RC23087 (W0401-131) January 28, 2004
Computer Science

IBM Research Report

Policy-basiertes Management: State-of-the-Art und zukünftige Fragestellungen

Alexander Keller, Heiko Ludwig
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Policy-basiertes Management: State-of-the-Art und zukünftige Fragestellungen

Alexander Keller, Heiko Ludwig
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
USA
E-Mail: {alexk|hludwig}@us.ibm.com

1. Einleitung

Im Systemmanagement werden Policies als ein zentrales Werkzeug angesehen, um die kontinuierlich steigende Komplexität und Dynamik verteilter Systeme zu bewältigen und derzeit überwiegend manuell durchgeführte Aufgaben zukünftig maschinell zu erledigen. Durch die Verwendung von Policies erhofft man sich eine signifikante Automatisierung der Managementprozesse, da ein Administrator im Sinne eines „Management by Objectives“ Verhaltensregeln definiert, die ein Managementsystem in die Lage versetzen sollen, anhand dieser vorgefertigten Regeln selbständig zu entscheiden, wie es sich beim Eintreffen bestimmter Ereignisse verhält. Durch die Verwendung von Policies ist ein Managementsystem in der Lage, auf unterschiedlichste Ereignisse korrekt zu reagieren, ohne daß seine zugrundeliegende Programmlogik modifiziert werden muß [39]. Die folgende allgemein akzeptierte Definition von Policies spiegelt dieses Prinzip wieder: „*Policies are rules governing the choices in behaviour of a system*“ [35]. Eine Policy beschreibt üblicherweise eine Kombination aus Bedingung(en) und Aktion(en), die durch das Eintreffen eines Ereignisses getriggert werden. Ferner sind Policies persistent und beschreiben wiederholbare Verhaltensmuster; eine einmalig bzw. ad-hoc durch einen Administrator oder ein Managementsystem ausgeführte Aktion wird nicht als Policy betrachtet.

Das Konzept und der Nutzen von Policies sind intuitiv verständlich, da der Begriff „Policy“ in verschiedenen Ausprägungsformen sowohl in der IT-Welt, als auch außerhalb benutzt wird, wie die folgenden Beispiele unterschiedlicher Policies zeigen:

- **Sicherheitspolicies**: „Benutzer der Kategorie *Enduser* dürfen sich nur an Werktagen zwischen 9.00 Uhr und 17.00 Uhr in ein System einloggen.“
- **Quality of Service (QoS) Management Policies**: „wenn ein IP-Paket zur Kategorie Gold angehört, wird es der Warteschlange mit der höchsten Priorität zugeordnet.“
- **Geschäftsbedingungen (Business Rules)**: „Wenn ein Kunde für mehr als 1000 US$ Waren pro Woche ordert, wird ein Rabatt von 2% der Gesamtsumme gewährt“.
- **Dienstgütevereinbarungen (Service Level Agreements, SLAs)**: „Wenn eine Kundenapplikation weniger als 95% während eines Monats verfügbar ist, werden dem Kunden 100 US$ zurückerstattet.“

- **Interaktionspolicies**: „Der Zugriff auf eine Online Banking Applikation erfordert die Verwendung von Triple DES Verschlüsselung“.

Im Systemmanagement bestehen die primären Anwendungsbereiche von Policies traditionell in der Kontrolle des Zugriffs auf Ressourcen durch Benutzer sowie dem Konfigurationsmanagement von Systemen, Anwendungen und Netzkomponenten. Neue dynamisch zusammengesetzte verteilte Anwendungen, basierend z.B. auf Utility und Grid Computing Infrastrukturen sowie die damit einhergehenden geschichteten Dienstnutzer/Diensterbringer-Beziehungen erfordern darüberhinaus ein nahtloses Zusammenwirken von Policies und Dienstgütevereinbarungen (Service Level Agreements, SLAs).
In der Systemadministration werden Policies verwendet, um Zugriffsrechte von Benutzern für Systemressourcen zu definieren oder die auf einem zentralen Server definierten Konfigurationsinformation und –dateien (z.B. für DNS, NFS, NIS, Passwort) auf Zielsystemen zu replizieren [17]. Ein häufig eingesetztes Beispiel für letzteres ist die frei verfügbare *Cfengine* [3]. Cfengine–Algorithmen beruhen auf der Klassifizierung von Zielsystemen anhand mehrerer Parameter wie Systemarchitektur, Betriebssystem, Zugehörigkeit zu IP-Subnetzen, organisatorischer Zugehörigkeit usw. Cfengine-Konfigurationspolicies bestehen aus Bedingungen (die sich z.B. aus den o.g. Zielsystemklassen sowie Replikationsintervallen zusammensetzen können) und Aktionen (z.B. Kopieren von Dateien). Wenn ein auf einem Zielsystem laufender Cfengine-Agent feststellt, daß eine Bedingung erfüllt ist, wird die dazugehörige Aktion ausgefürt, indem die notwendigen Daten von einem Server geladen werden. Die Anforderung von Konfigurationsdaten geschieht dezentral durch die Zielsysteme.

Policy-based Computing hat sich im Laufe der letzten 10 Jahre zu einer eigenständigen Forschungsrichtung innerhalb des Systemmanagements entwickelt, der in kurzen Abständen Themenausgaben von Fachzeitschriften (siehe z.B. [12, 14]) gewidmet sind. Ein jährlich stattfindender Policy Workshop [24] ergänzt die seit langem regelmäßig stattfindenden Sessions zum Thema *Policy-based Management* in den IFIP/IEEE Konferenzen *Integrated Management (IM)* und *Network Operations and Management Symposium (NOMS)* [13]. Die reichhaltige Menge an vorhandener Literatur zum Thema Policy-based Management verbietet es, die unterschiedlichen Arbeiten in diesem Gebiet eingehend vorzustellen. Ziel dieses Beitrags ist vielmehr, einen Überblick über den gegenwärtigen Stand ausgewählter Aktivitäten zu geben und Einstiegspunkte in die Literatur zu vermitteln, sowie aktuelle Fragestellungen und neue Anforderungen vorzustellen.

Die Gliederung des Beitrags ist wie folgt: Abschnitt 2 stellt die grundlegenden Anforderungen an Policy-basiertes Management und die damit verbundenen Fragestellungen vor. Eine zentrale Rolle spielt hierbei naturgemäß die Policy-Spezifikation, deren gegenwärtige Ansätze sich in zwei Kategorien einteilen lassen: Einerseits sind dies *sprachbasierte* Ansätze, deren Ziel die Definition formaler Policy-Sprachen ist; *modellbasierte* Ansätze hingegen definieren – analog zu Management-Informationsmodellen – objektorientierte Modelle zur Darstellung von Policies, die durch Vererbung an spezifische Szenarien angepaßt werden. Wir stellen die bekanntesten

Vertreter beider Ansätze in Abschnitt 3 vor und beschreiben deren Anwendungen. Abschnitt 4 stellt neue Fragestellungen vor, die sich aus der zunehmenden Verwendung von SLAs ergeben, sowie as dem Einsatz neuer dienstorientierter Architekturen *(Service-oriented Architectures, SOA)*, wie zum Beispiel Web Services. Der Beitrag endet mit einer Zusammenfassung in Abschnitt 5.

2. Grundlegende Anforderungen und Fragestellungen

Klassische Fragestellungen im Bereich des Policy-basierten Managements (vgl. z.B. [7, 40, 41]) betreffen zunächst die Festlegung geeigneter Terminologien und Architekturen (Abschnitt 2.1). Von zentraler Bedeutung ist das Problem der Policy-Spezifikation (Abschnitt 2.2), da die Wahl des Beschreibungsformats nachhaltige Auswirkungen auf die Möglichkeiten zur Verfeinerung von Policies (*Policy-Refinement*, beschrieben in Abschnitt 2.3) sowie der Erkennung und Behebung von Policy-Konflikten (Abschnitt 2.4) hat.

2.1. Policy-Architekturen

Policy-Architekturen definieren die am Entscheidungsprozeß beteiligten Rollen und Systeme im Sinne eines Organisationsmodells sowie deren Interaktionen. Zentrale Fragestellungen sind hierbei, wie ein Policy-basiertes Managementsystem grundsätzlich strukturiert ist, wie die Schnittstellen der einzelnen Komponenten definiert werden und mit welchen Kommunikationsprotokollen Policy-Information ausgetauscht wird. Eine Voraussetzung hierfür ist die Festlegung einer geeigneten Terminologie, wie es die *Internet Engineering Task Force (IETF)* in [30] getan hat. Ebenfalls besteht inzwischen weitgehend Einigkeit, wie die Architektur eines Policy-basierten Managementsystems strukturiert ist. Die IETF definiert in den RFCs 2748 [26] und 2753 [27] eine einfache Architektur, die aus folgenden Komponenten besteht und in Abbildung 1 dargestellt ist:

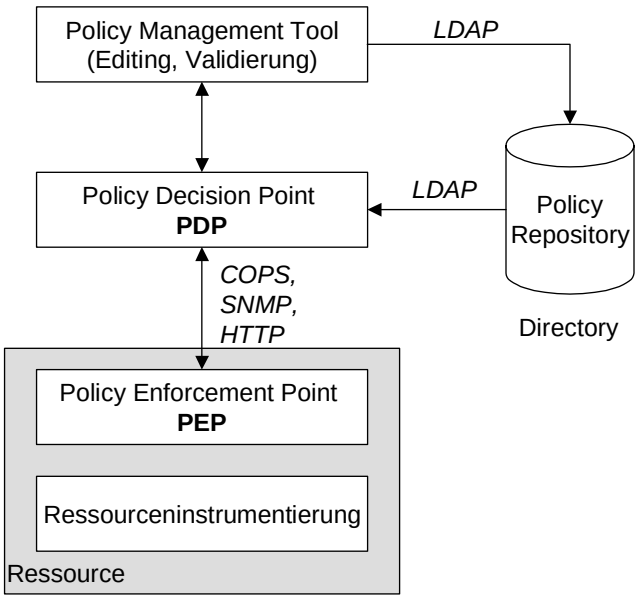

Abbildung 1: Die IETF Policy-Architektur (nach [26, 27])

- **Policy Management Tool**: Eine Komponente, mit der ein Administrator Policies definiert und validiert, sowie dem PDP und dem Policy Repository (s.u.) bekanntgibt. Ferner sollte ein Policy Management Tool die Verfeinerung von Policies unterstützen sowie prüfen, ob eine Menge von Policies widerspruchsfrei ist. Policy Management Tools können als eigenständige Komponenten implementiert oder in eine Managementplattform integriert sein.
- **Policy Decision Point (PDP)**: Eine Komponente, die Policies interpretiert und festlegt, welche Aktionen auf welchen Zielsystemen ausgeführt werden. Ein PDP nimmt von einen PEP (s.u.) kommende Ereignismeldungen entgegen, greift auf im Policy Repository gespeicherte Policies zu, erkennt und behebt zur Laufzeit Konflikte zwischen Policies (vgl. hierzu Abschnitt 2.4) und übermittelt die resultierenden (Konfigurations-)Aktionen an den PEP. Der Entscheidungsvorgang wird als *Policy-Refinement* bezeichnet (siehe Abschnitt 2.3) und geschieht durch eine im PDP laufende *Policy Engine*. In RFC 2753 wird der PDP als *Policy Server* bezeichnet; dieser Begriff hat sich jedoch nicht durchgesetzt.
- **Policy Enforcement Point (PEP)**: Eine Komponente, die beim Auftreten von Ereignissen den PDP benachrichtigt, sowie vom PDP kommende Policy-Aktionen interpretiert und ausführt, z.B. durch Setzen von Konfigurationsparametern, die von der Instrumentierung einer Ressource zur Verfügung gestellt werden.
- **Policy Repository**: Ein System zur persistenten Speicherung von Policies. In der Regel ist dies ein LDAP-Directory; alternativ kann ein Datenbanksystem verwendet werden. Da Policies systemübergreifend spezifiziert werden, ist es wichtig, daß sowohl der PEP als auch das Policy Repository die Speicherung von Policies in einem einheitlichen, standardisierten Beschreibungsformat unterstützen (vgl. Abschnitte 2.2 und 3).

PDP und PEP können entweder örtlich getrennt sein (wie in Abbildung 1), oder sich auf dem selben System befinden. Während ein PDP mehrere PEPs verwalten kann, ist einem PEP exakt ein PDP zugeordnet, um konfliktäre Aktionen auszuschließen. Zur Kommunikation zwischen PDP und PEP können unterschiedliche Protokolle verwendet werden, wie z.B. COPS [26], SNMP, HTTP oder telnet (bzw. proprietäre Kommandozeilen-Interfaces).

2.2. Policy-Spezifikation

Eine Policy-Spezifikation legt das Beschreibungsformat fest, mit dem Policies definiert werden. Die Definition von Policies wird durch den Administrator vorgenommen, der hierfür entweder direkt das vorgegebene Beschreibungsformat nutzt, oder sich spezieller graphischer Editoren bedient, die von einem benutzerfreundlichen Definitionsformat in eine maschinell verarbeitbare Policy-Notation abgebildet werden. Bei der Policy-Spezifikation gibt es gegenwärtig zwei Ansätze, deren Unterschiede im wesentlichen von der Beantwortung der Frage herrühren, ob Policies besser in einer eigenständigen Sprache oder sprachunabhängig, d.h. in einem Modell, spezifiziert werden sollen: Man unterscheidet deshalb zwischen sprach- und modellbasierten Ansätzen, auf die wir in den

Abschnitten 3.1 und 3.2 genauer eingehen. Für beide Ansätze gibt es Vorschläge, die auf funktionsfähige Implementierungen verweisen können.

2.3. Policy-Refinement und Policy-Hierarchien

Die Abbildung von abstrakten Policies in konkretere Policies wird in der Literatur als *Policy-Refinement* bezeichnet. Ziel ist hierbei, eine Transformation von Business Policies, die von (wirtschaftlichen) Zielvorgaben (in der Literatur als *Goals* bezeichnet) wie SLAs abgeleitet sind, in Technische Policies, oftmals (Sequenzen von) Konfigurationsaktionen, vorzunehmen. Die Beziehungen zwischen Zielvorgaben und technischen Policies bilden eine *Policy-Hierarchie* [22]. Die Analyse dieser Beziehungen gibt Aufschluß darüber, ob Policies die definierten Ziele unterstützen und die am Policy-Prozeß beteiligten Subjekte in der Lage sind, die Policies umzusetzen (d.h. über ausreichende Rechte verfügen). Ferner sollte es möglich sein, zu bestimmen, welche Policies von Änderungen der Zielvorgaben betroffen sind bzw. ob sich aus Änderungen der Zielvorgaben Policy-Konflikte ergeben (siehe hierzu Abschnitt 2.4). Diesen Anforderungen steht jedoch das generelle Problem entgegen, daß eine Automatisierung dieses Vorgangs im allgemeinen Fall nicht möglich ist [36], weil das Spektrum möglicher Policy-Aktionen von vornherein nicht bekannt ist und insbesondere von technischen und bereichsspezifischen Gegebenheiten abhängt (wie z.B. Art und Anzahl der Konfigurationsparameter von Managementressourcen). Für spezielle Anwendungsfälle ist es möglich, sogenannte *Refinement Patterns* zu definieren, die zumindest eine teilweise Automatisierung zulassen bzw. einen Administrator bei der Verfeinerung von Policies unterstützen [1].

2.4. Erkennung und Behebung von Policy-Konflikten

Policy-Konflikte entstehen, wenn unterschiedliche Policies für dieselben Subjekte und Zielsysteme Aktionen definieren, die zueinander im Widerspruch stehen. Komplexe Systeme erfordern naturgemäß eine große Anzahl von Policies, was die manuelle Überprüfung der Widerspruchsfreiheit einer Policymenge erschwert. Ein zusätzliches Problem ist die in der Einleitung angesprochene hohe Dynamik heutiger verteilter Systeme, die das Einfügen neuer Policies in eine Policymenge und die Modifikation sowie das Entfernen existierender Policies erfordert, was wiederum Auswirkungen auf die Widerspruchsfreiheit der Menge haben kann. Idealerweise sollte ein Policy-System in der Lage sein, vor jeder Modifikation einer Menge von Policies automatisch zu prüfen, ob die Gesamtmenge auch nach der Modifikation widerspruchsfrei ist.

Offensichtlich ist das Problem der automatischen Erkennung (und anschließenden Behebung) von Policy-Konflikten eng mit Fragestellungen der mathematischen Logik und der Konstruktion von Theorembeweisern verwandt. In der Tat basierten erste Ansätze (speziell im Bereich der Sicherheitspolicies) auf der Prädikatenlogik erster Stufe, jedoch erwies sich dies als nicht tragfähig, da notwendige logische Operatoren (wie die Negation) in Zusammehang mit rekursiven Regeln zu unentscheidbaren Programmen führen können [8]. Entsprechende Modifikationen der Prädikatenlogik erster Stufe (wie z.B. die sog. *Stratified Logic*) zeigen erste Erfolge in spezifischen Szenarien (wie z.B. der Sicherung eines relationalen Datenbanksystems gegen unbefugte Lese- und

Updateoperationen), jedoch ist man von einer allgemein anwendbaren Notation, die die automatische Erkennung von Policy-Konflikten gestattet, noch weit entfernt.

Alternative Ansätze zur Konflikterkennung reduzieren das Problem auf einen domänenspezifischen Teilaspekt; wie z.B. Policies für das Netzmanagement oder SLAs: Einfache Mechanismen zur Konflikerkennung beruhen hier meist auf einem Vergleich identisch strukturierter Elemente (wie z.B. Zeit- und Datumsfeldern, Dienstklassen oder Netzadressen) sowie der Policy-Bedingung bzw. der Policy-Aktion. Verma [40] schlägt die Zerlegung der Policy-Bedingung in abhängige und unabhängige Teile vor, die einen n-dimensionalen Raum beschreiben, der dann mit Hilfe geometrischer Algorithmen auf Überschneidungen untersucht wird. In der Praxis verlagert man jedoch gegenwärtig die Veranwortung für die Widerspruchsfreiheit einer Policymenge auf den Administrator, wobei die Möglichkeit zur Bildung von Ressourcen-, Nutzer- und Dienstklassen diese Aufgabe etwas vereinfacht.

Alle zuvor genannten Ansätze beruhen auf der statischen Analyse der Syntax einer Menge von Policies. Bislang noch kaum erforscht sind die Wechselwirkungen von Policies mit dem Zustand eines Systems, um sicherzustellen, daß die unterschiedlichen Zustände eines Systems keine Konflikte zur Laufzeit hervorrufen. Hierfür ist jedoch eine Überprüfung der Policymenge hinsichtlich aller möglichen Systemzustände erforderlich, was zusätzliche Komplexität mit sich bringt.

Zur Behebung erkannter Konflikte beruhen gängige Ansätze meist auf der Zuweisung von Prioritäten zu Policies, wobei im Falle eines Konfliktes jeweils die Policy mit der höchsten Priorität ausgeführt wird [28]. Prioritäten können den unternehmerischen Wert oder die Wichtigkeit einer Policy bzw. der von der Policy betroffenen Ressource(n) wiederspiegeln. Allerdings ist hierbei nicht auszuschließen, daß es zu einem Konflikt mit Policies gleicher Priorität kommt, insbesondere wenn die Definition von Policies von unterschiedlichen Personen vorgenommen wird. Andere Heuristiken beruhen darauf, der jeweils restriktiveren und/oder spezifischeren Policy den Vorrang zu geben, um im Zweifelsfalle – durch Unterlassen einer Aktion – auf „Nummer sicher“ zu gehen.

Es ist offensichtlich, daß die Erkennung von Policy-Konflikten sowie deren Behebung ein Bereich bleibt, der aufgrund seiner hohen Komplexität auch in Zukunft wesentliche Forschungsanstregungen erfordert, um zu praktikablen Lösungen zu kommen.

3. Ansätze zur Policy-Spezifikation

Zur Definition eines geeigneten Policy-Beschreibungsformates sind im Laufe der Zeit zahlreiche Lösungsvorschläge erarbeitet worden: Frühe Arbeiten zur Policy-Spezifikation, wie z.B. die von Wies [42], beruhen auf der Definition von Policy-Templates. Solche Schablonen bestehen aus einer vordefinierten Anzahl von Datenfeldern mit festgelegter Semantik, die von einem Administrator ausgefüllt und anschließend persistent gespeichert werden. Eine Policy-Engine ist dann in der Lage, bei Eintreffen eines Ereignisses die Policy-Templates auszuwerten und zu entscheiden, welche Aktionen ausgeführt werden sollen. Die Attraktivität des Template-Konzepts besteht in der Präzision der Policy-Definitionen und damit vereinfachter maschineller

Verarbeitung. Dieser Vorteil wird jedoch durch eine Beschränkung der Flexibilität erkauft, so daß sich der Einsatz von Templates auf sehr spezifische Szenarien beschränkt. Templates können darüber hinaus in sprachbasierten Ansätzen als Editierhilfe verwendet werden.

In jüngster Zeit haben sich im wesentlichen zwei verschiedene Ansätze zur Policy-Spezifikation durchgesetzt, die flexibler, aber auch deutlich komplexer als Templates sind: Der grundlegende Unterschied zwischen *sprachbasierten* und *modellbasierten* Ansätzen besteht in der Frage, ob es günstiger ist, Policies explizit in einer interpretierbaren Sprache oder implementierungsunabhängig zu spezifizieren. Wir werden zunächst zwei sprachbasierte Ansätze, *Ponder* sowie das *Web Services Policy Framework*, vorstellen, und behandeln anschließend das *Policy Core Information Model (PCIM)* und seine neuesten Erweiterungen. PCIM wurde gemeinsam von der Internet Engineering Task Force (IETF) sowie der Distributed Management Task Force (DMTF) entwickelt.
Auch wenn noch keine dieser Arbeiten zum gegenwärtigen Zeitpunkt ein breites Anwendungsspektrum abdeckt, so besteht Hoffnung, daß die bisher in der Praxis verwendeten Policy-Dokumente in natürlicher Sprache [9] in nicht allzu ferner Zukunft durch automatisierte Policy-basierte Managementsysteme ersetzt werden.

3.1. Sprachbasierte Ansätze

Wir stellen im folgenden die beiden derzeit am weitesten verbreiteten sprachbasierten Ansätze vor. Hierbei handelt es sich um *Ponder*, eine Sprache, die am Imperial College London zur Spezifikation von Sicherheits- und Management-Policies entwickelt wurde, sowie das *Web Services Policy Framework*, das von einem Industriekonsortium erarbeitet wurde und gegenwärtig dem W3C zur Standardisierung vorliegt. Ein weiterer Vertreter einer Policy-Sprache ist die *Policy Description Language (PDL)* [23], die in den Bell Laboratories entstanden ist und in Lucent Switching Produkten verwendet wird. Eine Übersicht über weitere Policy-Sprachen mit dem Fokus auf Kommunikationsnetze ist in [37] enthalten.

3.1.1. Ponder

Ponder ist eine deklarative, objektorientierte Sprache zur Spezifikation von Sicherheits- und Managementpolicies, die am Imperial College entwickelt wurde [6]. Die Sprache basiert auf Forschungsergebnissen, die über einen Zeitraum von 10 Jahren erarbeitet wurden.

Eng mit dem Begriff einer Policy verbunden ist in Ponder das Domänenkonzept, das die Gruppierung von Objekten ermöglicht, auf die sich Policies beziehen. Solche (gegebenenfalls verschachtelten oder sich überschneidenden) Gruppierungen können entweder nach organisatorischen und geographischen Gesichtspunkten erfolgen, oder auch nach dem Typ der Objekte. Eine Domäne hat Referenzen auf die in ihr enthaltenen Objekte. Objekte können entweder *Subjects* oder *Targets* sein. Subjekte sind Akteure (Personen oder automatisierte Manager-Komponenten), die in der Regel nach organisatorischen Kriterien in Subjektdomänen (*Subject Domains*) zusammengefaßt werden und Policies auf Targets bzw. Targetdomänen anwenden. Subjektdomänen

entsprechen Benutzergruppen wie z.B. Netzadministrator, Systemverwalter, Manager und abstrahieren von konkreten Personen. Die Definition von Policies auf der Basis von Subjektdomänen hat den Vorteil, daß Policies unverändert bleiben, wenn sich das Subjekt einer Person (d.h. ihre Funktion innerhalb einer Organisation) ändert.

Ponder unterscheidet zwischen sieben verschiedenen Policy-Typen, von denen mittels Vererbung weitere Typen abgeleitet werden können.Aus Platzgründen beschränken wir uns auf die drei im Systemmanagement am häufigsten verwendeten Arten.

Autorisierungspolicies (*Authorization Policies*) sind Sicherheitspolicies und dienen der Zugriffskontrolle, um Ressourcen vor unbefugtem Zugriff zu schützen. Autorisierungspolicies definieren, welche Aktionen ein Subjekt auf einem Target ausführen darf (positive Autorisierungspolicies), und welche ihm untersagt sind (negative Autorisierungspolicies). Zum Beispiel legt die folgende, [5] entnommene, negative Autorisierungspolicy fest, daß es Trainee Test Ingenieuren untersagt ist, zwischen 9 und 17 Uhr Performance Tests auf Routern auszuführen. Die Policy wird in der Domäne `/negativeAuth` gespeichert.

```
inst auth– /negativeAuth/testRouters {
      subject /testEngineers/trainee ;
      action performance_test() ;
      target <routerT> /routers ;
}
```

Obligationspolicies (*Obligation Policies*): Eine Obligation ist eine ereignisgesteuerte Regel, die Bedingung(en) und Aktion(en) enthält. Obligationspolicies spezifizieren die Aktionen, die von Subjekten auf Targets ausgeführt werden, wenn bestimmte Ereignisse eintreffen. Das folgende Beispiel einer Ponder-Obligationspolicy illustriert das Triggern der Policy durch das Ereignis „dreifacher fehlerhafter Login derselben UserID“, sowie die beiden durch den Sequenzoperator `->` verknüpften Aktionen „sperre user mit UserID in der Domäne `/Nregion/users`“ sowie anschließendes „Logging der UserID“; letztere wird durch den Sicherheitsadministrator `SecAdmin` ausgeführt.

```
inst oblig loginFailure {
      on 3*loginfail(userid) ;
      subject s = /NRegion/SecAdmin ;
      target <userT> t = /NRegion/users ^ {userid} ;
      do t.disable() -> s.log(userid) ;
}
```

Zusammengesetzte Policies (*Composite Policies*): Composite Policies sind ein Mittel zur Gruppierung zusammengehöriger Policies. Insgesamt kennt Ponder 4 Arten von Composite Policies: *Groups*, *Constraints*, *Management Structures*, und *Roles*. So fassen zum Beispiel *Roles* Aufgaben und Berechtigungen von Subjekten zusammen. Eine Rolle ist demzufolge eine Gruppierung von Autorisierungs- und Obligationspolicies (s.o.), die dasselbe Subjekt haben. Das folgende, aus [6] entnommene, Beispiel einer Role-Definition, bestehend aus zwei Instanzen von Obligationspolicies sowie einer Autorisierungspolicy, verdeutlicht das Ponder-Rollenkonzept. Der Rollentyp

`ServiceEngineer` eines Mobilfunkproviders ist zuständig für die Entgegennahme von Kundenbeschwerden und –anfragen. Er bedient sich hierzu einer Datenbank `CallsDB`, die Informationen über Kunden sowie deren Gespräche speichert. Die erste Obligationspolicy `serviceComplaint` wird durch ein Kundenbeschwerde-Ereignis getriggert, das die Anschlußnummer des Kunden als Parameter hat. Daraufhin führt der Service Engineer eine Sequenz von 3 Aktionen auf dem Target `callsDb` aus.

```
type  role ServiceEngineer (CallsDB callsDb) {

      inst oblig serviceComplaint {
            on      customerComplaint(mobileNo) ;
            do      t.checkSubscriberInfo(mobileNo, userid) ->
                    t.checkPhoneCallList(mobileNo) ->
                    investigate_complaint(userId);
            target t = callsDb ; // calls register }

      inst oblig deactivateAccount { . . . }

      inst auth+ serviceActionsAuth { . . . }
}
```

Für eine vollständige Beschreibung aller Policy-Typen und ihrer Notation wird auf die ausführliche Ponder-Sprachspezifikation [5] verwiesen. Implementierungen eines Ponder Policy Editors und Compilers sowie des Ponder Management Toolkits sind frei verfügbar [25]. Das Ponder-Toolkit wird kontinuierlich weiterentwickelt und in unterschiedlichen Szenarien auf seine Anwendbarkeit in der Praxis untersucht: [19] beschreibt die Verwendung von Ponder zur Abbildung von Dienstgütespezifikationen auf Policies für *Differentiated Services (DiffServ)* Netze; darauf aufbauend wird in [20] ein Ansatz für das Management CIM-instrumentierter DiffServ-Router vorgestellt.

3.1.2. Web Services Policy Framework

Das Web Services Policy Framework (WS-Policy) wurde in den vergangenen 2 Jahren federführend von Microsoft und IBM als Teil einer *Service-Oriented Architecture (SOA)* entwickelt. Das Ziel des Frameworks ist, Policies eines Dienstes (Web services) potentiellen Dienstnutzern zur Verfügung zu stellen, damit dessen Diensteigenschaften und Nutzungsvoraussetzungen interessierten Benutzern bekannt sind, bevor sie den Dienst nutzen. Das Framework läßt dem Spezifizierer großen Spielraum hinsichtlich der Objekte, die veröffentlicht werden können und geht demzufolge über die übliche Interpretation des Begriffs der Policy im Systemmanagement hinaus.

WS-Policy besteht aus einer Reihe von Spezifikationen: Das *Web Services Policy Framework* [43] beschreibt eine Sprache für Policy-Ausdrücke unabhängig vom eigentlichen semantischen Inhalt einer Policy. *Web Services Policy Attachment* [44] spezifiziert, wie eine Policy mit Ihrem Subjekt assoziiert werden kann, beispielsweise einem Dienst oder einer einzelnen Operation in einer *Web Service Description Language (WSDL)* Spezifikation eines Dienstes. Zur Beschreibung des eigentlichen Inhalts der Policy müssen domänenspezifische Beschreibungsmethoden benutzt werden. Ein erster solcher Ansatz ist der *Web Services Security Policy* Spezifikationsentwurf [45]. Weitere,

zum Beispiel für Transaktionen, sind in Diskussion. Als Erweiterungsmechanismus dient die Subtypisierung der Beschreibungselemente, die in XML definiert sind.

WS-Policy definiert Policy-Ausdrücke als Verknüpfung von zugesicherten Eigenschaften (*Assertions*). Assertions beschreiben Anforderungen (*Requirements*), Fähigkeiten (*Capabilities*) und Präferenzen (*Preferences*). Die Beschreibung des Inhalts der Assertion ist domänenspezifisch. Der Typ der Assertion – Requirement oder Capability – und die Preference werden als Attribute der Assertion beschrieben. Assertions werden mit Policy-Operatoren verknüpft und gruppiert. Der `All` Operator definiert, dass alle Assertions, auf die er sich bezieht, gültig sein müssen. `ExactlyOne` und `OneOrMore` haben die entsprechende Semantik. Operatoren können auch ineinander verschachtelt sein. Das folgende Beispiel aus [43] illustriert die WS-Policy Syntax:

```
<wsp:Policy xmlns:wsse="..." xmlns:wsp="...">
     <wsp:ExactlyOne>
         <wsse:SecurityToken wsp:Usage="wsp:Required"
                             wsp:Preference="100">
             <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
         </wsse:SecurityToken>
         <wsse:SecurityToken wsp:Usage="wsp:Required"
                             wsp:Preference="1">
             <wsse:TokenType>wsse:X509v3</wsse:TokenType>
         </wsse:SecurityToken>
     </wsp:ExactlyOne>
</wsp:Policy>
```

Die Policy besagt, daß genau eine Assertion erfüllt sein muß, nämlich entweder die Assertion „SecurityToken“ mit dem TokenType „Kerberosv5TGT“ oder die Assertion „SecurityToken“ mit dem TokenType „X509v3“. Beide Assertions sind Requirements an den Benutzer des Dienstes, ausgedrückt durch das „Usage“-Attribut. Kerberos-Tokens werden X509v3-Tokens vorgezogen, was durch die größere Präferenzzahl ausgedrückt wird. Kurz gesagt: Ein Benutzer kann zur Nutzung des Dienstes entweder Kerberos oder X509v3 Tokens benutzen, jedoch sollte Kerberos die erste Wahl sein.

Die WS-PolicyAttachment Spezifikation beschreibt, wie eine Policy mit ihrem Subjekt verknüpft werden kann. Eine Policy kann entweder in eine WSDL-Datei an das Element hinzugefügt werden, auf das es sich bezieht, oder sie kann außerhalb eines WSDL-Kontexts spezifiziert werden. Mittels eines *Attach*-Ausdrucks kann dann das Subjekt auf domänenspezifische Art referenziert werden, zum Beispiel durch Angabe eines eindeutigen Namens oder einer URL.

3.2. Modellbasierte Ansätze

3.2.1. Das IETF/DMTF Policy Modell

Das gemeinsam von den Policy Working Groups der IETF und der DMTF entwickelte *Policy Core Information Model (PCIM)* [10, 28] ist ein generisches objektorientiertes Informationsmodell, das auf dem *Common Information Model (CIM)* [4] basiert. Es

erweitert CIM um Funktionalität zur Definition von Policies und stützt sich auf andere, im sog. CIM *Core Model* definierte Objektklassen (wie z.B. *ManagedElement*, *System*, *AdminDomain*, *Collection*) ab. Das Modell ist generisch, weil domänenspezifische Erweiterungen (z.B. für IPSec, DiffServ und IntServ, siehe Abschnitt 3.2.2) durch Subklassenbildung von PCIM-Klassen vorgenommen werden.

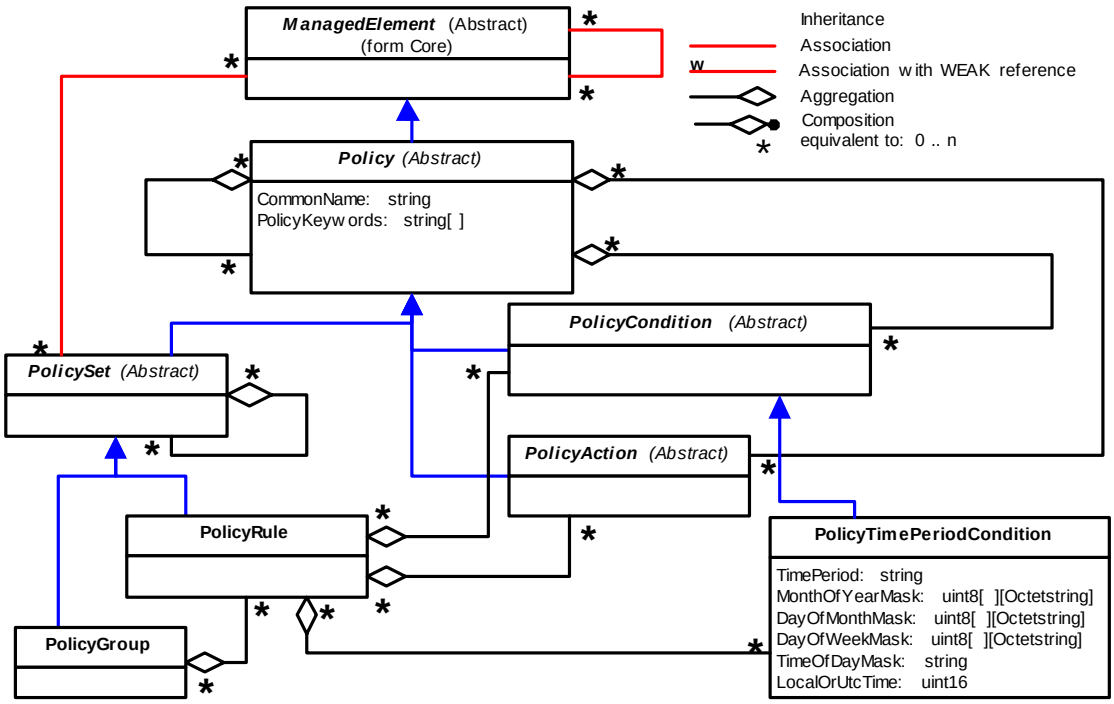

Abbildung 2: Das IETF/DMTF Policy Modell (Ausschnitt) [10, 28]

Abbildung 2 zeigt einen Ausschnitt der zentralen Klassen des aktuellen PCIM-Modells, wobei wir aus Platzgründen sowohl auf die Namen der Assoziationen und Aggregationen, als auch auf den Großteil der Klassenattribute verzichtet haben. Die abstrakte Klasse *Policy* ist die Basisklasse des PCIM und abstrahiert von wesentlichen Policy-Elementen (stellt also nicht die Policy selbst dar) wie *PolicyCondition*, *PolicyAction* und *PolicySet*. Eine einzelne Policy wird in PCIM als *PolicyRule* bezeichnet und kann eine beliebige Menge von Policy-Bedingungen und -aktionen enthalten (siehe die beiden Aggregationen mit Kardinalität *, d.h. [0..n], die *PolicyRule* jeweils mit *PolicyCondition* und *PolicyAction* assoziieren). RFC 3060 betont demzufolge die Wiederverwendbarkeit von Policy-Bedingungen und Policy-Aktionen (ggf. in mehreren Policy-Regeln), da sie jeweils als eigenständige Objektklassen modelliert sind und in einer beliebigen Anzahl von Policy-Regeln verwendet werden können. Mehrere *PolicyRules* können zu einer *PolicyGroup* zusammengefaßt werden, wobei auch hier gilt, daß eine *PolicyRule* in mehreren *PolicyGroups* enthalten sein kann. Ferner ist ersichtlich, daß eine Policy im IETF/DMTF-Sinne ein Ausdruck der Form `if <condition(s)> then <action(s)>` ist, wobei *PolicyConditions* logische Ausdrücke in konjunktiver oder disjunktiver Normalform sind. Eine *PolicyAction* muß ausgeführt werden, wenn die Auswertung der *PolicyCondition(s)* `true` ergibt. Die Spezifikation konkreter Aktionen geschieht nicht in PCIM, sondern bleibt den domänenspezifischen, d.h. von PCIM

abgeleiteten, Policy-Modellen vorbehalten. PCIM selbst kennt lediglich drei einfache Arten von Aktionen (*NetworkPacketAction*, *RejectConnectionAction* und *VendorPolicyAction*) und bietet eine auf dem *Composite* Design Pattern basierende Möglichkeit, mehrere *PolicyActions* zu einer *CompoundPolicyAction* zusammenzufassen. Zu beachten ist hierbei, dass PCIM keine Vorgaben bezüglich des Formats der Terme macht, aus denen sich *PolicyConditions* und *PolicyActions* zusammensetzen.

Im Vergleich zu Ponder Obligations-Polices (siehe Abschnitt 3.1.1) fehlt in PCIM eine explizite Ereignisbedingung, die die Auswertung einer Policy veranlaßt. Es wird angenommen, daß das Triggern von Policies implizit geschieht d.h. entweder durch den Administrator, durch Auftreten einer speziellen Verkehrssituation (z.B. Überlast, IntServ-Verbindungsaufbauanfrage), oder durch Eintreffen von Datenpaketen. Zusätzlich kann der Gültigkeitszeitraum einer Policy definiert werden: Hierfür hat PCIM eine spezielle Policy-Bedingung, die durch die Klasse *PolicyTimePeriodCondition* festgelegt und ebenfalls in Abbildung 2 dargestellt ist. Um zu bestimmen, wann eine *PolicyRule* gültig ist werden die letzten fünf Attribute der Klasse *PolicyTimePeriodCondition* mit einem logischen `UND` verknüpft und die entstehende Maske über den in *TimePeriod* definierten Zeitraum gelegt, womit komplexe Zeitaussagen wie „*Im Gesamtzeitraum vom 8. Januar 2003, 7 Uhr bis zum 31. Dezember, 23 Uhr 59 gilt die Policy in den Monaten Mai, Juli und Dezember jeweils am ersten und letzten Tag des Monats, sofern dies ein Montag ist, von 9 Uhr 15 bis 21 Uhr 30. Es gilt die Ortszeit des Systems.*“ gemacht werden können. *PolicyTimePeriodCondition* kann somit flexibel in einem logischen Ausdruck mit anderen Bedingungen kombiniert und in einer *PolicyRule* zusammengefaßt werden.

3.2.2. Erweiterungen und Anwendungen des PCIM Modells

PCIM wird kontinuierlich weiterentwickelt, was in der DMTF durch eine alle 6 bis 9 Monate neu erscheinende Version der CIM Schemas (gegenwärtig ist dies die Version 2.8) geschieht. So entspricht PCIM [28] dem Entwicklungsstand des CIM Policy Schemas in der Version 2.4. Bei der IETF hingegen führen Revisionen und Erweiterungen bereits publizierter RFCs zur Vergabe einer neuen RFC-Nummer, sodaß die seit dem Erscheinen von PCIM erfolgten Änderungen in den *PCIM Extensions (PCIMe)* [31] dokumentiert sind. Folglich beschreibt PCIMe anstelle des überarbeiteten Modells alle Modifikationen an der PCIM-Klassenstruktur, einzelnen Objektklassen, deren Attributen und Assoziationen, die im Zeitraum von Februar 2001 bis Januar 2003 erfolgt sind. Die Modifikationen reflektieren einerseits Änderungen am CIM Policy Modell, die durch Modell-Implementierungen notwendig wurden; der überwiegende Teil der zahlreichen Änderungen resultiert aus den Anforderungen drei neuer domänenspezifischer Policy-Modelle, die von PCIM (bzw. PCIMe) abgeleitet sind. Diese drei neuen Modelle sind:

- Das *IPsec Configuration Policy Information Model (ICPM)* [32]. Es behandelt die Definition von Policies für IP Security (IPsec) und das Internet Key Exchange Protocol (IKE). ICPM entspricht weitgehend dem CIM IPsec Policy Schema Version 2.8.
- Das *Policy Quality of Service (QoS) Information Model (QPIM)* [33], das sich mit der Definition von Policies für IntServ und DiffServ befaßt.

- Das *Information Model for Describing Network Device QoS Datapath Mechanisms (QDDIM)* [34]. Es basiert auf QPIM und erweitert es um Mechanismen zur Konfiguration der QoS Mechanismen von Netzkomponenten. Zu QPIM und QDDIM existieren derzeit keine entsprechenden CIM Schemas.

Es existieren einigen Beispiele erster Anwendungen von PCIM für QoS-Management in DiffServ-Netzen: [21] bechreibt die Implementierung eine PCIM-Systems zur Konfiguration von DiffServ-Netzkomponenten. Der PDP, implementiert auf der Basis der IETF Script MIB [29], spielt die Rolle eines Mid-level Managers, der aus Policies geeignete Werte für Konfigurationsparameter von in Linux implementierten DiffServ-Routern ableitet. Die Konfigurationsparameter werden mittels SNMP gesetzt.
[11] beschreibt einen Policy-basierten Ansatz zur Dimensionierung von Diffserv-Netzen, dem eine netzweite und damit systemübergreifende Betrachtungsweise zugrunde liegt.

4. Neue Anforderungen

Der folgende Abschnitt stellt 2 Szenarien vor, die durch die Verwendung neuer Diensterbringermodelle und -technologien neue Fragestellungen von hoher praktischer Relevanz aufwerfen. Wir gehen zunächst auf das Problem der Abbildbarkeit von SLAs auf Policies ein, das in Zusammenhang mit der steigenden Verbreitung von Web und Application Service Hosting eine zunehmende Brisanz erhält. Anschließend stellen wir neue Anforderungen an die Berechnungsalgorithmen für QoS-Kennzahlen vor, die sich aus der Aggregation von Diensten ergeben.

4.1. Abbildung von SLAs auf Policies

Service Level Agreements und Policies stehen in engem Zusammenhang. Während Policies in der Regel das Verhalten von Systemen innerhalb einer Organisation spezifizieren, beinhalten SLAs Service-Garantien, die eine Organisation einer anderen gibt. Bei Organisationen kann es sich sowohl um Einheiten desselben Unternehmens, als auch um unterschiedliche Unternehmen handeln. Inhaltlich überschneiden sich Sprachen und Modelle für SLAs, wie zum Beispiel *Web Service Level Agreement (WSLA)* [15, 18], in weiten Bereichen mit Policies. Einige Autoren, zum Beispiel Verma [38], unterscheiden begrifflich nicht zwischen externer Sicht und interner Verhaltensregel.

Werden Dienste extern auf der Basis von SLAs angeboten, stellt sich die Frage der SLA-Transformation in interne Policies, die dann vom Policy-Managementsystem des Diensterbringers interpretiert werden können. Allerdings weicht die externe Sicht auf einen Dienst in der Regel von der inneren Sicht ab: Im Fall eines Web Service oder eines Web Hosting Dienstes werden im SLA zu erreichende Ziele, beispielsweise die *durchschnittliche Antwortzeit (DAZ)*, definiert. Das System, das den Dienst erbringt, besteht aus einer Menge von Application-Servern mit der Dienst-Applikation, einem Dispatcher, der Benutzeranfragen Servern zuordnet und einem Workload Manager (dem *Policy Decision Point*, siehe Abschnitt 2.1), der das Verhalten des Dispatchers (des *Policy Enforcement Points*) konfiguriert. Abbildung 3 illustriert diese typische Konfiguration, wie sie zum Beispiel in [16] beschrieben wird.

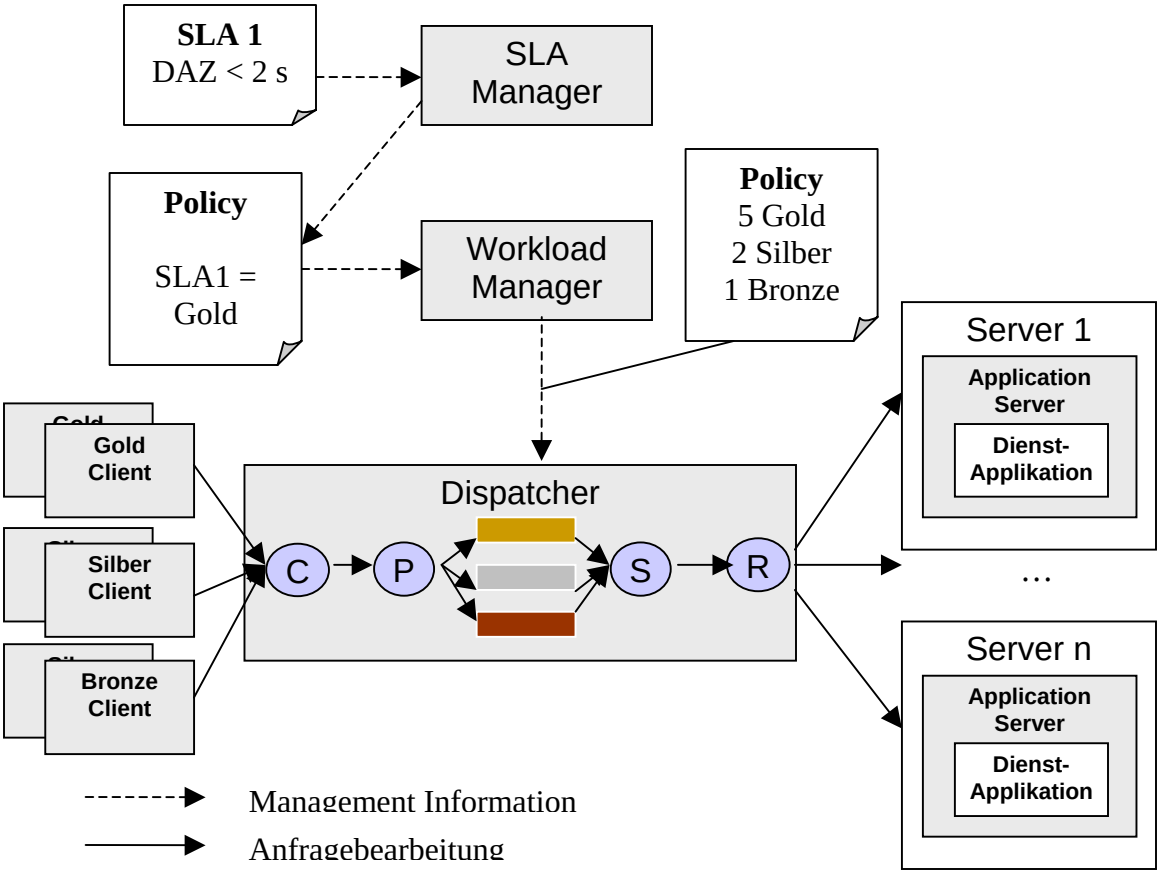

Abbildung 3: Architektur eines Web Hosting Systems

Der Dispatcher arbeitet mit einer Menge von Warteschlangen, die jeweils einer Dienstklasse entsprechen. Client-Anfragen werden

1. klassifiziert (C) und einer Warteschlange zugeordnet.
2. Ferner wird geprüft, ob die Warteschlange noch freie Kapazität hat (P).
3. Der Scheduler entnimmt den Warteschlangen Anfragen (S),
4. der Router (R) ordnet sie einem verfügbaren Server zu.

Um das Verhalten des Dispatchers zu konfigurieren, können zwei Konfigurationsparameter gesetzt werden:

- die jeweiligen Längen der Warteschlangen, und
- das Verhältnis, in dem der Scheduler den jeweiligen Warteschlangen Anfragen entnimmt.

Das interne Verhalten des Systems hängt also davon ab, welchen Dienstklassen die Anfragen zugeordnet sind und in welchem Verhältnis sie den Warteschlangen entnommen werden. Dies ist Gegenstand der internen Policy. Durch einen SLA Manager müssen nun die in einem SLA gegebenen Garantien in interne Policies übersetzt werden. Dies kann auf Basis einer (manuellen) Klassifikation von Antwortzeiten zu Serviceklassen automatisch geschehen. Aus der Menge von SLAs und ihrer erwarteten und realisierten Anfragevolumina leitet der Workload Manager die Warteschlangengewichte ab. Einen Algorithmus hierfür stellen zum Beispiel Levy et al. vor [16]. Das Beispiel illustriert die Notwendigkeit und Praktikabilität der

Unterscheidung zwischen Innensicht und Aussensicht des Dienstmanagements sowie den Zusammenhang zwischen SLAs und Policies. Die Übersetzung erfolgt in aller Regel nicht vollständig automatisch.

4.2. Dienstorientierte Architekturen: Web Services

Wie im vorherigen Abschnitt diskutiert, ergänzen sich SLAs und Policies, indem Policy-basierte Systeme sich gut als Umsetzungsplattformen für SLAs eignen. Im Umfeld von Web Services und dienstorientierten Architekturen (*Service-oriented Architecture, SOA*) im Allgemeinen werden Applikationen aller Art und Größe als Dienste aufgefaßt, die unter anderem über Organisationsgrenzen hinaus angeboten werden können. Diese Dienste werden in der Regel mit Qualitätsgarantien versehen, sei es auf der Basis einer SLA-Sprache wie WSLA, oder aber WS-Policy. Ein wichtiger Nutzen der SOA ist die Kombinierbarkeit von Diensten zu neuen, aggregierten Diensten. Aus Sicht des Anbieters eines solchen aggregierten Dienstes stellt sich nun die Frage, welche QoS-Kennzahlen der Dienst besitzt und welche QoS-Garantien realistisch sind.

Das in Abbildung 4 dargestellte Beispiel zeigt einen Wertpapierhandelsdienst, der aus anderen – unter Umständen externen – Diensten zusammengesetzt ist. Die Komposition erfolgt in aller Regel durch einen Workflow, der z.B. in der *Business Process Execution Language for Web Service (BPEL4WS)* [2] beschrieben ist und von einem Workflow Management System ausgeführt wird.

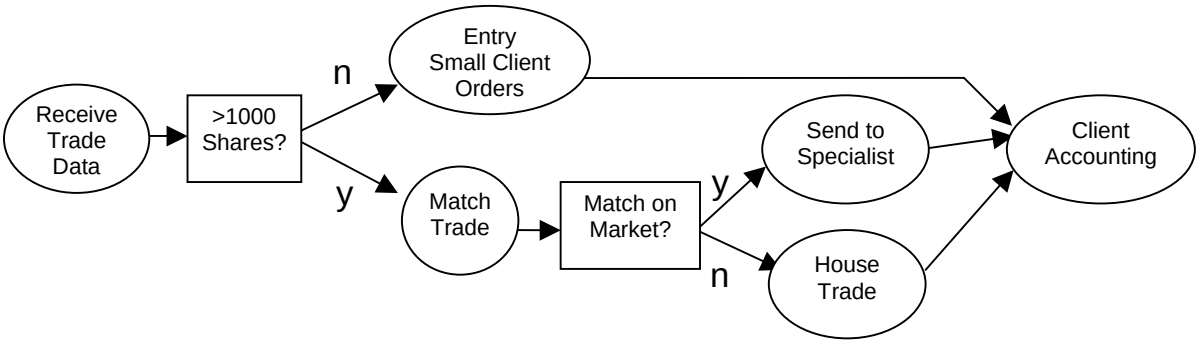

Abbildung 4: Dienstaggregation am Beispiel von Web Services

Der Workflow umfaßt mehrere Schritte in alternativen Zweigen. Jeder Schritt ist ein eigenständiger Web Service mit eigenen QoS-Kennzahlen, wie z.B. durchschnittliche Antwortzeit und Verfügbarkeit. Dabei stellt sich die Frage, welche QoS-Kennzahlen der kombinierte Service als Ganzes besitzt. Häufig werden diese stochastisch ausgedrückt, wie zum Beispiel: „*in 95% der Anfragen muß die Antwortzeit unter 200ms liegen*“ oder „*Die Verfügbarkeit über den Betrachtungszeitraum beträgt mindestens 98%*“. Durch die alternativen Pfade im Workflow ist die Bearbeitungszeit vom gewählten Pfad abhängig, sodaß durchschnittliche Antwortzeiten entsprechend der Wahrscheinlichkeit eines gewählten Pfades variieren. Es ist daher ausgesprochen schwierig, anhand lediglich *einer* Prozentgröße eine Antwortzeit zu garantieren, mit der die Dienstnutzer einverstanden sind. Möglicherweise müssen mehrere Intervalle angegeben werden, z.B. *„die Antwortzeit ist für 60% aller Fälle unter 200ms, für 85% unter 500ms, und für 95 % unter 750ms“*.

In Bezug auf die QoS-Kennzahl Verfügbarkeit stellt sich ebenfalls ein Problem: Prinzipiell ist die Verfügbarkeit das Produkt der Verfügbarkeiten einzelner Web Services auf einem Pfad. Mit zunehmender Anzahl von integrierten Diensten erreicht man sehr schnell einen Wert, der von Kunden nicht mehr akzeptiert wird. Die Auswirkungen dieses Problems sind umso gravierender, wenn Web Services über mehrere Ebenen von Diensterbringern aggregiert werden. Heutige Ansätze zum Policy-basierten Management tragen wenig zur Lösung dieses Problems bei. Eine mögliche Lösung liegt in der Nutzung der SOA zur dynamischen Auswahl von gleichwertigen Dienstimplementierungen unterschiedlicher Anbieter zur Laufzeit. Gegenstand einer Policy könnte dann sein, unter welchen Voraussetzungen Dienste gewählt werden, z.B. abhängig vom bisherigen Verlauf eines Workflows oder den unterschiedlichen Preisen von Dienstanbietern. Diese Problemstellung ist jedoch in der aktuellen Diskussion noch weitgehend unberücksichtigt.

5. Zusammenfassung und Ausblick

Policy-basiertes Management ist ein Bereich des Systemmanagements, der seit mehr als 10 Jahren intensiv erforscht wird. Während bei der Definition einer geeigneten Terminologie und einer generischen Architektur von Policy-Managementsystemen inzwischen ein breiter Konsens besteht, gibt es bei der Policy-Spezifikation zwei unterschiedliche Ansätze: Einerseits sind dies sprachbasierte Ansätze, deren Ziel die Definition formaler Policy-Sprachen ist, die den Policy-Verfeinerungsprozeß sowie die automatische Erkennung und Behebung von Policy-Konflikten unterstützen sollen. Modellbasierte Ansätze hingegen definieren – analog zu Management-Informationsmodellen – objektorientierte Modelle zur Darstellung von Policies, die durch Vererbung an spezifische Szenarien angepaßt werden. Der wichtigste Vertreter des modellbasierten Ansatzes ist das DMTF/IETF Policy Core Information Model (PCIM) und seine Erweiterungen. Zur Validierung beider Ansätze wird überwiegend das Szenario Policy-basierten QoS-Management von DiffServ-Netzen und deren Komponenten gewählt. In beiden Fällen konnte deren Anwendbarkeit auf spezifische Szenarien nachgewiesen werden, wobei man jedoch von einer allgemeingültigen Lösung des Problems noch weit entfernt ist.

Das Problem der Erfüllung von Dienstgütevereinbarungen (SLAs) hängt mit Policy-basiertem Management zusammen, jedoch bestehen deutliche Unterschiede: Während bei SLAs die externe Sichtweise eines Dienstnutzers auf ein verteiltes System im Vordergrund steht, behandeln Policies die konsistente Transformation abstrakter Policies in konkrete Aktionen, wie z.B. das Setzen von Konfigurationsparametern, innerhalb eines verteilten Systems. Die Frage der Abbildbarkeit von Dienstgütevereinbarungen auf Policies ist einerseits eng mit der Fragestellung des Policy-Refinements verwandt, jedoch ignoriert eine Gleichsetzung von SLAs mit abstrakten Policies die Tatsache, daß sich z.B. in Service Provider-Umgebungen Zielkonflikte zwischen SLAs ergeben können. Folglich ist es möglich, daß bereits die Eingabeparameter des Policy-Entscheidungsprozesses Inkonsistenzen aufweisen, was sich in widersprüchlichen abstrakten Policies wiederspiegelt und das Policy-Refinement unmöglich macht. Auch hier sind noch

erhebliche Forschungsanstrengungen nötig, um zu universell anwendbaren Lösungen zu kommen.

Schließlich haben wir am Beispiel der Aggregation von Diensten zusätzliche Fragestellungen identifiziert, die sich durch die Verwendung neuer dienstorientierter Architekturen (wie z.B. Web Services) ergeben: Alternative Ausführungspfade innerhalb aggregierter Dienste machen es ausgesprochen schwer, Aussagen über die zu erwartende Verfügbarkeit oder die Antwortzeiten des Gesamtdienstes zu treffen. Während erste Arbeiten in diesem Bereich begonnen haben, besteht auch hier in der Zukunft großer Forschungsbedarf.

6. Literatur

[1] P. Bertrand, R. Darimont, E. Delor, P. Massonet, A. Van Lamsweerde, **GRAIL/KAOS: An Environment for Goal-driven Requirements Engineering**, Proceedings of the 20th International Conference on Software Engineering, IEEE-ACM, April, 1998

[2] Business Process Execution Language for Web Services Version 1.1. Second Public Draft Release, BEA Systems, International Business Machines Corp., Microsoft Corp., SAP AG, Siebel Systems, May 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/.

[3] M. Burgess. Cfengine: A site configuration engine. *Computing Systems*, 8(3), 1995. USENIX Association, http://www.cfengine.org.

[4] W. Bumpus, J. Sweitzer, P. Thompson, A. Westerinen, R. Williams: **Common Information Model - Implementing the Object Model for Enterprise Management**. J. Wiley & Sons, 2000

[5] N. Damianou, N. Dulay, E. Lupu, M Sloman, **Ponder: A Language for Specifying Security and Management Policies for Distributed Systems**, The Language Specification, Version 2.3, Imperial College Research Report DoC 2000/1, Department of Computing, Imperial College, London, Oktober 2000. http://www-dse.doc.ic.ac.uk/policies/ponder.shtml

[6] N. Damianou, N. Dulay, E. Lupu, M Sloman, **The Ponder Specification Language**, Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001

[7] N. Damianou, A.K. Bandara, M Sloman, E. Lupu, **A Survey of Policy Specification Approaches**, Technischer Bericht, Department of Computing, Imperial College, London, April 2002

[8] E. Dantsin et al., **Complexity and Expressive Power of Logic Programming**, 12the Annual IEEE Conference on Computational Complexity, IEEE Press

[9] B. Dijker, **A Guide to Developing Computing Policy Documents**, Short Topics in System Administration, Volume 2, SAGE (The System Administrators Guild)

[10] **CIM Policy MOF Specification 2.8**, http://www.dmtf.org/standards/documents/CIM/CIM_Schema28/mofs/CIM_Policy28.mof

[11] P. Flegkas, P. Trimintzios, G. Pavlou, A. Liotta, **Design and Implementation of a Policy-based Resource Management Architecture**, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), IFIP Conference Proceedings 246, Kluwer Academic Publishers, 2003

[12] **Special Issue on Policy-Based Networking**, IEEE Network, Volume 16, Number 2, März/April 2002

[13] **IFIP/IEEE International Symposium on Integrated Network Management (IM)** und **IFIP/IEEE Network Operations and Management Symposium (NOMS)**, http://www.comsoc.org/confs/im/index.html

[14] **Special Issue on Policy Based Management of Networks and Services**, Journal of Network and Systems Management, Volume 11, Number 3, September 2003

[15] A. Keller, H. Ludwig, **The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services**, Journal of Network and Systems Management, Special Issue on "E-Business Management", Volume 11, Number 1, Plenum Publishing Corporation, März 2003

[16] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. N. Tantawi, A. Youssef: **Performance Management for Cluster Based Web Services**, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), IFIP Conference Proceedings 246, Kluwer Academic Publishers, 2003

[17] T. A. Limoncelli, C. Hogan, **The Practice of System and Network Administration**, Addison-Wesley, Reading, MA, 2001.

[18] H. Ludwig, A. Keller, A., Dan, R. King, R. Franck, R., **A Service Level Agreement Language for Dynamic Electronic Services**, Journal of Electronic Commerce Research, Volume 3, Issue 1&2, Kluwer Academic Publishers, März, 2003

[19] L. Lymberopoulos, E. C. Lupu, M. S. Sloman, **An Adaptive Policy Based Framework for Network Services Management**, Journal of Network and Systems Management, Special Issue on Policy Based Management of Networks and Services, Volume 11, Number 3, September 2003

[20] L. Lymberopoulos, E. Lupu and M. Sloman, **Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework.** IFIP/IEEE Network Operations and Management Symposium (NOMS 2004), April 2004

[21] P. Martinez, M. Brunner, J. Quittek, F, Strauss, J. Schönwälder, S. Mertens, T. Klie, **Using the Script MIB for Policy-based Configuration Management**, Proceedings of the 2002 IEEE/IFIP Network Operations and Management Symposium, IEEE, April 2002.

[22] J.D. Moffett, M. Sloman, **Policy Hierarchies for Distributed Systems Management**, IEEE Journal of Selected Areas in Communications, Vol. 11, No 9, Dezember 1993

[23] R. Lobo, R. Bhatia, S. Naqvi, **A Policy Description Language**, Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI), AAAI Press / MIT Press, Juli 1999

[24] **Policy Workshop: International Workshop on Policies for Distributed Systems and Networks**, http://www.policy-workshop.org

[25] **Ponder: A Policy Language for Distributed Systems Management**, http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml

[26] D. Durham, Ed., J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry**, The COPS (Common Open Policy Service) Protocol,** Request for Comments 2748, Internet Engineering Task Force, Januar 2000

[27] R. Yavatkar, D. Pendarakis, R. Guerin, **A Framework for Policy-based Admission Control,** Request for Comments 2753, Internet Engineering Task Force, Januar 2000

[28] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, **Policy Core Information Model -- Version 1 Specification**, Request for Comments 3060, Internet Engineering Task Force, Februar 2001

[29] D. Levi, J. Schönwälder, **Definitions of Managed Objects for the Delegation of Management Scripts**, Request for Comments 3165, Internet Engineering Task Force, August 2001

[30] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser, **Terminology for Policy-Based Management**, Request for Comments 3198, Internet Engineering Task Force, November 2001

[31] B. Moore, **Policy Core Information Model (PCIM) Extensions**, Request for Comments 3460, Internet Engineering Task Force, Januar 2003

[32] J. Jason, L. Rafalow, E. Vyncke, **IPsec Configuration Policy Information Model**, Request for Comments 3585, Internet Engineering Task Force, August 2003

[33] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, B. Moore, **Policy Quality of Service (QoS) Information Model,** Request for Comments 3644, Internet Engineering Task Force, November 2003

[34] B. Moore, D. Durham, J. Strassner, A. Westerinen, W. Weiss, **Information Model for Describing Network Device QoS Datapath Mechanisms**, Request for Comments 3670, Internet Engineering Task Force, Januar 2004

[35] M. Sloman, **Policy driven Management for Distributed Systems**, Journal of Networks and Systems Management, Volume 2, Number 4, Dezember 1994

[36] M. Sloman, E. Lupu, **Security and Management Policy Specification**, IEEE Network, März 2002

[37] G.N. Stone, B. Lundy, G.G. Xie, **Network Policy Languages: A Survey and a New Approach**, IEEE Network, Januar 2001

[38] D. Verma, **Supporting Service Level Agreements on IP Networks**, MacMillan Technical Publishing, 1999

[39] D. Verma, **Policy-based Networking: Architecture and Algorithms**, New Riders, 2000.

[40] D. Verma, **Simplifying Network Administration using Policy based Management**, IEEE Network, März 2002

[41] R. Wies, **Policies in Network and Systems Management – Formal Definition and Architecture**, Journal of Networks and Systems Management, Volume 2, Number 1, März 1994

[42] R. Wies, **Policies in Integrated Network and System Management**, Dissertation, Ludwig-Maximilians-Universität München, Shaker Verlag, Aachen, 1995

[43] D. Box, F. Curbera, M. Hondo, et al., **Web Services Policy Framework (WS-Policy)**, Version 1.1, Mai 2003.

[44] D. Box, F. Curbera, M. Hondo, et al., **Web Services Policy Attachment (WS-PolicyAttachment)**, Version 1.1, Mai 2003.

[45] A. Nadalin, G. Della-Libera, P. Hallam-Baker et al., **Web Services Security Policy (WS-SecurityPolicy)**, Draft, Dezember 2002.

7. Kurzbiographien der Autoren

Dr. Alexander Keller ist seit 1999 Research Staff Member im Autonomic Computing Department am IBM T.J. Watson Research Center in Yorktown Heights, USA. Nach seinem Diplom in Informatik an der Technischen Universität München 1994 promovierte er dort 1998 am Lehrstuhl für Technische Informatik/Rechnernetze über CORBA-basiertes Enterprise Management. Er ist Autor von mehr als 40 referierten Publikationen im Bereich des Integrierten Managements verteilter Systeme. In 2003 war er Co-Chair des *14th International Workshop on Distributed Systems: Operations & Management (DSOM)* Workshops. Sein besonderes Interesse gilt Fragen des Dienst- und Anwendungsmanagements, sowie der Definition und automatisierten Umsetzung von SLAs. Er ist Mitglied der IEEE und der USENIX Association, sowie der DMTF CIM Application Working Group.

Dr. Heiko Ludwig studierte Wirtschaftsinformatik an der Otto-Friedrich-Universität Bamberg von 1987 - 1992 und promovierte 1997 dort am Lehrstuhl für Büro- und Verwaltungsautomation über Koordinationssysteme. Von 1996 bis 2001 arbeitete er als Wissenschaftlicher Mitarbeiter am IBM Forschungslabor in Zürich und wechselte dann an das IBM T.J. Watson Research Center in New York. Dort beschäftigt er sich mit vertrags- und policy-basierten Systemen im Umfeld von Web Services und Grid. Er ist Autor zahlreicher referierter Artikel, mehrerer Buchkapitel, ist regelmäßig in Programmkommitees und als Gasteditor tätig und arbeitet in der GRAAP Working Group des Global Grid Forums am WS-Agreement Standard.