem bestimmten Zeitpunkt angeboten werden und vermittelt die entsprechende Information an den Client weiter.

Trading-Konzept

Daß es bei einem so umfangreichen Netz, wie dem Internet nicht mit einem einzigen zentralen Objekt zur Verwaltung getan ist, ist verständlich. Es ist vorgesehen, daß mehrere Trading-Objects parallel existieren (Objektorientiert betrachtet handelt es sich eher um mehrere Instanzen eines Trading Objektes) Ein solches Trading-Object bedient in erster Linie seine unmittelbare Umgebung. Dabei soll jeder Nutzer einen *Scope* mit angeben. Es gibt also eine Vorstellung von 'räumlicher Nähe', zumindest aus netztopologischer Sicht. Scope

Ein solcher Scope bestimmt, in welchem Umfang der Client Dienstangebote vermittelt bekommen soll; zum Beispiel alles, was zu diesem Scope paßt, alles, was der lokale Trader anzubieten hat. Ein Scope enthält eine Art Kurzbeschreibung welche Eigenschaften der gesuchte Dienst erfüllen muß. Damit kann er dann auch mit entfernteren Trading-Objekten direkt oder indirekt (über seinen lokalen Trader) kommunizieren.

Da ein Trader-Object unter diesen Umständen möglicher Weise einen sehr umfangreichen Datenbestand verwalten muß, ist es bereits im ersten Entwurf der Klassenbibliothek vorgesehen, mit Datenbanken zu arbeiten.

Organisationsmodell

CORBA-OM-neu.tex

Da CORBA Trading Services auch auf den Einsatz in sehr großen Netzen optimiert ist, benötigt man verschiedene Mechanismen, um die Menge an Informationen verteilt zu speichern. Dabei wird ein lokaler Ansatz verwendet. Das bedeutet: es werden nicht alle Informationen an alle Komponenten weitergereicht. Informationen sind meist in der Umgebung des Services, zu dem sie gehören, zu finden. In der Regel macht es keinen Sinn, wenn jemand in Berlin etwas auf einem Drucker in New York ausdruckt. Sollte dies doch einmal nötig sein, dann kann er explizit die Informationen für einen Druckservice in New York anfordern. Dies ist dann Aufgabe des Trading Services andere Trading Service Objekte zu kontaktieren und herauszufinden, wer einen solchen Druck-Service anbieten kann.

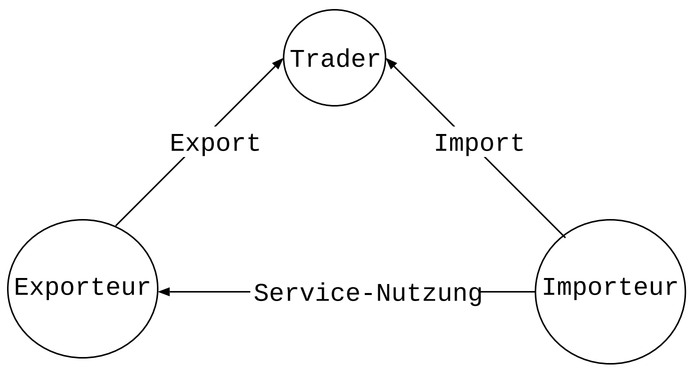

Abbildung 3.8: Corba: Trading-Service

Exporter = Dienstanbieter/Service

Import/Export Ein Exporter ist ein Object, welches sich beim Trader als Service registrieren läßt. Es enthält einen oder mehrere *Service Types*, die er dem Trader bekannt gibt. Ein Service Type enthält immer einen Interface-Typen, der das konkrete Interface-Verhalten definiert ('*computational signature*') sowie kein, ein oder mehrere '*property types*'. Diese beschreiben das Verhalten des Services und alle anderen Informationen, sofern sie nicht schon in der Signatur enthalten sind.

Ein Importeur (Importer) ist ein Object, welches den Trader nutzt um nach Services zu suchen, die gewissen Kriterien entsprechen. Der Trader vergleicht diese Charakteristiken mit den Beschreibungen, die er von den versch. Services bekommen hat und liefert dem anfragenden Objekt die Location des am besten passenden Services. Siehe dazu auch Abb. 3.8.

Abbildung 3.9 veranschaulicht welche Komponente welcher anderen Komponente Informationen sendet: Der Exporter sendet dem Trader die Informationen über die

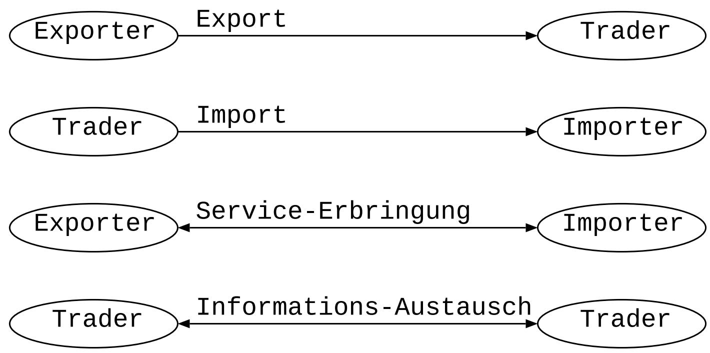

Abbildung 3.9: Corba: Informations-Austausch

Services, die er anbietet. Diese Informationen reicht der Trader an den Importer auf Anfrage weiter. Mit diesen Informationen kann der Importer direkt mit dem Exporter kommunizieren, bzw. der Exporter kann dem Importer Daten senden, bzw. für ihn einen Service erbringen. Es ist zusätzlich vorgesehen, daß Trader untereinander die Service-Informationen der ihnen bekannten Services austauschen.

Verknüpfung mehrerer Trader untereinander Es ist eine Verknüpfung mehrerer Trader untereinander in der Spezifikation des CORBA Trading Services vorgesehen. Werden verschiedene Trader miteinander verbunden, so bieten beide ihre Dienste den Importern (Clients) des jeweils anderen Traders mit an. (Siehe Abbildung 3.10, gekennzeichnet durch die gestrichelten Linien)

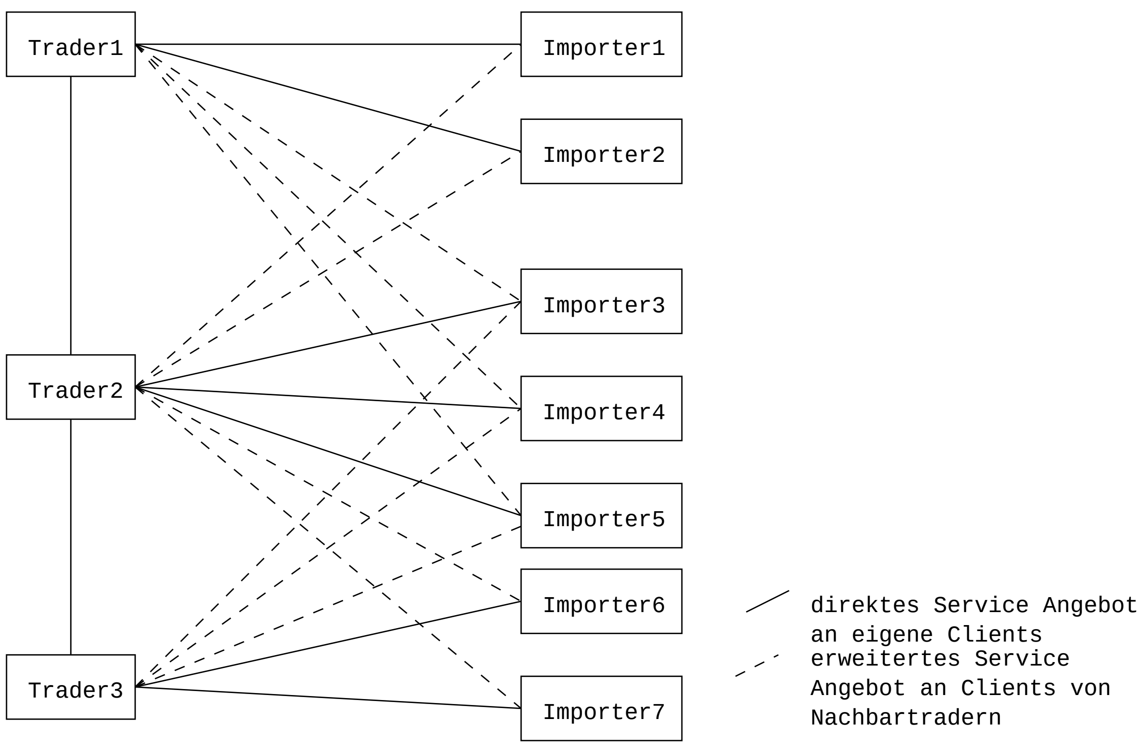

Abbildung 3.10: Corba: Bezug Trader/Importer bei mehreren Tradern

Jeder Trader sieht dabei nur eine begrenzte Anzahl seiner Nachbar-Trader, nämlich die, mit denen er selbst explizit verknüpft ist. Die Verlinkung wird über einen Gerichtete Graphen (*directed graph*) realisiert, den man den *trading graph* nennt. Solche Verlinkungen sind in beliebigem Umfang und auf beliebige Entfernungen hinweg (über Domaingrenzen und Netzgrenzen hinweg) möglich.

Link *Links* repräsentieren Pfade für die Verteilung vom Quell-Trader zum Ziel-Trader. Ein Link beschreibt das Wissen, welches ein Trader über einen anderen Trading Service, der ihn nutzt, hat.

Scopes Neben der Relativierung der Anforderungen an den Exporter seitens des Importers, kann der Importer durch die Verwendung von Scopes den Suchraum erweitern. Der Standard-Auswahl-Bereich für Exporter ist der jeweilige Trader und seine direkten Nachbarn. Dieser Bereich kann erweitert werden, indem der Importer mittels Scopes angibt, wie viele Knoten weiter nach 'außen' gesucht werden darf (siehe Abb. 3.11).

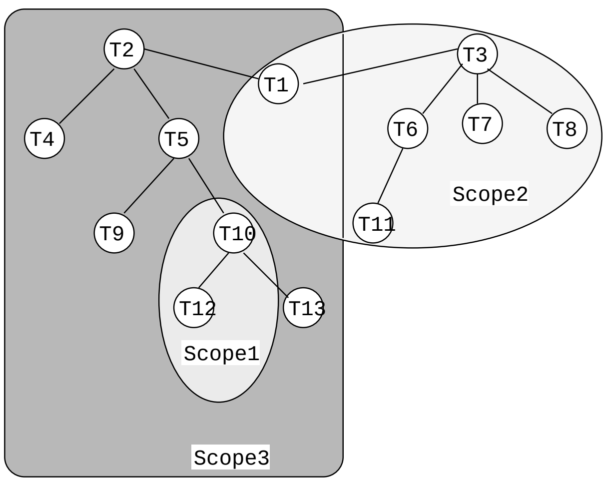

Abbildung 3.11: Corba: Erweiterung des Standard-Suchbereiches durch Scopes

In Abbildung 3.11 wird ein Trader mit T und einer laufenden Nummer bezeichnet. Wirkungsbereiche von Scopes sind farbig hinterlegt und enthalten den Namen des betreffenden Scopes als Scope und laufende Nummer.

Scope1 ist der Default-Bereich aus Sicht eines Importers, der direkt mit Trader12 (T12) verbunden ist. Es wird der direkte Trader (T12) gefragt, sowie der nächste direkt verbundene (T10).

Scope2 könnte zum Beispiel entstehen, wenn ein Importers von T6, T7 oder T8 aus mit Erweiterung um einen Knoten suchen würde. Der gleiche Bereich träfe aber auch für einen Client zu, der mit T11 verbunden ist und mit Erweiterung um drei Knoten suchen würde.

Scope3 ist ein erweiterter Bereich aus Sicht eines Importers, der an T12 oder T13 angebunden ist und besagt, daß bis zu 4 übergeordnete Trader gefragt werden dürfen; T1 ist enthalten.

Informationsmodell

CORBA-IM-neu.tex

Die Hauptaufgabe vom CORBA Trading Service ist es, einen Dienst zu vermitteln und somit die Nutzung dieses Dienstes zu ermöglichen.

Dazu muß sich jeder Dienst (genauer: jeder Server, der einen Dienst erbringen kann), der vermittelt werden soll, bei einer lokalen Zentrale (*Trader*) anmelden und dort eine möglichst genaue Beschreibung von sich ablegen. Diese Beschreibung kann unter anderem folgende Informationen enthalten:

- Typ des Dienstes (Druck-Service, Email-Service...)
- Einzelheiten zum Diensttyp (0-Druck, Endlospapier, POP3-Server...)
- räumliche Angaben (3. Stock, Zimmer 250, München, Öttingenstraße...
- Quality of Service (20 Seiten pro Minute, 2Mbit-Außenanbindung...)
- Angaben zum zugehörigen Interface
- eigene Policy-Regeln

Dienstvermittlung

Soll ein Dienst genutzt werden, muß sich der Dienst-Nutzer (Client) an einen Trader wenden und den gesuchten Dienst so genau wie möglich spezifizieren (z.B. Drucker, 3. Stock, schwarz/weiß). Der Trader ermittelt dann, anhand aller Informationen zu Diensten die sich bereits angemeldet haben, den für dieses Fall am besten passenden Dienst. Die Informationen zu diesen Dienst liefert der Trader an den Anfragenden zurück. Mit den erhaltenen Informationen ist dieser jetzt in der Lage den Dienst direkt zu nutzen.

Link

Ein Link wird verwendet, wenn mehrere Trader untereinander Daten austauschen. Er enthält unter anderem folgende Informationen (siehe dazu auch Seite 16-11 in [OMG97]):

- ein Lookup-Interface
- ein Register Interface
- Regeln, wie dem Link zu folgen ist, falls nichts anderes angegeben wurde
- Regeln, wie dem Link zu folgen ist, die andere Angaben beschränken können

Scopes

Scopes dienen zur Erweiterung oder Eingrenzung der Suche. Sie werden in der Regel vom Client an den Trader gesendet um die Suchkriterien zu relativeren. So kann z.B. ein Scope die Angabe enthalten, daß alle Suchkriterien erfüllt sein müssen oder daß nur ein Teil der Kriterien zu erfüllen sind. Es kann ebenfalls darüber informiert werden, wenn ein Client auch Services nutzen würde, die nicht direkt von diesem einen Trader angeboten werden, sondern von einem Trader in mehr oder weniger unmittelbarer Nachbarschaft.

Funktionsmodell

CORBA-FM-neu.tex

Corba Trading Services ist ein etwas umfangreicheres Konstrukt zur Vermittlung von Services, als es das DHCP-Protokoll ist.

Die Aufgaben eines Trading-Services beschränken sich nicht nur auf die Vergabe und das Management von IP-Adressen, sowie die Weiterleitung fest vorkonfigurierter allgemeiner Informationen.

Vielmehr muß ein Trading-Service dynamisch wechselnde Services verwalten. Er muß die Informationen neuer Services dynamisch annehmen können und auf der anderen Seite für einen Nutzer aus einen optimalen Service aus seinem Angebot auswählen.

TRADER_SUCHEN Der Trader ist eine zentrale Stelle im CORBA-Trading-Service. Er sammelt alle Informationen und Anfragen und 'vermittelt' den passenden Kontakt. Damit dies möglich ist, müssen alle Teilnehmer einen Trader finden. Dem Trader kann dann ein Service die Informationen zu den Diensten, die er anbietet senden. Ein Client kann dem Trader seine Beschreibung des Dienstes senden, den der Client benötigt.

SERVICEDATEN_ANFORDERN Hat ein Client einen Trader gefunden, so kann er von ihm Daten zu einem speziellen Service anfordern.

SERVICE_ANMELDEN Hat ein Service einen Trader gefunden, so kann er diesem die für seinen Service spezifischen Daten senden und sich somit bei diesem Trader registrieren.

SERVICE_ABMELDEN Ist ein Service nicht mehr verfügbar, so kann er sich bei dem Trader, bei dem er sich angemeldet hatte, auch wieder abmelden. Als Folge dieser Abmeldung, werden seine Informationen nicht mehr mit Suchanfragen von Clients verglichen.

SERVICE_SUCHEN Hat ein Trader von einem Client eine Beschreibung für einen Service bekommen, so muß er aus allen bei ihm gemeldeten Services einen (oder mehrere) Services auswählen, die mit ihren Beschreibungen den Anforderungen des Clients möglichst nah kommen. Dabei ist es auch möglich, daß nicht nur Services betrachtet werden, die sich direkt bei diesem Trader angemeldet haben, sondern auch solche, die sich bei benachbarten Tradern gemeldet haben.

SCOPES_NUTZUNG Mit Scopes beschränkt der anfragende Client die vom Trader zu durchsuchende Datenmenge. Mit Scopes kann der anfragende Client beispielsweise angeben, ob er alle auffindbaren Services bekommen möchte oder nur die, die von seinem direkten Trading-Object vermittelt werden oder nur die Services, die genau auf seine Anforderungsbeschreibung passen.

POLICIES_UND_PROPERTIES_EINSETZEN Es ist möglich einem Trader, sowie einem einzelnen Service Regeln aufzuerlegen (Policies).

Dabei kann bestimmt werden, wie mit den Property-Angaben umzugehen ist: 'mandatory', 'readonly' oder beides. (Abb. 3.12).

Weiterhin kann man Properties nach einem vorgegebenen Schema ([OMG97] Kapitel 16.2.4) selbst definieren.

Policies geben dem Administrator die Möglichkeit Eigenschaften von Services zur Laufzeit zu manipulieren. Es gibt zwei grundlegende Arten von Policies:

1. Policies, die die Regeln zur Suche beeinflussen (siehe auch 3.4.2 auf Seite 54)
2. Policies, die die Funktionalität von Operationen einschränken (oder erweitern)

Es gibt Policies speziell für Trader, Links, und Importeure.

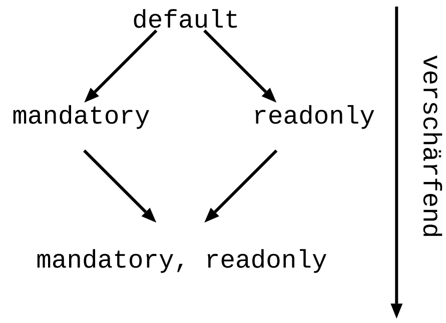

Abbildung 3.12: Corba: Properties-Hierarchie

Protokoll-Besonderheiten und Details

Das Konzept der CORBA Trading Services ist speziell auf größere Netzwerke wie das Internet ausgelegt. Das bedeutet, daß bei der Entwicklung der Mechanismen auf eine Netzgröße dieser Art speziel geachtet wurde und entsprechende Mechanismen zur hierarchischen Gliederung des Systems entwickelt wurden. Beispiele hierfür sind Scopes, sowie die vorgesehene Anbindung von Datenbanken und die Verteilung auf mehrere Trader.

Die Vergabe einer IP-Adresse, sowie gezielte Angaben von Boot-Informationen sind im derzeit vorliegenden Protokoll [OMG97] noch nicht realisiert. Allerdings ist bereits ein spezielles Object als reserviert markiert, welches später einmal zum 'Booten von Extern' (bootstrap) verwendet werden soll. Wie dies genauer aussehen wird, wird Gegenstand einer der nächsten Versionen dieses Protokolls sein.

Stand der Technik

CORBA Trading Services sind ein sehr junges Konzept. Demnach gibt es kaum Implementierungen, die es verwenden. Allerdings wird derzeit von verschiedenen Unternehmen an solchen Projekten gearbeitet, so daß man in naher Zukunft mit einer ganzen Reihe CORBA-fähiger Services rechnen kann.

Die Arbeiten werden zum großen Teil von der OMG koordiniert. Dort wird man auch viele freie Beispiel-Implementierungen (neben den schon Existierenden) finden können.

3.4.3 SLP

SLP.tex

Das *SLP (Service Location Protocol)* bietet eine Umgebung zum Auffinden und Auswählen von Netzwerk-Diensten. Zum Zeitpunkt der Erstellung dieses Arbeitsabschnittes ist ist Version1 verabschiedet (RFC 2165 [VGPK97]). Version2 wurde vom IETF im Juni 1999 verabschiedet (RFC 2608 [GPVD99]). Diese Arbeit bezieht sich vorwiegend auf SLPv1.

In SLPv2 wurden sehr umfangreiche Veränderungen vorgenommen. Diese Veränderungen treffen nahezu alle Bereiche: Scopes, Formate und Definitionen vieler Flags, sowie einen neuen Nachrichtentyp: SAAdvert für die Kommunikation ohne Directory Agents.

Client Im Folgenden ist der Client der Rechner, der einen Dienst nutzen soll.

Server Ein Server kann ein System sein, welches diesen Dienst direkt anbietet oder dem anfragenden Client mitteilen kann, welches System diesen Service anbietet (Auskunftei).

Nutzer Unter Nutzer wird hier ein externer Benutzer verstanden, in der Regel ein Mensch, der am Client arbeitet. Der Nutzer muß spezifizieren, welche Dienste er nutzen möchte (z.B. Druckservice auf dem Drucker im Nachbarzimmer).

Designziel

Computer, denen bereits ein Zugang zu einem IP-basierten Netzwerk zur Verfügung steht, sollen Dienste selbständig auffinden und nutzen können. Dabei soll keine manuelle Hilfe von einem Administrator nötig sein. Der optimale zu nutzende Dienst-Leister (z.B. Drucker) soll allein anhand einer Reihe Kriterien, die der Nutzer bestimmen kann, automatisch gefunden, konfiguriert und genutzt werden.

Dabei soll anhand der Informationen über die Existenz, räumliche Lage (Drucker) und Konfiguration von Netzwerk-Diensten der zu nutzende Service ermittelt werden. Diese Informationen sollen natürlich-sprachlich abgegeben werden können.

Der Nutzer soll lediglich eine Beschreibung des gewünschten Dienstes, sowie einige Rahmen-Daten angeben müssen und bekommt automatisch als Ergebnis Zugriff auf diesen Dienst (sofern vorhanden). Sämtliche Konfigurations-Arbeit soll für den Nutzer unsichtbar und automatisch ablaufen. Die Vergabe einer IP-Adresse ist nicht vorgesehen. Für diese Aufgabe wird explizit auf 'andere' Protokolle verwiesen.

Dabei soll eine zentrale Konfiguration und Steuerung aller Regeln möglich sein. Es soll möglich sein, Policies bezüglich Sicherheit, Multicast-Routing und Organisation der Dienste festzulegen und durchzusetzen.

SLP wurde designed, um Firmen-Netzwerke mit 'Shared Services' zu bedienen. Eine Nutzung über das gesammte Internet hinweg oder in Netzen mit mehreren tausend Clients ist nicht vorgesehen.

Das Service Location Protokoll basiert auf der Vorstellung, daß alle Diensterbringer und alle Dienstnutzer (also Clients und Server) als Agenten betrachtet werden. Somit wird meist ein Client als UserAgent bezeichnet und ein Service als ServiceAgent. Ein DirectoryAgent bildet einen speziellen Auskunfts-Service.

Nachdem SLP nur eine allgemeine Dienstvermittlung beschreibt, muß in einer konkreten Verwendung durch einen Dienst dieser Dienst, bzw. sein Typ gekennzeichnet (benannt) werden können. Dies wird mit 'ServiceType' umgesetzt.

Ein Scope ist ein Bezeichner, um einen speziellen Bereich des Netzwerkes zu benennen.

Um Lokalitäten zu benennen (Rechnername, Port) verwendet man hier ausschließlich URLs (Uniform Ressource Locator). Der URL ist im Zusammenhang mit dem allgemeinen Web-Hype berühmt geworden und hat folgende Grundstruktur (wie in RFC 1738 [BLMM94] beschrieben):

```
protokoll://[(host.)*]rechner.domain[:port]/<pfad>/<dokument>
```

Informationsmodell

SLP-IM.tex

Will ein Service Agent seine Dienste im Netz anbieten, so muß er sich bei einem Vermittlungs-Service (*Directory Agent*) registrieren. Dieser Directory Agent vermittelt dann den Service entsprechend weiter. Dazu muß der zu vermittelnde Service Agent dem Directory Agent Attribute liefern, die sein Angebot so gut wie möglich beschreiben und nach denen ein Service-Suchender (*User Agent*) fragen könnte. Beispiele für solche Attribute können sein: Drucker, Laser, Farbe, 3. Stock - es handelt sich um einen Farb-Laser-Drucker, der im 3. Stock steht.

Der User Agent schickt einen *Request* (Anfrage), in dem er den gesuchten Dienst beschreibt. Dieser Request enthält eine sehr menschliche Beschreibung wie: Drucker, 2. Stock, Laser, schwarz/weiss. Diese Information soll mit einer Schnittstelle direkt vom Benutzer erfaßbar sein. Dabei soll der Nutzer keine Computerspezifische Ausdrucksweise erlernen müssen.

Es ist jetzt Aufgabe des Directory Agents anhand seiner Attribute den passendsten Service aus den verschiedenen Beschreibungen herauszufinden und an den User Agent zu vermitteln. Es müssen nicht immer alle Attribute der Anfrage erfüllt werden, sondern nur möglichst viele, bzw. ähnliche. Wenn es z.B. in 2. Stock keinen Drucker gibt, aber im 3. Stock und im Keller, sollte der Drucker im 3. Stock angeboten werden (bei gleicher sonstiger Eignung).

Mit diesen Informationen wird ebenfalls die Ansprechadresse des Dienstes (z.B. Name und IP-Adresse, falls es sich um einen Netzwerkdrucker handelt), an den User Agent übertragen. Dieser kann jetzt, wenn er den Dienst nutzen möchte, direkt mit dem zugehörigen Service Agent kommunizieren.

Zusätzlich ist es im SLP-Protokoll vorgesehen, daß ein Service Agent seine Existenz und seine Attribute nicht an einen Directory Agent übermittelt, sondern hin und wieder selbständig im Netz über einen Broadcast-Mechanismus bekannt gibt. Diese Technik ist für kleine Netze mit wenigen Dienstangeboten gedacht.

Organisations- und Funktionsmodell

SLP-OM.tex

User Agent und Service Agent alleine

Anwendungen auf dem Client werden als *User Agents* betrachtet und Dienste als *Service Agents*. *Directory Agents* bieten eine Skalierbarkeit (scalability) des Protokolls.

Der User Agent schickt einen SrvRqst (Service Request), in dem die Eigenschaften des gesuchten Services angegeben werden. Als Antwort erhält der User Agent die Adressen aller Dienste im Netzwerk, die seine Anforderungen erfüllen. (siehe Abb. 3.13)

Es ist erlaubt, daß ein User Agent sich direkt an einen Service Agent wendet. In diesem Fall wird der Request als *Multicast* betrachtet. Service Agents, die eine Anfrage nach einem Service erhalten, den sie anbieten können, schicken ihre Antwort als *unicast-reply*, sie enthält lediglich die Adresse des Services.

Verwendung eines Directory Agent

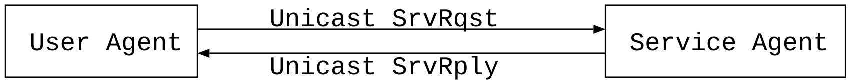

Abbildung 3.13: SLP: Direkt-Vertrieb

In größeren Netzwerken können ein oder mehrere *Directory Agents* verwendet werden. Ein solcher Directory Agent fungiert als Cache. Die Service Agents senden Registrierungs-Meldungen (SrvReg) an den Directory Agent. Diese Registrierungs-Meldungen enthalten Informationen über alle Services, die der Agent anbieten kann. Als Antwort bekommen sie ein SrvAck. Das Angebot muß von Zeit zu Zeit erneuert werden. User Agents schicken ihre Anfragen in diesem Fall an Direcory Agents und nicht an Service Agents. Voraussetzung hierbei ist, daß ein Directory Agent bekannt sein muß. (Abb. 3.14)

Abbildung 3.14: SLP: Mit Directory Agent

Für User- und Server Agents gibt es zwei Möglichkeiten, einen Directory Agent zu finden (siehe auch Abb 3.15)

1. Wenn sie starten, schicken sie einen Multicast Service Request an den Directory Agent-Service.

2. Der Directory Agent sendet hin und wieder unaufgefordert ein Directory-Agent-Angebot (*Directory Agent Advertisement* im Netz.)

In beiden Fällen bekommen die User Agents vom Directory Agent (*DA*) ein Directory Agent Advertisement (DAAdvert).

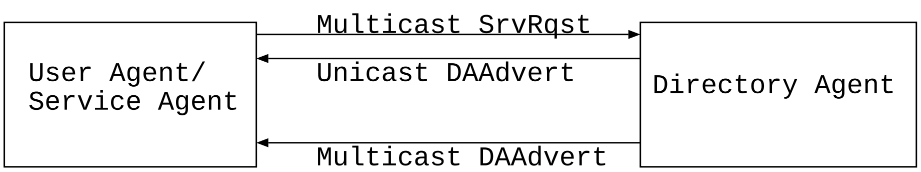

Abbildung 3.15: SLP: Auffinden eines Directory Agent für User und Service Agent

Scopes

Einzelne Dienste werden mit Hilfe von *Scopes* zusammengefaßt. Scopes sind Arrays, die Dienste beschreiben, welche 'administratively identified' wurden. Ein Scope kann beinhalten:

- eine Adresse
- eine administrative Gruppe
- eine Angabe über eine Lage in einer Netzwerk-Topologie
- weitere Kategorien

Einem Service und Directory Agent ist immer ein Scope zugeordnet. User Agents ist normaler Weise einer zugeordnet, was dazu führt, daß der Agent ausschließlich Dienste dieser (seiner) Gruppenzugehörigkeit auffinden kann. Alternativ darf der User Agent auch ohne Scope konfiguriert werden. In diesem Fall findet er alle Dienste im Netzwerk. (Abb. 3.16)

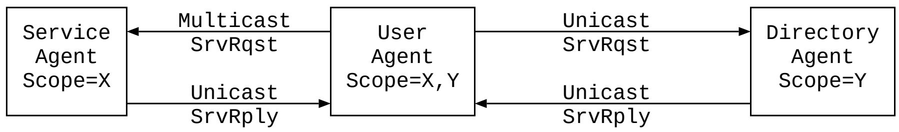

Abbildung 3.16: SLP: Verwendung von Scopes

Protokoll-Besonderheiten und Details

In letzter Zeit wurde vielfach der Wunsch nach digitalen Signaturen laut. Digitale Signaturen werden verwendet, um festzustellen, ob die übertragenen Daten auf ihrem Weg manipuliert wurden. Als Nebenprodukt ergibt sich eine Authentifizierung des Absenders, da nur dieser für eben diese Daten-Kette eine solche Signatur erzeugen kann. Deshalb ist die Verwendung digitaler Signaturen im SLP vorgesehen. Dort wird sie im DAAdvert Verwendung finden.

SLP ist bezüglich zur Verfügung stehender Protokolle sehr frei. So wird ausdrücklich festgelegt, daß Kommunikation, die über Multicast verteilt werden soll, alternativ auch über Broadcast abgewickelt werden kann.

Ein ebenfalls wichtiges Thema ist die Angabe wo sich der Service befindet. Diesen Zweck erfüllen URLs sehr gut. Deswegen wird zur Angabe von Adressen im Protokoll ausschließlich die Form des *Uniform Ressource Locator (URL)* verwendet.

Das IETF hat verfügt, daß aus Kompatibilitätsgründen SLPv2 abwärtskompatibel zu SLPv1 sein muß. Dies war nötig, da sich SLPv1 bereits verbreitet hat, SLPv2 aber sehr unterschiedlich aussehen sollte, quasi eine Neuentwicklung geworden war.

Stand der Technik

Es gibt verschiedene Implementierungen verschiedener Hersteller. Diese beschränken sich derzeit (soweit feststellbar) weitestgehendst auf SLP Version 1. Allerdings arbeiten auch gerade während diese Arbeit entsteht Gruppen schon an Implementierungen für SLPv2. So hat zum Beispiel SUN angekündigt, daß das neue Solaris 7.0 SLPv2 verwenden wird.

SLP ist etwa zur gleichen Zeit entwickelt worden wie DHCP. Trotzdem hat es sich bis jetzt nicht annähernd so gut durchgesetzt und ist sehr viel seltener in der Praxis zu finden. Mögliche Ursachen hierfür könnten sein, daß das Protokoll in seiner Endumsetzung deutlich komplexer ist, als DHCP.

Als Ausgleich für seine Komplexität bietet das Service Location Protokoll eine ganze Reihe von Features, die bereits beim Design des Protokolls vorgesehen waren:

Proxyfähigkeit: Die Verwendung von Proxies ist in Form von DirectoryAgents vorgesehen.

Erreichbarkeit: Die Kommunikation läuft über Unicast/Multicast, auch Broadcast ist vorgesehen. Dies sollte überall realisierbar sein, bzw. an entsprechenden Filtergeräten (Switches) aktivierbar. Damit sind verschiedene Kommunikationswege bereits im grundlegenden Design vorgesehen. Man kann somit auf eine andere Technik zurückgreifen, wenn zum Beispiel die allgemeinen Policies die Verwendung von Broadcast nicht zulassen.

Anmeldung: Das Protokoll ansich sieht umfangreiche Möglichkeiten vor, einen Service im Netz an zentralen Stellen anzumelden um ihn somit besser vermitteln zu können und einem Client das Auffinden eines speziellen Services deutlich zu vereinfachen. Siehe dazu auch Abb. 3.15 und Abb. 3.16.

Policyfähigkeit: Mit Hilfe der Scopes (Abb 3.16) lassen sich teilweise auch Policies umsetzen.

Kommunikation mit anderen Programmen: Eine direkte Kommunikation mit Programmen (z.B. Abrechnungs-Agenten oder Ähnlichem) ist im Protokoll nicht vorgesehen. Allerding ist dies bei der reinen Dienstvermittlung, wie sie bei SLP vorliegt auch nicht nötig. Die entsprechenden Agenten werden wie alle anderen Services behandelt und können sich anmelden und an Clients vermittelt werden. Ein solcher Client kann natürlich wiederum ein Service sein, der zur Erbringung seiner Dienstleistung einen entsprechenden anderen Service nutzen muß. Diesen Service kann er mit Hilfe vom SLP selbständig finden.

Modularität: Das Protokoll und sein Ablauf, sowie das Aussehen der einzelnen Pakete, in denen Informationen übermittelt werden, sind festgeschrieben. Über die Dienstleistung, die die einzelnen zu vermittelnden Services erbringen, wird keine Festlegung getroffen. Es kann jeder beliebige bereits existierende Dienst vermittelt werden, solange er sich in seiner Kommunikation mit dem vermittelnden Directory Agent an das Service Location Protokoll hält. Auch die Verwendung von Netzwerkprotokollen (Ethernet, ATM) ist nicht explizit festgelegt. Es ist lediglich festgelegt, welche Funktionen diese Protokolle erfüllen müssen.

3.4.4 JINI

JINI.tex

Jini ist, genauso wie die CORBA Services (in dieser Arbeit wurde der Trading Service beschrieben), kein echtes Protokoll, wie man es zum Beispiel bei DHCP oder SLP oder anderen netzwerknahen Protokollen hat.

Jini baut vielmehr auf einem beliebigen Protokoll auf und ist eher eine Festlegung der Art und Weise, wie die Kommunikation einzelner Komponenten untereinander auszusehen hat und welche Informationen übertragen werden sollen als eine Festlegung, wie welches übertragene Bit auszusehen hat.

Mit *Jini (Java Intelligent Network Infrastructure)*, sollen Geräte unterschiedlicher Art über Netz miteinander kommunizieren können und managebar sein. Klassisches Beispiel:

Der Toaster schaltet sich automatisch ein, sobald die Kaffee-Maschine fertig ist.

Entwickelt wird Jini von SUN. Sämtliche Dokumentationen und Spezifikationen befinden sich unter http://www.sun.com/jini/. Jini basiert auf Java. Dies bedeutet, daß alle Komponenten der Plattform in Java programmiert sind. Zur Kommunikation untereinander wird die RMI-Schnittstelle von Java verwendet.

Designziel

Vorstellung bei der Entwicklung von Jini war ein zentral managebarer Haushalt. Komplett, mit allen darin verwendeten Geräten. Dabei können alle Teilnehmer miteinenander kommunizieren und sich so ggf. koordinieren.

Die Grundidee von Jini trifft in erster Linie alle externen Geräte, die Strom verbrauchen: Fernseher, Waschmaschine, Toaster, Drucker, Scanner, Modem, Kühlschrank. Das klassische Beispiel, welches die Entwickler von Jini (SUN) anführen, ist die Kommunikation von Haushaltsgeräten miteinander (der Toaster startet erst, wenn die Kaffeemaschine fertig ist) und Haushaltsgeräte mit dem Menschen (der Wohnungsbesitzer wird informiert, wenn die Alarmanlage ausgelöst wird).

In der Praxis wird Jini derzeit getestet um Nomadischen Systemen automatischen Netz-Zugang zu gewähren. Als Beispiel wird hier oft ein Konferenz-Szenario verwendet, in dem sich Teilnehmer in einem Raum, oder über Videokonferenz treffen und mit Hilfe des Jini-Systems alle lokal verfügbaren Dienste automatisch zur Verfügung gestellt bekommen (Video-Beamer, Drucker). Dieses Angebot kann dann je nach Client noch um externe Geräte erweitert werden (Drucker zu Hause). ([Mic99b])

Dabei werden mit Hilfe des Jini-Systems alle nötigen Informationen zur Nutzung eines Services an den entsprechenden Client übertragen, bis hin zu speziellen Schnittstellen-Implementierungen für externe Geräte.

Organisationsmodell

JINI-OM.tex

Jini wird von Sun auf der Basis von Java aufgebaut. es handelt sich um ein relativ komplexes, in Java programmiertes System aus verschiedenen Services, die Informationen sammeln, zur Verfügung stellen, weiterleiten und abfragen.

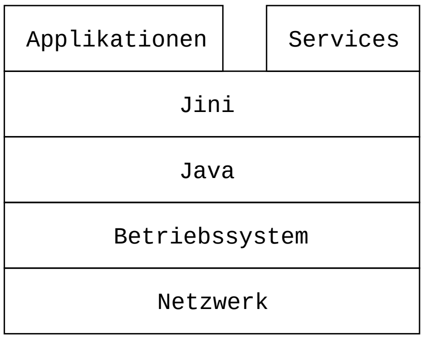

Abbildung 3.17: Einordnung von Jini

Die Jini-Technologie kann in zwei Kategorien unterteilt werden: Infrastruktur und Distributed Programming. Zusätzlich laufen Netzwerk-Services ganz oben auf Jini-Technologie (siehe Abbildung: 3.17 in Anlehnung an Bild 1 des Jini White Paper)

Der Client interagiert mit einem Service über ein Set wohl definierter Interfaces, welche in Java geschrieben sind. Diese Interfaces definieren eine Ansammlung von Methoden, welche zur Kommunikation mit dem Service verwendet werden können.

Ein Clientfindet über einen 'Lookup Service' einen anderen Service und automatisch alle nötigen Protokollinformationen, um den Service zu nutzen.

Informationsmodell

JINI-IM.tex

Das weit gefächerte Angebot an Diensten, die man mittels Jini an Clients verteilen lassen kann, wird auf die Grundidee von Dienstleistern, die Dienstleistungen anbieten, heruntergebrochen. Idee: Das Gerät X bietet einen Dienst Y an.

Dazu benötigen alle teilnehmenden Geräte einen Netzzugang, sowie entsprechende Zusatzfunktionen, die die Kommunikation im Netz ermöglichen.

Über einen Lookup-Service können Dienstleister ihre Dienste anbieten, aber auch Dienstleistungen abfragen. Ein Lookup-Service ist ein zentraler Service, an den sich ein Dienstanbieter, der einen Dienst anbietet, wendet. Er sendet dem Lookup-Service die Informationen welchen Dienst er anbietet, sowie einige zusätzliche Eigenschaften, seines individuellen Service-Angebotes, zum Beispiel, daß er sich im 2. Stock des Gebäudes X befindet.

Ein Dienstsuchender kann einem Lookup-Service eine Anfrage nach einem speziellen Dienst senden. Diese Anfrage enthält den Typ des Dienstes, sowie einige 'Wunschparameter', z.B. daß der Dienst sich im 2. Stock des Gebäudes X befinden soll. Dieses Feature ist beispielsweise bei Druckern in weitläufigen Gebäuden hilfreich. Der Lookup-Service sendet dem Client alle nötigen Informationen zu dem gesuchten Dienst zurück. Nun kann der Client direkt mit dem Dienst kommunizieren und ihn nutzen.

Als 'nötige' Informationen wird hier nicht nur eine Adresse und Service-Beschreibung verstanden, sondern auch konkreter Code, der die Schnittstelle zum Server enthält. Der Client bekommt sozusagen den Treiber geliefert. Dies ist möglich, da Jini auf der Basis von Java realisiert ist und somit Programminformationen im allgemein gültigen Java-Byte-Code übertragen werden und verwendet werden können.

Dieser Code enthält das in Java programmierte Interface des Services, sowie alle Methoden die Nutzer und Applikation aufrufen müssen um den Service zu nutzen. Weiterhin enthält es alle anderen den Service beschreibenden Attribute.

Die Möglichkeit, Code vom Service Provider zum Lookup Service zu transportieren und von dort zum Client, gibt dem Service Provider große Freiheiten bezüglich der Kommunikation zwischen Client und Service; er kann sie gestalten, wie er es für richtig hält, mit Verschlüsselung, ohne Verschlüsselung, mit einem anderen Protokoll, welches speziell auf die Übertragung der jeweiligen Datentypen abgestimmt ist oder mit einem bereits verbreiteten Protokoll.

Zusätzlich ist es möglich, den Status eines Dienstleisters zu erfahren und ggf. passend zu reagieren. Z.B. der Drucker meldet, daß eine Inspektion fällig ist oder daß das Papier in Schacht 3 verbraucht wurde.

Voraussetzung dafür ist natürlich, daß jeder teilnehmende Dienstleister das Protokoll Jini implementiert hat.

Jini bietet Mechanismen um Devices, Services und User in ein bestehendes Netzwerk zu integrieren. Sich an ein Jini-System anzubinden oder wieder abzukoppeln, ist oft ein vollautomatischer Vorgang.

Das Herzstück von Jini ist ein Protokollpaar: discovery und lookup. Discovery und lookup werden zu unterschiedlichen Zeitpunkten verwendet.

Funktionsmodell

JINI-FM.tex

Der allgemeine Ablauf der Kommunikation bei Jini sieht eine Discovery-Aktion, einen Lookup-Schritt und einen letzten Schritt vor, der dem Client letztlich die Nutzung des Service-Interfaces mitteilt.

Discovery wird verwendet, um einen Service beim Jini-Lookup-Service anzumelden, zum Beispiel weil ein Device aktiviert wurde.

Lookup wird aktiviert, wenn ein Client einen Service anhand einer Beschreibung sucht.

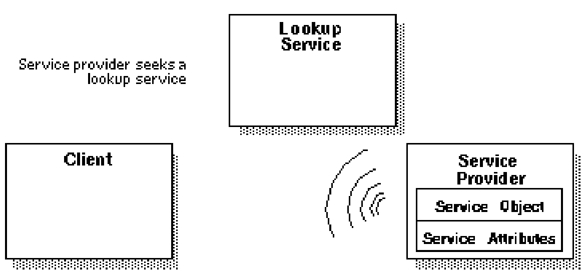

Abbildung 3.18: Discovery aus [Mic99d]

Abbildung 3.18 verdeutlicht den Discovery-Vorgang. Discovery ist der Vorgang einen neuen Service zum Jini-System hinzuzufügen. Im Service Location Protokoll kommt dies der Anmeldung der Services beim Directory Agent gleich. Der Service-Provider ist die Quelle des Services. Als ersten Schritt sucht der Service-Provider nach einem Lookup Service, indem er eine Broadcast Nachricht verschickt, die seine Existenz verbreitet.

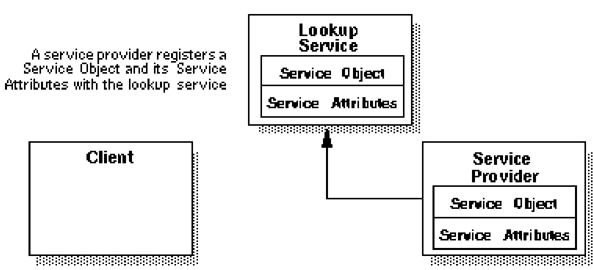

Abbildung 3.19: Discovery aus [Mic99d]

Danach wird ein Service Object für diesen Service in den Lookup Service geladen (siehe Abbildung 3.19).

Dieses Service Object enthält das in Java programmierte Interface des Services, sowie alle Methoden und beschreibenden Attribute die der Nutzer zur Verwendung des Services benötigt.

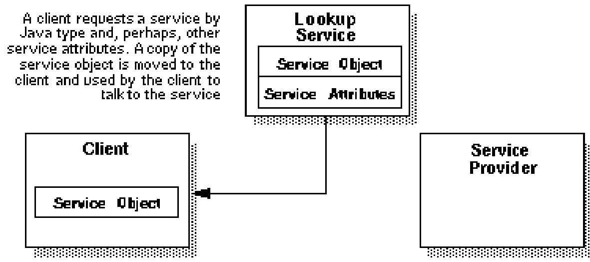

Abbildung 3.20: Lookup-Schritt aus [Mic99d]

Clients müssen in der Lage sein einen Lookup Service zu finden, allerdings darf diese Aufgabe auch an Dritte delegiert werden. Hat der Client einen Lookup Service gefunden, werden die benötigten Daten, die zuvor vom Service-Provider auf den Lookup Service übertragen wurden, vom Lookup Service an den Client gesendet. Siehe dazu auch Abb 3.20.

Ein Client lokalisiert einen passenden Service anhand seines Types, zusammen mit beschreibenden Attributen, welche im Interface des Lookup Services verwendet wurden.

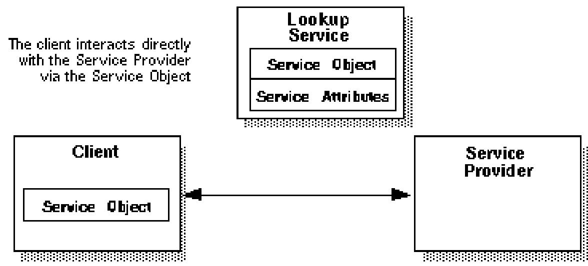

Abbildung 3.21: invoking a Service aus [Mic99d]

Der letzte Schritt ist, den Service zu starten, siehe dazu auch Abbildung 3.21. Der Code wird in den Client geladen. Er kann ein proprietäres Protokoll zwischen Client und dem ursprünglichen Service Provider implementieren. Unterschiedliche Implementierungen des selben Service Interfaces können völlig verschiedene Interaktionsprotokolle verwenden.

Weiterhin ist es möglich ein User Interface in Form eines Java Applets über den Lookup Service verbreiten zu lassen.

Kommunikationsmodell (nach [Mic99c])

JINI-KM.tex

Das Jini-System ist nur eines von vielen Systemen der Welt des sogenannten Plug-and-Participate.

Andere Technologien sind zum Beispiel Bluetooth (IBM Corporation, Intel Corporation, Nokia Corporation, Telefon AB L.M. Ericsson and Toshiba Corporation.), JetSend (Hewlett-Packard.) und HAVi (GUNDIG A.G., Hitachi, Ltd., Matsushita Electric Industrial Co., Ltd. (Panasonic), Philips Electronics N.V., Sharp Corporation, Sony Corporation, Thomson Multimedia S.A. und Toshiba Corporation.)

Das Beispiel in Abbildung 3.22 zeigt, wie Jini mit anderen Systemen zusammenarbeiten soll. Als Beispiel wurde die Kooperation zwischen einer Kamera und einem Drucker gewählt.

Auf der linken Seite ist ein Device dargestellt, welches einen Service nutzen soll. In diesem speziellen Beispiel handelt es sich um eine Kamera, die einen Drucker nutzen soll um das aufgenommene Bild auszudrucken.

In der Mitte ist das Device, welches den Service - in diesem Fall Drucken - anbietet, dargestellt.

Normaler Weise wird der Service lokal ausgeführt. Manchmal allerdings, muß der Service mit einem externen Device kommunizieren. Dieses Externe Device kann zum Beispiel ein

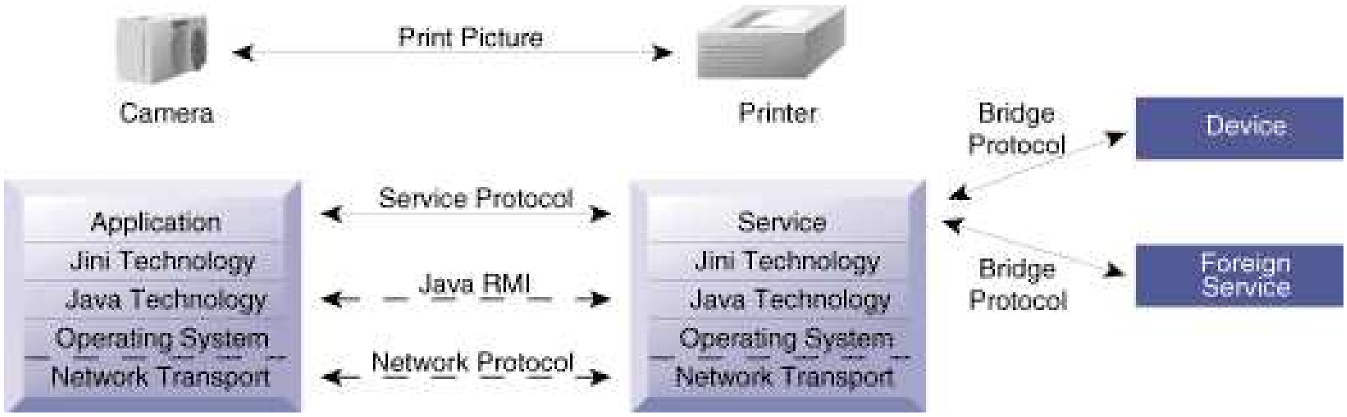

Abbildung 3.22: Jini: Interoperability aus [Mic99c]

Drucker sein sein, der noch kein Java unterstützt oder ein Device, welches die Jini Technologie nicht unterstützt. Dieser Fall ist auf der rechten Seite der Abbildung dargestellt. Die Pfeile beschreiben die unterschiedlichen Ebenen der benötigten Kommunikationen.

Im ersten Schritt, lokalisiert die Kamera den Druck Service indem sie Jini Technologie nutzt, welche die *RMI (Java Remote Method Invocation)* verwendet, welche wiederum ein beliebiges Netzwerk Protokoll, das vom lokalen Betriebssystem unterstützt wird, verwendet.

Die Jini Technologie verlangt keinerlei spezielles Betriebssystem oder speziellen Netzwerktyp. Die Kamera kann genauso gut über Infrarot oder Funk wie über ein physikalisches Kabel angebunden sein. Die meisten Netz Spezifikationen sind für spezielle Netzwerktypen entwickelt, z.B. TCP/IP für Ethernet, IrDA für Infrarot, IEEE 1394 (Firewire)... Sie alle unterstützen die Jini Technologie. Dementsprechend unterstützen auch alle Betriebssysteme die Funktionen, die für RMI und somit wiederum für Jini benötigt werden.

Nachdem die Kamera jetzt den Druck Service lokalisiert hat um das Bild zu drucken, läd und startet sie den Java-Code, den der Druck Service ihr anbietet (In diesem Beispiel wird ohne Lookup-Service gearbeitet). Dieser Code verwendet das darunterliegende Netzwerk und eventuell die RMI Technologie um das Druck-Protokoll, welches zum Übertragen und Drucken des Bildes benötigt wird, zu implementieren.

Das Service Protokoll, welches eine Applikation und ein Service zur Kommunikation untereinander verwendet, kann ein bereits existierendes Protokoll sein oder aber auch ein neues (proprietäres) Protokoll, welches diejenige Firma bestimmt, die den Service implementiert.

Einige der 'emerging network technologies' basieren auf neuen Service Protokollen, welche deutlich intelligenter und flexibler sind, als bereits bestehende Protokolle. Diese können ebenfalls sehr gut mit der Jini-Technologie zusammen arbeiten. Einige dieser Technologien definieren ihre eigenen Protokolle um Devices zu lokalisieren und mit ihnen zu kommunizieren. Für sie muß der Netzwerkservice als Bridge fungieren. Er muß das Protokoll der Anfrage in das Protokoll umsetzen, welches im anderen Netzwerk verwendet wird und die Anfrage an das Device weiterleiten. Dies setzt voraus, daß der Service auf einem Device läuft, welches zusätzlich mit dem anderen Netzwerk verbunden ist.

Stand der Technik

Sun macht den Jini technology source-Code unter dem *SCSL(Community Source Code License)* Modell frei verfügbar ([Mic99a]).

Zudem ist auf dem Webserver von Sun bereits eine Version der Jini-Plattform erhältlich.

Diese Jini Implementierung ist ein eigenständiges, kleines Management-System. SUN arbeitet derzeit in Kooperation mit anderen Herstellern an ersten Jini-fähigen Haushaltsgeräten. Es ist also damit zu rechnen, daß dieses Protokoll in der Zukunft vermehrt Verwendung findet.

Kapitel 4

Abstrakte Dienstbeschreibung eines Management-Systems

dienstbeschreibung.tex

Kapitelüberblick:

- In diesem Kapitel soll dargestellt werden, wie ein Managementsystem allgemein aussehen kann. Dabei werden Vorstellungen entwickelt, wie sich Policies und allgemeine Managementregeln in einem allgemeinen und offenen System integrieren lassen. Dabei werden Abläufe von Authentifizierung, Dienstvermittlung und Abrechnung unter Beachtung von Policies dargestellt.
- Es werden alle nötigen Abläufe innerhalb eines Managementsystems, sowie ihre benötigten Funktionen dargestellt.
- Ziel des Kapitels ist es auf Kapitel 5 vorzubereiten, welches sich mit der konkreten Umsetzung und Integration der verschiedenen Managementprotokolle in einer Plattform befaßt.

Nachdem die zur Verfügung stehenden Protokolle und Standards beschrieben wurden, stellt sich nun die Frage ob und inwieweit man sie gleichzeitig nutzen kann.

Warum will man alle diese Protokolle miteinander verwenden? Kann man nicht einfach sagen: 'Wir verwenden nur Jini.' oder 'Wir verwenden nur SLP.'? Man kann. Wenn man sich aber die praktische Umsetzung dieser Idee betrachtet, stellt man folgendes fest: Eine solche Festlegung würde voraussetzen, daß man vom Kunden erwartet, daß dieser die angegebene spezielle Plattform verwendet und installiert hat. Bei der Vielfalt der inzwischen bekannten Systeme ist es sehr schnell möglich, daß ein Kunde genau eines der anderen Systeme verwendet und installiert hat. Möglicher Weise, weil dieses Protokoll von der Unterkunft des Kongresses von vor drei Monaten verwendet wurde. Jedes einzelne Protokoll ist in seiner Umsetzung komplex. Deshalb kann man von einem Kunden nicht erwarten, daß er für einen dreitägigen Hotelaufenthalt das komplette Managementsystem auf seinem Nomadischen System austauscht.

Um also optimale Verfügbarkeit zu bieten, muß man mehrere verbreitete Systeme unterstützen.

Dabei stellt sich die Frage, wie man mehrere, in sich abgeschlossene Protokolle oder gar Plattformen, in einem großen System zusammenfügen kann.

Deswegen soll hier in einem ersten Schritt analysiert werden, wie ein Managementsystem aufgebaut ist und arbeitet. Nur dann kann man festlegen, an welchen Stellen man verschiedene Protokolle verwenden kann und inwieweit es möglich ist, alle Protokolle parallel zu verwenden oder bei Bedarf gegeneinander auszutauschen.

Hat man einen Überblick über die Funktionsweise des Managementsystems und weiß man, wo es möglich wäre externe Protokolle einzubinden, ist eine grundlegende Richtung für ein großes, umfassendes Managementsystem entstanden.

Mit diesem Wissen kann man Details designen und implementieren.

4.1 Begriffsklärung

dienstbeschreibung-allgemeines.tex

Nutzer Ein *Nutzer* bezeichnet in der Regel einen Menschen. Der Nutzer arbeitet an einem Computer und *nutzt* mit diesem Computer Dienste, die ihm ein Computernetz anbietet. Häufig wird das Wort *Nutzer* für 'Computer des Nutzers' verwendet.

Endsystem Ein Endsystem oder Endgerät ist ein Computer oder ein anderes Peripheriegerät, an welchem keine weiteren Geräte angeschlossen sind. Beispiele für Endsysteme sind:

- Drucker
- Geldautomat
- Surfstation
- Lautsprecher
- Kamera

Dagegen sind Router, Switches und Hubs keine Endgeräte in diesem Sinne.

(Computer-)Netzwerk Unter einem Netz, oder Netzwerk oder Computernetzwerk versteht man eine Anzahl Computer, die miteinander verbunden sind. In einem Netzwerk hat jeder Teilnehmer theoretisch die Möglichkeit jeden beliebigen anderen Teilnehmer über dieses Netzwerk zu erreichen und mit ihm zu kommunizieren.

Service Ein Service wird oft auch als *Dienst* oder Dienstleistung bezeichnet. Er wird von einem oder mehreren Servern erbracht und von einem Client genutzt, wobei in diesem Moment auch ein anderer Service als Client fungieren kann. Services sind beispielsweise:

- Email-Service
- News-Service
- Web-Service

- Druck-Service
- Fax-Service

Client Ein Client ist ein Computer oder Programm, welcher eine Dienstleistung in Anspruch nimmt. Manchmal wird auch ein Nutzer als Client bezeichnet oder ein Service, der einen anderen Service nutzt.

Server Ein Server ist ein Computer oder Computerprogramm, welches eine oder mehrere Dienstleistungen anbietet. Das Wort *Server* ist doppelt belegt:

- Zum Einen ist ein Server ein Computer, der Services anbietet. Dabei kann es sich um mehrere Services gleichzeitig handeln.
- Zum anderen wird aber auch der Erbringer eines einzelnen, expliziten Services als Server bezeichnet.Um Verwechslungen zu vermeiden, wird dieser Server in der vorliegenden Arbeit auch als Service-Server bezeichnet.

Nomadisches System Unter einem *Nomadischen System* versteht man in der Regel ein Endgerät, welches innerhalb eines Netzwerkes 'wandert' oder temporär entfernt oder hinzugefügt wird. Beispiele für Nomadische Systeme sind:

- Notebooks
- Arbeitsplatzrechner, die den Ort wechseln, wenn der dazugehörige Mitarbeiter den Ort (z.B. neue Abteilung) wechselt
- portable Drucker
- Vorführsysteme, die in verschiedenen Schulungsräumen verwendet werden.
- Telefone, welche das lokale Netzwerk nutzen (IP-Telefon)

Managementsystem Ein Managementsystem ist ein Verbund von Servern und Services. Seine Aufgaben bestehen darin, Administrations-Aufgaben gegenüber der angebotenen Services und der Nutzer dieser Services zu übernehmen, bzw. an einer zentralen Stelle zu konzentrieren.

4.2 Das Managementsystem und seine Hauptklassen im Überblick

dienstbeschreibung-hauptklassen.tex

Abbildung 4.1 zeigt einen Überblick über die Objekte des Managementsystems und in welcher Beziehung sie zueinander stehen. Die meisten Klassen, die im Managementsystem miteinander verknüpft werden erben direkt von der Klasse Top, die hier nicht weiter ausgeführt ist.

Die Startklasse **Managementsystem** erbt von der Klasse Top.

Sie enthält lediglich ein Attribut ID, welches es erlaubt verschiedene Instanzen dieses Managementsystems parallel zu verwenden.

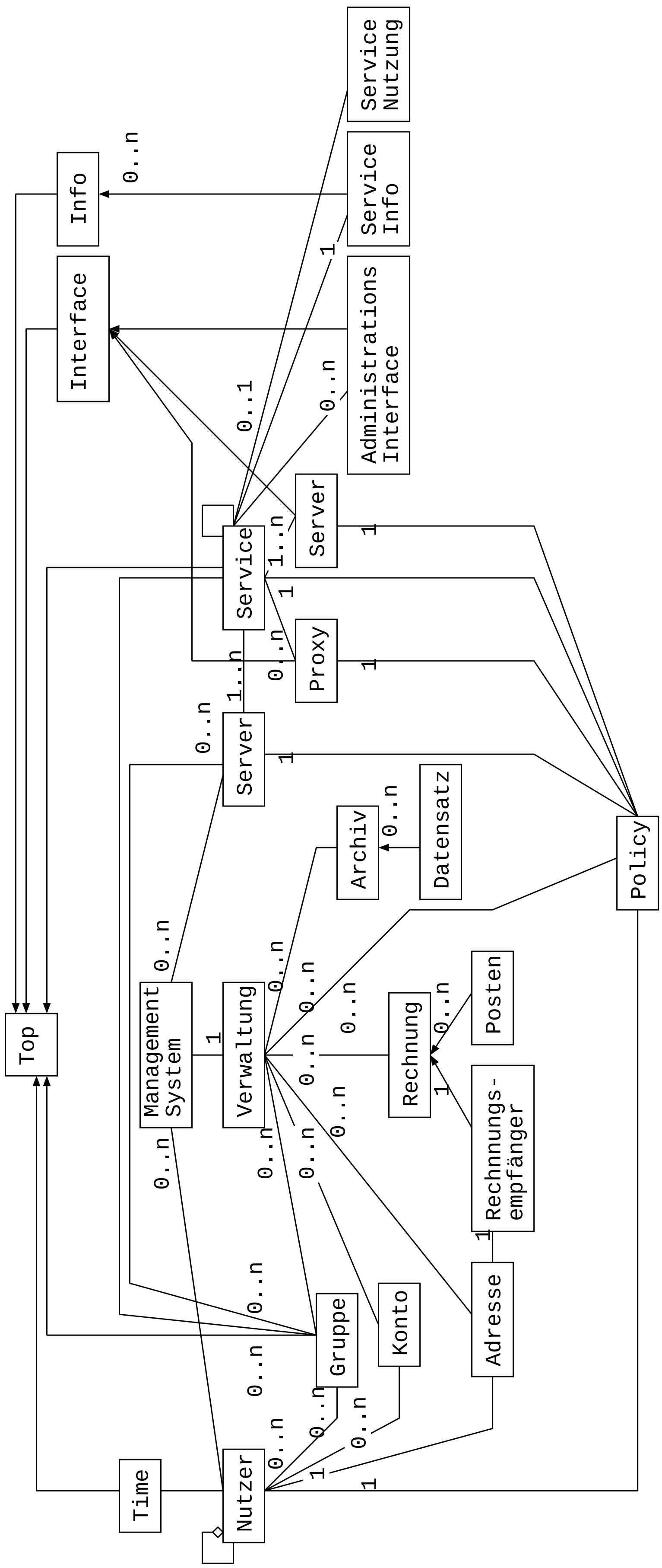

Abbildung 4.1: Managementsystem - Überblick

Direkt mit der Klasse **Managementsystem** assoziiert, sind die Klassen **Nutzer**, **Verwaltung** und **Server**. Nutzer

Nutzer: Die Klasse **Nutzer** (Abb 4.2) enthält keine eigenen Unterklassen.

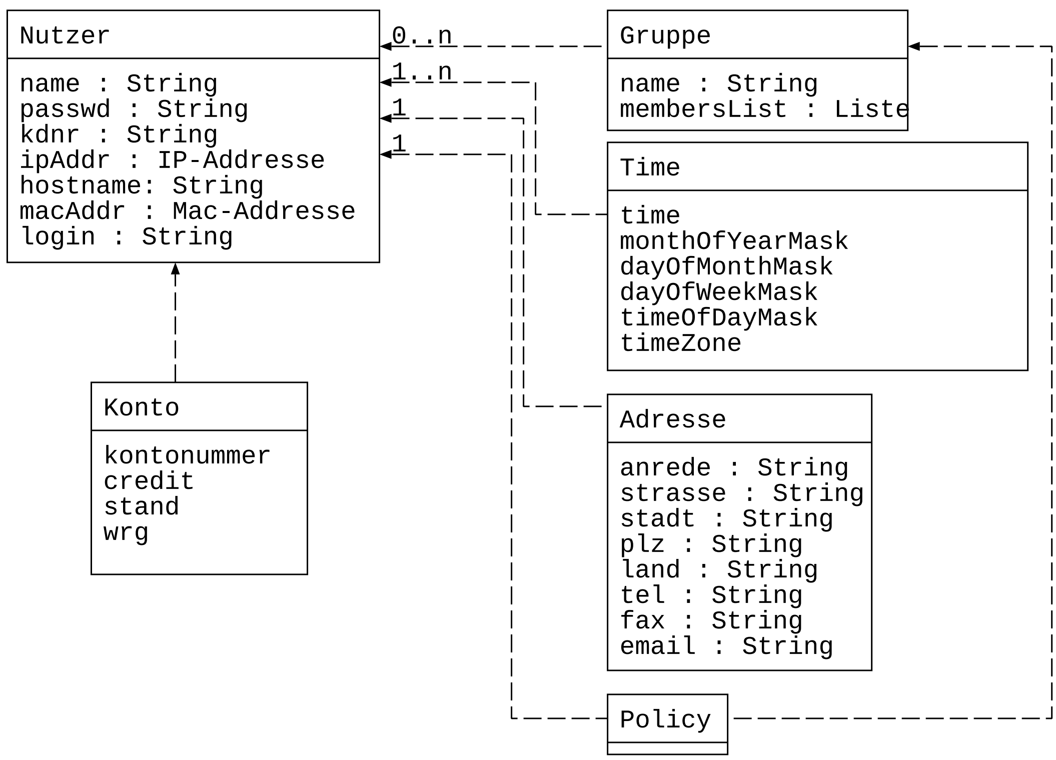

Abbildung 4.2: Klasse Nutzer

Ihre Attribute beschreiben einen einzelnen Nutzer. Dabei werden sowohl Informationen über den menschlichen Nutzer als auch Informationen über das Nomadische System (Notebook), welches dieser Nutzer nutzt, hinterlegt:

name enthält den Namen des Nutzers, dieser muß nicht eindeutig sein

login enthält den eindeutigen Loginnamen des Nutzers, den er benötigt, wenn er sich im Managementsystem authentifizieren möchte.

passwd enthält das Passwort des Nutzers, welches benötigt wird, um den Nutzer zu authentifizieren.

kdnr ist ein eindeutiger String

ipAddr enthält die IP-Adresse, die das Nomadische System des Kunden erhalten soll (sofern diese festgelegt wurde)

hostname enthält den vom Kunden für sein Nomadisches System gewünschten Hostnamen

macAddr enthält die Mac-Adresse der verwendeten Ethernetkarte im Nomadischen System. Diese Angabe dient zur zusätzlichen Authentifizierung.

Der Klasse **Nutzer** werden Instanzen der Klassen **Konto**, **Adresse**, **Policy** und **Time** zugeordnet.

Jedem Nutzer wird ein eigenes 'virtuelles' **Konto** zur Verfügung gestellt. Dieses Konto kann für Abrechnungsdienste verwendet werden. Konto

kontonummer enthält die eindeutige Bezeichnung dieses Kontos

credit enthält einen Vermerk über den Creditrahmen, den der Kunde mit diesem Konto hat

stand enthält den aktuellen Kontostand

wrg enthlt die Information in welcher Währung die angegebenen Beträge sind Adresse

Von jedem Nutzer wird zur eindeutigen Identifizierung in der realen Welt eine Adresse erfaßt. Zu diesem Zweck gibt es die allgemeine Klasse **Adresse**. Sie enthält folgende Attribute:

anrede Wie der Kunde anzureden ist, sowie sein vollständiger Name; Herr, Frau, Titelangaben

strasse Straße der Postanschrift

stadt Stadt der Postanschrift

plz Postleitzahl der Stadt

land Land, in dem sich die Postadresse befindet.

tel enthält die Telefonnummer des Kunden

fax enthält die Faxnummer des Kunden

email enthält die Emailadresse des Kunden Policy

Um Sondervereinbarungen mit dem Kunden festhalten zu können, gibt es die Klasse **Policy**. Jedem einzelnen Nutzer muß eine individuelle Policy zugeordnet werden können. In ihr werden Sondervereinbarungen bezüglich Zugriffsrechte, Abrechnungsmodalitäten und Sonderbehandlungen (Quality Of Service) festgehalten. Sie wird später detaillierter beschrieben (siehe Seite 87). Time

Die Klasse **Time** bestimmt Zeiträume in denen die Konfigurationsdaten dieses entsprechenden Nutzers gültig sein sollen. Die Klasse **Time** ist eine allgemein verwendete Klasse. Sie wurde nach dem Vorbild der Klasse PolicyConditionTime aus dem Policy-Modell (Seite 24) entwickelt und leitet sich von der Klasse **Top** ab. Sie enthält die folgenden Attribute:

time enthält eine Start- und eine Endzeitangabe der Form: yyyymmddhhmmss; beide Angaben werden mit ':' voneinander getrennt.

monthOfYearMask enthält die Angabe, in welchen Monaten des Jahres diese Angaben Gültigkeit haben sollen. Dabei wird eine Zeichenkette aus 12 Zeichen verwendet, wobei jedes Einzelzeichen einen Monat darstellt. Soll die Regel an diesem Monat gelten, wird diese Stelle mit 1 besetzt, wenn nicht, dann mit 0.

dayOfMonthMask enthält die Angabe, in welchen Tagen des Monats diese Angaben Gültigkeit haben sollen. Dabei wird eine Zeichenkette aus 31 Zeichen verwendet, wobei jedes Einzelzeichen einen Tag darstellt. Soll die Regel an diesem Tag des Monats gelten, wird diese Stelle mit 1 besetzt, wenn nicht, dann mit 0.

dayOfWeekMask enthält die Angabe, an welchen Wochentagen diese Angaben Gültigkeit haben sollen. Dabei wird eine Zeichenkette aus 8 Zeichen verwendet, wobei jedes Einzelzeichen einen Tag darstellt. Soll die Regel an diesem Tag gelten, wird diese Stelle mit 1 besetzt, wenn nicht, dann mit 0.

timeOfDayMask enthält die Angabe, zu welchen Stunden eines Tages diese Angaben Gültigkeit haben sollen. Diese wird durch eine Start- und Endzeit angegeben, die im Format: hhmmss abgelegt wird; beide Zeitangaben werden mit ':' voneinander getrennt.

timeZone enthält eine Angabe, auf welche Zeitzone sich die Uhrzeitangaben beziehen.

Verwaltung: Die Klasse **Verwaltung** dient, wie ihr Name schon andeutet, Verwaltungszwecken. Hier finden sich alle Klassen, die zur Koordination, Organisation oder Kommunikation mit der Außenwelt benötigt werden. Die Verwaltung hat den Überblick über alle Gruppen, die im Managementsystem angelegt wurden. Verwaltung

Eine Übersicht über die Klasse Verwaltung und Ihre Assoziationen findet sich in Abbildung 4.3. Group

Die Klasse **Group** leitet sich von der Klasse Top ab und ist eine Container-Klasse. In ihr kann man verschiedene Instanzen von Nutzern, Policies, Services oder wieder Gruppen zusammenfassen. Dementsprechend sind auch ihre Attribute ausgelegt:

name ist ein eindeutiger String, der diese Gruppe benennt.

membersList ist eine Liste der Instanzen, die mit dieser Gruppe zusammengefaßt werden.

type ist eine Angabe, von welchem Typ (Nutzer, Policy, Service) die einzelnen Gruppenmitglieder sind. Es ist nicht vorgesehen, Instanzen verschiedenen Types in einer Gruppe zusammenzufassen.

Die Klasse Verwaltung enthält Verweise auf alle Instanzen von **Konto**. Dies dient der allgemeinen Verwaltung nach außen hin. Man gewinnt durch diesen zusätzlichen Aufwand die Möglichkeit, einen Überblick über alle existierenden Konten zu bekommen, diese nach bestimmten Kriterien (Rechnungsempfänger, Nutzer, Wert) zu sortieren und zu analysieren. es handelt sich um einen Vorteil für die Buchhaltung des Unternehmens, welches das Managementsystem betreibt. Eine Beschreibung der Klasse **Konto** wurde bereits auf Seite 74 gegeben.

Ebenfalls zu Verwaltungszwecken werden unter der Klasse Verwaltung alle Instanzen von **Adressen** angelegt. Auch diese Klasse wurde auf Seite 74 bereits dargestellt.

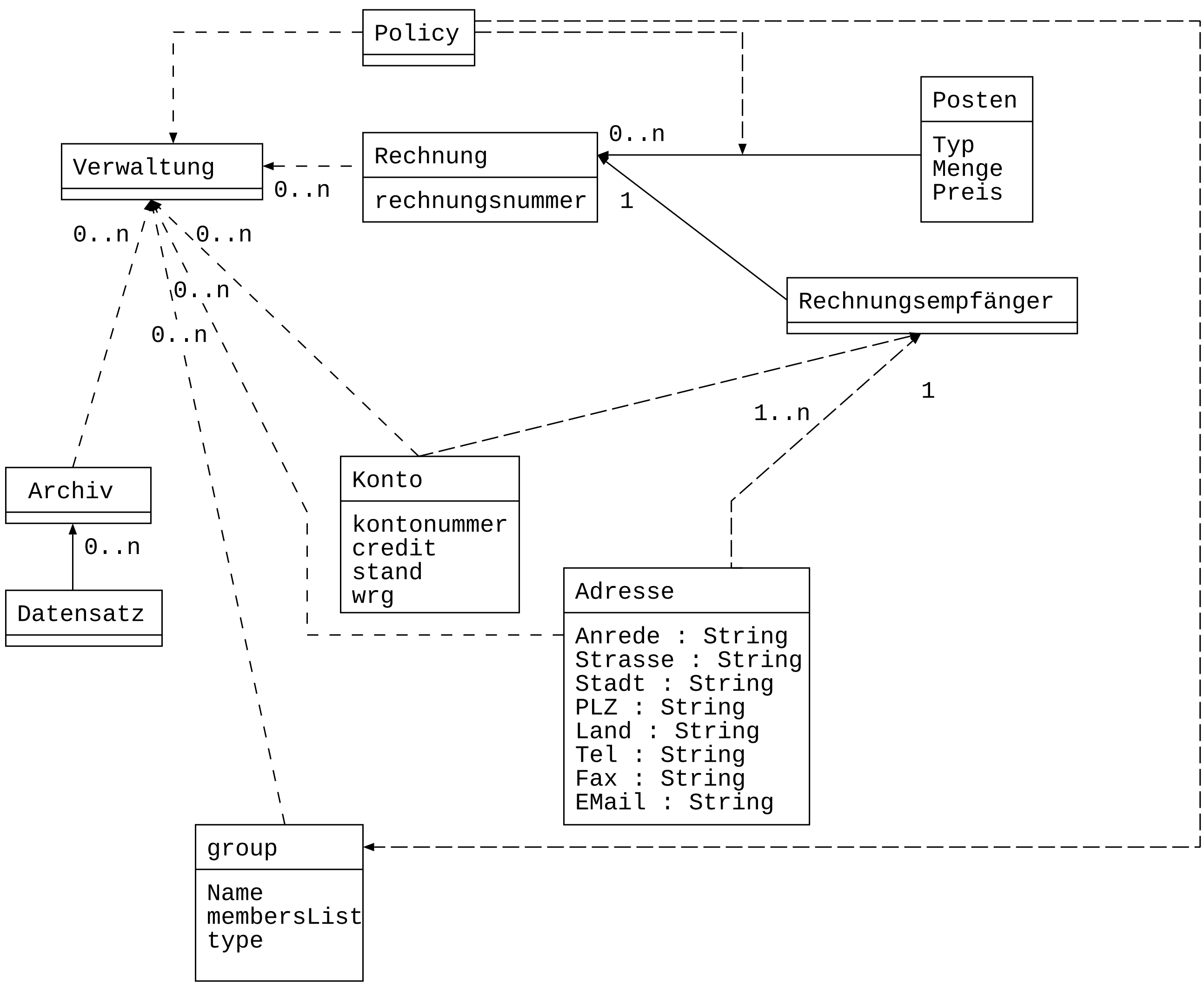

Abbildung 4.3: Klasse: Verwaltung

Ebenfalls von Top leitet sich die Klasse **Rechnung** ab. Sie besitzt das Attribut Rechnungsnummer und die beiden Unterklassen **Posten** und **Rechnungsempfänger**. Posten

Die Klasse **Posten** hat folgende Attribute:

typ enthält eine Beschreibung des entsprechenden Postens, zum Beispiel: Telefongebühren

menge enthält die Anzahl Einheiten, die mit diesem Abrechnungsposten abgerechnet werden. Eine solche Einheit ist individuell zum Postentyp zu sehen, zum Beispiel: Minuten, Megabyte oder PapierseitenA4.

Preis enthält den Preis, den diese Menge von dem entsprechenden Typ ergibt.

Bei der Rechnungsstellung spielt die Policy wieder eine Rolle, da hier Sondervereinbarungen, die auch Preise (Rabatte, Freieinheiten) betreffen entsprechend angerechnet werden müssen und in den einzelnen Rechnungsposten verrechnet werden.

Jede Rechnung hat einen Rechnungsempfänger. Dieser muß nicht zwingend identisch dem Nutzer sein. Beispiel: Die Firma übernimmt die gesamten Hotelkosten eines ihrer Angestellten. Dann ist der Angestellte der Nutzer, aber die Firma der Rechnungsempfänger. Aus diesem Grund wird dem Rechnungsempfänger ein eigenes **Konto** (Beschreibung siehe Seite 74) zugeordnet, sowie eine **adresse** (Beschreibung siehe Seite **??**).

Auch für alle Instanzen der Klasse **Policy** enthält die Klasse Verwaltung einen Vermerk. Dies ist für interne Verwaltungszwecke innerhalb des Managementsystems gedacht. Die Klasse Verwaltung hat somit die Möglichkeit jederzeit angeben zu können, welche Policies existieren. Archiv

Nicht weiter ausgeführt ist der Bereich **Archiv**. Auch er ist unterhalb der Klasse **Verwaltung** angesiedelt. Er hat als direkte Unterklasse eine Klasse **Datensatz**. Dieser nicht weiter ausgearbeitete Bereich ist für Backup-Zwecke innerhalb des Managementsystems gedacht. Er wird im Laufe der Arbeit auch nicht weiter ausgearbeitet werden. Server

Server: Unter der Klasse **Server** werden alle Instanzen aktiver Service-Erbringer zusammengefaßt. Dabei ist die obere Klasse Server eine Instanz, mit der eine Gruppe von Services assoziiert sind. (siehe dazu auch Abb. 4.4) Server

Die Klasse **Server** hat folgende Attribute:

name ist eine eindeutige Bezeichnung für diese Instanz der Klasse Server.

ServiceList ist eine Liste die Referenzen auf alle, mit diesem Server assoziierten Services enthält.

active enthält TRUE, wenn dieser Server aktiv ist. Ist er inaktiv enthält active FALSE, alle Services dieses Servers sind damit ebenfalls inaktiv (unabhängig von ihren eigenen Policies oder activ-Stati). Service

Ein **Service** ist ein Dienstleister von einem ganz speziellen Typ. Beispiele hierfür sind: Email, Fax, Netz-Zugang, DHCP, Accounting, Billing, Policy. Ein Service hat

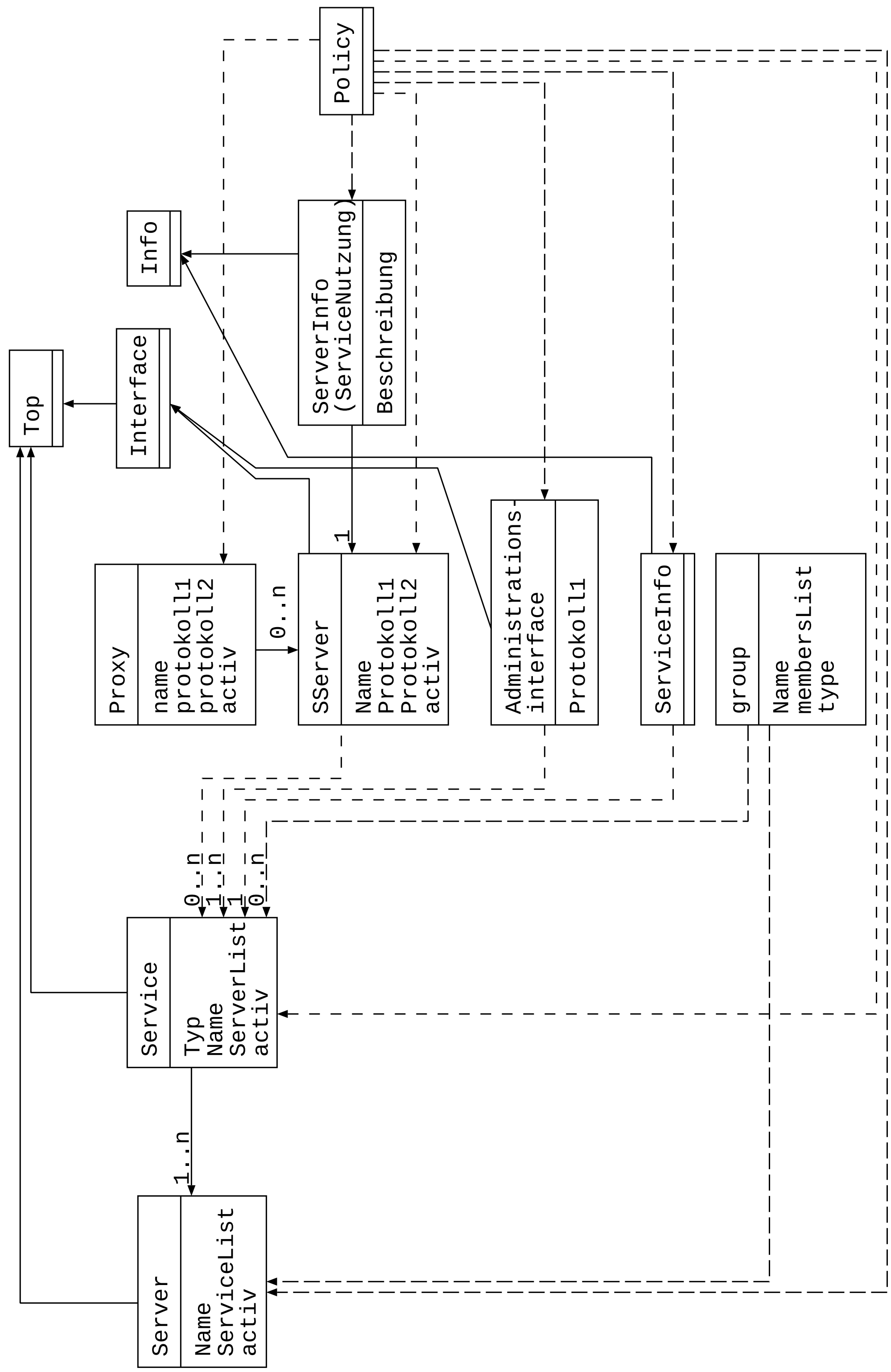

Abbildung 4.4: Klassen: Server und Service

die Aufgabe, Aktionen auf Anfrage zu steuern (Druck-Service) oder Daten auf Anfrage zu liefern (Email-Service) oder zu bestätigen (Authentisierer).

Jeder **Service** beschreibt einen erbringbaren Service. Erbracht wird dieser Service im Endeffekt von den Klassen **SServer** (wie Service-Server) und **Proxy**. **Service** hat folgende Attribute:

name enthält eine eindeutige Bezeichnug für die Instanz dieses Services.

type enthält eine Beschreibung, welchen Servicetyp dieser Service leistet, bzw. welchen Service-Typ seine Unterklassen erbringen. Diese Angabe des Types soll der Angabe eines MIME-Type entsprechen. ([FB96a], [FB96b], [Moo96], [FKP96], [FB96c])

serverList enthält eine Liste der diesem Service zuzuordnenden Server.

active entspricht dem Active-Attribut der Klasse Server. Enthält es den Wert FALSE sind alle mit dieser Instanz der Klasse Service assoziierten Server und Proxies deaktiviert.

Die Klassen Server und Service leiten ich direkt von der Oberklasse **Top** ab. SServer

Jedem Service ordnet man einen Server zu, der hier als **SServer** bezeichnet wurde, um Verwechslungen mit der Klasse Server zu vermeiden. Dieser **SServer** ist der eigentliche Erbringer der Dienstleistung. Er wird oft auch als Interface bezeichnet und deshalb, genauso wie die ebenfalls im Diagramm befindliche Klasse **Administrationsinterface** von einer Oberklasse Interface abgeleitet. Die Klasse **SServer** enthält folgende Attribute:

name enthält eine eindeutige Bezeichnung für diese Instanz der Klasse SServer.

protokoll1 enthält einen Vermerk auf das Protokoll mit welchem der Server mit dem Managementsystem (also den anderen Services und Klassen des Systems) kommunizieren soll. Diese Angabe kann entfallen, wenn intern ein festes Protokoll Verwendung findet.

protokoll2 beschreibt das extern gesprochene Protokoll, also dasjenige Protokoll, mit dem der eigentliche Service erbracht wird. Dieses kann unterschiedlich sein, da zum Beispiel Email-Service mit dem POP3-Protokoll oder dem SMTP-Protokoll erbracht werden kann.

activ entspricht dem activ-Attribut der Klassen Server und Service. Ist es mit dem Wert FALSE belegt, so ist dieser SServer inaktiv.

proxyList enthält eine Liste aller Proxies, die zu dieser Instanz der Klasse SServer gehören.

Proxy

Die Klasse **Proxy** ist eine direkte Unterklasse von **SServer**. Sie enthält die folgende Attribute:

name enthält einen eindeutigen Namen

protokoll1 beschreibt das interne Protokoll, also jenes Protokoll, mit dem der Proxy mit dem SServer kommuniziert

protokoll2 beschreibt das externe Protokoll, mit dem der Proxy mit dem Nomadischen System kommuniziert. Dies wird benötigt, wenn der Proxy Protokollumsetzungen machen soll.

activ gibt an, ob der Proxy aktiv ist.

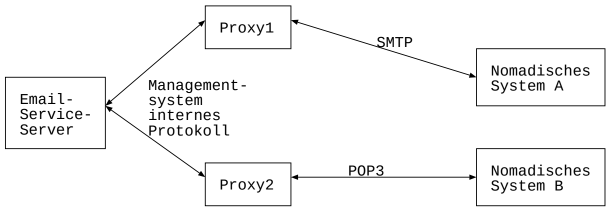

Abbildung 4.5: Einsatz von Proxies zur Protokollumsetzung

Es ist möglich in den Proxies Protokollumsetzungen durchzuführen (auch wenn dies nicht die ursprüngliche Aufgabe von Proxies ist). Dies bedeutet, daß man einem Service zwei (oder mehr) 'passende' Proxies 'vorschaltet' und man dann für POP3 und SNMP nur noch eine einzige Service-Implementierung benötigt. Die Übersetzung vom einen ins andere Protokoll würde dann im Proxy getätigt werden müssen. Siehe Abbildung: 4.5

Diese Idee läßt sich nur bei ähnlichen Protokollen durchführen. Bei grundlegend verschieden funktionierenden Protokollen ist dies nicht möglich.

Die Klasse **ServerInfo** ist eine Unterklasse der Klasse **Info**, welche sich von **Top** ableitet. ServerInfo

ServerInfo enthält eine menschenlesbare Beschreibung des Services. Die zu diesem Service gehörende **Policy** kann dort auch Hinweise auf ihre Regeln ablegen. ServiceInfo

Die Klasse **ServiceInfo** leitet sich, genauso wie die Klasse **ServerInfo** von der Oberklasse **Info** ab und enthält eine menschenlesbare Beschreibung der Klasse Service.

Damit Services, Server und SServer gruppierbar sind, kann die Klasse **Group** auf Instanzen dieser Klassen verweisen.

4.3 Abläufe

klassen-modelle-ablaeufe.tex

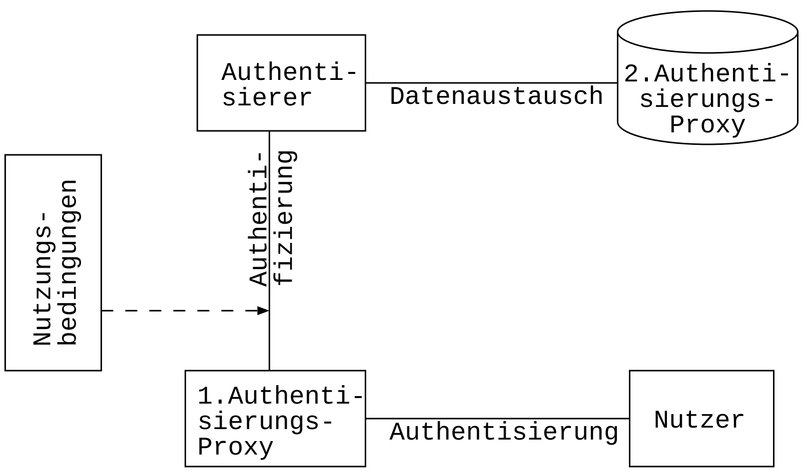

Abbildung 4.6: Informations- und Organisationsmodell der Authentisierung

4.3.1 Authentisierung

Will ein Nutzer mit Services im System kommunizieren und diese nutzen, so muß er sich authentisieren. Das bedeutet, er muß dem System mitteilen, daß er er selbst ist.

Organisationsmodell

Der Nutzer sendet seine Daten an einen 1. Authentisierungsproxy zum authentisieren. Dieser sendet sie an einen Authentisierer, welcher sich gegebenenfalls die Gegendaten zum Vergleich von einem 2. Authentisierungsproxy zusenden läßt. Anhand der Nutzerbedingungen, denen der entsprechende Nutzer unterliegt, bestimmt der Authentisierer, ob die Authentisierung von ihm authentifiziert werden kann oder nicht.

Informationsmodell

Eine Authentisierung geschieht in der Regel dadurch, daß zwischen Nutzer und Computer ein gemeinsames Geheimnis ausgetauscht wird. Dieses Geheimnis kann ein Passwort sein oder ein Fingerabdruck oder auch eine Stimmanalyse des bedienenden Menschen oder des Services, der die Funktion des Clients übernimmt. Dieses Geheimnis wird dann als Daten an den 1. Authentisierungsproxy gesendet.

Das System läßt sich nun von einer dritten Instanz bestätigen, daß die Daten, die es bekommen hat mit dem übereinstimmen, was der Nutzer zur Authentisierung benötigt. Diesen Vorgang (der Bestätigung) nennt man Authentifizierung (Abb 4.6). Die eigentliche Entscheidung, ob die Authentisierung erfolgreich war, trifft der Authentisierer. Er erhält die Daten, die dem Geheimnis entsprechen, welches der Nutzer abgesendet hat und er kann (falls nicht vorhanden) die Daten, die dem Geheimnis entsprechen, welches der Nutzer angeben sollte, von einer entfernten Stelle abfragen, welche er mittels dem 2. Authentisierungsproxy erreichen kann. Stimmen beide Informationen überein, hat der Nutzer das richtige Geheimnis abgegeben und in Verbindung mit der Nutzerbedingung, wird eine Antwort an den 1. Authentisierungsproxy gesendet, der diese dann an den Nutzer weitersendet.

Funktionsmodell

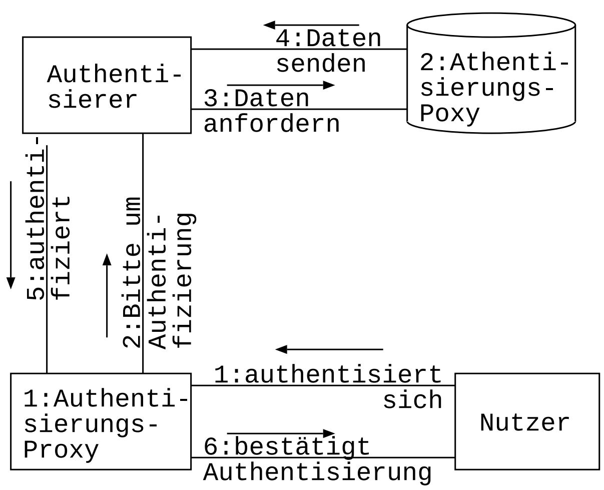

Abbildung 4.7: Funktionsmodell der Authentisierung

Abbildung 4.7 stellt diesen Ablauf im Detail dar: Der Nutzer wendet sich an einen Authentisierungsproxy und sendet seine Authentisierungsdaten dort hin (1). Der Proxy reicht diese Daten an einen Authentifizierer weiter, damit der sie authentifizieren kann (2). Der

Authentifizierer fordert nun die Daten aus einer zentralen Sammelstelle (hier als 2. Authentisierungsproxy bezeichnet) an (3+4). Stimmt das Ergebnis mit dem überein, was der Authentisierungsproxy weitergereicht hat, wird eine Authentifizierung bestätigt und diese Bestätigung an den 1. Authentisierungsproxy zurückübermittelt (5). Bei dieser Rückübermittlung wird die Policy (im Bild 4.7) des jeweiligen Nutzers mit einbezogen. Sie könnte eine Nutzung des Services durch eine entsprechende Regel unterbinden.

Der Authentisierungsproxy schickt im letzten Schritt (6) das Ergebnis der Authentifizierung zur Authentisierung des Nutzers an den Nutzer. Dieser erhält damit die Information, ob seine Authentisierung erfolgreich war.

4.3.2 Service-Vermittlung und Nutzung

Organisationsmodell

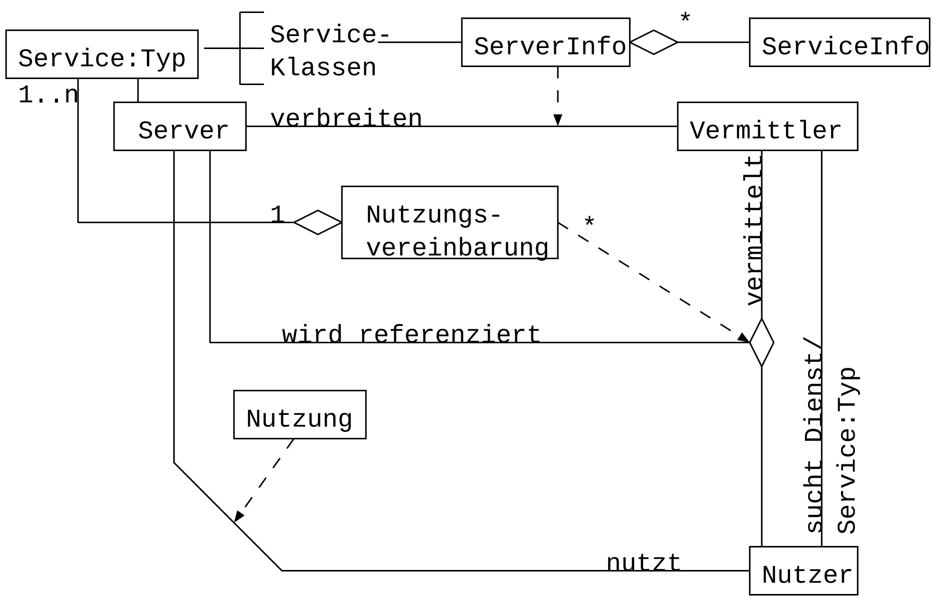

Abbildung 4.8: Information- und Organisationsmodell der Dienstvermittlung und Dienstnutzung

Bei der Dienstvermittlung muß der Nutzer sich einen Dienst vermitteln lassen (siehe Abb 4.8). Dies geschieht, indem er einen Dienstvermittler kontaktiert. In Ausnahmefällen (zum Beispiel bei sehr kleinen Netzen) ist es möglich, daß die Services sich sozusagen selbst vermitteln und regelmäßig ihre Informationen im Netzwerk verbreiten um somit den Nutzern direkt mitzuteilen, welche Eigenschaften sie haben.

Der Vermittler vermittelt lediglich die Informationen, die der Nutzer benötigt um eben diesen vermittelten Service-Server zu nutzen. Die eigentliche Service-Nutzung muß der Nutzer selbst vornehmen. Es ist allerdings auch die Möglichkeit vorgesehen, daß der Vermittler den Service informiert, daß er den Nutzer gerade an ihn vermittelt hat und somit ein eventuell unnötig langer neuer Verbindungsaufbau zum Server entfällt.

Informationsmodell

Die ServiceInfo und ServerInfo, die der zu vermittelnde Service verbreitet enthält Informationen über den Typ des angebotenen Services, Beschreibung des Service-Interfaces, Beschreibungen der räumlichen Lage der Service-Erbringung (z.B. 2.Stock Zimmer 234), sowie Preisinformationen und sonstige Sonderregelungen für diesen Service.

Ein Nutzer sendet an den Vermittler eine Anfrage nach einem Service eines speziellen Types und mit besonderen Eigenschaften. Solche besonderen Eigenschaften können Angaben über den Ort sein, an dem der Service erbracht werden soll oder Angaben über die Geschwindigkeit oder über Kosten, die anfallen würden. Beispiel für Zusatzeigenschaften zu einem Druck-Service: der Drucker soll im 2. Stock stehen, er soll Farbe ausdrucken können und pro ausgedruckte Seite nicht mehr als 1.00DM an Kosten verursachen.

Hat der Vermittler einen passenden Service zu diesen Anforderungen gefunden (dabei müssen nicht zwingend alle Angaben erfüllt sein), sendet er die Daten des entsprechenden Services an den Nutzer zurück. Dabei spielt wieder die Policy (als Nutzerbedingung bezeichnet) eine Rolle, da diese die endgültige Übermittlung der Daten verhindern kann, z.B. wenn der Nutzer diesen speziellen Service nicht nutzen darf.

Nun kann der Nutzer anhand dieser Daten vom Service-Server eine konkrete Diensterbringung verlangen. Der Service-Server überprüft selbst noch einmal anhand seiner Policy und der des Nutzers, ob der Nutzer wirklich berechtigt ist diesen Service zu nutzen (oder ob er auf gut Glück einen entsprechenden Dienstauftrag an den Service vergeben hat und hofft, daß dieser einfach ausgeführt wird).

Funktions-
modell

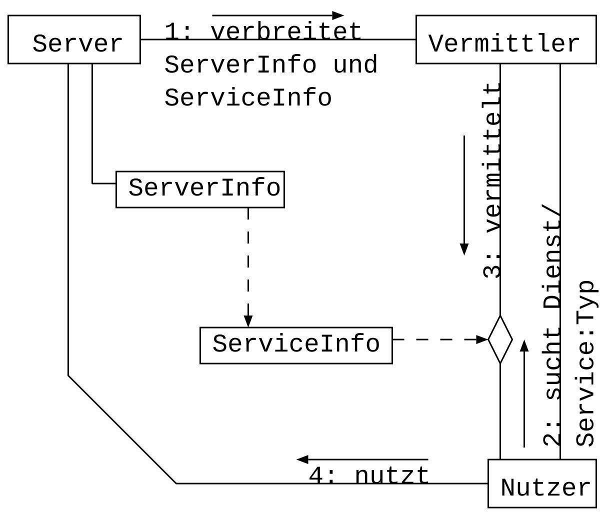

Abbildung 4.9: Funktionsmodell der Service-Vermittlung

Abbildung 4.9 stellt das Funktionsmodell der Service-Vermittlung grafisch dar.

Bei der Service-Vermittlung muß der zu vermittelnde Service sich in einem 1. Schritt bei den Vermittlern anmelden. Dies geschieht, indem er seine eigene ServiceInfo und ServerInfo verbreitet. Der Vermittler empfängt diese Information und kann nun den Service vermitteln. Im 2. Schritt muß ein potentieller Service-Nutzer einen Service suchen. Dazu schickt er eine entsprechende Anfrage an den Vermittler. Dieser sucht anhand seiner gesammelten ServiceInfos und ServerInfos den passendsten Service heraus und sendet die entsprechenden Daten an den Nutzer zurück, vermittelt also somit den Service.

Alternativ gibt es die Möglichkeit, daß der Vermittler die Anfrage des Nutzers gleich direkt an den passenden Server weitereicht. Der Service wird dann referenziert und der Nutzer muß keine neue Verbindung zum Service-Server aufbauen.

Schritt 4 ist dann die aktive Nutzung des Services durch den Nutzer.

4.3.3 Abrechnung

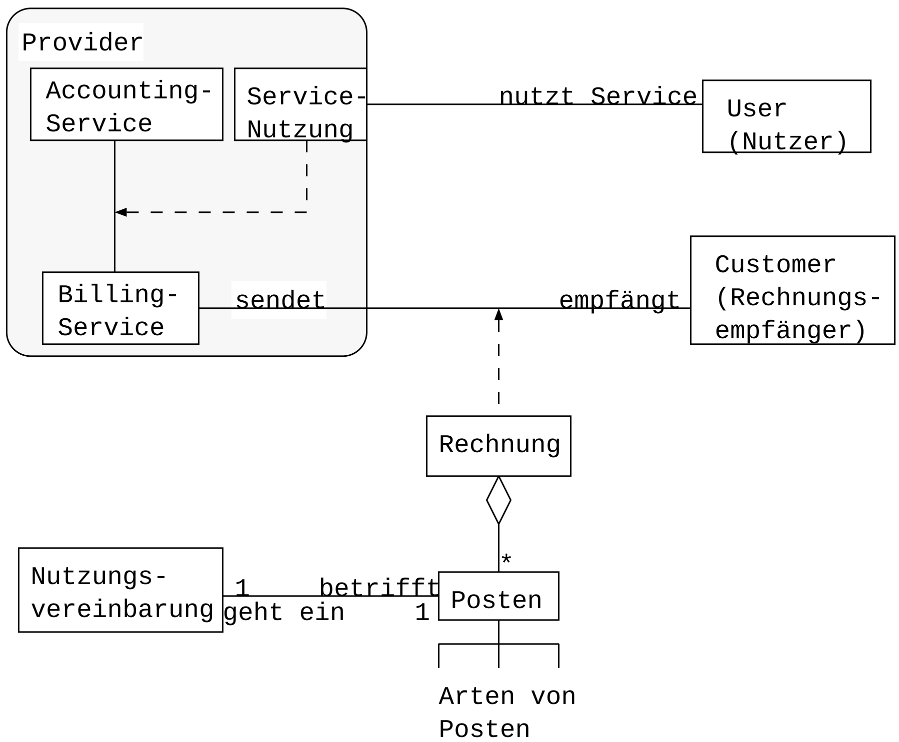

Abbildung 4.10: Informations- und Organisationsmodell der Abrechnung

Die Abbildung 4.10 stellt den Abrechnungsweg dar.

Organisationsmodell

Nutzt ein Nutzer einen Service, muß diese Nutzung registriert und abgerechnet werden. Dazu wird vor, während der Service-Nutzung oder danach - je nach Policy - der Verbrauch an einen Accounting-Service gemeldet. Dieser Accounting-Service reicht seine gesammelten Daten (z.B. einmal die Stunde) an einen Billing-Service weiter. Der produziert eine Rechnung, die dem Rechnungsempfänger zugestellt wird. Der Rechnungsempfänger muß dabei nicht notwendiger Weise mit dem Kostenverursacher (Nutzer) identisch sein.

Informationsmodell

Will ein Nutzer einen Service nutzen, so erhält er (je nach Policy) davor oder danach den Auftrag den abzurechnenden Betrag (Menge und Preis) der Service-Nutzung zu bestätigen. Tut er dies nicht, erhält er keine Dienstleistung oder das Problem muß an einen menschlichen Vermittler weitergereicht werden.

Durch diese Bestätigung erkennt der Nutzer an, daß er genau diese bestätigte Menge an Service nutzen möchte, bzw. genutzt hat. Dieser bestätigte Wert wird dann vom Accounting-Service an einen Billing-Service weitergereicht. Dieser sammelt gegebenenfalls mehrere solche Accountingdaten bevor er dann diese zu einer Rechnung für den Rechnungsempfänger des jeweiligen Dienstnutzers zusammenstellt.

Die Rechnung selbst setzt sich aus Posten zusammen, die aus den Daten des Billing-Services in Kombination mit den Nutzervereinbarungen (Policy) besteht. Diese Teil-Daten bilden einzelne Posten. Jeder unterschiedliche Tarif bildet beispielsweise einen eigenen Posten:

- 20 Telefoneinheiten zwischen 9 und 18 Uhr (Bussines-Tarif) zu je 12 Pfennige,
- 10 Telefoneinheiten zwischen 20 und 2 Uhr (Abend-Tarif) zu je 7 Pfennige,
- 10 Seiten Laserdruck A4 zwischen zwischen 5 und 9 Uhr (Schnell-Tarif) zu je 15 Pfennige,
- 2 mal zusätzlicher Zimmerservice zu je 80.00 DM,
- 2 Nächte Einzelzimmer zu je 250.00DM

Funktionsmodell

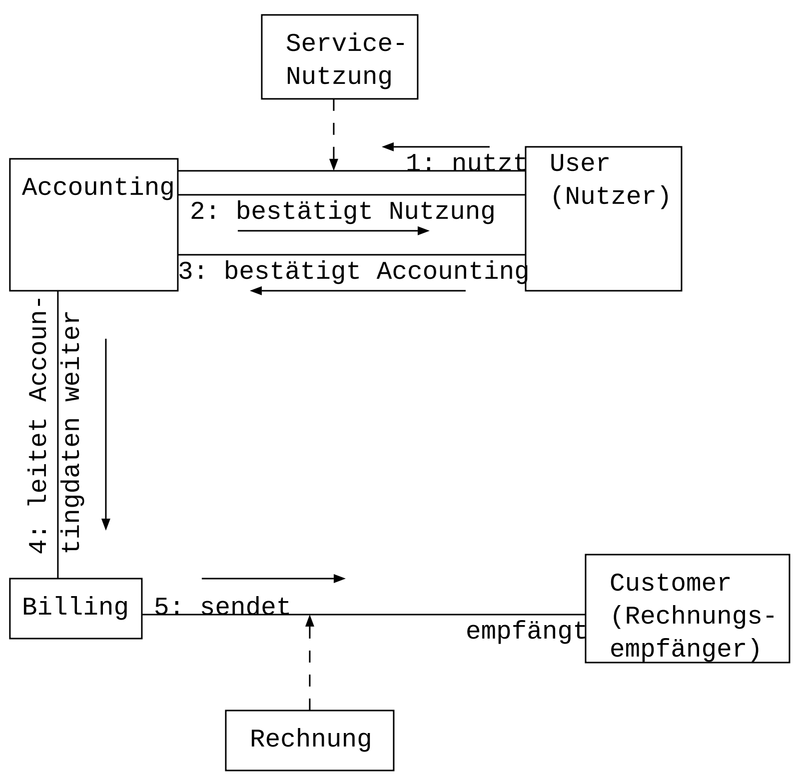

Abbildung 4.11: Funktionsmodell der Dienstabrechnung

Den genauen Ablauf einer solchen Abrechnung stellt Abbildung 4.11 dar.

Der Nutzer nutzt einen Service, bzw. will einen Service nutzen (Schritt 1). Von dieser Service-Nutzung wird der Accounting-Service informiert, der dem Nutzer den abzurechnenden Wert zum Bestätigen zusendet (Schritt 2). Der Nutzer bestätigt den angegebenen Betrag (Schritt 3). Der Accounting-Service leitet die vom Nutzer bestätigten Daten an einen Billing-Service weiter (Schritt 4). Dieser errechnet daraus eine Rechnung und sendet sie an den zugehörigen Rechnungsempfänger (Schritt 5). Der die Rechnung dann begleicht (nicht angegebener Schritt 6)

4.4 Policies im Managementsystem

Policies sind wichtig, wenn ein Server entscheiden (lassen) muß, ob er den angeforderten Service erbringen darf oder nicht. Ein Server kann aus folgenden Gründen die Erbringung eines Services verweigern:

- der Auftraggeber ist nicht berechtigt diesen Service zu nutzen
- der Auftraggeber will die anfallenden Kosten nicht tragen
- eine globale Policy verbietet die Erbringung dieses Services um diese Uhrzeit
- der Service kann nicht erbracht werden, weil der Server nicht mehr genug freie Ressourcen hat (Druckpapier verbraucht).
- die Größe des Auftrages widerspricht den Policy-Regeln dieses Services (eine Email darf nicht größer als 500 KiloByte sein.)
- der Server hat momentan nicht genug Kapazitäten diesen Auftrag abzuarbeiten (der Prozessor ist ausgelastet)

Policies können nicht nur Einschränkungen festlegen, sondern auch die Ausführung von Aktionen veranlassen. Mögliche Beispiele hierfür sind:

- Ist die Außenleitung zu mehr als 80% ausgelastet, muß eine weitere Leitung geschaltet werden
- Hat ein Nutzer drei mal sein Passwort falsch eingegeben wird der Account gesperrt und der Service-Leiter bekommt eine Nachricht.
- Trifft ein Fax für die Nummer des Sekretariats ein, wird es automatisch ausgedruckt und eine Email an die Sekretärin geschickt.

4.4.1 Policy-Typen

policy-typen.tex

In der Arbeit von [Die99] werden die einzelnen Policies in zwei Untergruppen gegliedert: dynamische Policies und statische Policies.

dynamische Policies Dynamische Policies werden durch das Eintreten eines passenden Ereignisses ausgelöst. Dabei können abhängig von Umgebungsparametern unterschiedliche Aktionen ausgelöst werden. Zum Beispiel können, wenn der Nutzer X sich im System anmeldet, abhängig davon, wie ausgelastet das System allgemein zu diesem Zeitpunkt ist, unterschiedliche Aktionen ausgelöst werden (wie z.B. der Aufbau einer zusätzlichen Außenanbindung).

statische Policies Statische Policies werden durch die Erfüllung vorgegebener Konditionen ausgelöst. Diese Konditionen entsprechen fest vorgegebenen Ereignissen. Treten diese ein, tritt die Policy-Regel in Kraft. Solche Ereignisse können zum Beispiel Uhrzeiten sein, nach denen erst bestimmte Ressourcen von bestimmten Nutzern genutzt, bzw. nicht mehr genutzt werden dürfen: z.B. zwischen 22 und 6 Uhr darf kein

Nadel-Drucker verwendet werden - Nachtruhe. Statische Policies lösen immer auf ein bestimmtes Ereignis hin (es ist 12:00 Uhr) eine feste Folge von Aktionen aus.

4.4.2 Policy en Detail

policy-en-detail.tex

Dieser Abschnitt geht genauer auf die Struktur der benötigten Policies ein. Dabei wird auch das Zusammenspiel von Policy und Abrechnung beleuchtet.

Das allgemeine Vorgehen bei der Nutzung von Policies wurde bereits auf Seite 23 im Zusammenhang mit dem Policy-Modell [ESWW99] dargestellt.

Das dort beschriebene Modell ist für den Einsatz in dem hier erstellten Managementsystem sehr gut geeignet. Da dieses Modell aber in seiner Gesamtheit viel umfangreicher ist, als wir es hier benötigen, wird es wie folgt gekürzt (Abb. 4.12):

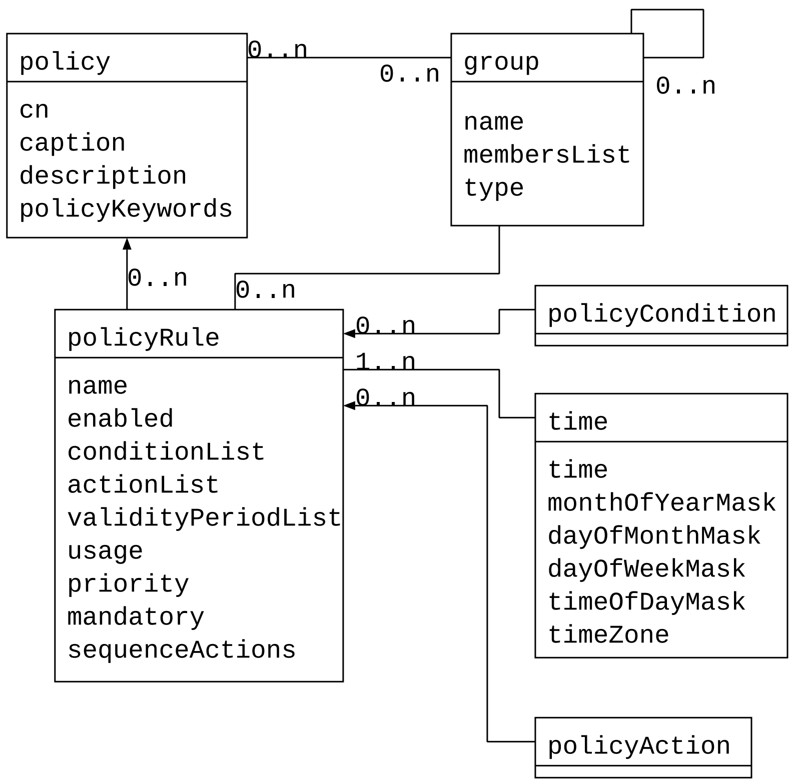

Abbildung 4.12: Policy-Klassenmodell für Managementsystem

Hierbei wird die Klasse **policyTimePeriodCondition** verallgemeinernd als **Time** bezeichnet und als direkte Unterklasse der Klasse **Verwaltung** betrachtet. Dies hat den Vorteil, daß sehr viele andere Klassen, die ebenfalls eine Zeitregelung benötigen auf diese Klasse zurückgreifen können. Das Gleiche gilt für die Klasse **policyGroup**. Sie wird in **group** umbenannt und ebenfalls direkt der Klasse **Verwaltung** unterstellt.

Der Zusammenhang zwischen Policy, Nutzung und Rechnung wird in Abb 4.13 verdeutlicht. Die Nutzung eines Dienstes kann von der Nutzungsvereinbarung (Policy) verschieden beeinflußt werden:

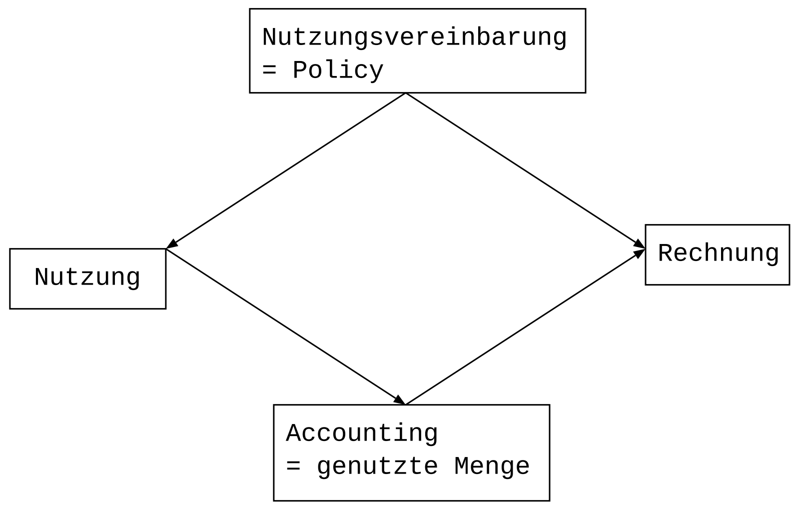

Abbildung 4.13: Zusammenhang zwischen Policy, Nutzung und Abrechnung

- die Nutzung wird verhindert
- die Nutzung wird beschleunigt (höhere Priorität)
- die Nutzung wird verzögert (niedere Priorität)
- die Nutzung wird normal abgearbeitet

Aus der Nutzung resultiert eine genutzte Menge, welche beim Accounting(Service) gesammelt wird und aus der dann eine Rechnung generiert wird. Dabei nimmt die Policy bei der Rechnungsgenerierung wieder Einfluß auf die jeweiligen Posten. Dies kann den Preis betreffen:

- bevorzugte Service-Erbringung ist teurer als
- niedriger privilegierte Service-Leistungen
- spezielle Zahlungsmodalitäten (Ratenzahlung, Abbuchung, Rechnung...) oder Preise für einen Kunden

Solche Besonderheiten können vom Kunden speziell im Voraus verhandelt werden und müssen am Ende bei der automatischen Abrechnung berücksichtigt werden.

Als konkretes Policy-Modell bietet sich hier das weiter oben beschriebene Modell von Elleson, Ed, Strassner, Weiss und Westerinen an (2.4.4 auf Seite 22).

Dieses Modell wird für unsere Zwecke hier etwas vereinfacht. Einige Klassen des Policy-Modells wurden bereits für andere Klassen dieses Managementsystemes verwendet. Dies betrifft insbesondere die Attribute der Klassen: **Group** (Seite 75), deren Attribute der Klasse **PolicyGroup** (Seite 25) entlehnt sind, sowie die Klasse **Time** (Seite 74), für deren Attribute die Klasse **policyTimePeriodCondition** (Seite 26) Vorbild war.

Abbildung 4.14 zeigt einen ausführlichen Überblick über die neuen Policy-Klassen.

Die Oberklasse ist die Klasse **Policy**. Mehrere Instanzen der Klasse Policy können in der Container-Klasse **Group** zu einer Gruppe zusammengefaßt werden.

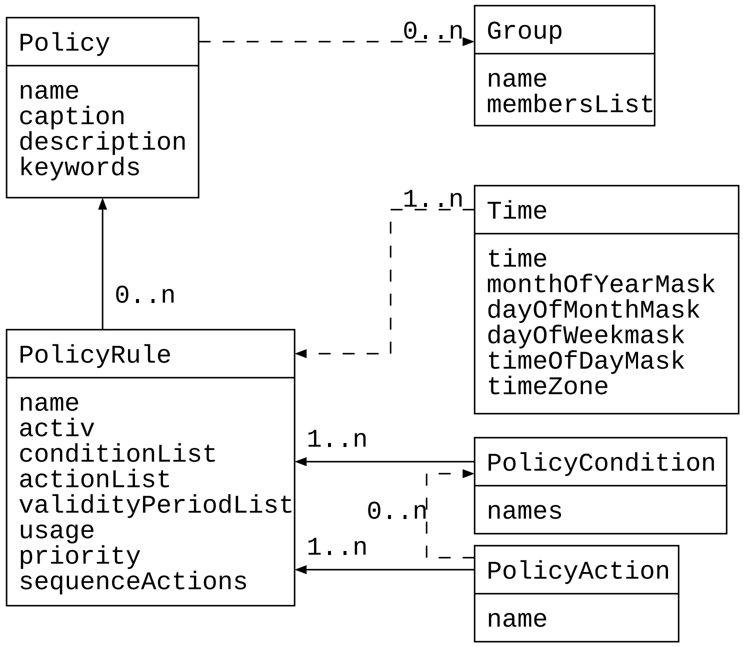

Abbildung 4.14: new-policy-modell.eps

Direkte Unterklasse der Klasse **Policy** ist die Klasse **PolicyRule**. Eine Policy kann mehrere Instanzen der Klasse PolicyRule haben. Die Klasse PolicyRule kann eine oder mehrere Assoziationen der Klasse **Time** haben.

Direkte Unterklassen zur Klasse PolicyRule sind die Klassen **PolicyCondition** und **PolicyAction**. Im Vergleich zum existierenden Schema von Strassner (2.4.4, S. 22) ist es hier vorgesehen, daß man eine PolicyAction mit einem konkreten PolicyCondition assoziiert und nicht mit einem PolicyRule (die Vererbung bleibt wie im Vorbildmodell). Policy

Attribute der Klasse **Policy**:

name enthält einen eindeutigen Namen dieser Klasseninstanz

caption enthält eine kurze Beschreibung der Klasse

description enthält eine ausführliche Beschreibung der Klasse

keywords enthält eine Liste von Schlüsselworten, die vom Administrator angegeben werden, um eine einfachere Zuweisung des Objekts an einen Client zu unterstützen.

Attribute der Klasse **PolicyRule**:

name enthält einen eindeutigen Namen dieser Klasseninstanz

activ gibt an, ob diese PolicyRule activ ist oder nicht.

conditionList enthält eine Liste aller Instanzen von **PolicyConditions**, die mit dieser Instanz der Klasse PolicyRule assoziiert werden.

actionList enthält eine Liste aller Instanzen von **PolicyActions**, die mit dieser Instanz der Klasse PolicyRule assoziiert werden.

validityPeriodList enthält eine Liste aller Instanzen von **Time**, die mit dieser Instanz der Klasse PolicyRule assoziiert werden.

usage ist eine menschenlesbare Angabe, wie diese Policy-Regel zu verwenden ist.

priority Gibt eine Priorität dieser PolicyRule an.

sequenceActions enthält die Angabe, wie die Regeln dieser PolicRule auszuführen sind: mandatory, reccomended, dontCare

Policy-Condition

Attribute der Klasse **PolicyCondition**:

name enthält einen eindeutigen Namen der Instanz dieser Klasse

Policy-Action

Attribute der Klasse **PolicyAction**:

name enthält einen eindeutigen Namen der Instanz dieser Klasse

Mit dieser Policy-Klasse sind alle nötigen Policies, die innerhalb eines Managementsystems benötigt werden, beschreibbar:

- man kann Policies benennen
- Policies sind Gruppierbar
- einzelnen PolicyRules kann man eine Zeit zuordnen, zu der sie gültig sein sollen.
- einzelnen PolicyRules kann man eine oder mehrere PolicyConditions angeben
- einzelnen PolicyConditions kann man eine oder mehrere PolicyActions zuweisen
- man kann einzelnen PolicyRules Prioritäten zuweisen
- man kann allgemein angeben in welcher Reihenfolge, die jeweiligen PolicyConditions der PolicyRules abgearbeitet werden sollen

4.4.3 Policy-Konflikte

policy-conflicts.tex

Ein Policy-Konflikt liegt vor, wenn mehrere Policy-Regeln auf ein Ereignis zutreffen, diese sich in ihrer Ausführung aber gegenseitig ausschließen würden.

Beispiele:

1. Die speziellen Regeln für Kunde A besagen, daß er jeder Zeit den Drucker X verwenden darf. Die allgemeinen Regeln besagen allerdings, daß der Drucker X nicht von 9-12 Uhr verwendet werden darf. Wenn Kunde A jetzt zwischen 9 und 12 Uhr den Drucker X verwenden will, erhält man einen Policy-Konflikt zwischen der Policy für Kunde A und der allgemeinen Policy.
2. Ein Ereignis hat nach einer allgemeinen Regel die Sperrung des kompletten Netzwerkes als Folge. Gleichzeitig löst dieses Ereignis noch eine andere Policy-Regel aus, nach der eine Nachricht über das Netz an einen Verantwortlichen gesendet werden soll. Beide Regeln stehen im Konflikt zueinander. Werden sie in beliebiger Reihenfolge ausgeführt, kann die Nachricht unter Umständen nicht mehr versandt werden.

Im ersten Beispiel hatte man einen direkten Konflikt zweier Policies miteinander (*Intra-Policy Conflict* [Die99]). Im zweiten Beispiel hat man einen indirekten Konflikt, der erst bei der Ausführung deutlich wird (*InterPolicy Conflict* [Die99]).

Für beide Typen von Konflikten muß man eine Regel schaffen, wie sie zu behandeln sind. So wäre es zum Beispiel möglich, einzelne Policies höher zu priorisieren als andere, so daß die höher priorisierten Policy-Regeln Vorrang vor den niedrigeren haben.

Zur Konfliktlösung bei Policies gibt es mehrere Arbeiten. Die hier aufgeführten Lösungsmöglichkeiten wurden von den Angaben in den Arbeiten [ESM99a], [ESWW99] und [CIM99] in der Regel bestätigt.

Priorität

Man kann den einzelnen Policy-Regeln Prioritäten geben. Dann werden im Falle eines Konflikts zuerst die höher priorisierten Regeln abgearbeitet und zum Schluß die niedrig priorisierten.

Wäre im Fall 1 die kundenspezifische Regel höher priorisiert, als die allgemeine Hausregel, könnte der Kunde A auch zur Sperrzeit den Drucker X verwenden.

Im Fall 2 würde die Regel, eine Nachricht zu versenden höher priorisiert sein, als das Sperren des Netzwerkes. Würde man dann das Netzwerk sperren, nachdem die Nachricht abgeschickt wurde, käme sie aus dem Netz heraus. Würde man keine Konfliktbehandlung durchführen, könnte es passieren, daß die Nachricht erst nach der Sperrung des Netzwerkes abgeschickt werden würde. Sie würde ihr Ziel nicht mehr erreichen.

Reihenfolge

Man kann festlegen, in welcher Reihenfolge die Aktionen der einzelnen policyRules auszuführen sind. So wäre es zum Beispiel denkbar, daß zuerst alle Administrator-Policies erfüllt würden, dann alle Nutzer-Policies und zum Schluß die globalen Policies und die der Services.

Dieser Punkt wäre für Fall 2 eine sinnvolle Lösung. Man gibt vorher an, in welcher Reihenfolge Aktionen von verschiedenen Policy-Regeln abzuarbeiten sind. Dazu muß man im Vorhinein wissen, welche Aktionen bei welchen Ereignissen zusammentreffen und sich gegenseitig behindern. Mit diesem Wissen kann man regeln, daß zuerst die Nachricht verschickt wird und dann das Netzwerk gesperrt.

Beide Möglichkeiten der Konfliktbehandlung werden ebenfalls in [ESWW99], [ESM99a] und [CIM99] aufgeführt und bestätigen damit, daß die an dieser Stelle erarbeiteten Möglichkeiten zur Konfliktlösung sinnvolle Ansätze sind.

4.4.4 Priorität und Reihenfolge

In der Praxis ist eine Kombination aus beiden Lösungsansätzen sinnvoll. Dabei erhält die höher priorisierte Policy-Reaktion Vorrang vor einer niedriger priorisierten. Sie wird also vorher ausgeführt. Zusätzlich erhält sie das Recht ausgeführt zu werden, auch wenn eine niedriger priorisierte Regel die Ausführung unterbindet.

Dies hätte im Fall 1 zur Folge, daß der Drucker trotz Sperre genutzt werden kann. Im Fall 2 würde die Nachricht vor der Sperrung des Netzwerkes verschickt werden. Sollte dann das auslösende Ereignis allerdings nochmals auftreten, würde die Nachricht aus Fall 2 zwar abgeschickt werden, aber das bereits gesperrte Netz nicht verlassen können. Dies ist kein großes Problem, da die Nachricht bei der ersten Sperrung bereits erfolgreich versendet werden konnte. Mit einer Sperrung des gesperrten Netzes erreicht man keinen neuen Zustand.

Die Dokumente [ESM99a], [ESWW99] und [CIM99] kommen zu dem selben Ergebnis, allerdings anhand anderer Beispielbetrachtungen.

4.5 Entwicklung eines Beschreibungs-Modells für Policies im Dienstmanagement

dienstbeschreibung-regelsyntax.tex

4.5.1 Begriffsklärung

Aus den Überlegungen über die Funktionsweise eines Managementsystems und der Vielfältigkeit von Policies, deren Konflikten und möglicher Konfliktbehandlung, kann man nun eine Beschreibung der Policy-Regeln entwickeln.

Diese Beschreibung soll alle wichtigen Informationen beinhalten, die über Services, Sonderregelungen und Nutzer im laufenden Betrieb des Managementsystems von Interesse sein könnten.

Nutzer oder Client Als Nutzer oder Client wird zum Zeitpunkt der Beschreibung des Managementsystems und seiner Policies der menschliche Nutzer zusammen mit seinem Nomadischen System - in der Regel ein Notebook - verstanden. Während des laufenden Betriebes kann es vorkommen, daß auch ein Systeminterner Service als Nutzer oder Client bezeichnet wird. Dies passiert genau dann, wenn er in der Rolle eines Clients von einem Server Daten erfragen muß und somit als Client einen Dienst nutzt.

Dienst oder Service Ein Dienst oder Service sind die Abstrakten Begriffe für eine Dienstleistung. Diese wird in der Regel direkt von einer Komponenten des Managementsystems (dem Server) erbracht. Spricht man in diesem Zusammenhang von 'einem Client, der einen Dienst nutzt', so meint man, daß dieser Client mit einem Server (direkt oder über einen Proxy) kommuniziert und Daten austauscht. Ein Service hat immer einen Service-Typ, sowie eine Reihe weiterer Eigenschaften, die ihn genauer spezifizieren. Ein solcher Service-Typ wäre zum Beispiel Email, eine weitere Spezifikation wäre POP3.

Managementsystem Unter dem Managementsystem wird die Gesamtheit aller Komponenten eines lokal begrenzten Netzwerk-Bereiches betrachtet. So gehören Drucker genauso zum Managementsystem, wie verschiedene Server und Services. Allgemein kann man sagen, daß das Managementsystem alle Komponenten eines lokalen Netzwerkes umfaßt, die zentral managebar sind. Dazu gehören nicht nur Services und Peripheriegeräte (Drucker), sondern auch Hardwarekomponenten wie Router und Switches.

Netzwerk Ein Netzwerk oder Netz bezeichnet den kompletten lokalen Netzbereich. Er umfaßt auch nichtmanagebare Komponenten oder bei Bedarf die Komponenten eines benachbarten Managementsystems.

Die im Folgenden entwickelte Beschreibung soll es ermöglichen, folgende Arbeitsgänge zu erfassen. Damit ergibt sich die Möglichkeit diese Arbeitsgänge zu automatisieren.

Automatische Konfiguration von Diensten vereinfachen Die automatische Konfiguration von Diensten soll mit Hilfe des Regelsets vereinfacht werden. Es sind grundlegende allgemeine Konfigurationsdaten wie IP-Adresse des Services, Name des Services, Typ des Services, maximal und minimal anbietbarer Quality of Service enthalten. Damit lassen sich Services allgemein in ein Netzwerk integrieren. Detailkonfigurationen müssen weiterhin je nach Service individuell ausgeführt werden. Dabei werden bei der Konfiguration des jeweiligen Dienstes die Angaben entsprechend eingelesen und der Service grundlegend konfiguriert.

Festlegen von Zugangsdaten für Clients Aus dem Regelset sollen die für den Client typischen technischen Spezifikationen enthalten sein. Dies umfaßt seinen Usernamen, sein Passwort, seine MAC-Adresse, seine IP-Adresse oder seinen Hostname. Sein bürgerlicher Name kann ebenfalls angegeben werden, ist aber für den laufenden Betrieb im Netzwerk nicht relevant. Außerdem sind an dieser Stelle noch keine buchhaltungstechnischen Angaben vorgesehen, wobei nichts dagegen spricht, das Modell passend zu erweitern.

Auf die Zugangsdaten wird zugegriffen, wenn ein Nutzer sich authentisieren will. Ebenfalls werden die Daten benötigt um zu prüfen, ob dieser spezielle Nutzer Vorrechte irgendeiner Art hat, wie möglicher Weise ein Recht auf bevorzugte Abarbeitung seiner Druck-Jobs. Auch bei der Abrechnung können diese Daten benötigt werden, wenn eine Service-Leistung abgerechnet wird, für die ein spezieller Preis im Vorfeld ausgehandelt wurde (z.B. 20 Drucker-Seiten frei)

Festlegen allgemeiner Policy-Regeln Der Hauptgrund, weswegen dieses Modell entwickelt wird, ist der, daß Policy-Informationen vermittelt werden sollen. Die meisten Informationen, die man über ein System angeben kann, kann allerdings schon auf die ein oder andere Weise als Policies bezeichnen, z.B. die Regelung, daß ein Client eine IP-Adresse dynamisch zugewiesen bekommt. Dabei kann man die Regel ganz speziell dem entsprechenden Client zuweisen.

Viel Wichtiger ist es oft noch, allgemeine, systemweite Regeln aufstellen zu können. Dies soll mit dem hier entwickelten Modell möglich gemacht werden.

Festlegen spezieller Policy-Regeln Keine Regel ohne Ausnahmen. Es soll Möglich sein, allgemeine Policy-Regeln zu verfeinern oder gar temporär außer Kraft zu setzen. Dafür benötigt man individuelle Regelangaben, die man einem speziellen Service oder einem speziellen Service-Nutzer zuordnen kann.

Erleichterung der Dienstauswahl Um die Zuordnung: 'Dienstwunsch des Clients' und 'Dienstangebot des Servers' einander zuzuordnen, soll man direkt bei der allgemeinen Beschreibung des Dienstes diese Angaben ablegen und verändern können. Das erleichtert den späteren Überblick.

Abrechnungshilfe Es kann wichtig sein, für verschiedene Services verschiedene Rechnungsstellen einführen zu müssen. Diese Besonderheit sollte direkt im Regelset anzugeben sein. Ebenfalls soll man Besonderheiten bei der Abrechnung beachten können (z.B. einen speziell ausgehandelten Preis für eine Service-Leistung)

4.5.2 Darstellung der Regelsyntax

Um diese Angaben so einfach wie möglich halten zu können, soll eine Sprache verwendet werden, die leicht in maschinell zu verarbeitende Regeln umzusetzen ist. Andererseits soll sie aber auch für den menschlichen Benutzer noch les- und schreibbar sein.

Aus diesen Gründen wurde sich für XML als Grundlage für diese Beschreibungssprache entschieden. XML wurde für die Entwicklung von Sprachen, die Regeln definieren und Zusammenhänge beschreiben, entwickelt und ist deswegen sehr gut für diese Zwecke geeignet. Der Vorteil von XML gegenüber einer eigenen proprietären Entwicklung liegt darin, daß Außenstehende sich nicht mit einer völlig fremden Beschreibung auseinander setzen müssen, sondern das bekannte Regelwerk von XML verwenden können. Somit sind die mit XML erstellten Regeln schneller nachvollziehbar. Weiterhin sind bereits fertige Parser für XML-generierte Sprachsets erhältlich.

Das hier verwendete Prinzip ist bereits von der HTML-Syntax her vertraut. Dabei werden Angaben mit Hilfe von *Tag*s gemacht. Ein solches Tag hat ein Beginn-Tag sowie meistens ein End-Tag. Allerdings müssen End-Tags nicht immer angegeben werden. Zu manchen Tags existiert auch kein End-Tag. Im Beginn-Tag kann man zusätzlich Attribute angeben. Diese sind von der Form `name="Wert"`.

Beispiel für ein Tag:

```
Beginn-Tag: <TAG>
End-Tag:    </TAG>
Attribute: <TAG name1="wert1" name2="wert2">
```

Dabei beschreibt der Begriff TAG um was es sich handelt, also als was die folgenden Werte vom Leser zu interpretieren sind.

XML benötigt eine allgemeine Definition, diese ist in dieser Arbeit in der Beschreibungssprache DTD abgefaßt. In Zukunft kann man auch UML statt DTD verwenden.

Für den hier zu verwendenden Zweck benötigt man Möglichkeiten, um folgende Angaben in XML vornehmen zu können (siehe auch Abb. 4.15:

- Beschreibung eines Nutzers
- Beschreibung eines Services
- Beschreibung einer Policy

- Gruppenbildung von Nutzern, Services, Policies oder Gruppen.
- Angabe von Gültigkeitszeiträumen
- Angaben von Sondervereinbarungen (QOS)
- Gruppierung von Gültigkeitsbereichen, also abgeschlossenen logischen Systemen

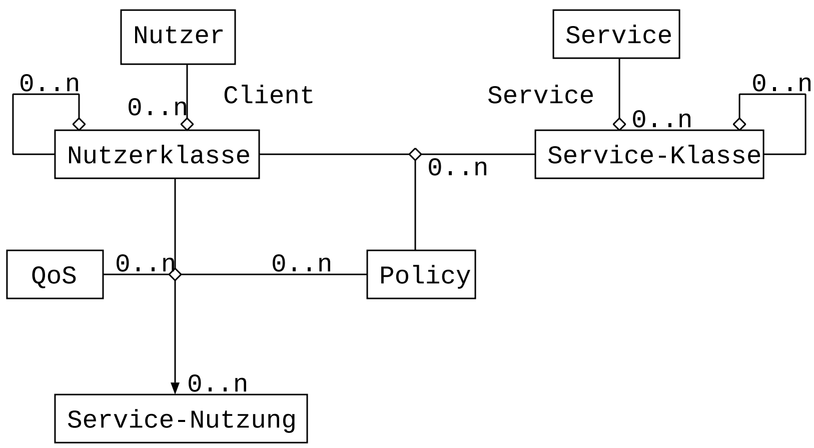

Abbildung 4.15: Policy: allgemeines Object-Modell

Aus diesen Anforderungen kann man eine Grammatik entwickeln. Das Objektmodell dieser Entwicklung stellt Abbildung 4.16 grafisch dar.

Die Ausgangsklasse ist **DOMAIN**. Sie besitzt die Unterklassen:

- **POLICY** welche selbst als direkte Unterklassen die Klassen **PR** (PolicyRule) mit deren Unterklassen **PC** (PolicyCondition) und **PA** (PolicyAction) besitzt. Instanzen der Klasse Policy kann von den Klassen **CLIENT SERVICE** und **GROUP** referenziert werden.
- **GROUP** besitzt als direkte Unterklasse die Klasse **INSERT** welche ebenfalls von den Klassen **CLIENT** und **SERVICE** aus referenziert werden kann.
- **TIME** besitzt selbst keine weiteren Unterklassen. Dafür werden Instanzen von ihr von den Klassen **PR** und **CLIENT** referenziert.
- **CLIENT** beschreibt das Profil eines speziellen Kunden. **CLIENT** hat als direkte Unterklasse die Klasse **QOS** in der spezielle Service-Vereinbarungen für diesen Kunden vermerkt werden können. Weiterhin referenziert die Klasse **CLIENT** Instanzen der Klassen **TIME**, **INSERT** und **POLICY**. Damit ist es möglich Vereinbarungen, die von allgemeinen Regeln abweichen direkt beim Kundenprofil zu vermerken.
- **SERVICE** hat zwei weitere Unterklassen und sie referenziert Instanzen der Klasse Policies. Damit werden spezielle Zusatzpolicies für genau diesen Service angegeben. Ihre Unterklasse **ADMININT** dient der Angabe wie das Administrationsinterface dieses Services angesprochen werden soll. Die Unterklasse **INFO** kann zusätzliche Informationen zum Service vermitteln.

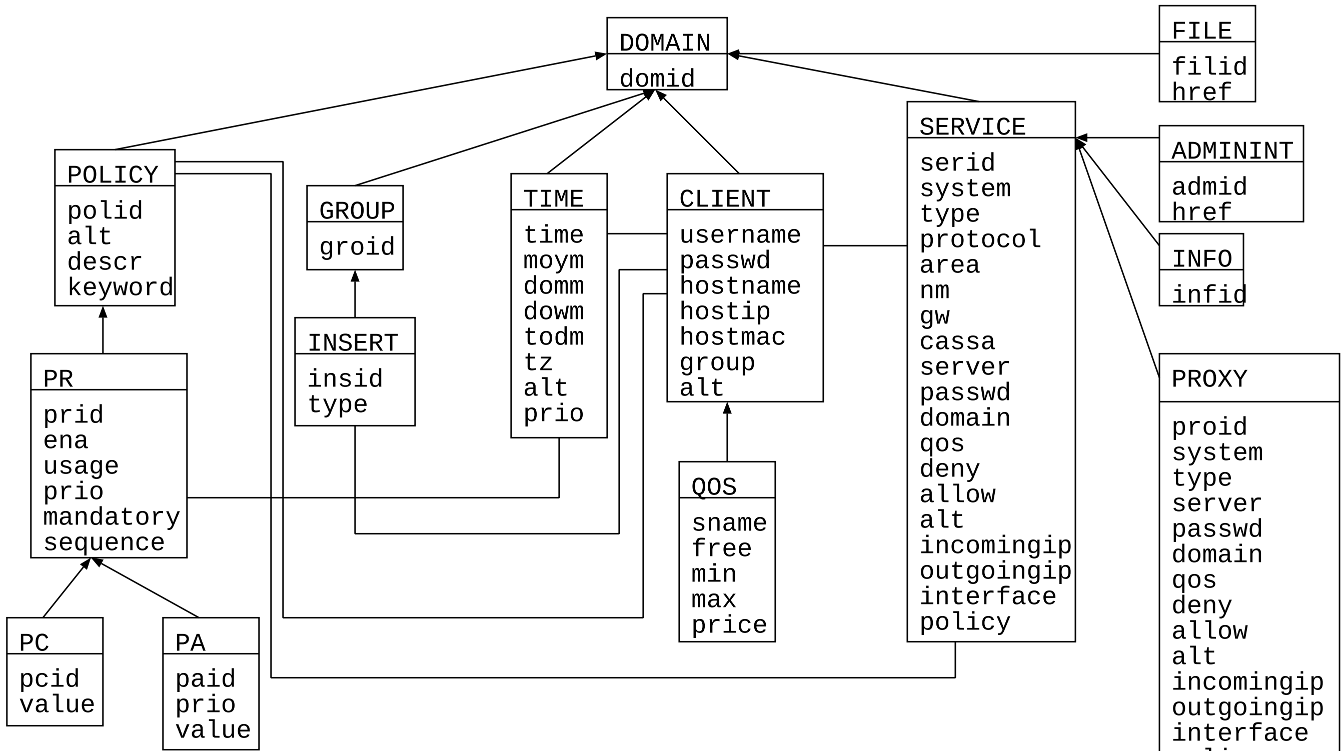

Abbildung 4.16: Object-Modell der DTD

- **FILE** ist eine Klasse, die es erlaubt, Definitionen, die Speichertechnisch in einem anderen File abgelegt sind, quasi an dieser Stelle zu übernehmen. Die Klasse hat somit auch keinerlei Unterklassen.

Ein Nebeneffekt, der bei der Entwicklung des Regelsets aufgetreten ist, ist der, daß viele Policy-Angaben bereits direkt bei der Angabe der jeweiligen Definition enthalten sind. Dies verkürzt die zu erstellende Datenmenge bei der Verwendung sehr einfacher Policy-Regeln erheblich, da die Darstellung einer kompletten Policy-Regel sehr umfangreich ist.

Allgemeine Dokumentstruktur

Ein XML Dokument beginnt immer mit den Zeilen:

```
<?xml version="1.0"?>
<!DOCTYPE DOMAIN SYSTEM "policies.dtd">
```

Sie definieren, daß es sich um XML in der Version 1.0 handelt. In DOCTYPE wird angegeben, wie das Root-Tag heißt (in vorliegendem Beispie heißt es DOMAIN). SYSTEM in Verbindung mit einer URL, die auf ein DTD-File zeigt, gibt an daß es eine Beschreibung zur verwendeten XML gibt und wo sie liegt.

```
<DOMAIN name="hotel">
</DOMAIN>
```

Das DOMAIN-Tag gibt an, um welches zu managende System es sich handelt. Als DOMAIN in diesem Sinne kann zum Beispiel ein Hotel verstanden werden, oder eine Bibliothek oder ein großer Konzern. Alle Angaben, die zwischen einem `<DOMAIN>` und einem `<DOMAIN>` stehen, beziehen sich auf diese eine zu managende Einheit.

Beschreibung eines Services

Die Beschreibung eines Services trifft in der Praxis mit der Beschreibung eines konkreten Dienstleisters zusammen. Deshalb wird auch hier nicht separat zwischen Service und einzelne zugehörige Server unterschieden.

Eine Service-Beschreibung besteht primär aus der Information zum Service selbst. Diese Informationen werden als Attribute im Service-Tag angegeben. Weiterhin kann ein Service eine beliebige Anzahl an Policy, Administrationsinterfaces, sonstige Zusatzinformationen und Proxies enthalten.

Ein Beispiel für einen POP3-Server:

```
<SERVICE
serid="ServiceName"
system="hotel"
server="pop3-I"
type="mail"
protocol="POP3"
area="network"
domain="hotel.de">
</SERVICE>
```

Das obige Beispiel beschreibt einen einfachen POP-3-Service. Dieser gehört zum Hotel. Der Server selbst wird angesprochen mit dem Namen pop3-I.hotel.de. Der Name setzt sich zusammen aus dem Attribut domain und dem Attribut server. Das Attribut protocol gibt an, daß der Service POP3 spricht und das area-Attribut zeigt, daß es sich um einen Netzwerkdienst handelt.

Mit dem Service-Tag sollen alle möglichen Services innerhalb eines Netzwerkes beschreibbar sein. Deshalb existiert eine umfangreiche Liste an möglichen Attributen:

serid enthält einen eindeutigen Namen innerhalb des Managementbereiches für diesen Service.

system gibt an, zu welchem Managementbereich der Service gehört. Der Wert von system ist identisch mit dem Wert des Attributes name des DOMAIN-Tag. Es soll zur Referenzierung des Services von Außerhalb des Managementsystems verwendet werden.

type gibt an um welche Art Service es sich handelt. Solche Servicetypen sind: email, fax, news, internet, web, telephone, nameservice, dns... Der Servicetyp soll an den allgemeinen Sprachgebrauch angelehnt sein.

protocol ist das nach Außen hin verwendete Protokoll. Es wird implizit angenommen, daß nur Services definiert werden, die intern das allgemeine Protokoll des Managementsystems verwenden, so daß dieses hier nicht weiter beachtet werden muß.

area gibt an, ob es sich um einen Netzwerkdienst oder um einen lokalen Service handelt. Dieses Attribut existiert, weil zum Zeitpunkt der Entwicklung dieses Regelsets nicht vollständig ausgeschlossen werden konnte, ob man auf diesem Wege nicht auch Services managen und vermitteln muß, die keine Netzwerkdienste sind, sondern lokal am betreffenden Nomadischen System angeschlossen sind. Zugelassene Werte des Attributes sind: lokal und network.

nm ist zum Eintragen einer Netmask vorgesehen. Dieses Feld soll vorwiegend der automatischen Konfiguration des Services selbst dienen.

gw ist zum Eintragen eines Gateways vorgesehen. Dieses Feld soll vorwiegend der automatischen Konfiguration des Services selbst dienen.

cassa gibt an, welcher Accounting-Server für diesen Dienst zuständig ist. Es soll die Möglichkeit geben, daß verschiedene Services auch verschiedene Accounting-Server nutzen.

server enthält den direkten Server-Namen des Services.

passwd enthält ein gecryptetes Passwort, welches zur Konfiguration des Services nötig sein kann.

domain enthält die Domain-Angabe des Servers. Die Attribute server und domain bilden zusammen den kompletten Namen des jeweiligen Services.

qos enthält minimale und maximale Quality of Service, den dieser Server anbieten kann.

deny enthält Clients und Services denen der Zugriff auf diesen Service generell verboten ist. Dadurch können sehr einfache Policy-Regeln bereits an dieser Stelle eingesetzt werden.

allow enthält Clients und Services denen der Zugriff auf diesen Service generell erlaubt ist. Dadurch können sehr einfache Policy-Regeln bereits an dieser Stelle eingesetzt werden.

incomingip bezeichnet die externe IP-Adresse des Services. Die Sichtweise ist immer vom Managementsystem aus. Dieses Attribut kann z.B. bei der Verwendung eines NAT-Services verwendet werden.

outgoingip bezeichnet die interne IP-Adresse des Services. Die Sichtweise ist immer vom Managementsystem aus. Dieses Attribut kann z.B. bei der Verwendung eines NAT-Services verwendet werden.

interface enthält Informationen zum Konfigurationsinterface.

Das End-Tag zu SERVICE ist verpflichtend. Zwischen Beginn- und End-Tag dürfen folgende Tags verwendet werden: POLICY, ADMININT, INFO, PROXY, INSERT.

POLICY gibt die Möglichkeit explizite Policies für den betreffenden Service anzugeben.

ADMININT enthält einen eindeutigen Namen, sowie eine URL die auf das Administrationsinterface verweist.

INFO enthält zusätzliche Informationen zum Service, die die Dienstvermittlung an den Nutzer vereinfachen sollen, z.B.:

```
<INFO infid="i1">3.Stock</INFO>
<INFO infid="i2">Farblaser</INFO>
<INFO infid="i3">rotes Druckpapier</INFO>
```

PROXY entspricht seitens der Attribute ziemlich genau dem SERVICE-Tag und beschreibt Proxies, die zu diesem Service gehören.

INSERT gibt die Möglichkeit einzelne Policies oder Gruppen von Policies an dieser Stelle zu benennen. Diese sind dann für diesen Service gültig.

Beschreibung eines Nutzers

Ein Nutzer wird grundsätzlich als Client aufgefaßt:

```
<CLIENT username="sdemmel"
passwd="ak3mwe34"
hostname="nekobil"
hostip="dynamic"
group="student"
alt="Simone Demmel">

<TIME time="19991212000000:19991231000000">

</CLIENT>
```

username entspricht dem Nutzernamen, der zur Anmeldung im System benötigt wird.

passwd ist das zu verwendende Passwort. Es soll aus Sicherheitsgründen verschlüsselt abgespeichert werden.

hostname gibt den Namen des Nomadischen Systems an, welches der Nutzer verwendet.

hostip gibt an, ob die IP-Adresse automatisch dem Nomadischen System zuzuweisen ist oder, wenn nicht, welche Adresse zugewiesen werden soll.

hostmac enthält die MAC-Adresse des Nomadischen Systems. Dieser Wert dient zur Kontrolle und eindeutigen Identifizierung des Nomadischen Systems.

group Gibt an, ob der Client einer Nutzergruppe zugeordnet werden soll. Diese Nutzergruppe ist unabhängig von der Zuordnung zu einer Service-Gruppe.

alt gibt die Möglichkeit Zusatzinformationen zum Nutzer abzulegen, z.B. seinen Namen.

Das End-Tag ist zwingend. Zwischen Beginn- und End-Tag sind folgende Tags erlaubt: POLICY, INSERT, SERVICE, QOS, TIME

POLICY erlaubt es, spezielle Policies für diesen Nutzer festzulegen.

SERVICE erlaubt es, spezielle Services anzugeben, die dieser Nutzer nutzen darf.

qos gibt an, ob und wenn ja welche speziellen Quality of Service Vereinbarungen mit diesem Nutzer getroffen wurden.

TIME gibt an, von wann bis wann dieser Nutzereintrag gültig ist. Es ist mindestens eine TIME-Angabe pro CLIENT nötig.

INSERT gibt die Möglichkeit, an dieser Stelle Services, Policies, oder Gruppen von Services oder Policies anzugeben, die für diesen Nutzer Gültigkeit haben sollen.

Beschreibung von Policy-Regeln

Eine Policy besteht aus einer Anzahl Policy Rules, welche wiederum aus Policy Conditions und Policy Actions, sowie einem Gültigkeitszeitraum besteht.

```
<POLICY polid="policyname"
alt="Kurzbeschreibung"
descr="ausführliche Beschreibung"
keyword="Schlüsselworte">
<PR prid="pr1"
ena="TRUE"
usage="Verwendungszweck"
prio="5"
mandatory="TRUE">

<PC pcid="pc1"
value="ip 123.123.123.123">
<PA paid="pa1"
prio="5"
value="execute /usr/bin/mailanadmin.pl">
<TIME time="19990510000000:19990611000000>
</PR>
</POLICY>
```

Bei der Policy-Konstruktion und den verwendeten Attributen wurde sich soweit möglich an das System von Strassner [ESM99c] und [ESM99a] gehalten (inzwischen existiert eine Version 5 [ESM99b]), dafür sind die Vorgängerversionen nicht mehr offiziell verfügbar.

Policy-Regeln einem Service oder einem Nutzer zuordnen

Einem Service oder Nutzer kann man auf zwei verschiedenen Wegen Policy-Regeln zuweisen.

1. Man verwendet das POLICY-Tag zwischen dem Beginn- und dem End-Tag des Service- bzw. Client-Tags.

2. Man verwendet das INSERT-Tag und verweist konkret auf eine Policy.

Als Beispiel die Verwendung von INSERT:

```
<CLIENT ...>
<TIME ...>
<INSERT insid="#policy1"
type="policy">
</CLIENT>
```

Im angegebenen Beispiel existiert eine Policy mit `groid="policy1"`. Als insid wird hier eine URL angegeben, die auf die zu verwendende Policy verweist. Mit dem Attribut type gibt man an, um welchen Typ es sich bei dem an dieser Stelle einzufügenden Objekt handelt. Zulässige Werte für type sind: client, service, group, policy.

Zeitangaben mit TIME

Mit dem TIME-Tag kann man Gültigkeitsdauer und Gültigkeitsperioden von Policies und Nutzern angeben.

TIME hat folgende Attribute:

time gibt den Gültigkeitsbereich an. Dabei wird ein Startdatum und ein Enddatum angegeben. Beide Daten werden mit ':' voneinander getrennt. Das Datum selbst ist im Format: yyyymmddhhmmss.

moym gibt an, an welchen Monaten der Eintrag Gültigkeit haben soll. Die Notation erfolgt über 12 Stellen, welche die 12 Monate repräsentieren und mit 0 oder 1 belegt werden. Ist eine Stelle mit 1 belegt, bedeutet dies, daß die Regel an diesem Monat Gültigkeit hat. Beispiel: 110011100011 zeigt an, daß in den Monaten Januar, Februar, Mai, Juni, Juli, November und Dezember die Regel gültig ist und an den anderen Monaten nicht.

domm gibt an, an welchem Tag im Monat der Eintrag Gültigkeit haben soll. Die Notation erfolgt über 31 Stellen die die Monatstage repräsentieren und mit 0 oder 1 belegt werden.

dowm gibt an, an welchem Wochentag der Eintrag Gültigkeit haben soll. Die Notation erfolgt über 7 Stellen, wobei die erste Montag repräsentiert. Jede Stelle wird wiederum mit 0 oder 1 belegt.

todm gibt an zu welcher Uhrzeit eines Tages der Eintrag Gültigkeit haben soll. Dazu wird eine Start- und eine Enduhrzeit angegeben und beide Zeitangaben mit ':' voneinander getrennt. Die Zeitangaben haben die Form: hhmmss. Beispiel: 121500:143000 bezeichnet einen Zeitraum zwischen 12:15:00 und 14:30:00.

tz gibt an, in welcher Zeitzone man sich befindet.

alt stellt Raum für einen menschenlesbaren Hinweis zur Verfügung.

prio gibt die Möglichkeit, dieser Zeit-Angabe eine Priorität zuzuweisen. Dadurch ist es möglich Policy-Konflikte leichter zu bearbeiten.

Die Belegung der Attribute wurde nach der Belegung in der Klasse PolicyTimeConditions aus [ESM99c] gewählt.

Quality Of Service spezifizieren

Das Quality Of Service-Tag erlaubt es, zusätzliche Vereinbarungen mit dem Nutzer bezüglich Service-Qualitäten anzugeben.

Das QOS-Tag hat kein Endtag und folgende Attribute:

sname enthält einen eindeutigen Identifikator für die betreffende Quality of Service Anweisung.

free enthält Freieinheiten, die der Benutzer vereinbart hat.

min enthält Angaben zu minimaler Service-Qualität.

max enthält Angaben zu maximaler Service-Qualität.

price enthält die Angabe wie viel pro Service-Einheit bezahlt werden muß. Dabei muß im Service selbst festgelegt werden, was eine Abrechnungseinheit darstellen soll. Mögliche Abrechnungseinheiten wären:

- Angefangenes Megabyte
- Angefangene Minute
- Angefangene Telefon-Gebühren-Einheit
- Angefangene Seite Papier
- Anfrage an den Server

Gruppenbildung

Oft macht es Sinn, daß man bestimmte Services oder Policies zu einer Gruppe zusammenfaßt um später nur noch auf die gesamte Gruppe zu verweisen. Zu diesem Zweck wird das GROUP-Tag verwendet. Dieses Tag hat genau ein Attribut, welches die eindeutige Bezeichnung dieser Gruppe enthält und bei späteren Verweisen auf eben diese Gruppe verwendet wird.

Innerhalb der Gruppe wird dann mit dem INSERT-Tag gearbeitet.

Dabei ist zu beachten, daß man in einer Gruppe nur Inserts des gleichen Typs verwendet, da es sonst zu Kollisionen kommen kann.

Als Beispiel soll eine Service-Gruppe gebildet werden. Diese soll Services mit der serid pop3-I, news-I und internet-I miteinander verbinden:

```
<GROUP groid="servicegruppeI">
<INSERT insid="#pop3-I" type="service">
<INSERT insid="#news-I" type="service">
```

```
<INSERT insid="#internet-I" type="service">
</GROUP>
```

Diese Gruppe kann man jetzt zum Beispiel innerhalb der CLIENT-Tags wie folgt 'einbetten':

```
<CLIENT ...Attribute>

<INSERT insid="#servicegruppeI" type="group">

...weitere INSERTs, und TIME...

</CLIENT>
```

Die Verwendung von FILE

Um die Übersichtlichkeit auch in größeren zu managenden Umgebungen zu gewährleisten, kann man mit dem FILE-Tag ein externes File an eben dieser Stelle 'einbetten'. Dieses File muß ein XML-Fragment sein.

Auf diese Art und Weise kann man Files anlegen, die die Service-Definitionen beinhalten, Files die die Gruppenbildungen enthalten und Files die Policies und Clients enthalten. Diese kann man dann mit dem File-Tag zu einem einzigen großen File zusammenfassen:

Beispiel:

```
<?xml version="1.0"?>
<!DOCTYPE DOMAIN SYSTEM "policies.dtd">

<DOMAIN domid="testhotel">
<FILE href="hotel-3-user.xml">
<FILE href="hotel-3-services.xml">
<FILE href="hotel-3-groups.xml">
</DOMAIN>
```

Kapitel 5

Realisierung für das Managementsystem NoCsControl

realisierung.tex

Kapitelüberblick:

- Das Kapitel erarbeitet ein Realisierungskonzept. Dabei sollen die in den vorangegangenen Kapiteln entwickelte Ideen im bereits existierenden Masa-Projekt des Lehrstuhles eingebunden werden.
- Es wird ein allgemeiner Lösungsansatz dargestellt.
- Das Kapitel bereitet die Implementierungsansätze des folgenden Kapitels vor.

Nachdem im vorigen Kapitel betrachtet wurde, wie ein solches Managementsystem aussehen kann, soll in diesem Kapitel ein Realisierungskonzept aufgestellt werden.

5.1 Konzept von Services

realisierungs-konzept.tex

Als Erstes benötigt man eine etwas andere Sichtweise auf Services. Services waren bisher Komponenten, die ein Interface (SServer) enthielten, der die zur Erbringung dieses Dienstes nötige Kommunikation mit der Außenwelt abwickelt (Abb. 5.1).

Dieses Konzept wird leicht verändert und der Service erhält drei Interfaces. Eines zur Kommunikation mit dem Managementsystem, eines zur Kommunikation mit der Außenwelt (hier wird für Außenstehende der eigentliche Dienst erbracht) und eines zur Konfiguration des Services.

Da die Konfiguration des Services und des zugehörigen Servers, praktischer Weise von Seiten des Managementsystemes aus vorgenommen werden soll, verschmelzen diese beiden Interfaces zu einem einzigen (siehe Abbildung 5.2).

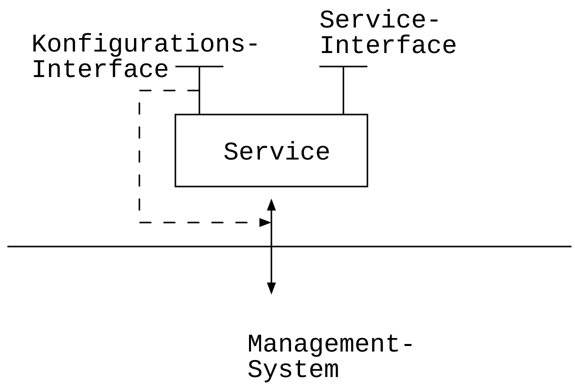

Abbildung 5.1: Service mit 3 Interfaces

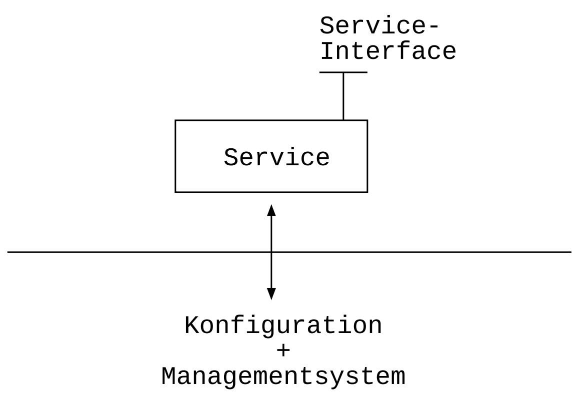

Abbildung 5.2: Service mit verschmolzenem Interface

5.2 Integration verschiedener Protokolle in einem Managementsystem

realisierung-multiprotokoll.tex

Die bisherige Entwicklung und Beschreibung von Managementsystemen beschränkt sich auf die Existenz einzelner Komponenten, Hierbei stand die Frage: 'Welche Komponenten muß das System besitzen?' im Vordergrund. Weiterhin wurde beschrieben, inwieweit einzelne Komponenten auf andere zugreifen und miteinander kommunizieren um Daten auszutauschen.

Damit ergibt sich jetzt folgendes allgemeines Bild (Abb 5.3) über den Grundaufbau einer Realisierung eines Managementsystems:

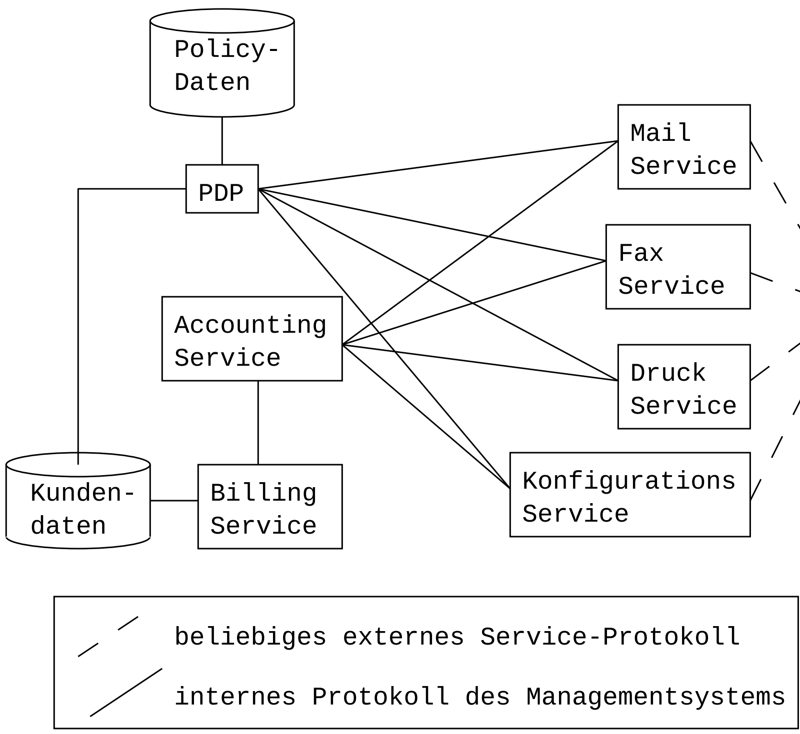

Abbildung 5.3: management-system1.eps

Ein Nomadisches System kommuniziert mit Services (konkreter mit dem SServer, der praktisch betrachtet der eigentliche Diensterbringer ist). Diese Services werden über ihr service-eigenes Protokoll angesprochen. So kann der Mail-Service zum Beispiel über POP3 oder SMTP angesprochen werden. Dazu benötigt man entweder mehrere SServer-Interfaces an einem Mailservice oder mehrere unabhängig voneinander arbeitende Services.

Letzteres ist die modularere Version, die es erlaubt, komplett unterschiedlich arbeitende Protokolle hinzuzufügen, ohne ein bereits vorhandenes ändern zu müssen.

Für den Druck- und Fax-Service werden ebenfalls jeweils unterschiedliche Protokolle verwendet (eines für den Druckservice, ein anderes für den Fax-Service).

Ein Beispiel, inwieweit die selben Services über verschiedene Protokolle erbracht werden können, liefern Abbildung 5.6 auf Seite 110 und 5.5 auf Seite 109.

Die Verwendung eines einheitlichen internen Kommunikationsprotokoll wie in Abbildung 5.6 auf Seite 110 vorgestellt, birgt den Vorteil, daß man nicht zusätzlich auch alle internen Komponenten für alle denkbaren Protokolle auslegen muß. Sollte dies trotzdem nötig werden, kann man jederzeit entsprechende Services hinzufügen.

Verwenden Nomadische Systeme unterschiedlicher Managementplattformen (zum Beispiel Jini oder SLP), so kann man dies wie folgt realisieren:

Die Komponente, die das Nomadische System direkt anspricht, verhält sich, wie die entsprechende Komponente der Managementplattform. Für alle weiteren Aufgaben (Abrechnung, Policy-Service-Nutzung...) verwendet sie das allgemeine interne Managementprotokoll.

Man kann auch alle Komponenten ihr eigenes Managementprotokoll verwenden lassen. Dabei würde man entsprechend viele Managementsysteme parallel zeitgleich verwenden. Dabei muß man darauf achten, daß an mindestens einer Stelle die Plattformen auf gemeinsame Daten zurückgreifen und so indirekt miteinander kommunizieren. Geschieht dies nicht, ist es möglich, daß ein Nomadisches System (absichtlich oder unabsichtlich) wechselweise die verschiedenen Plattformen nutzt und so eventuelle Freieinheiten mehrfach ausschöpfen kann, bzw. am Ende nicht berechnet bekommt.

5.3 Konfiguration der Services

realisierung-konfig.tex

In vielen Rechner-Netzen wurde beim Aufbau nicht an den Einsatz spezieller Management-Tools gedacht. Diese werden häufig erst dann in Betracht gezogen, wenn es sich um ein großes Netz (einige hundert Rechner, über mehrere Standorte verteilt) handelt. Die meisten Netze wurden für wenige Hosts designed und sind im Laufe der Zeit gewachsen. In ihnen wird traditionell jeder Dienst einzeln dort installiert, konfiguriert, überwacht und manipuliert, wo er läuft (d.h. wo er erbracht werden soll). Dabei ist es nicht vorgesehen, diese Aufgabe zentral zu managen. Oder von einem Managementsystem überwachen zu lassen. Diese Notwendigkeit wird meist erst sehr viel später festgestellt.

In den Anfängen des Internet (und Intranet) zum Beispiel sind solche gewachsenen Netzwerke die Regel. Somit haben auch fast alle Services keinerlei Schnittstellen, die über eigene, lokale Konfigurationsfiles hinausgehen oder für ein umfangreiches Management designed worden wären.

Das Problem, welches sich daraus ergibt, ist die Frage, wie man in die vorhandenen Dienste eine zusätzliche Management-Schnittstelle integrieren kann. Dabei hat man grundsätzlich zwei Lösungsansätze:

- Man kapselt den vorhandenen Code ein und erschafft einen zusätzlichen Codeteil, der einerseits eine Managementschnittstelle implementiert, andererseits die Kommunikation, die an den Server gesendet wird (und umgekehrt) abfängt und auf einem anderen Weg weiterreicht. Auf diesem Wege kann man eine künstliche Managementschnittstelle für einen bereits vorhandenen und implementierten Dienst erstellen.

Dies geht nur bei Diensten, die man auf diese Art und Weise einfach abkapseln kann, d.h. von der Außenwelt abschneiden kann, so daß sie selbst keinen direkten Zugriff mehr auf das Netz und die dort übertragenen Daten haben (Abb. 5.4.

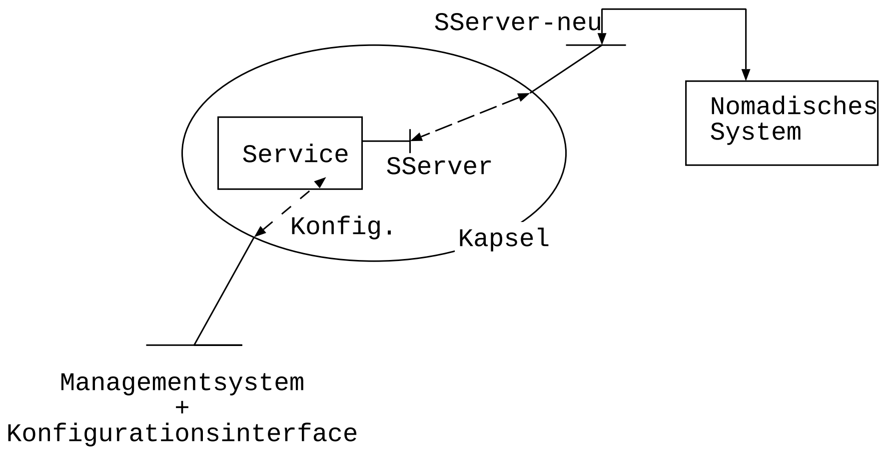

Abbildung 5.4: komplett eingekapselter Service

Bei normalen Services läßt sich dies regeln, indem der Original-Service auf einem Nicht-Standard-Port auf Daten wartet und das neue Interface auf dem Standardport agiert und die Daten bei Bedarf entsprechend an den anderen Port weiterleitet und anders herum ebenfalls wieder zu sich senden läßt und dann erst 'ausliefert'. Es fungiert sozusagen als Proxy für das Nomadische System. Das zusätzliche Konfigurationsinterface greift dann direkt auf das Konfigurations-File des Services zu, manipuliert es und veranlaßt, daß der Service selbst bei Bedarf neu gestartet wird oder anderweitig dazu veranlaßt wird, die neue Konfiguration einzulesen. Alle Konfigurations-Aspekte, die im Original-Service nicht vorgesehen sind (Verwendung von Policies, Kommunikation mit einem Policy-Service), muß das neu erstellte, dem Original-Service vorgestellte, Interface leisten.

- Man schreibt ein neues Programm für den entsprechenden Dienst und integriert hier eine Management-Schnittstelle von vorn herein.

Ansatz eins hat den Vorteil, daß man bereits existierende Services weiterverwenden kann. Diese muß man eventuell umkonfigurieren, aber ansonsten können sie normal weiter laufen. Allerdings ist der Ansatz, durch ein Zusatzprogramm auf eines oder mehrere Konfigurations-Files zuzugreifen, nicht elegant und sehr fehleranfällig.

Ansatz zwei würde die Kapselung des Services verhindern. Dafür muß man die komplette Implementierung des jeweiligen Services neu vornehmen.

In beiden Ansätzen muß man darauf achten, daß das verwendete Protokoll der Schnittstelle für die Kommunikation mit dem Managementsystem und die Konfiguration von allen anderen benötigten Diensten und Komponenten verwendet werden kann (also passende Implementierungen dieser Dienste vorhanden sind) Siehe dazu auch Abbildung 5.5:

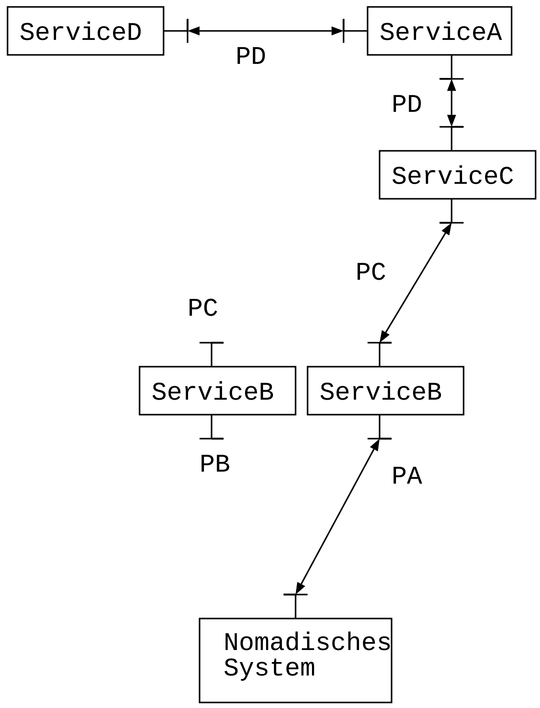

Abbildung 5.5: Verwendung mehrerer unterschiedlicher Protokolle

Dem Nomadischen System stehen zur Nutzung des ServicesB zwei Protokolle zur Verfügung (PA und PB). ServiceB reicht seine vom Nomadischen System mit Protokoll PA empfangene Daten an ServiceC weiter. Dies geschieht über Protokoll PC. ServiceC kommuniziert auf seiner 'internen' Seite mit ServiceA über Protokoll PD und der wiederum über Protokoll PD mit ServiceC. Wichtig bei der Installation einer solchen Service-Kette ist es, daß, soll ServiceB ServiceC nutzen können, die 'interne' Schnittstelle von ServiceB jeweils Protokoll PC 'sprechen' muß, da sonst die Kette unterbrochen wird.

Dieses Problem vermeidet man, indem man ein einheitliches Protokoll innerhalb der Management-Komponenten (alle Services zusammen, ohne dem Nomadischen System bilden das Managementsystem) ein einheitliches Protokoll verlangt (siehe Abbildung 5.6).

Die Konfiguration des einzelnen Services kann jetzt (im Objektmodell gedacht) direkt am Service angesetzt werden, da jeder Service nur ein Interface 'nach außen' und eines 'nach innen' mit integriertem Management-Interface besitzt.

Jeder Service soll, je nach internen Managementsystem, ein eigenes Konfigurationsinterface besitzen. Dieses Interface soll folgende Grundfunktionen anbieten:

- Abfragen des Zustandes des Services
- starten und stoppen des Services
- Eingabe sonstiger Konfigurationen des Services
- auslesen der aktuellen Konfiguration des Services
- Eingabe spezieller Policies für diesen Service

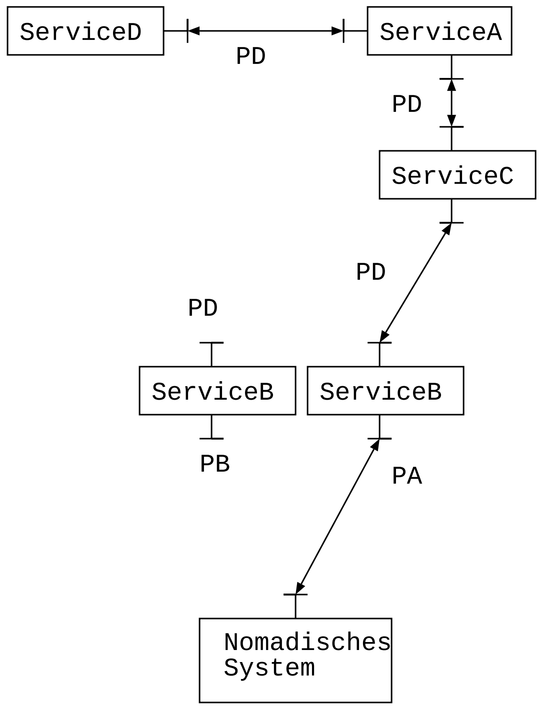

Abbildung 5.6: Verwendung mehrerer unterschiedlicher Protokolle

5.4 Aufbau des Managementsystems

realisierung-managementsystem.tex

Um ein Grundgerüst für ein Managementsystem zu erhalten, müssen folgende Schritte realisiert werden:

Für alle zu verwendenden Services müssen folgende Zusatzfunktionen zum eigentlichen Funktionsumfang des Services definiert werden:

- Kommunikation mit einem Policy-Service
- Kommunikation mit einem Abrechnungs-Service (Accounting-Service)

Als nächstes müssen für alle Services Prototypen implementiert werden.

Eine zentrale grafische Repräsentation aller Services, die zum Managementsystem gehören, sowie entsprechende Konfigurationsmöglichkeiten, muß geschaffen werden.

Ein Großteil dieser Aufgaben wurde bereits im Projekt MASA ([**?**]) realisiert.

Einzelne MASA-Komponenten werden als Agenten bezeichnet. Sie kommunizieren untereinander über CORBA und sind in der Regel in Java implementiert.

In Masa sind bereits folgende Services realisiert:

XXXX

Als Ergebnis dieser Arbeit soll der bereits vorhandene Policy-Agent auf die in dieser Arbeit entwickelten Möglichkeiten, erweitert werden. Weiterhin soll als Prototyp ein DHCP-Agent hinzugefügt werden, der die erweiterte Policy nutzt.

Kapitel 6

Implementierung

implementierung-neu.tex

6.1 Policy-Agent

Der bereits existierende Policy-Agent besteht aus drei Hauptklassen: **PersistenceService**, **EnforcementObjectFactory** und **PolicyFactory**. Weiterhin existieren eine Reihe Nebenklassen.

6.1.1 PersistenceService

Dieser Service lädt Policies aus einem File. Das bedeutet, er erhält den Namen des zu öffnenden Files vom dazugehörigen Applet. er öffnet das File und lädt seinen Inhalt. Diesen reicht er an die PolicyFactory weiter.

6.1.2 PolicyFactory

Erhält die Eingaben, die der PersistenceServer weitergeleitet hat und bildet daraus Corba-Objekte. Diese Objekte können dann von er EnforcementObjectFactory ausgewertet werden. Dabei werden die Policies noch einheitlich in Zielorientierte, Operational und Strategic Policies untergliedert. Diese Untergliederung existiert im einzuarbeitenden Policy-Design nicht. Hier werden Policies nach Zugehörigkeit zu Klassen, bzw. Nutzergruppen gegliedert. So gibt es Policies, die einen Nutzer betreffen oder einen Service.

Es handelt sich also um ein vollständig anderes Beschreibungssystem. Dieses soll auch weiterhin beibehalten werden, da die anderen Agenten, sofern sie den Policy-Service nutzen nicht umgestellt werden.

Es muß also ein zusätzliches System implementiert werden.

6.1.3 EnforcementObjectFactory

Die EnforcementObjectFactory wertet die vorhandenen Anfragen und Policies aus und liefert die entsprechenden Entscheidungen als Ergebnis.

6.2 Änderungen und Erweiterungen

Es müssen also alle drei Services erweitert werden. Der Persistence-Service muß die neue Policy-Syntax einlesen können. Das bisherige File war in komprimierter Datenbankform abgelegt und wird mit einer entsprechenden Java-Funktion eingelesen.

Das neue File liegt im Klartext und ASCII-Format vor.

Zur Umsetzung der Policy-Informationen im neuen Format (XML) gibt es einen Parser von IBM, den SAX-Parser. Dieser kann XML-Files einlesen und liefert entsprechende CORBA-Objekte.

Der bereits existierende Code muß also um die SAX-Schnittstelle erweitert werden, sowie um eine Möglichkeit erkennen zu können, ob es sich um das alte Policy-Format oder das neue Format handelt.

Der EnforcementObjectFactory muß um die in dieser Arbeit entwickelte Policy erweitert werden, da er bisher nur die konkreten Policy-Typen erkennt und auswerten kann.

Weiterhin muß das Applet angepaßt werden. Es ist zulässig einzelne Policy-Regeln über das Applet anzugeben. Dieses Interface müsste komplett umgestellt werden, damit beide Formen der Eingabe zulässig sind. Die neue Eingabeform ist in der Beschreibungssprache XML.

6.2.1 Der SAX-Parser

Der SAX-Parser ist ein Javabasiertes Produkt zum Parsen von XML-Files.

Um ihn einzusetzen wären folgende Code-Fragmente nötig:

```
import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;

public class MyHandler extends HandlerBase {

public void startElement (String name, AttributeList atts)
{
System.out.println("Start element: " + name);
}

    public void endElement (String name)
    {
```

```
        System.out.println("End element: " + name);
    }

}
```

In der Klasse MyHandler wird der DocumentHandler verwendet. er erhält Events zu den verschiedenen Ereignissen. Was mit diesen Events geschehen soll, wird in der Klasse myHandler implementiert.

Es gibt folgende Events:

characters(char[, int, int)]Erhält eine Nachricht, für jeden Buchstaben

endDocument() Erhält eine Nachricht, wenn das zu parsende Dokument zu Ende ist.

endElement(String) Erhält eine Nachricht, wenn ein Element-Ende gelesen wird.

ignorableWhitespace(char[, int, int)]Erhält eine Nachricht, wenn White-Spaces gelesen werden, die im Normalfall ignoriert werden können.

processingInstruction(String, String) Erhält eine Nachricht, wenn die Ausführung einer Aktion (processing instruction) im zu parsenden Dokument gelesen wird.

setDocumentLocator(Locator) Erhält ein Objekt, um den Ausgangspunkt eines Events im Dokument zu erhalten.

startDocument() Erhält eine Nachricht, wenn ein neues Dokument beginnt.

startElement(String, AttributeList) Erhält eine Nachricht, wenn ein Element-Start gelesen wird.

Um diese HandlerBase zu nutzen, müßte folgendes implementiert werden:

```
import org.xml.sax.Parser;
import org.xml.sax.DocumentHandler;
import org.xml.sax.helpers.ParserFactory;

public class SAXApp {

static final String parserClass = "com.microstar.xml.SAXDriver";

    public static void main (String args[])
        throws Exception
    {
     Parser parser = ParserFactory.makeParser(parserClass);
        DocumentHandler handler = new MyHandler();
        parser.setDocumentHandler(handler);
        for (int i = 0; i < args.length; i++) {
         parser.parse(args[i]);
        }
     }
}
```

6.3 Client

Der DHCP-Agent besteht aus seiner Hauptklasse DHCPMobileAgent, der Klasse RequestHandler und DhcpServer. Weiterhin werden noch die Spezialklassen IpAddress und DhcpRequest implementiert.

Arbeitsweise

Der DHCP-Server lauscht auf Port 68. Empfängt er einen Datenstrom, so wird dieser an den RequestHandler weitergereicht. Der RequestHandler übergibt den Datenstrom an die Klasse DhcpRequest, welche als Ergebnis alle Teilbereiche des DHCP-Paketes zur Verfügung stellt, sowie analysiert um welchen Typ Request es sich handelt.

Policy-Agent

Der RequestHandler startet nun eine Anfrage an den Policy-Agenten, in der er übermittelt, daß er die IP-Adresse zum Client mit der entsprechenden MAC-Adresse zugewiesen bekommen möchte. Dabei gibt es zwei Antwortmöglichkeiten. Der Policy-Agent kann mit einem String antworten, der eine IP-Adresse enthält oder mit dem Vermerk 'dynamic'.

Liefert der Policy-Agent eine IP-Adresse, so wird ein Antwort-Strom mit dieser IP-Adresse generiert (DhcpRequest.reply). Im anderen Fall wird eine IP-Adresse aus dem Adresspool des DHCP-Agenten verwendet.

IP-Adresspool

Dieser Adresspool wird von der Klasse DhcpServer verwaltet. Dabei handelt es sich um drei Arrays mit Instanzen der Klasse IpAddress. Diese Arrays werden nach Freien Adressen, reservierten Adressen und Momentan verwendeten Adressen untergliedert.

Hat der Agent keine freien IP-Adressen mehr zur Verfügung, so sendet er keine Antwort.

Applet

Das dazugehörige Applet soll die Möglichkeit bieten, den Agenten zu stoppen und zu starten, sowie ihm IP-Adressen für seinen internen Pool ein und auszutragen.

Kapitel 7

Zusammenfassung und Ausblick

IPv6 IPSec

A

```
<!ELEMENT DOMAIN  - ((CLIENT)+|(GROUP)+|(POLICY)+|(FILE)+)+ - DOMAIN>
<!ATTLIST DOMAIN
domid CDATA #IMPLIED <!-- system name -->
>

<!ELEMENT FILE - 0 EMPTY>
<!ATTLIST FILE
href %URL <!-- file to insert here -->
>

<!ELEMENT ADMININT - 0 EMPTY>
<!ATTLIST ADMININT
admid CDATA #IMPLIED <!-- name of the interface -->
href %URL #IMPLIED <!-- URL of interface -->
>

<!ELEMENT INFO - - CDATA - INFO><!-- any kind of additional Info -->
<!ATTLIST INFO
infid CDATA #IMPLIED <!-- name of INFO -->
>

<!ELEMENT POLICY - - (PR)+ - POLICY><!-- Policy-Group in Group abhandeln -->
<!ATTLIST POLICY
polid CDATA #IMPLIED
alt CDATA #IMPLIED <!-- caption in object model -->
descr CDATA #IMPLIED <!-- description -->
keyword CDATA #IMPLIED <!-- list of keywords seperated by ";" -->
>
```

```
<!ELEMENT PR - - ((PC)+|(TIME)+|(PA)+)+ - PR>
<!-- Policy Rule -->
<!ATTLIST PR
prid CDATA #IMPLIED
ena TRUE|FALSE #TRUE
usage CDATA #IMPLIED <!-- policyRuleUsage -->
prio CDATA #IMPLIED <!-- policyRulePriority -->
mandatory TRUE|FALSE #IMPLIED <!-- policyRuleMandatory -->
sequence mandatory|
recommendet|
dontcare #mandatory <!--
  policyRuleSequencedActions
-->
>

<!ELEMENT PC - 0 EMPTY>
<!ATTLIST PC
pcid CDATA #IMPLIED <!-- unique name -->
value CDATA #IMPLIED <!-- condition -->
>

<!ELEMENT PA - 0 EMPTY>
<!ATTLIST PA
paid CDATA #IMPLIED <!-- name -->
prio CDATA #IMPLIED <!-- priority -->
value CDATA #IMPLIED <!-- action -->
>

<!ELEMENT TIME - O EMPTY>
<!ATTLIST TIME
time CDATA #IMPLIED <!-- ptpConditionTime -->
moym CDATA #IMPLIED <!-- ptpMonthOfYearMask
                                                -->
domm CDATA #IMPLIED <!-- ptpDayOfMonthMask -->
dowm CDATA #IMPLIED <!-- ptpDayOfYearMask -->
todm CDATA #IMPLIED <!-- ptpTimeOfDayMask -->
tz CDATA #IMPLIED <!-- ptpTimeZone -->
alt CDATA #IMPLIED <!-- alternate Text  -->
prio CDATA #IMPLIED <!-- priority to use in
     PRULE from
     policyRulePriority -->
>

<!ELEMENT QOS - 0 EMPTY>
<!ATTLIST QOS
sname CDATA #IMPLIED <!-- Service name -->
```

```
free %Text #IMPLIED <!-- any kind of free instances -->
min CDATA #IMPLIED <!-- any kind of minimum -->
max CDATA #IMPLIED <!-- any kind of maximum -->
price CDATA #IMPLIED <!-- price per unit -->
>


<!ELEMENT CLIENT - - ((POLICY)+|(INSERT)|(SERVICE)|(QOS))*&(TIME)+ - (CLIENT)>
<!-- Nutzer in class modell -->
<!ATTLIST CLIENT
username CDATA #IMPLIED
passwd CDATA #IMPLIED
hostname CDATA #IMPLIED
hostip (%IPAddr|"dynamic") #IMPLIED
hostmac CDATA #IMPLIED
group CDATA #IMPLIED
alt CDATA #IMPLIED
>

<!ELEMENT SERVICE - - ((POLICY)+|(ADMININT)+|(INFO)+|(PROXY)+|(INSERT)+) - SERVICE>
<!-- any kind of agent -->
<!ATTLIST SERVICE
serid CDATA #IMPLIED
system CDATA #IMPLIED
type %ContentType #IMPLIED
protocol CDATA #IMPLIED <!-- protocoll
  used by this
  service -->
area ("network"|"lokal") #IMPLIED
nm %IPAddr #IMPLIED
gw %IPAddr #IMPLIED
cassa %URI #IMPLIED
server %URI #IMPLIED
passwd CDATA #IMPLIED
domain CDATA #IMPLIED
qos CDATA #IMPLIED <!-- information about minimal and
maximal quality of service served
by this server -->
deny CDATA #IMPLIED <!-- goes to policy-rules -->
allow CDATA #IMPLIED <!-- goes to policy-rules -->
alt CDATA #IMPLIED
incomingip %IPAddr #IMPLIED
outgoingip %IPAddr #IMPLIED
interface CDATA #IMPLIED
>
```

```
<!ELEMENT PROXY - O EMPTY>
<!ATTLIST PROXY
proid CDATA #IMPLIED
system CDATA #IMPLIED
type %ContentType #IMPLIED
server CDATA #IMPLIED <!-- name of
corresponding server -->
passwd CDATA #IMPLIED
domain CDATA #IMPLIED
protocol CDATA #IMPLIED <!-- protocoll
  used by this
  service -->
qos CDATA #IMPLIED <!-- information about minimal and
maximal quality of service served
by this server -->
deny CDATA #IMPLIED <!-- goes to policy-rules -->
allow CDATA #IMPLIED <!-- goes to policy-rules -->
alt CDATA #IMPLIED
incomingip %IPAddr #IMPLIED
outgoingip %IPAddr #IMPLIED
interface CDATA #IMPLIED
policy %URL #IMPLIED
>



<!ELEMENT GROUP - - (INSERT)+ - (GROUP)>
<!ATTLIST GROUP
groid CDATA #IMPLIED
>

<!ELEMENT INSERT - O EMPTY>
<!ATTLIST INSERT
insid %URL #IMPLIED <!-- Unique URL -->
type (client|service|group|policy) #IMPLIED
<!-- ELEMENT-name of
 inserted Object -->
>
```

B

```
<?xml version="1.0"?>
<!DOCTYPE DOMAIN SYSTEM "policies.dtd">
<!--
<?xml version="1.0" standalone="yes"?>
-->
```

```
<DOMAIN domid="testhotel">
<!-- KundenKonfiguration -->

<CLIENT username="u1"
    passwd="02oawyZdjhhpg"
    hostname="kunde1"
    hostIP="dynamic"
    group=""
    alt="User Eins vom Testhotel">

    <TIME   time="19990101000000:19991231235959"
        moym="110011100011"
        dowm="1111100"
        todm="090000:180000"
        alt=""></TIME>

    <QOS sname="#drucker-papier"
        free="10"></QOS>

    <INSERT insid="#multimedia"
        type="group"></INSERT>
</CLIENT>

<CLIENT username="u2"
    hostname="kunde2"
    hostIP="dynamic"
    group="Konferenz"
    alt="User Zwei vom Testhotel">

    <TIME   time="19990101:19990110"
        moym="100000000000"
        alt=""></TIME>

    <INSERT insid="#standard"
        type="group"></INSERT>

</CLIENT>

<CLIENT username="u3"
    hostname="kunde3"
    hostIP="192.168.1.100"
    group="Konferenz"
    alt="User drei vom Testhotel">

    <TIME   time="19991101:20000229"
        moym="110000000011"
```

```
        alt=""></TIME>

    <INSERT insid="#standard"
        type="group"></INSERT>

</CLIENT>
<!-- Services -->

<SERVICE serid="drucker-papier"
    system="testhotel"
    type="printer"
    area="network"
    server="lpr1"
    domain="hotel.de"
    alt="lokaler Printserver fuer Papierdrucke">
</SERVICE>

<SERVICE serid="fax"
    system="testhotel"
    type="fax"
    area="network"
    server="fax"
    domain="hotel.de"
    alt="Allgemeiner Fax-Service">

</SERVICE>
<SERVICE serid="email"
    system="testhotel"
    type="mail"
    area="network"
    server="mail"
    domain="hotel.de"
    alt="lokaler Mailserver">

</SERVICE>
<SERVICE serid="news"
    system="testhotel"
    type="news"
    area="network"
     server="news"
    domain="hotel.de"
    alt="lokaler Newsserver">

</SERVICE>
<SERVICE serid="internet"
    system="testhotel"
```

```
    type="network"
    area="network"
    nm="255.255.255.240"
    gw="10.0.0.2"
    domain="hotel.de"
    alt="Allgemeiner Internet-Zugang">
</SERVICE>


<!-- Service-Gruppen -->

<GROUP groid="multimedia">
    <INSERT insid="drucker-papier" type="service"></INSERT>
    <INSERT insid="fax" type="service"></INSERT>
</GROUP>

<GROUP groid="standard">
    <INSERT insid="email" type="service"></INSERT>
    <INSERT insid="news" type="service"></INSERT>
    <INSERT insid="internet" type="service"></INSERT>
</GROUP>

</DOMAIN>
```

Index

Literaturverzeichnis

[AD93] ALEXANDER, S. und R. DROMS: *RFC 1533: DHCP Options and BOOTP Vendor Extensions*. RFC, IETF, Oktober 1993.

[AD97] ALEXANDER, S. und R. DROMS: *RFC 2132: DHCP Options and BOOTP Vendor Extensions*. RFC, IETF, März 1997.

[BLMM94] BERNERS-LEE, T., L. MASINTER und M. MCCAHILL: *RFC 1738: Uniform Resource Locators (URL)*. RFC, IETF, Dezember 1994.

[CIM99] DMTF: *The CIM Tutorial*, 1999.

[Cla82] CLARK, D. D.: *RFC 815: IP datagram reassembly algorithms*. RFC, IETF, Juli 1982.

[Die99] DIEDERICH, THOMAS: *Verzeichnisdienste und ihre Anwendungen im Netz- und Systemmanagement - Directory-basiertes Netzmanagement - Policy-basiertes Bandbreitenmanagement am Beispiel CISCO/3Com*. Hauptseminar, Technische Universität München, 07 1999.

[Dro93] DROMS, R.: *RFC 1534: Interoperation Between DHCP and BOOTP*. RFC, IETF, Oktober 1993.

[Dro97] DROMS, R.: *RFC 2131: Dynamic Host Configuration Protocol*. RFC, IETF, März 1997.

[ESM99a] ELLESSON, ED, JOHN STRASSNER und ROBERT MOORE: *Policy Framework LDAP Core Schema*. Internet Draft, IETF, 08 1999.

[ESM99b] ELLESSON, ED, JOHN STRASSNER und ROBERT MOORE: *Policy Framework LDAP Core Schema*. Internet Draft, IETF, 10 1999.

[ESM99c] ELLESSON, ED, JOHN C. STRASSNER und ROBERT MOORE: *Policy Framework Core Information Model*. Internet Draft, IETF, 03 1999.

[ESWW99] ELLESSON, ED, JOHN STRASSNER, WALTER WEISS und ANDREA WESTERINEN: *Terminology for describing network policy and services*. Internet Draft, IETF, 03 1999.

[FB96a] FREED, N. und N. BORENSTEIN: *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC, IETF, November 1996.

[FB96b] FREED, N. und N. BORENSTEIN: *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.* RFC, IETF, November 1996.

[FB96c] FREED, N. und N. BORENSTEIN: *RFC 2049: Multipurpose Internet Mail Extension (MIME) Part Five: Conformance Criteria and Examples.* RFC, IETF, November 1996.

[FKP96] FREED, N., J. KLENSIN und J. POSTEL: *RFC 2048: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures.* RFC, IETF, November 1996.

[GPVD99] GUTTMAN, E., C. PERKINS, J. VEIZADES und M. DAY: *RFC 2608: Service Location Protocol Version 2.* RFC, IETF, Juni 1999.

[HAN99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integriertes Management vernetzter Systeme — Konzepte, Architekturen und deren betrieblicher Einsatz.* dpunkt-Verlag, ISBN 3-932588-16-9, 1999.

[Hei99] HEILBRONNER, S.: *Konzeption einer Managementarchitektur für die Unterstützung nomadischer Systeme (Arbeitstitel).* Dissertation, Ludwig-Maximilians-Universität München, September 1999.

[Hor84] HORNIG, C.: *RFC 894: Standard for the transmission of IP datagrams over Ethernet networks.* RFC, IETF, April 1984.

[Mic99a] MICROSYSTEMS, SUN: *Community Licensing.* Jini Licensing, Sun, 01 1999.

[Mic99b] MICROSYSTEMS, SUN: *Jini Technology Vision.* Jini Overiew, Sun, 01 1999.

[Mic99c] MICROSYSTEMS, SUN: *JiniTM Technology and Emerging Network Technologies.* Jini Specification, Sun, 01 1999.

[Mic99d] MICROSYSTEMS, SUN: *JiniTM Technology Architectural Overview.* Jini Overview, Sun, 01 1999.

[Moo96] MOORE, K.: *RFC 2047: MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text.* RFC, IETF, November 1996.

[MRU92] MALIS, A., D. ROBINSON und R. ULLMANN: *RFC 1356: Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode.* RFC, IETF, August 1992.

[OMG97] *CORBAservices - Trader Service (chapter 16).* OMG Specification formal/97-12-23, Object Management Group, Dezember 1997.

[Pos84] POSTEL, J.: *RFC 895: Standard for the transmission of IP datagrams over experimental Ethernet networks.* RFC, IETF, April 1984.

[Pro97] PROVAN, D.: *RFC 2241: DHCP Options for Novell Directory Services.* RFC, IETF, November 1997.

[R9̈9] RÖLLE, H.: *Authentisierung und Autorisierung für das Java/CORBA–Agentensystem MASA.* Diplomarbeit, Technische Universität München, August 1999.

[Ste93] STEENSTRUP, M.: *RFC 1479: Inter-Domain Policy Routing Protocol Specification: Version 1.* RFC, IETF, Juli 1993.

[Su81] SU, Z.: *RFC 781: Specification of the Internet Protocol (IP) timestamp option.* RFC, IETF, Mai 1981.

[VGPK97] VEIZADES, J., E. GUTTMAN, C. PERKINS und S. KAPLAN: *RFC 2165: Service Location Protocol.* RFC, IETF, Juni 1997.