Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Fakultät für Informatik

Institut für Telematik
Cooperation & Management
Prof. Dr. Sebastian Abeck

Erfassung von Abhängigkeiten zwischen IT-Ressourcen in einem Service-Kontext

Diplomarbeit
von

Ingo Pansa

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

Bearbeitungszeit: 01. September 2007 – 29. Februar 2008

Universität Karlsruhe (TH) · Institut für Telematik · Geb. 20.20 · Zirkel 2 · 76131 Karlsruhe
Universität Karlsruhe (TH) · Abteilung Technische Infrastruktur · Geb. 50.34 · Am Fasanengarten 5 · 76131 Karlsruhe

Ehrenwörtliche Erklärung

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

Karlsruhe, den 29. Februar 2008

Ingo Pansa

Danksagung

Eine Reise selbst über tausende von Meilen beginnt mit einem einzigen Schritt.

Sprichwort, unbekannter Autor

Für die Möglichkeit der Entstehung der vorliegenden Arbeit möchte ich mich herzlich bei Herrn Prof. Dr. Sebastian Abeck bedanken.

Zu absolut tiefstem Dank verpflichtet bin ich Herrn Klaus Scheibenberger, der mir während der Erarbeitung der Themenstellung stets Hilfreich zur Seite gestanden ist und in unzähligen spannenden Diskussionsrunden letztlich die Neugierde für den Themenkomplex IT-Management geweckt hat.

Danken möchte ich ebenso Herrn Christian Mayerl, Kai-Moritz Hühner und Jews-Uwe Gaspar, da ich durch die Mitwirkung in der "Arbeitsgruppe IT-Management" im Jahre 2007 neben einem Einblick in aktuelle Entwicklungen in den Komplex IT-Management auch gelehrt wurde, eigene Gedanken in Diskussionen zu vertreten und Konzepte wissenschaftlich zu erarbeiten.

Nicht unerwähnt lassen an dieser Stelle möchte ich all diejenigen Menschen, die in kleinem wie auch großem Maße zum Gelingen dieser Arbeit beigetragen haben.

Vor allem sind dies:

Meine lieben Eltern Dieter und Elisabeth; Nadja, Peter, Sebastian S., Olaf, Andreas, Christoph Z., Christoph M., Stephan, Jörg, Jan-Michael;

Inhaltsverzeichnis

1 EINLEITUNG 8
1.1 Einführung in das Themengebiet 8
1.2 In der Arbeit behandelte Fragestellungen 10
1.3 Abgrenzung zu bestehenden Arbeiten 11
1.4 Erzielbare Verbesserungen 12
1.5 Modellhafter IT-Dienst und Szenario 12
1.6 Gliederung der Arbeit 14
2 GRUNDLAGEN 16
2.1 IT-Management 16
2.1.1 Unterteilung der Managementebenen 16
2.1.2 Unterteilung der Managementfunkionen 22
2.2 Technisch-orientierte Beschreibungsansätze 24
2.3 Strukturierende Beschreibungsansätze 30
2.3.1 Extensible Markup Language (XML) 31
2.3.2 Ressource Description Framework (RDF) 32
2.4 Managementarchitektur 35
3 STAND DER TECHNIK 39
3.1 Analyse des Begriffs Abhängigkeit 39
3.2 Analyse strukturierender Ansätze 48
3.2.1 Datacenter Markup Language (DCML) 48
3.2.2 CIM Modellaustausch 52
4 ENTWURF EINES KONZEPTS ZUR MODELLIERUNG VON RESSOURCENABHÄNGIGKEITEN 53
4.1 Motivation und Vorgehen 53
4.2 Generisches Infrastrukturmodell 57
4.3 Metamodell zum generischen Infrastrukturmodell 59
4.4 Konkretisierung des generischen Ansatzes 63
4.4.1 Modellierung von Fehlerzuständen 63
4.4.2 Implementierung im CIM 64
4.5 Zusammenfassung 66
5 MODELLIERUNG UND ANALYSE VON BEZIEHUNGEN UND ABHÄNGIGKEITEN 67
5.1 Szenario 67
5.2 Modellierungsansätze für Ressourcen und Abhängigkeiten 68
5.2.1 Vorgehensmodell 69
5.2.2 CIM und Generischer Ansatz 70
5.2.3 RDF/XML 72
5.3 Zusammenfassung 75
6 ENTWURF EINER MANAGEMENTARCHITEKTUR 76
6.1 Motivation 76
6.2 Anforderungen 77
6.3 Modell einer Managementarchitektur 80
6.3.1 Komponentenmodell 81
6.3.2 Management Node 82
6.3.3 Managed Node 84
6.3.4 Managed Nodes Directory 85
6.4 Zusammenfassung 85
7 PROTOTYPISCHE IMPLEMENTIERUNG, EVALUATION UND ERGEBNISSE 87
7.1 Details der Implementierung 87
7.1.1 Einschränkungen im Rahmen eines Prototyps 87
7.1.2 Implementierung des Szenario 87
7.1.3 Implementierungen von Managementkomponenten 89
7.2 Evaluation 92

7.3 Zusammenfassung 94
8 ZUSAMMENFASSUNG UND AUSBLICK 95
8.1 Zusammenfassung 95
8.2 Ausblick 96
ANHÄNGE 98

1 EINLEITUNG

1.1 Einführung in das Themengebiet

Betrachtet man die Bedeutung von *Informationstechnologie (IT, Information Technology)* und deren Auswirkungen auf Bereiche wie die Geschäftswelt, Unterhaltung, Kommunikation aber auch das Privatleben einer Person, stellt man fest, dass so gut wie jeder Bereich mehr oder weniger stark durch rechnergestützte Verarbeitung beeinflusst wird.

Besonders stark kommt dieser Einfluss in der Geschäftswelt zu tragen, kaum eine Branche kann heutzutage ohne rechnergestützte Systeme Geschäfte tätigen. Im Jahre 2005 etwa setzten 84% aller deutscher Unternehmen rechnergestützte Systeme in ihren Geschäftsabläufen ein [DSB06]. Mit dem Aufkommen des Internets und der weltweiten Vernetzung von Rechnersystemen innerhalb der letzten 15 Jahre ist aber auch die gesamte Wirtschaft zunehmend vernetzt worden, so waren im Jahre 2005 etwa 79% der Unternehmen ans Internet angeschlossen [DSB06]. Durch die Möglichkeiten der weltweiten Vernetzung sind schließlich komplett neue Geschäfts-Möglichkeiten entstanden, beispielsweise für Firmen, die Funktionalitäten, welche durch IT erbracht werden, als Dienstleistungen *(Services)* anbieten und verkaufen.

Bereits seit einiger Zeit gehen Unternehmen dazu über, u.a. rechnergestützte (systembasierte) Funktionalitäten, die nicht unmittelbar dem Kerngeschäftsbereich zuzuordnen sind, an externe Dienstleister zu vergeben und bedarfsgerecht – als IT-Service - wieder einzukaufen (sogenanntes *Outsourcing*, [Sö06]). Dazu werden sowohl qualitative wie auch nicht qualitative Parameter der eingekauften Funktionalität in Form von *Dienstleistungsvereinbarungen (DLV, Service Level Agreements, SLA's)* zwischen dem Dienstnehmer und dem Dienstleister vertraglich fixiert. Zentrale Fragestellung auf Seiten des Dienstleisters ist es, wie die ausgehandelten Parameter der SLA's durch die in einen IT-Dienst eingebundenen IT-Ressourcen abgedeckt werden können, bzw. welche Parameter überhaupt in Betracht kommen.

Dass ein systembasierter Dienst korrekt, d.h. innerhalb der im SLA vereinbarten Parameterbereiche, erbracht wird, wird vom *Service Level Management (SLM)* überwacht, welches ein Teil des IT-Managements ist. Das IT-Management umfasst alle Maßnahmen, die für einen unternehmenszielorientierten, effektiven und effizienten Betrieb eines verteilten IT-Systems und seiner Ressourcen erforderlich ist [HAN99].

Da das IT-Management stark mit den Geschäftsprozessen eines Unternehmens verbunden ist und damit maßgeblich am Geschäftserfolg eines Unternehmens Teil hat (Stichwort *IT-Alignement*, [Ser07]), rücken jüngste Entwicklungen einen neuen Managementansatz in den Vordergrund – *Business-Driven IT Management* (BDIM). Dieser stellt das Unternehmens- und das IT-Management in einen sehr engen Bezug:*„ ... The focus of BDIM is on the business: IT must help the business to achieve its goals, contributing to its results in a measurable way. ...“* ([SMS+06]). Die entscheidende Aussage hierbei ist, dass der Einfluss eines systembasierten IT-Dienstes in den Metriken der Geschäftswelt bemessen werden muss, um ihren tatsächlichen Anteil am unternehmerischen Erfolg aufzeigen und bewerten zu können. [SMS+06]

Da ein Service an sich immateriell ist, bedeutet die (zunehmende) Service-Ausrichtung, dass diese Bewertung aus Unternehmenssicht auf die Ressourcen, die einen Service erbringen, fortgesetzt werden muss. Wie in [MHG+07] gezeigt, können die Abhängigkeiten zwischen Service- und Geschäftsprozessebene anhand bestimmter Metriken berechnet werden, falls die zugrunde liegenden Abhängigkeiten einfach und bekannt sind. Für den Betreiber einer den IT-Services zugrundeliegenden IT-Infrastruktur ist jedoch (auch) der Zusammenhang zwischen Ressourcen der Ebenen Netz, System und Anwendungen und deren Bezug zur Serviceebene wichtig, zumal sich im Hintergrund von steigendem Kostendruck auch die genaue Rolle einer Ressource innerhalb einer Dienstleistung belegen lassen muss. Mit der Durchdringung von IT

und daher einhergehend umfangreicheren IT-Infrastrukturen wird diese Aufgabe zunehmend schwieriger. Dazu ist es jedoch unabdingbar, das der Betreiber einer IT-Infrastruktur sowohl weiß, welche IT-Ressourcen im Kontext von welchen IT-Diensten eingebunden sind (vertikale Ausrichtung) als auch deren Beziehungen untereinander kennt (horizontale Ausrichtung). Mit der Durchdringung von IT und daher einhergehend umfangreicheren IT-Infrastrukturen wird diese Aufgabe aufgrund der Komplexität des Themas zunehmend schwieriger.

Bisher wurde versucht, durch Aufteilung der unterschiedlichen Problemdomänen in Netzwerk-, System- und Anwendungsmanagement der wachsenden Kompliziertheit zu begegnen. Mit der Orientierung am Service wird aber gefordert, die Wechselwirkungsverknüpfungen *(interdependencies)* zwischen diesen unterschiedlichen Ebenen im Rahmen eines Service-Kontexts ganzheitlich zu erfassen und zu managen. Dazu kann der Begriff Service als Verknüpfungskriterium betrachtet werden, mittels diesem ein Zusammenhang über die unterschiedlichen Ebenen hergestellt werden kann. Abbildung 1 verdeutlicht diese Auffassung.

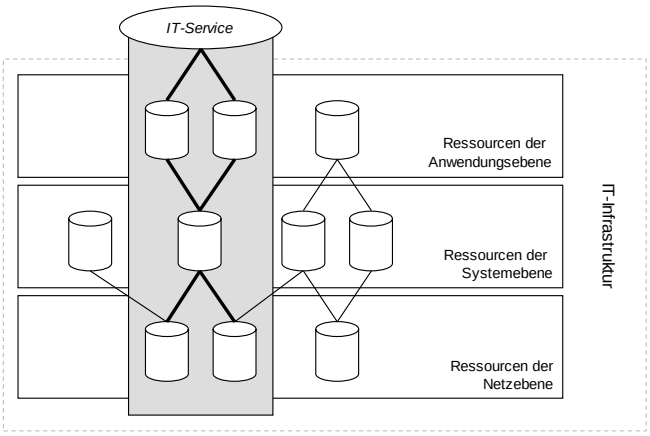

Abbildung 1 Service als Filterkriterium

Bisher kam hier vor allem die Top-Down Modellierung von der Serviceschnittstelle hinab zu den Infrastrukturressourcen mit dem ad hoc Wissen, welcher Service welche Ressource benötigt und welche Beziehungen zwischen welchen Ressourcen bestehen, zum Einsatz. Dabei muss von den zuständigen Administratoren deren Wissen exakt in die Modellierung der Infrastruktur einfließen um eine möglichst große Überdeckung zwischen Realität und Modell zu erreichen. Oftmals entstehen hier aber Fehler wenn man sich beispielsweise der Bedeutung einer Ressource innerhalb eines IT-Dienstes nicht bewusst ist. Erschwerend kommt hinzu, dass Beziehungen zwischen Ressourcen zum Teil vielleicht bereits zur Entwurfszeit bekannt sind (statischer Aspekt), aber ebenso wie die Ressourcen selbst zur Laufzeit Veränderungen (dynamischer Aspekt) unterliegen. Umgekehrt werden Beziehungen zwischen Ressourcen auch zum Teil erst im operativen Betrieb, zur Laufzeit, anhand ihres dynamischen Verhaltens erkannt. Gerade dieser dynamische Aspekt ist sehr wichtig für den qualitätsgesicherten Betrieb von IT-Diensten, da dieses dynamische Verhalten von Ressourcen und ihrer gegenseitigen Beziehungen einerseits durch die Nutzung der Dienste generiert wird, aber andererseits auch wieder auf das Verhalten der Dienste gegenüber dem Nutzer zurück wirkt.

Gesucht wird demnach ein Konzept, um das Laufzeitverhalten von Beziehungen zwischen IT-Ressourcen in Form von Abhängigkeiten innerhalb der Infrastruktur maschinell-bearbeitbar zu erfassen und darzustellen, um darauf aufbauend dieses verbesserte Wissens zu nutzen, um beispielsweise bekannte statische Modelle zu verifizieren bzw. zu erweitern.

Bestehende Arbeiten im Bereich der Beschreibung und maschinellen Verarbeitung von Abhängigkeiten ([KK01], [MHG07], [KBK00], [AGR+05], [CDS01]) beziehen sich zumeist nur auf die reine Erfassung von Zusammenhängen innerhalb der Serviceebene oder versuchen ohne hinreichend genaue Untersuchung des Begriffes Abhängigkeit Ressourcen in den Kontext

von IT-Diensten zu stellen. Mit der vorliegenden Arbeit wird versucht, zunächst den Begriff *Abhängigkeit* in Bezug auf die Dynamik während der Produktion eines IT-Dienstes zu untersuchen, um anschließend zu betrachten, wie das Wissen um Abhängigkeiten im Rahmen eines integrierten Managementansatzes strukturiert zu Weiterverarbeitung präsentiert werden kann.

1.2 In der Arbeit behandelte Fragestellungen

Während im Rahmen von integrierten Managementansätzen bisher die statische Top-Down Modellierung der Zusammenhänge zwischen IT-Diensten und IT-Ressourcen im Vordergrund steht, taucht vermehrt die Frage auf, wie sich Abhängigkeiten innerhalb eines Servicekontextes auch automatisiert erkennen und erfassen lassen (siehe auch [En01]). Bisher existieren eine Reihe von Ansätzen, die auf spezielle Systeme oder Umgebungen beschränkt sind [KBK00, Li03, CDS01, KK01].

Durch die Formulierung eines generischen Ansatzes wird daher untersucht, wie sich die zur Produktionszeit eines IT-Dienstes ergebenden Wechselbeziehungen modellieren lassen, um im Rahmen eines Gesamtkonzepts verschiedene Aspekte bezüglich des Managements von vereinbarten Dienstleistungen zu unterstützen. Dies, um zum einen die in einem SLA festgeschriebenen qualitativen technischen Parameter der Dienstleistung durch das Überwachen der Ressourcen und deren Wechselbeziehungen sicherzustellen (dynamischer Aspekt), und zum anderen um eine Übersicht der voneinander abhängigen Komponenten der Infrastruktur zu bekommen (statischer Aspekt). Letztgenannter Punkt ist beispielsweise im Rahmen von *Change Management* Prozessen relevant. Um diese Anforderungen zu unterstützen, ist im Rahmen von integrierten Managementansätzen die Information über Beziehungen und Abhängigkeiten auf Modellebene zu erfassen, um einerseits die Datenhaltungen von bereits bestehenden spezialisierten Tools zugänglich zu machen, andererseits eine ganzheitliche Sicht über die bestehenden Datenbestände zu erreichen.

Motiviert wird diese Fragestellung der möglichst präzisen Erfassung von Zusammenhängen innerhalb einer IT-Infrastruktur beispielsweise durch Ansätze wie *BDIM* (*Business-Driven IT Management*). Dafür ist es erforderlich, die Zusammenhänge zwischen *Business*- und *Service*-Metriken festmachen zu können. Da ein IT-Service von IT-Ressourcen erbracht wird, ist aber ebenso die Kenntnis der Zusammenhänge von Ressourcen-Metriken und damit notwendigerweise die der Ressourcen innerhalb der Infrastruktur, in Bezug auf einen IT-Service, von essentieller Bedeutung. Nur auf Basis dieser Kenntnisse ist schließlich der Einfluss einer Ressource auf den IT-Dienst und schließlich bis zur Geschäftsebene durchgängig sichtbar zu machen. Die nachfolgende Abbildung verdeutlicht diesen grundlegenden Zusammenhang.

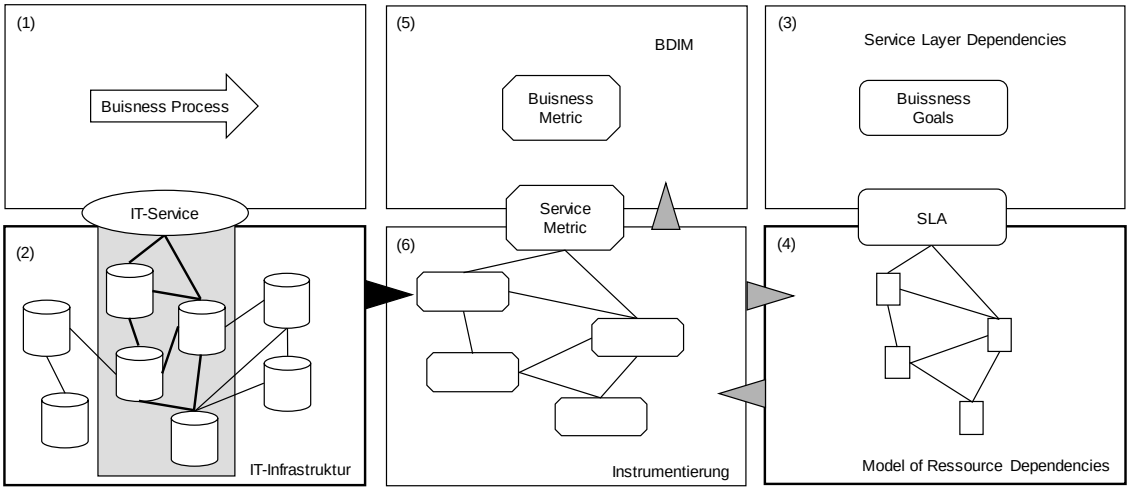

Abbildung 2 Motivation für den Ansatz

Zwischen (1) und (2) tritt der IT-Service als Bindeglied zwischen Geschäftsprozess und den Ressourcen einer Infrastruktur auf. Ähnlich lässt sich das SLA als Bindeglied in der Modellierung der Abhängigkeiten einer Infrastruktur (*Resource Dependencies*) zwischen (3) und (4) auffassen wie auch die *Service Metric* die Kopplung zwischen Metriken der Ressourcenebene (*Metric Dependencies*) und Vorgaben der Unternehmensführung (*Business Metric*) erreicht (zwischen (5) und (6)).

Die genaue Kenntnis über die Abhängigkeiten der Ressourcenebene ist somit Voraussetzung, um als Betreiber einer Infrastruktur *überhaupt* korrekte Zusicherungen über die Erbringung einer IT-Dienstleistung treffen zu können (Fixierung in einem SLA). Zusätzlich entsteht die Möglichkeit, den Einfluss beliebiger Ressourcen am Geschäftserfolg eines Unternehmens beurteilen zu können (BDIM Ansatz).

Im Fokus dieser Arbeit steht somit die Frage, durch welche Modellierungskonzepte der Schritt zwischen (2, Realität) und (4, genaues Modell) vollzogen werden kann, wobei sowohl Wissen aus dem Entwurf und der Implementierung (statischer Aspekt) wie auch Wissen aus der Laufzeit (dynamischer Aspekt) herangezogen werden und einer Managementarchitektur zur Verfügung gestellt werden soll. Dabei interessiert zunächst vor allem die Fragestellung, wie die Managementinformation im Informationsmodell repräsentiert wird, und wie die Information der technischen Sicht für strukturierte Weiterverarbeitung präsentiert werden kann.

Es wird ein generischen Modell entwickelt, mit dem die IT-Ressourcen einer IT-Infrastruktur in den Kontext eines IT-Dienstes gestellt werden können, um Verknüpfungen wie zwischen (5)+(6) oder (3)+(4) zu ermöglichen. Dieses generische Modell bezieht sich zunächst auf die technische Sicht und deren Modellierungskonzepte, wird aber im Hinblick auf den Entwurf eines integrierten Managementansatzes durch abstrahierende Beschreibungsansätze ergänzt. Die vorliegende Arbeit präsentiert demnach ein *Gesamtkonzept*, um die Erfassung von Wechselbeziehungen zwischen IT-Infrastrukturressourcen zu unterstützen, um dieses Wissen über die technische Sicht hinaus zur Verfügung zu stellen.

Dabei fallen eine Reihe grundsätzlicher Fragen ab, die im Rahmen dieser Arbeit behandelt werden:

- Wie kann der Begriff „Abhängigkeit“ erfasst werden und im Kontext der Produktion eines systembasierten IT-Dienstes definiert werden?
- Wie werden die Abhängigkeiten innerhalb einer Infrastruktur erfasst und modelliert?
- Welche Technologien existieren, um die gewonnene Information abzulegen, bzw. wiederzuverwenden?
- Und letztendlich: Wie sieht eine konkrete Architektur aus, die die oben genannten Fragestellungen implementiert?

Im Vordergrund bei all diesen Betrachtungen steht dabei die Frage, wie die hierzu notwendige Information im Rahmen eines Informationsmodells beschrieben werden kann. Dies stellt eine wichtige Grundlage im Hinblick auf automatisierbare Ansätze dar.

1.3 Abgrenzung zu bestehenden Arbeiten

Thematisch eng verwandt ist die vorliegende Arbeit u.a. mit [MHG07, CDS01, DMTF03]. Während in [MHG07] von bekannten funktionalen Zusammenhängen innerhalb einer *SOA* (*Service-Orientierte Architektur*) ausgegangen wird, anhand deren Metriken analysiert und berechnet werden können, sind die Zusammenhänge auf Ebene der Netz-, System- und Anwendungsebene nicht immer offensichtlich, weshalb eine Erfassung der Zusammenhänge über eine Analyse von Metriken fehlschlagen kann. Die gleiche Argumentation kann auch in Bezug auf das *Metrics Model* in [DMTF03] angebracht werden. Die Arbeit in [CDS01] stellt in gewissem Maße den Ausgangspunkt für die Überlegungen zum generischen Ansatz in Kapitel 4 dar, während die vorliegende Arbeit den konzeptionellen Gedanken vertieft und Abhängigkeiten

ebenso zwischen Zustandsänderungen auffasst, die Untersuchung jedoch verfeinert in dem Sinne, das eine wirkliche Trennung zwischen *Entitäten*, *Attributen* und *Attributwerten* erreicht wird.

1.4 Erzielbare Verbesserungen

Ein automatisierbares Vorgehen bei der Erfassung der Ressourcenabhängigkeiten bringt somit folgende Vorteile mit sich:

- Genaues Wissen über Zusammenhänge innerhalb der Infrastruktur unter Einbeziehung des Laufzeitaspekts, ohne die bestehenden funktionalen Zusammenhänge der Infrastrukturkomponenten genau analysieren zu müssen.
- Verifikation der statischen Modellierung (Bottom-Up, Punkt (4) in Abbildung 2)
- Unterstützung der Ausrichtung des IT-Managements an den Vorgaben der Unternehmensführung (BDIM, (6) nach (5))
- Klarheit bei der Formulierung von Dienstleistungsvereinbarungen ((4) nach (3))
- Verständnis über die Charakteristika von Abhängigkeiten

Mit dem Entwurf einer Managementarchitektur wird somit ein vollständiges Konzept entwickelt, um Wechselbeziehungen zwischen IT-Ressourcen innerhalb einer IT-Infrastruktur erfassen zu können.

1.5 Modellhafter IT-Dienst und Szenario

Die *Abteilung technische Infrastruktur*, im nachfolgenden nur noch ATIS genannt, ist innerhalb der *Universität Karlsruhe* IT-Dienstleister der Fakultät für Informatik. Zu den grundsätzlichen Aufgaben zählen neben der Bereitstellung der Netzwerkinfrastruktur für die Anbindung der unterschiedlichen Informatikinstitute an das Internet verschiedene systembasierte Dienste wie E-Mail oder VPN [ATIS07a]. Gerade der Dienst E-Mail ist auch für die unterschiedlichen Tätigkeiten im Forschungsumfeld inzwischen ein geschäftskritischer Dienst geworden, dessen Verwaltung mehrere Mitarbeiter übernehmen [ATIS07b].

Gegenüber den Nutzern der IT-Dienste kommt die Notwendigkeit auf, die Leistungserbringung qualitativ erfassen und bewerten zu können. Hierzu ist jedoch, wie in der Einführung bereits erwähnt, der genaue Aufbau der Infrastruktur sowie der Einfluss einer Ressource bei der Erbringung eines IT-Dienstes genau zu kennen.

Um verschiedene Lösungsansätze für diese Problematik zu bewerten, wird jeweils am Beispiel des Mailsystems überprüft, welche Möglichkeiten existieren, die darin vorhandenen Abhängigkeiten und Wechseleinflüsse der Ressourcen im Kontext des E-Mail Dienstes beschreiben zu können. Dazu ist es notwendig zu verstehen, wie der E-Mail Dienst aufgebaut ist.

Prinzipiell lässt sich der E-Mail Dienst in die zwei Teildienste „E-Mail senden“ und „E-Mail lesen“ aufteilen [Pa07]. Für beide Teildienste kommen jeweils unterschiedliche Protokolle zum Einsatz (*smtp* für die E-Mail Zustellung, *imap*/*pop* für den Zugriff auf das E-Mail Postfach). Größtenteils ist jedoch die gleiche physikalische Hardware und Infrastruktur in die beiden Teildienste eingebunden.

Über zwei unterschiedliche Schnittstellen (E-Mail Programm auf dem Rechner des Benutzers, Webbrowser mit Webinterface [Horde]) hat der Benutzer des E-Mail Dienstes die Möglichkeit, seine abgelegten E-Mails zu lesen. Als Zugangsprotokolle stehen dabei sowohl *pop* [POP] (*Post Office Protocol*) wie auch *imap* [IMAP] (*Internet Mail Access Protocol*) zur Verfügung, die Kommunikation erfolgt nach erfolgreicher Authentifizierung verschlüsselt. Der Benutzer kommt mit wenigen technischen Parametern in Verbindung. Bis auf einige Konfigurationseinstellungen an der Dienstschnittstelle (Portnummern für das ausgewählte

Kommunikationsprotokoll müssen bekannt und entsprechend eingestellt werden) kennt der Benutzer lediglich die physikalische Größe seines Postfaches, welches momentan (Stand August 2007) maximal 200MB umfassen darf.

Während der E-Mail (Teil-)Dienst sich aus Sicht des Nutzers relativ einfach präsentiert, ist die Sicht für den Betreiber des Dienstes auf die zu Grunde liegende Infrastruktur wesentlich komplizierter. Abbildung 3 gibt einen übersichtlichen Blick auf die in den E-Mail Teildiensten eingebundenen IT-Ressourcen wieder. Basisdienste wie der DNS-Dienst zur Namensauflösung wie auch die eingebundenen Netzwerkkomponenten werden der Einfachheit halber hier nicht weiter betrachtet.

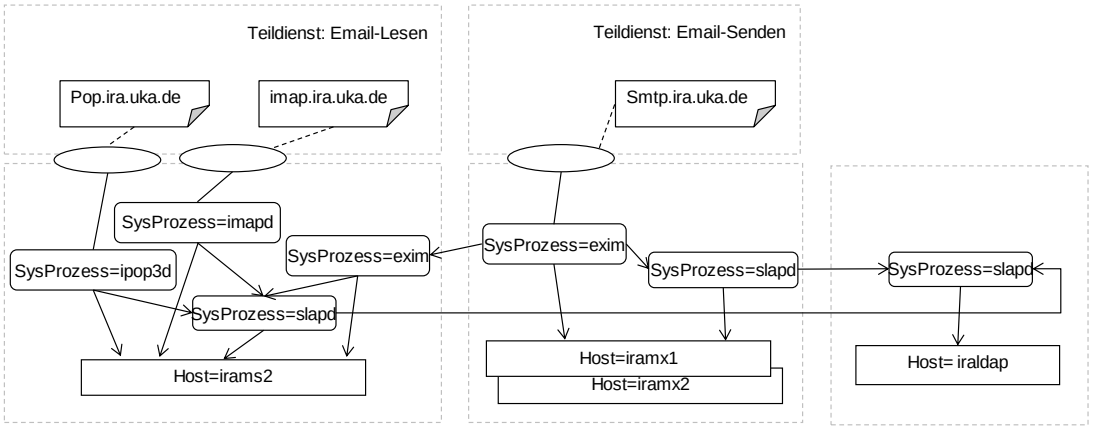

Abbildung 3 Betreibersicht auf den E-Mail Dienst (Übersicht)

Neben dem zentralen Mailserver irams2 sind noch zwei weitere Systeme aus Gründen der Redundanz, die ein- und ausgehenden Mailverkehr transportieren (iramx1/iramx2), eingebunden. Diese beiden Systeme bilden die Schnittstelle gegenüber externen Mailsystemen. Der zentrale Verzeichnisdienst auf Basis von *openldap* ist redundant auf mehrere Server ausgelegt, wobei Lesezugriffe auf die lokale Serverkopie des ldap-Datenbestandes durchgeführt werden, Schreibzugriffe dagegen zentral auf den ldap-Datenbestand auf dem Host iraldap.

Im Rahmen der vorliegenden Arbeit wird anlehnend an das vorgestellte *reale* Szenario folgendes *vereinfachtes* Szenario untersucht (in Abbildung 4). Dabei wird die Untersuchung auf wenige wesentliche IT-Ressourcen konzentriert, an deren Verhalten die gestellten Fragestellungen jedoch einfach nachvollzogen werden können.

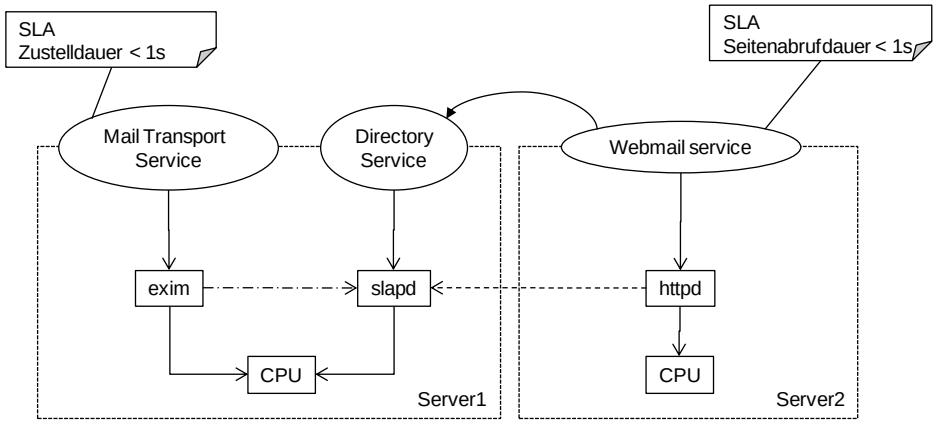

Abbildung 4 Vereinfachtes Szenario

Betrachtet werden zwei Serversysteme *Server1* und *Server2* mit den logischen IT-Ressourcen *exim*, *slapd* und *httpd* (Systemprozess-basierte Netzdienste) sowie die physikalische Ressource *CPU*. Die angebotenen Dienste *Mail Transport Service* werden durch den *Mail Transfer Agent*

exim, der *Directory Service* durch den ldap-Server *slapd* sowie der *Webmail Service* durch den *apache* Webserver *httpd* realisiert.

Im Rahmen der Nutzung der Dienste *Mail Transport* sowie *Webmail Service* werden mit Nutzern der Dienste zugesicherte Qualitätslevel in Form von (technischen) Dienstleistungs-Vereinbarungen (technischen SLA) vereinbart. Durch die Inanspruchnahme der Dienstleistungen und der damit verbundenen Produktion des immateriellen *IT-Dienstes* durch die zugrunde liegende IT-Infrastruktur ist von besonderem Interesse, inwiefern sich die Einhaltung der Dienstleistungsvereinbarungen überwachen lässt, indem die zugrunde liegende Dynamik aufgrund der Inanspruchnahme der IT-Ressourcen erfasst, modelliert und somit für das Management nutzbar gemacht werden kann.

Dabei spielt das Wissen um Beziehungen, wie bereits argumentiert, eine entscheidende Rolle. Im Beispiel des Szenarios haben Wechselwirkungen aufgrund von Laufzeitverhalten beispielsweise Auswirkungen auf die erzielbare Dienstqualität an der Webmail-Schnittstelle. Dies ist insofern kritisch, da ein Dienstnutzer hier unmittelbar in Kontakt mit der Dienstschnittstelle kommt und Verzögerungen bei der Auslieferungsdauer von Webseiten spürt. Konkret besteht eine Wechselwirkung zwischen der durch den E-Mail-Transportdienst verursachten CPU-Last und der erzielbaren Dienstleistungsqualität der Webmail-Schnittstelle. Bezogen auf den IT-Dienst bedeutet dies, es besteht ein Zusammenhang zwischen den Zustandsänderungen der IT-Ressourcen und dem erzielbaren Qualitätslevel eines IT-Dienstes.

1.6 Gliederung der Arbeit

Kapitel 1 führt in die Thematik der Arbeit, ein und schärft den Sinn für die Notwendigkeit der Erfassung von Abhängigkeiten. Neben einer generellen Einführung in das thematische Umfeld wird mit dem E-Mail-Dienst der ATIS ein konkreter systembasierter IT-Dienst vorgestellt und vereinfacht eingeführt (Szenario).

Im zweiten Kapitel werden grundlegende Ansätze erläutert, die für das weitere Verständnis der Arbeit zwangsläufig wichtig sind. So gehört neben einer Einführung in Begriffe des IT-Managements die Untersuchung einer generischen Managementarchitektur, eines konkreten Informationsmodells auch die Betrachtung von strukturierenden Beschreibungsansätzen.

Kapitel 3 ordnet diese Arbeit in den Kontext bestehender Ansätze ein und deckt deren Vorteile wie auch deren offensichtliche Schwächen auf.

In Kapitel 4 wird ein generischer Ansatz untersucht, der sowohl statische, vorab bekannte, wie auch sich erst zur Laufzeit ergebende Zusammenhänge erfassen kann. Hierzu wird zunächst ein einfaches generisches Informationsmodell betrachtet, um anschließend die Begriffe Relation und Abhängigkeit innerhalb des Meta-Modells zu definieren. Von diesen Überlegungen ausgehend wird ein konkretes Erweiterungsschema für das *CIM* definiert.

Das *CIM* Erweiterungsschema aus Kapitel 4 wird in Kapitel 5 konkret an den IT-Ressourcen des Szenarios angewendet. Weiterhin wird eine Verbindung zwischen der technischen Sicht und dem darauf aufbauenden strukturierenden Ansatz geschaffen, indem ein *RDF Schema* zur Umsetzung der technischen Sicht in eine strukturierende Sicht entworfen wird.

In Kapitel 6 werden Komponenten einer *WBEM*-basierten Managementarchitektur diskutiert, um ein verteiltes Management einer Infrastruktur bezüglich der an der Produktion eines IT-Dienstes beteiligten IT-Ressourcen zu ermöglichen. Diese Managementarchitektur bildet die Grundlage für den prototypischen Demonstrator in Kapitel 7.

Kapitel 7 geht auf Details eines entwickelten prototypischen Demonstrators ein und diskutiert die Ergebnisse der Auswertung des generischen Ansatzes auf technischer Ebene bezüglich der

Erfassung von Laufzeitabhängigkeiten und den Ergebnissen der Integration der technischen Managementdaten in den auf *RDF*-basierten strukturierenden Beschreibungsansatz.

Im 8. Kapitel werden die wesentlichen Erkenntnisse dieser Arbeit zusammengefasst und ein Ausblick auf mögliche auf dieser Arbeit aufbauende Untersuchungen gegeben.

Ergänzt wird diese Arbeit durch folgende Anhänge:

(1) Literaturverzeichnis zu den Quellangaben
(2) Glossar der fachbezogenen Begriffe
(3) Verzeichnisse (Abbildungsverzeichnis, Tabellenverzeichnis)
(4) Literaturanalysen von ausgewählten Arbeiten
(5) *MOF*-Datei des *CIM Extension Schemas*
(6) *RDF* Schema für die Modelltransformation *CIM* generischer Ansatz – *RDF*
(7) Java-*Codelisting* der im Rahmen der prototypischen Umsetzung implementierten Komponente *CIM2RDF*

2 GRUNDLAGEN

Nach der thematischen Einführung sowie der Vorstellung konkreter IT-Dienste und der zugrunde liegenden Infrastruktur werden nun Konzepte und Technologien betrachtet, die für ein weiteres Verständnis notwendig sind. Zunächst werden Begriffe des IT-Managements eingeführt, um anschließend Ansätze aus dem Bereich IT-Management, Informationsbeschreibung (Informationsmodell) und Informationsverknüpfung hinsichtlich der Modellierung der für die Unterstützung des Dienstmanagements notwendigen Eigenschaften zu untersuchen. Die teilweise sehr umfangreichen Themengebiete werden hierbei unter den für die weitere Arbeit relevanten Gesichtspunkten betrachtet, hauptsächlich gehören hierzu die Begriffe IT-Dienst, IT-Ressource und Abhängigkeit.

2.1 IT-Management

Durch die Bereitstellung von IT-Diensten und die damit verbundene Positionierung als IT-Dienstleister muss sich dementsprechend auch das unterstützende Management diesem Trend anpassen. Dazu wurden eine Reihe Standards geschaffen, in welchen der Service als produzierende Komponente eine zentrale Rolle spielt. Durch die zunehmende Ausrichtung am Begriff IT-Service und den damit verbundenen Konsequenzen müssen auch die entsprechenden Managementaufgaben dahingehend ausgerichtet werden. Das IT-Management hat zur Aufgabe, die angebotenen und erbrachten IT-Dienstleistungen zu verwalten und die Administration dahingehend zu unterstützen. Diese Aufgaben umfassen neben der Planung und Bereitstellung der Dienste eine Zusicherung über zu erwartende qualitative Zusicherungen, Überwachung und Steuerung und letztlich Abrechnung der erbrachten Dienstleistung. Charakteristisch für IT-Dienste sind dabei ihr immaterielles Erscheinungsbild und die damit einhergehenden Schwierigkeiten, die angesprochenen Aufgaben zu bewältigen

Das IT-Management umfasst alle Maßnahmen, die für einen unternehmenszielorientierten, effektiven und effizienten Betrieb eines verteilten IT-Systems und seiner Ressourcen erforderlich sind. [HAN99]. Das Verständnis, dass durch verteilte IT-Systeme IT-Dienste bereitgestellt werden, dient dazu, die Geschäftsziele bzw. die Geschäftsprozesse die durch IT realisiert werden, als Ziel des Betriebs einer IT-Infrastruktur zu betonen. Demzufolge ist natürlich auch die entsprechende Ausrichtung der notwendigen Maßnahmen, d.h. des IT-Managements, auf diese IT-Dienste erforderlich. Für die Strukturierung der Maßnahmen existieren unterschiedliche Prozessrahmenwerke, u.a. das im IT-Dienstleistersektor geläufige Rahmenwerk *IT Information Library* (ITIL, [ITIL]). Um die IT-Dienste mit vereinbarten Leistungsniveaus, d.h. mit vereinbarter Qualität, auch gesichert anbieten zu können, müssen die Ressourcen und dementsprechend auch das Management der IT-Ressourcen – das Infrastrukturmanagement –in Bezug zu den angebotenen IT-Dienstleistungen gestellt werden.

2.1.1 Unterteilung der Managementebenen

Zunächst lässt sich eine Einordnung der Managementaufgaben in unterschiedliche Ebenen durchführen. Innerhalb einer IT-Infrastruktur lassen sich Managementaufgaben anhand der unterschiedlichen Charakteristika der eingesetzten IT-Ressourcen identifizieren. Zur Vereinfachung der anfallenden Aufgaben lassen sich dabei grob vier unterschiedliche Ebenen ausmachen [HAN99], welche in Abbildung 5 dargestellt sind. Hierbei sind die einzelnen Ressourcen der unterschiedlichen Ebenen den jeweiligen Managementbereichen zugeordnet, die Anbindung an Geschäftsprozesse und Geschäftsprozessmanagement ist nur schematisch dargestellt.

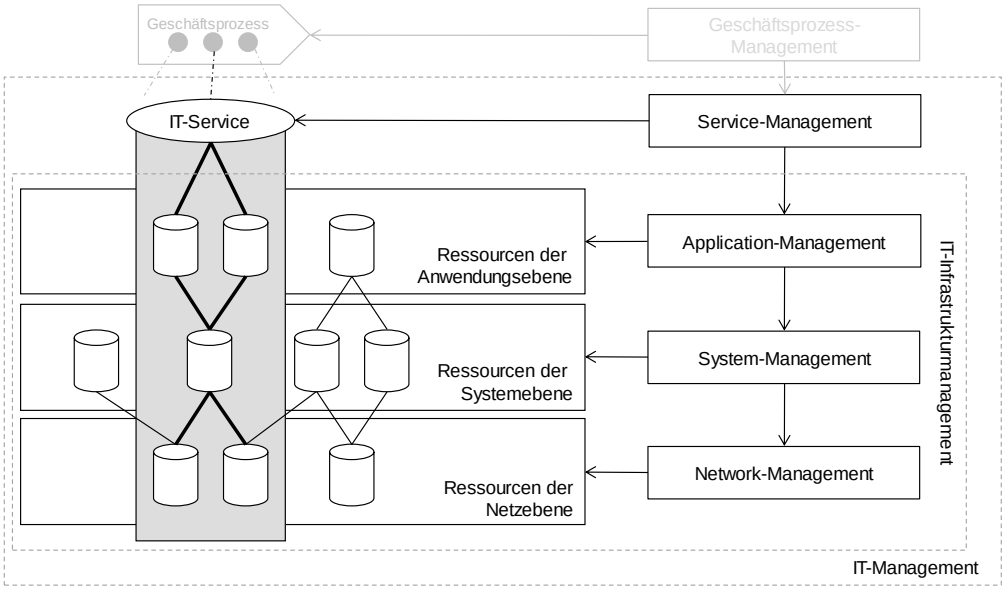

Abbildung 5 Unterteilung der Managementebenen

Im Rahmen einer auf den IT-Dienst ausgerichteten Managementarchitektur werden die unterschiedlichen Ebenen in einem holistischen Kontext betrachtet. Die einzelnen Managementebenen bauen demnach aufeinander auf. Zunächst werden Details des IT Service Managements untersucht, anschließend die unterschiedlichen Ebenen des IT-Infrastrukturmanagements.

IT Service Management (ITSM) nach ITIL

Das im Einleitungskapitel beschriebene Szenario im Rahmen der Problemstellung dieser Arbeit wird durch einen IT-Dienstleister getragen, daher wird für die Untersuchung des *IT Service Managements* (ITSM) nachfolgend das im IT-Dienstleisterbereich geläufige Prozessframework *ITIL* naher erläutert.

Die *Information Technologie Infrastructure Library* (*ITIL*) ist ein prozessorientiertes Rahmenwerk, um die für ein *IT-Service Management* notwendigen Bedingungen zu schaffen. Ein Prozess ist hierbei die chronologische Abfolge aller Schritte, die zur Erstellung eines Produktes erforderlich sind [Ol06]. Das konkrete Produkt im Kontext der IT (aus Sicht des Nutzers) ist demnach der IT-Dienst. Einheitliche Prozessrahmenwerke ermöglichen es Betreiben, IT-Dienste gesichert und zuverlässig anzubieten, wobei die notwendigen Managementaufgaben nachvollziehbar sind. Zu diesen Rahmenwerken zählt beispielsweise eben die *ITIL* [ITIL].

Im Wesentlichen ist die *ITIL* eine Sammlung von Lösungsansätzen, die sich in der Vergangenheit bewährt haben (*best practices*), und wird in textuell-beschriebener Form angeboten, wenngleich mittlerweile eine Reihe von kommerziell verfügbaren Managementlösungen existieren, die die Verwaltung der von der *ITIL* definierten Prozesse unterstützt.

Die *ITIL* beschreibt nicht, *wie* die konkreten Managementprozesse im ITSM aussehen, sondern hauptsächlich, *was* überhaupt getan werden muss, und welche Schnittstellen zwischen diesen unterschiedlichen Managementprozessen bestehen. Im Kern der aktuellen Version der *ITIL* steht dabei der Begriff *Service Lifecycle*. In Anlehnung an den Demingkreis (*PDCA*, *Plan-Do-Check-Act*, nach Edward Deming) sind qualitätssicherende Maßnahmen keine abgeschlossenen Einheiten sondern unterliegen einer ständigen Verbesserung (siehe auch [BSIa]). Dementsprechend unterliegt auch der IT-Dienst einem stetigen Wandel, wird justiert und den aktuellen Bedürfnissen der Servicekonsumenten ständig (neu) angepasst. Somit stellt die

Lebensdauer eines IT-Dienstes keinen begrenzten Zeitabschnitt vom Design über die Implementierung bis zur Nutzung dar, sondern wird zur Laufzeit überwacht, um daraus Rückschlüsse für eine eventuelle Verbesserung im Servicedesign schließen zu können. Die neuen (verbesserten) Dienste müssen in die IT-Infrastruktur eingebracht werden, um so für einen Konsumenten verfügbar gemacht zu werden. Hierbei muss wiederum der betriebliche Ablauf überwacht werden, um die Einhaltung der ausgehandelten Leistungsparameter sicherstellen zu können.

Dieser Lebenszyklus lässt sich nach *ITIL Version 3* in drei Abschnitte einteilen:

(1) *Service Design*: beschreibt Prozesse, die zur Entwicklung von IT-Diensten und den zugehörigen Managementaufgaben relevant sind. In Bezug auf IT-Ressourcen und deren Beziehungen untereinander fällt hier Wissen ab, das sich der Entwurfszeit zuordnen lässt und daher statischen Charakter hat.
(2) *Service Transition*: beschreibt die Prozesse, die notwendig sind, um neue und/oder verbesserte IT-Dienste in den Betrieb einer Infrastruktur zu bringen.
(3) *Service Operation*: definiert notwendige Prozesse zur Auslieferung eines IT-Dienstes inklusive den Support-technischen Fragestellungen. In Bezug auf IT-Ressourcen und deren Beziehungen untereinander fällt hier Wissen ab, das sich der Laufzeit zuordnen lässt und daher dynamischen Charakter hat.

Umschlossen werden diese drei Teile der *ITIL Version3* durch den *Continual Service Improvement* [ITILa], also jenen Maßnahmen und Prozessen, die dahingehend wirken, dass der Lebenszyklus eines IT-Dienstes ständig von neuem beginnt und das erreichte Wissen in die stetige Verbesserung von IT-Diensten zurück fließt.

Ziel der vorliegenden Arbeit ist es, zu verstehen, welche Abhängigkeiten sich zwischen IT-Ressourcen im Kontext eines IT-Dienstes ergeben (Ressourcenabhängigkeiten), um hierdurch unmittelbar Abweichungen von der vereinbarten Qualität erkennen, einordnen und eingrenzen zu können. Das Wissen über Abhängigkeiten spielt unmittelbar im Prozess der Qualitätsverbesserung eine wichtige Rolle, will ein Infrastrukturbetreiber herausfinden, an welchen IT-Ressourcen eventuelle Engpässe zu verzeichnen sind, aber auch um seine statische Sicht auf die Infrastruktur für Änderungsprozesse evaluieren bzw. verbessern zu können.

Folgende *ITIL*-Teilprozesse unterstützen das Service-Management und damit auch die Fragestellung, welche IT-Ressourcen Wechselbeziehungen während der Produktion eines IT-Dienstes untereinander besitzen:

(1) *Incident Management*: befasst sich mit Störungen oder Ereignissen, die eine Minderung oder Unterbrechung des standardmäßigen Betriebs eines IT-Dienstes verursachen (Fehlfunktionen, Ausfall von Komponenten) [Ol06]. Wissen um genaue Beziehungen ist hier notwendig, um Fehler genau eingrenzen zu können und die Ursache für Fehler erkennen zu können (*Root-Cause Analysis*)
(2) *Problem Management*: befasst sich mit der Ursachenforschung von *Incidents* [Ol06]. Erkenntnisse des *Incident Managements* gehen in das *Problem Management* ein, u.U. wird während der Analysephase im Rahmen des Problem Managements eine nicht bekannte Wechselbeziehung zwischen Infrastrukturelementen aufgedeckt.
(3) *Change Management*: Im Rahmen der stetigen Verbesserung müssen Änderungsvorhaben geprüft, die Durchführung organisiert, überwacht und dokumentiert werden. [Ol06]. Die Handlungsaktionen im *Change Managementprozess* binden auch Wissen um Beziehungen zwischen Infrastrukturelementen ein.
(4) *Configuration Management*: Die gesamte IT-Infrastruktur eines Unternehmens wird in Form eines logischen Modells in einer *Configuration Management Database*

(CMDB) abgespeichert. Es werden nicht nur die einzelnen Komponenten selbst, sondern auch die Beziehungen dieser Komponenten untereinander abgelegt. [Ol06]

(5) *Service Level Management*: zentral für die vertragliche Fixierung der IT-Angelegenheiten zwischen internen und externen Partner. Die getroffenen Vereinbarungen müssen ständig kontrolliert werden [Ol06]. In diesem Managementprozess kommen die beteiligten Akteure unmittelbar mit Laufzeitaspekten in Kontakt und damit mit Wechselbeziehungen, die einen starken Bezug zur Laufzeit (Produktionszeit) eines IT-Dienstes besitzen.

(6) *Availability Management*: sorgt dafür, das IT-Dienste stets verfügbar sind und wirtschaftlich arbeiten [Ol06]

(7) *Release Management:* definiert die notwendigen Schritte, um Änderungen der IT-Infrastruktur in Bezug auf einen IT-Dienst gesichert und nachvollziehbar zu veröffentlichen.

(8) *Capacity Management:* Im Rahmen der Vereinbarung von Dienstgüte in einem SLA muß die zugesicherte Leistung auch real zur Verfügung stehen. Das *Capacity Management* umfasst die Schritte die notwendig sind, um Sicherzustellen das die entsprechenden Leistungskapazitäten zur Verfügung stehen.

Dabei können vor allem das *Incident Management* und das *Service Level Management* als Aufgaben herausgestellt werden, die im unmittelbaren Kontext zur Dynamik während der Produktion eines IT-Dienstes von Bedeutung sind.

Im *Incident Management* werden Störungen oder aktuelle Ereignisse, welche eine Abweichung von zugesicherten Leistungsparametern verursachen, erfasst. Ziel der Erfassung ist zum einen, alle Störungen zu dokumentieren um die Behebung von (wiederkehrenden) Störungen zu vereinfachen, zum anderen, mit dem gelernten Wissen eventuell zukünftige Störungen zu vermeiden. Ziel des *Service Level Management* ist es, zum einen vertragliche Zusicherungen in Form von SLA's zu dokumentieren, zum anderen durch Überwachung und steuernde Eingriffe die Einhaltung dieser Vorgaben zu erreichen.

SLM

Als wichtiger Aspekt bei der Untersuchung der Fragestellung, wie IT Dienste qualitativ gesichert angeboten werden können, spielt das *Service Level Management* (*SLM*) eine zentrale Rolle. Im *Service Level Management* wird die zugesicherte Dienstgüte einer Dienstleistung überwacht und bei Verletzung gegebenenfalls die notwendigen Schritte eingeleitet, um die Ursachen der Verletzung einzugrenzen und/oder zu beheben. Dazu werden die notwendigen Parameter eines IT-Dienstes in einer Dienstleistungs-Vereinbarung (DLV, engl. *Service Level Agreement, SLA*) in einem vertraglichen Rahmenwerk fixiert und sind somit Gegenstand der Überwachung. Da ein IT-Dienst ein immaterielles Produkt ist, findet die eigentliche Überwachung an Messpunkten der Ressourcenebene statt. Zentrale Frage ist daher, wie die ressourcenbezogenen Messwerte derart aggregiert werden können, dass sie in Deckung zu den in einem *SLA* zugesicherten Parametern stehen. Abhängigkeiten innerhalb dieser Ebene können zwischen unterschiedlichen IT-Diensten in Bezug auf einen Geschäftsprozess ausgemacht werden, beispielsweise wenn im Rahmen eines Geschäftsprozesses unterschiedliche IT-Dienste eingebunden werden, die in geordneter Reihenfolge genutzt werden müssen.

Das IT-Service Management befasst sich demnach mit den Prozessen und Vorgehensweisen, um IT-Dienste zielgerichtet, kundenfreundlich, kostenoptimiert zu erbringen [Ol06]. Dabei wird der Begriff IT-Dienst als atomarer Baustein innerhalb einer Prozess-orientierten Architektur eingesetzt, um die benannten Ziele zu erreichen.

Für Untersuchungen im Rahmen der vorliegenden Arbeit interessiert vor allem die genaue Kenntnis über Zusammenhänge zwischen IT-Ressourcen um das *SLM* zu unterstützen.

SLA

Der Mehrwert eines IT-Dienstes wird für einen Nutzer in Form von Diensteigenschaften, den Serviceparametern, erfasst. Zu diesen Serviceparametern werden Qualitätsangaben gemacht (Servicelevel). Die Diensteigenschaften beschreiben demnach zum Teil technische Parameter Funktionalitäten, aber auch nicht-technische Parameter, wie beispielsweise die Erreichbarkeit von Supportmitarbeitern. Die messbaren Parameter werden vertraglich fixiert und in der Dienstleistungsvereinbarung niedergeschrieben. Demnach werden hinter dem Begriff Qualität die messbaren Parameter der Dienstleistung maskiert, um so für den Dienstnutzer eine konkrete Zusage zu fassen, auf die er sich im Rahmen der Dienstnutzung verlassen kann.
Im Rahmen dieser Arbeit interessieren vor allem die messbaren Parameter der technischen Aspekte, um Zusagen für die Realisierung der technischen Schnittstelle zu erhalten. Beispiele für solche Qualitätszusagen sind Aussagen wie: „Garantierte Zustellzeit einer E-Mail innerhalb 100ms“ oder „Durchschnittliche Verfügbarkeit pro Jahr von 99,99%“. Dabei müssen bekannte Messinformationen von den IT-Ressourcen gesammelt und über Managementebenen hinweg aggregiert werden. Schließlich werden die qualitativen Parameter eines IT-Dienstes letztlich durch die zugrunde liegenden IT-Ressourcen bestimmt. Abbildung 6 verdeutlicht den Zusammenhang zwischen den Akteuren *Service User* und *Service Provider* in Bezug auf einen IT-Dienst und die technischen Parameter der IT-Ressourcen.

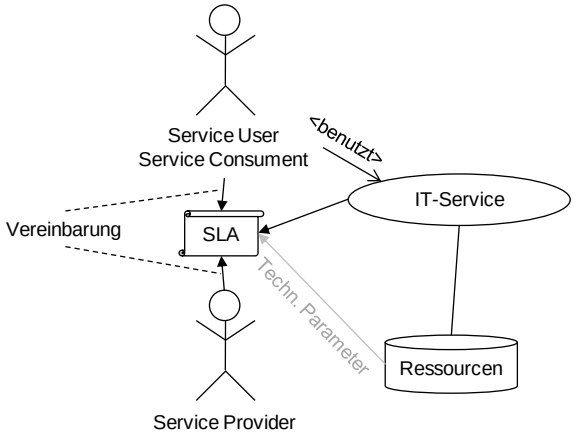

Abbildung 6 SLA als Vereinbarungsgegenstand zwischen Service User und Service Provider

Die einzelnen qualitativen Parameter zu einem IT-Dienst werden im *Service Level Agreement* gesammelt und vertraglich fixiert. Das *Service Level Agreement* stellt somit ein Dokument dar, das zum einen für den Nutzer eines IT-Dienstes das zu *erwartende* Leistungsspektrum, und zum anderen für den Produzent eines IT-Dienstes das *einzuhaltende* Leistungsspektrum definiert.

Somit konzentrieren sich die im vorigen Abschnitt angesprochenen Managementprozesse und Verfahren eines IT-Dienstleisters auf die Erfüllung der *Service Level Agreements*. Im Rahmen dieser Arbeit wird hierbei der technische Aspekt betrachtet, der sich mit einem *SLA* verknüpfen lässt, also jene Parameter, welche sich aus der zugrunde liegenden IT-Infrastruktur ableiten lassen. Dies bezieht sich auf Parameter, die sich direkt aus den entsprechenden IT-Ressourcen ableiten lassen, beispielsweise Parameter wie Netzwerkdurchsatz, Antwortzeiten oder Plattenkapazitäten.

Metrik

Um qualitative Zusagen über IT-Dienste geben zu können, müssen diese Zusagen intern überwacht und auf Korrektheit geprüft werden. Dies, um einerseits die Erbringung einer Dienstleistung einem Kunden gegenüber belegen zu können, aber auch um andererseits im Fall der Nichterbringung (unter der vereinfachten Annahme, das eine geringfügige Abweichung von den zugesicherten Parametern bereits einen Fehler darstellen) den Fehler eingrenzen zu können.

Die Überwachung findet dabei ausschließlich an den diensterbringenden IT-Ressourcen statt. Mit dem Ergebnis der Überwachung von materiellen IT-Ressourcen können Messwerte derart aggregiert werden, das Aussagen über den qualitativen Zustand eines IT-Dienstes möglich werden, um somit letztlich wiederum IT-Dienste im Rahmen von Geschäftsprozessen einsetzten zu können. Angestrebt wird demnach, die einfachen Messwerte (wie "CPU-Last" und "Netzwerkdurchsatz") hin zu qualitativen Aussagen zu aggregieren, um die im *SLA* festgeschriebenen Parameter zu realisieren (beispielsweise "Zustelldauer").

Messwerte sind in diesem Sinne Metriken, um eine Quantifikation bezüglich der Charakteristik eines Elements (in diesem Fall einer IT-Infrastrukturressource) vornehmen zu können. Metriken können demnach einfache Messwerte repräsentieren (atomare Metriken), wie auch komplexere Zusammenhänge wiederspiegeln, wobei die quantifizierende Maßzahl nicht direkt ermittelt (gemessen) werden kann, sondern Aufgrund von Berechnungsvorschriften berechnet wird (aggregierte Metrik).

Folgende Abbildung demonstriert diesen Sachverhalt.

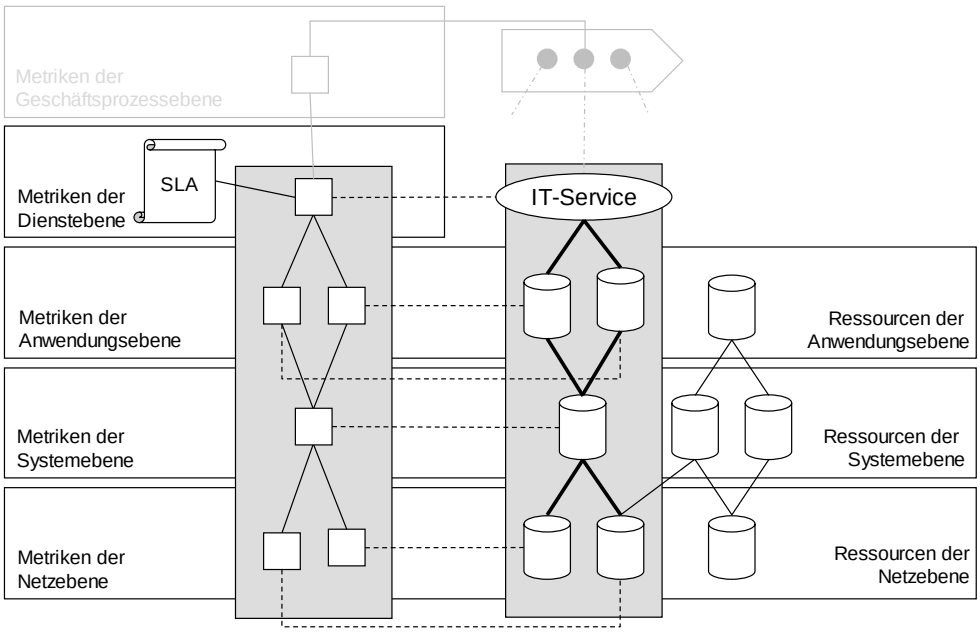

Abbildung 7 Aggregation der Metriken über Managementebenen hinweg

Angedeutet hierbei werden neben den Abhängigkeiten innerhalb der Infrastruktur auch Zusammenhänge zwischen Metriken der entsprechenden Komponenten. Zur Modellierung der entsprechenden Beziehungen existieren eine Vielzahl unterschiedlicher Ansätze [SO+01, SS+03, SM+02, MS+03, OMG06, DMTF03].

Während viele Arbeiten sich mit der Frage auseinander setzten, wie konkrete Berechnungen von Metriken im Falle von *bekannten* Abhängigkeiten der entsprechenden Komponenten auszuführen sind (beispielsweise [Hu07]), wird in der vorliegenden Arbeit angenommen, das die Beziehungen innerhalb einer Infrastruktur nicht zwangsläufig vollständig bekannt sein müssen und sich teilweise auch erst zur Laufzeit ergeben. Damit einhergeht die Feststellung, das Zusammenhänge, die aus dem Entwurf bekannt sind, sich erst zur Laufzeit qualitativ ausprägen in dem Sinne, das erst zur Laufzeit sich quantifizierende Maßzahlen angeben werden können. Zusammenhänge innerhalb der Infrastruktur spiegeln sich demnach auch in Zusammenhängen der entsprechenden Metriken wieder.

IT-Infrastrukturmanagement

Die restlichen drei Ebenen lassen sich zum IT-Infrastrukturmanagement zusammenfassen, da sie einheitlich aus Sicht der Serviceebene *IT-Ressourcen* darstellen.

Anwendungsmanagement

Die IT-Dienste werden in erster Linie von den Ressourcen der Anwendungsschicht produziert – dies umfasst sowohl die anwendungsrealisierenden Systemprozesse wie auch deren Datenbestände. Das Anwendungsmanagement befasst sich demnach mit der Überwachung, Steuerung und Instandhaltung von Softwareanwendungen. Bezogen auf den Laufzeitaspekt wird hier beispielsweise auf Warteschlangen, Puffer o.ä. operiert.

Innerhalb dieser Ebene ergeben sich zweierlei grundsätzliche Typen von Beziehungen:

(1) Statische (zur Entwurfs-/Installationszeit der Anwendung) bekannte Beziehungen. Dazu gehören Abhängigkeiten innerhalb der einzelnen Module einer Software wie auch Schnittstellenabhängigkeiten zwischen Anwendungen und dynamischen Funktions-Bibliotheken.
(2) Dynamische (zur Laufzeit der Anwendung) bekannte Beziehungen. Dazu gehören beispielsweise Abhängigkeiten, die sich durch Client/Server-Kommunikation ergeben und nicht ad hoc bekannt sind wie auch Auswirkungen einer Anwendung auf eine weitere aufgrund von konkurrierendem Ressourcenbedarf der System- und/oder Netzwerkebene.

Im Rahmen dieser Arbeit interessiert vor allem der dynamische Aspekt und somit die Frage, welche Abhängigkeiten sich zur Laufzeit zwischen Komponenten der Anwendungsebene ergeben, die qualitative Auswirkungen auf die zugesicherten Parameter einer IT-Dienstleistung haben.

Systemmanagement

Die abstrakten IT-Ressourcen der Anwendungsebene werden in letzter Konsequenz von den physikalisch-vorhandenen Ressourcen der Systemebene realisiert. Hierzu zählen CPU, Hauptspeicher, Festplatten, aber auch „aggregierte“ Ressourcen wie komplette Serversysteme. Das Management der Systemebene umfasst somit alle Maßnahmen, um einen einwandfreien Betrieb dieser Ressourcen zu ermöglichen. Wie auch beim Anwendungsmanagement können hier zwei unterschiedliche Typen von Beziehungen (statische und dynamische) ausgemacht werden.

Netzwerkmanagament

Das Netzwerkmanagement stellt die unterste der vier Ebene dar und umfasst die Maßnahmen zum Management der netzwerkgebundenen Infrastruktur. Dazu gehören neben der Überwachung der physikalischen Komponenten wie beispielsweise Router oder Hubs auch die Überwachung der Protokollimplementierungen auf den einzelnen Komponenten. Wie auch beim Anwendungsmanagement können hier zwei unterschiedliche Typen von Beziehungen (statische und dynamische) ausgemacht werden.

2.1.2 Unterteilung der Managementfunkionen

Innerhalb der unterschiedlichen Managementdisziplinen lassen sich die anfallenden Aufgaben gemäß [ISO89] in fünf Kategorien einteilen. Diese Einordnung steht orthogonal zur Unterteilung in Managementebenen und ist daher in jeder Ebene präsent. Wegen der Anfangsbuchstaben der englischen Bezeichnungen sind diese Disziplinen auch oftmals unter der Abkürzung *FCAPS* in der Literatur anzutreffen.

Die fünf Kategorien sind im Folgenden:

Fehlermanagement (Failure Management)

Fehlermanagement beschäftigt sich mit dem Entdecken, Eingrenzen und Beheben von anormalem Systemverhalten [HAN99]. Ein anormales Systemverhalten ist in diesem Sinne eine Abweichung der erbrachten Systemleistung im Vergleich zur zugesicherten Systemleistung. Beispielsweise kann ein Fehler der Ausfall einer Hardwarekomponente sein, wie auch eine Verletzung der zugesicherten Dienstgüte in einem IT-Dienst.

Der Definition nach lassen sich drei Teilaspekte im Fehlermanagement ausmachen. Um Fehler überhaupt entdecken zu können, muss eine vollständige Überwachung (*Monitoring*) der Infrastruktur in allen vier Teilebenen erfolgen. Die gesammelte Überwachungsinformation muss über die Grenzen der einzelnen Ebenen hinweg in Beziehung zueinander gestellt werden können, um Fehlerquellen einwandfrei identifizieren und eingrenzen zu können. Schließlich müssen Maßnahmen zur Verfügung gestellt werden, mit denen ein steuernder Eingriff möglich ist, um Fehlersituationen beheben zu können.

Das Fehlermanagement ist eine zentrale Herausforderung für jeden IT-Betreiber, stellt es doch die grundlegende Funktionsfähigkeit der Infrastruktur sicher, gleichwohl die Maßnahmen hierzu unterschiedlich in der Ausprägung und Komplexität sind. Gerade im Hinblick von wachsendem Kostendruck und Rationalisierungsbedarf werden die physikalischen Komponenten der IT-Infrastrukturen verstärkt massiv parallel in den Kontext unterschiedlicher virtueller IT-Ressourcen gestellt (Stichwort Virtualisierung). Ein Fehler innerhalb der physikalischen Komponenten kann demnach eine Vielzahl an Auswirkungen auf unterschiedliche Anwendungen der darauf aufbauenden Schicht haben, das rechtzeitige Erkennen, präzise Eingrenzen und wenn möglich rasche Beheben eines Fehlers ist das daher von entscheidender Bedeutung.

Genaues Wissen über die Abhängigkeiten der Komponenten einer IT-Infrastruktur ist somit eine direkte Forderung, die sich hieraus ableiten lässt.

Konfigurationsmanagement (Configuration Management)

In einer verteilten heterogenen Umgebung wird eine Vielzahl unterschiedlicher Komponenten eingesetzt – dies sind physikalische Hardwarekomponenten wie auch virtuelle Softwarekomponenten. Gemeinsam sind beiden Typen von Komponenten, das sie in gewissem Maße *konfigurierbar* sind, d.h. in ihrem äußeren Verhalten durch Parameter eingestellt werden können. Dieser Vorgang wird als konfigurieren bezeichnet, das Ergebnis, also die Summe der aktuell eingestellten gültigen Parameter, als Konfiguration [HAN99]. Die Konfiguration kann in der Regel getrennt von der Komponente persistent abgelegt werden, um so beliebige (abgelegte) Konfigurationen wiederherzustellen. Die Gesamtheit der einzelnen Konfiguration bestimmt im Endeffekt das Verhalten des Gesamtsystems, daher ist auch die Bezeichnung Konfiguration in Beziehung zu einem gesamten IT-System gebräuchlich.

Das Konfigurationsmanagement befasst sich demnach mit der Verwaltung, Überwachung und Steuerung der einzelnen Konfigurationen der Infrastrukturkomponenten. Hinzu kommt, dass das Konfigurationsmanagement in folgenden Aspekten eine unterstützende Rolle spielt:

(1) Im Rahmen von akuten Fehlerbehebungsmaßnahmen muss das Fehlermanagement eventuell steuernd in die Infrastruktur eingreifen, wodurch die Änderung von Konfigurationsparametern notwendig werden kann.
(2) Im Rahmen von Komponentenaktualisierungen müssen eventuell Konfigurationsdaten ausgetauscht werden. Hierbei ist vor allem das Wissen über grundlegende Beziehungen und Zusammenhänge der einzelnen Komponenten wichtig. Das Konfigurations-Management muss dementsprechend diesen Aspekt unterstützen.

Eine Datenbasis, die die beschriebenen Aspekte unterstützt (ablegen der Konfiguration, verfolgen von Änderungen, Analyse von Zusammenhängen und Auswirkungen von Konfigurationsparametern) ist daher von Vorteil.

Abrechungsmanagement/Benutzerverwaltung (Accounting Management)

Die Kosten, die durch die Bereitstellung von IT-Diensten entstehen, kann auf die Benutzer dieser Infrastruktur umgerechnet werden [HAN99]. Hierbei ist es notwendig, überhaupt eine Nutzungsberechtigung für entsprechende Infrastuktur-Komponenten/IT-Dienste zu erhalten. Das Abrechnungsmanagement ist demnach eng mit dem Benutzermanagement verbunden. Diese Managementfunktion spielt im Rahmen dieser Arbeit eine untergeordnete Rolle, wenn auch das im vierten Kapitel entworfene generische Modell für die Beschreibung von Abhängigkeiten prinzipiell in dieser Managementdisziplin angewendet werden kann.

Leistungsmanagement (Performance Management)

Das Leistungsmanagement ist von seiner Zielsetzung her die konsequente Fortführung des Fehlermanagements [HAN99]. Es kann als Verfeinerung des Fehlermanagements angesehen werden; während ersteres überhaupt dafür sorgt, das eine IT-Infrastruktur fehlerfrei funktioniert, unterstützt das Leistungsmanagement in Bezug auf qualitative Aspekte.

Diese qualitativen Aspekte werden durch *Service Level Agreements* an der Schnittstelle des Dienstes zum Nutzer hin dokumentiert und stellen somit ein gemeinsames Verständnis über die Qualität der zu erbringenden Leistung dar. Das Leistungsmanagement orientiert sich an den im SLA festgeschriebenen Qualitätsparametern und richtet die Managementfunktionalitäten dahingehend aus, das Abweichungen von diesen Parametern im Vorfeld der Dienstproduktion vermieden, mindestens aber erkannt werden. Der Erkennung einer Abweichung sollte eine Eingrenzung des Fehlers folgen (siehe Fehlermanagement), um somit für zukünftige Leistungsanforderungen die festgeschriebenen Qualitätsparameter wieder erfüllen zu können.

Zentrale Frage hierbei ist somit, wie die im SLA festgeschriebenen Parameter durch die überwachbaren Parameter aus der Infrastruktur heraus motiviert werden. Dies ist zum einen wichtig, da überhaupt untersucht werden muss, welche Aussagen im *SLA* getroffen werden können, zum anderen, um das Management an diesen Aussagen hin auszurichten.

Abhängigkeiten spielen hierbei eine entscheidende Rolle. Die Produktion einer IT-Dienstleistung wirkt sich „störend“ auf die erbringenden IT-Ressourcen aus (Rechenleistung wird in Anspruch genommen, Datenkommunikation zwischen Rechnerknoten findet statt, …), im Falle einer zunehmenden Nutzung eines IT-Dienstes und der damit verbundenden Abweichung von den zugesicherten Qualitätsparametern müssen eventuell Teile der Infrastruktur neu dimensioniert werden. Um die bestehenden Schwachstellen eindeutig zu identifizieren, muss jedoch der Einfluss jeder einzelnen Ressource auf die Produktion des IT-Dienstes bekannt sein. Da sich solche Beziehungen und gegenseitige Auswirkungen jedoch oft erst zur Laufzeit manifestieren, ist hierbei ein hochdynamischer Aspekt mit verbunden.

Sicherheitsmanagement (Security Management)

Das Sicherheitsmanagement umfasst all jene Maßnahmen, die die betriebliche Sicherheit einer IT-Infrastruktur gewährleisten, inklusive Autorisation und Authentifizierung von Nutzeridentitäten. Diese Managementfunktion spielt im Rahmen dieser Arbeit eine untergeordnete Rolle, wenn auch das im vierten Kapitel entworfene generische Modell für die Beschreibung von Abhängigkeiten prinzipiell in dieser Managementdisziplin angewendet werden kann.

2.2 Technisch-orientierte Beschreibungsansätze

Die zu verwaltenden Infrastrukturelemente – IT-Ressourcen und IT-Services, werden innerhalb einer Managementanwendung durch *Managed Objects* repräsentiert. Ein *Managed Object* steht

stellvertretend für ein Infrastrukturelement und definiert die Eigenschaften und Aktionen, die von einer Managementhandlung bezüglich des Infrastrukturelements möglich sind. Der Aufbau der *Managed Objects* und die Beziehungen untereinander wird im Informationsmodell einer Managementarchitektur beschrieben, weshalb das Informationsmodell eine grundlegende Stellung in den Betrachtungen in der vorliegenden Arbeit einnimmt. Da im Rahmen der Formulierung eines Konzeptes zur Erfassung der Wechselbeziehungen zwischen IT-Ressourcen ein konkreter Lösungsvorschlag erarbeitet werden soll, wird mit dem Common Information Model ein standardisierter Vertreter für ein (technisches) Informationsmodell näher untersucht.

Das *Common Information Model* (*CIM*) [DMTF05a] wurde im Hinblick auf die Plattformbindung schon existierender Management-Informationsmodelle von der *Distributed Management Task Force (DMTF)* mit der Vorgabe entwickelt, Plattform und Technologie unabhängig zu sein. Es sollte bestehende Informationsmodelle (SNMP, CIMP, DMI) ergänzen und vereinigen, nicht ersetzten. So kommt das *CIM* in Form eines konzeptionellen Modells daher. Aktuell liegt das CIM in Version 2.17.1 (Stand 27.Februar 2008).

Grundlegender Aufbau

Offensichtlichste Charakteristik des *CIM* ist der Objekt-orientierte Ansatz, der sich durch das komplette Informationsmodellmodell zieht. Repräsentiert werden die einzelnen Teile zum einen im grafischen Ausdruck durch *UML*-Diagramme, zum anderen in der maschinell bearbeitbaren Syntax *Managend Object Format (MOF,* [DMTF05a]*).*

Der grundlegende Aufbau des *CIM* wird in drei Ebenen aufgeteilt [DMTF05a]. Im Kern des Modells steht mit dem *Core Model* ein Informationsmodell, das in allen Bereichen des IT-Managements anwendbar ist. Es ist sowohl Plattform wie auch Technologie unabhängig.

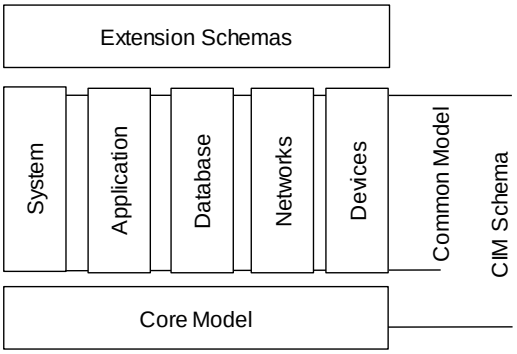

Abbildung 8 Übersicht CIM

Auf dem *Core Model* aufbauend definiert das *Common Model* ein Informationsmodell, das spezieller auf dedizierte Bereiche des IT-Managements zugeschnitten, gleichwohl Plattform und Technologie-unabhängig wie das *Core Model* ist. Die grundlegenden Modelle sind in die Bereiche *systems, applications, databases, networks* und *devices* eingeteilt.

Spezifiziert sind folgende Modelle (Common Models):

- *Application, Application-J2eeAppServer, Database, Device, Event, Interop, IPSecPolicy, Metrics, Network, Physical, Policy, Security, Support, System, User*

Zusammengenommen beschreiben das *Core Model* und die *Common Models* das *CIM Schema*.

Mit den *Extension Schemas* genannten Informationsmodellen kann auf dem *CIM Schema* aufbauend eine Abbildung in Technologie-spezifische Domänen vorgenommen werden. Hier haben Hersteller die Möglichkeit, Informationsmodelle für konkrete Produkte auszuliefern, um so bestehende Managementanwendung in Bezug auf deren Produkte erweitern zu können.

Meta Schema

Im Kern vom *CIM* steht das *CIM Meta Schema*. Es stellt eine formale Definition des Modells dar und definiert die Begriffe, deren Verwendung und Semantiken, die innerhalb der Modelle zum Einsatz kommen. Somit bezieht sich das Meta Schema auf den Aufbau des *CIM*, während sich die restlichen Schemata auf die zu beschreibende Managementinformation beziehen.

In folgender Abbildung 9 wird der Aufbau des Metaschemas als *UML*-Diagramm aufgezeigt.

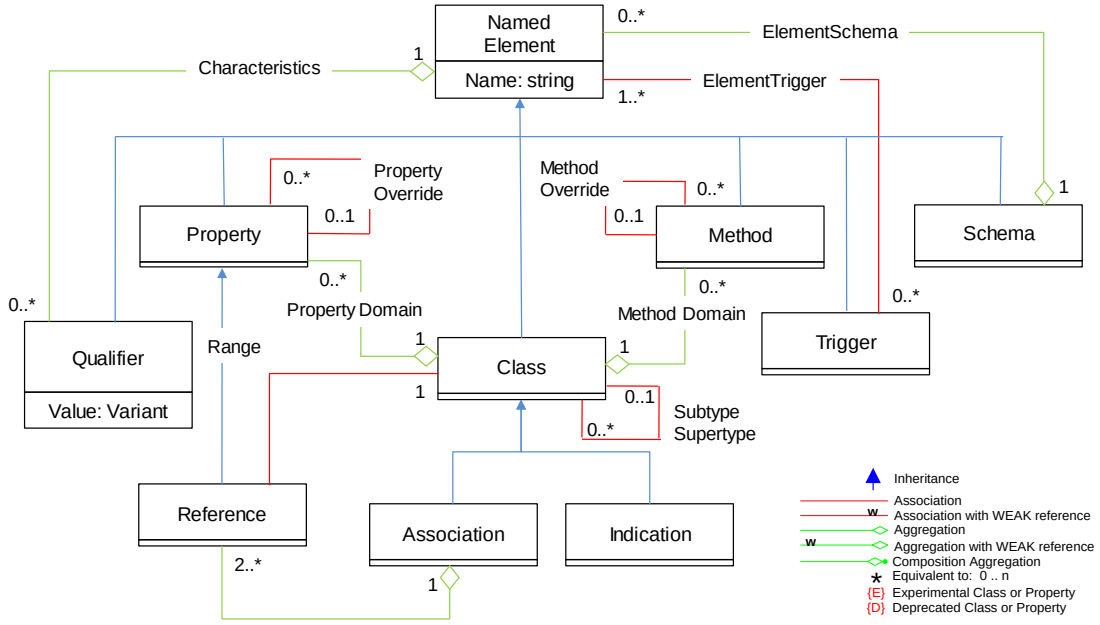

Abbildung 9 Übersicht über das CIM Meta-Schema (nach [DMTF05a])

Oberstes Element ist die abstrakte Klasse *Named Element*. Jedes Meta-Element des *CIM* wird durch diese Klasse beschrieben. Davon abgeleitet wird das zentrale Element *Class* (Klasse) welchem sowohl Eigenschaften (*Property*) wie auch Methoden (*Method*) zugewiesen werden können. [DMTF05a]. Managementobjekte werden demnach durch Eigenschaften und Methoden beschrieben, welche zusammengenommen eine Klasse von Gegenständen beschreiben. Dadurch ist es möglich, eine Objekt-orientierte Sicht auf die zu verwaltende Infrastruktur einzunehmen.

Ein *Trigger* überwacht Änderung im Zustand einer Klasse (beispielsweise Erschaffung, Löschung, Änderung oder Zugriff) oder Änderung im Zustand einer Eigenschaft (beispielsweise Änderung oder Zugriff) [DMTF05a]. *Indications* werden als Ergebnisse eines Triggers erstellt. Mit dem Konzept des Triggers kann somit nicht nur ein passives Überwachungskonzept entwickelt werden, bei dem Information über ein Managementobjekt mittels *get*/*set*-Funktionalitäten ermittelt wird, vielmehr können die Managementobjekte aktiv Information publizieren.

Core Model

Das Core Model spezifiziert die invarianten Teile des Informationsmodells und ist unabhängig von konkreten Produkten und technologischen Problemstellungen.

Folgende Abbildung zeigt eine Übersicht über den grundlegenden Aufbau der wichtigsten Klassen im Core Model (nach [DMTF00]).

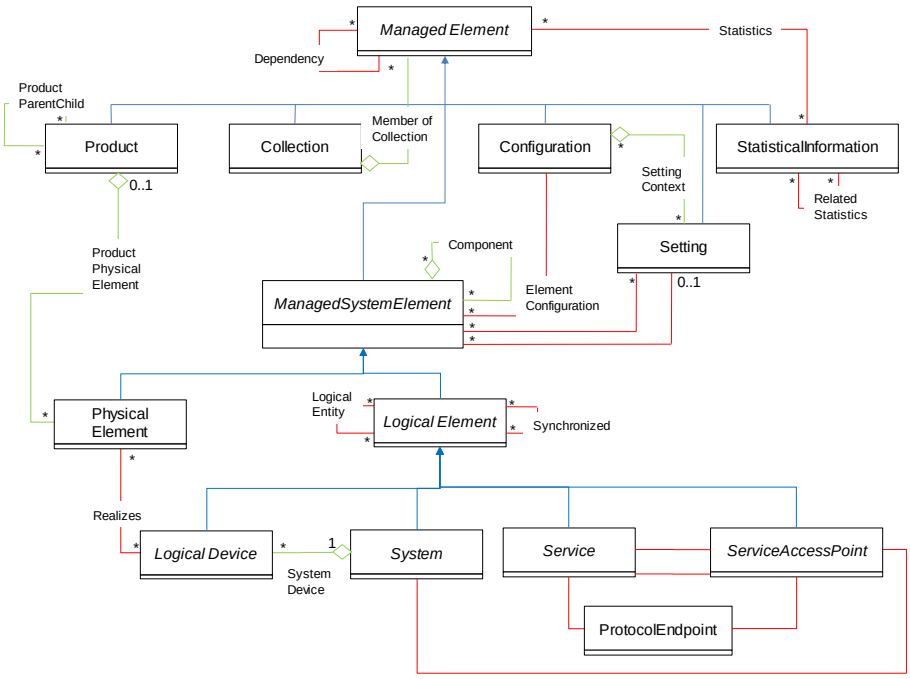

Abbildung 10 Übersicht über den Kernaufbau des Core Models (nach [DMTF00])

Die abstrakte Klasse *Managed Element* stellt die Wurzel im Klassenbaum von *CIM* dar. Sie agiert als Referenz für Assoziationen, die auf alle Elemente der Klassenhierarchie anwendbar sind. Im Kern des Modells steht die Klasse *ManagendSystemElement*. Sie repräsentiert alle Teile im weiteren Sinne eines IT-Systems, dies können sowohl einzelne Bauteile wie auch abstrakte netzbasierte Systemdienste sein. Aus dieser Klasse lassen sich die grundlegenden Klassen *Physical Element* und *Logical Element* ableiten, mit denen jeweils materielle wie auch immaterielle IT-Ressourcen modelliert werden können. Diese beiden Klassen stellen den zentralen Klassen für die Modellierung der Komponenten einer IT-Infrastruktur dar.

Für die unmittelbare weitere Verfeinerung in den *Common Models* sind die Klassen *Logical Device*, *System*, *Service*, *ServiceAccessPoint* und *ProtocolEndpoint* verantwortlich.

Abhängigkeiten

Beziehungen werden durch Assoziationen definiert. Eine Assoziationsklasse enthält als Attribut zwei oder mehr Elemente des Typs *Reference.* Das Element *Reference* spezifiziert somit die Rolle eines Elements innerhalb einer Assoziation. Eine Abhängigkeit innerhalb des CIM wird dann festgestellt, wenn eine Assoziation mit bestimmten Eigenschaften besteht. Wie diese Eigenschaften genau aussehen, ist nicht vorgeschrieben. Über die Verarbeitung der relevanten Managementinformation bezüglich Abhängigkeiten kann somit ein erster Anhaltspunkt für den strukturellen Aufbau eines verteilten Systems erstellt werden.

Da wie erwähnt die Assoziationen durch spezielle Assoziationsklassen zum Ausdruck gebracht werden, werden auch die entsprechenden Abhängigkeiten durch eigenständige Klassen beschrieben. Dies hat zur Konsequenz, dass *jede* Abhängigkeit durch eine *eigenständige* Klasse definiert wird, was wiederum eine Vielzahl unterschiedlicher Abhängigkeitsklassen nach sich zieht. Gemeinsam bei allen Klassen ist die Ableitung von der abstrakten Klasse `Dependency`. Abhängigkeiten werden im *CIM* generell an funktionalen Beziehungen (ein Objekt kann nicht

ohne das andere funktionieren) oder an existentiellen Beziehungen (ein Objekt kann nicht ohne das andere existieren) festgemacht.

Abbildung 11 gibt einen Ausschnitt der im *Core Model* definierten Abhängigkeiten wieder. Anhand dieses Diagramms ist der inflationäre Gebrauch des Klassenkonzept in Bezug auf die Modellierung von Abhängigkeiten erkennbar, schließlich wird jede Assoziation, und damit auch jede Abhängigkeit, durch eine eigenständige Klasse beschrieben.

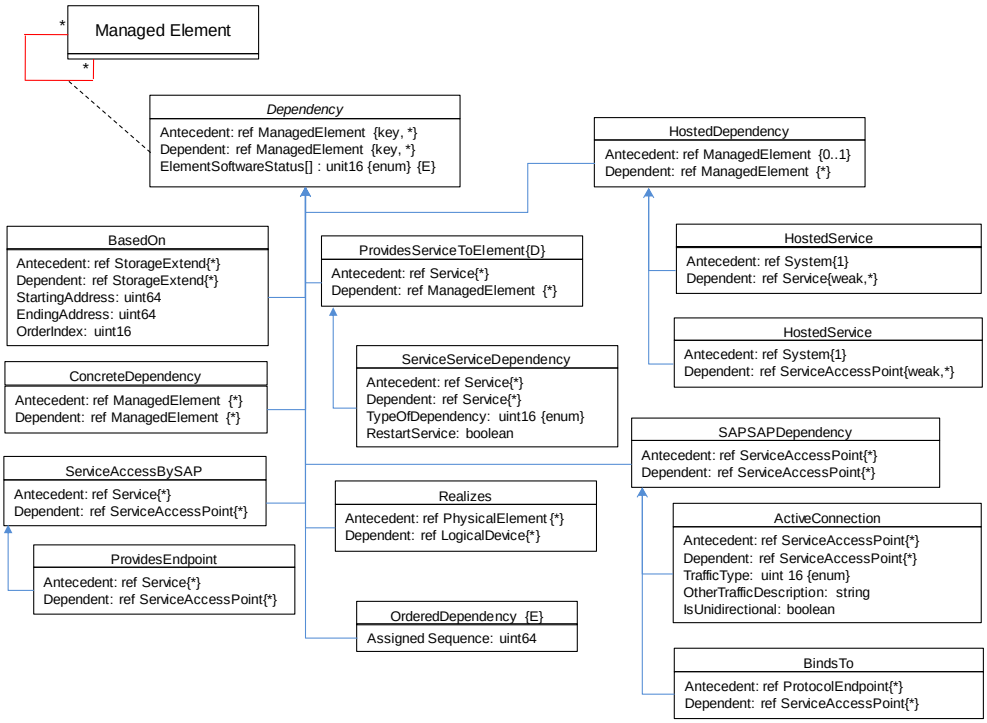

Abbildung 11 Übersicht der definierten Dependency-Klassen im Core Modell

Metriken

Die Klassen des *Core Model* bzw. der *Common Models* ermöglichen, die Objekte einer Infrastruktur zu beschreiben. Der Fokus liegt hierbei auf den *strukturellen* Gegebenheiten. Neben den statischen Merkmalen und Attributen eines Objekts sind jedoch auch gerade im Hinblick auf den Einfluss in die Produktion einer Dienstleistung die dynamischen Merkmale interessant. Hierunter verbergen sich Attribute wie Pufferfüllstände oder Umlaufzeiten. Da ad hoc nicht bekannt ist, welche dynamischen Merkmale an einer Komponente interessieren, ist die Modellierung dieses Aspektes aus den Klassen der Common Models ausgeklammert und mit dem *CIM Metrics Model* [DMTF03,DMTF06] in einem eigenständigen Modell niedergeschrieben.

Im Kern des Metrics Model stehen die Konzepte `UnitOfWork Model` und `BaseMetric Model`. Mit dem `UnitOfWork Model` wird ein logisches Element (der zu modellierenden Infrastruktur) mit einem Messwert (`BaseMetricValue`) verbunden. Der Fokus des `UnitOfWork Model` liegt hierbei auf der Modellierung von Antwortzeitverhalten von Anwendungen. Die Verbindung erfolgt nicht unmittelbar, sondern zunächst über die Definition einer Metrik (`BaseMetricDefinition`), welche über eine Spezialisierung dieser Klasse über `MetricDefinition` letztlich an die `UnitOfWork` gebunden wird. Dabei wird ähnlich der Trennung zwischen Metrik-Definition und Metrik-Wert eine Trennung zwischen der Definition der Arbeitsleistung (`UnitOfWorkDefinition`) und der eigentlichen Arbeitsleistung erreicht.

Nachfolgende Abbildung 12 zeigt die Zusammenhänge im *Metrics Model* auf.

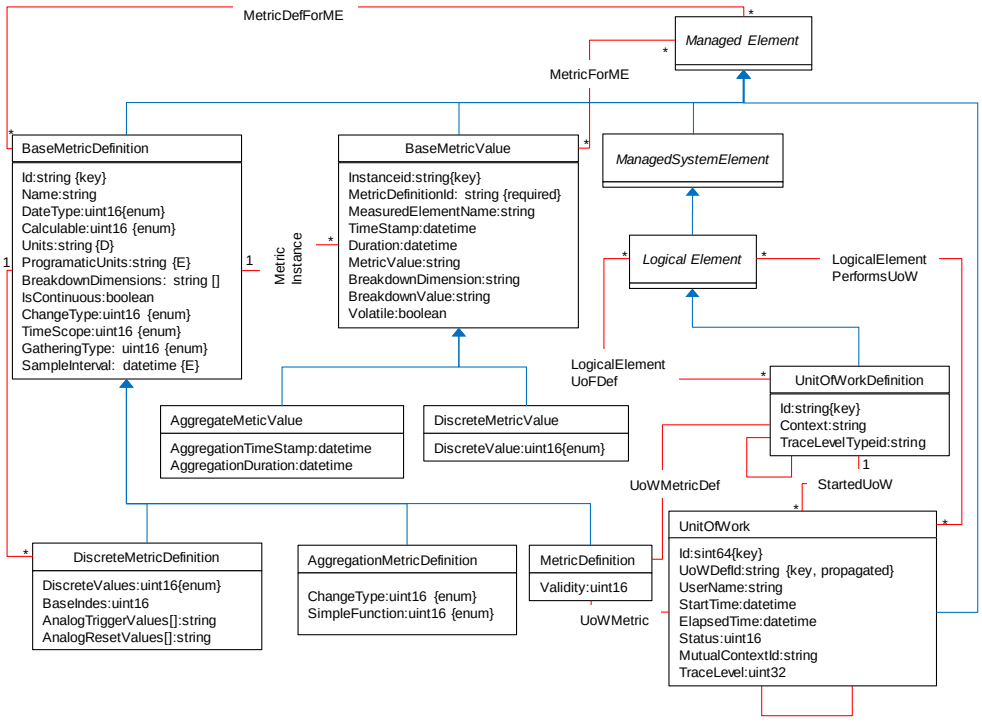

Abbildung 12 CIM Metrics Model [DMTF03,DMTF06]

Mit dem *CIM Metrics Model* existiert ein flexibles Modellierungskonstrukt, um dynamische Eigenschaften von Infrastuktur-Komponenten zu modellieren. [MH+07, Hue07] nimmt beispielsweise direkten Bezug zum *CIM Metrics Model*, und verbindet funktionale Abhängigkeiten in einer SOA mit den entsprechenden Metriken.

Jedoch zielt die Modellierung im *Metrics Model* auf Metriken, was nicht deckungsgleich mit dem Begriff Attributwert ist, der bei der Formulierung des generischen Ansatzes eine zentrale Rolle bei der Konzeption des Begriffes Ressourcenabhängigkeit einnimmt.

Managed Object Format (MOF)

Zur maschinellen Verarbeitung der durch das *CIM* spezifizierten Managementinformation eignet sich die Darstellung der UML Diagramme weniger. In [DMTF05] wird daher mit einer auf der *Interface Description Language* (*IDL* [OMG07]) basierenden Sprache die Syntax einer Beschreibungssprache zur Definition der Modellierungselemente des *CIM* festgelegt. Die Korrektheit einer im *MOF* vorliegenden Managementinformationsdefinition wird in der Regel durch einen *MOF*-Compiler validiert, um Inkonsistenzen bezüglich bestehender Definition zu vermeiden. Die Syntax von *MOF* selber ist in einer kontextfreien Grammatik festgelegt ([DMTF05.AppendixA]).

Zusammenfassung

Mit dem *Common Information Model* existiert ein umfassendes und standardisiertes Informationsmodell, um zu verwaltende IT-Ressourcen (wie auch den technischen Teil eines IT-Dienstes) zu beschreiben. Durch den Objekt-orientierten Ansatz können einfach Erweiterungen und Verfeinerungen eingebracht werden, ohne dass sich die grundsätzlichen Modelle und damit auf diesen Modellen aufbauende Managementapplikationen, ändern müssen. Durch die Möglichkeit, Assoziationen und damit auch Abhängigkeiten an allen Elementen der Modelle festzumachen, besteht die Möglichkeit, strukturelle und funktionale Zusammenhänge zu erfassen, die schon zur Entwurfszeit bekannt sind.

Dieser Ansatz wirkt sich jedoch nachteilig auf die Tatsache aus, das Abhängigkeiten, die sich zur Laufzeit durch konkrete Änderungen von Eigenschaften der Managementobjekte

manifestieren, nicht systematisch erfasst werden können. Zwar existiert mit dem *Metrics Model* ein *Common Model*, das die Modellierung von Metrikinformationen und damit auch Laufzeitaspekte zum Fokus hat, jedoch können keine Abhängigkeiten zwischen den Metriken modelliert werden in derart, dass sich die damit verbundenen Abhängigkeiten zwischen Infrastrukturkomponenten im Kontext einer Serviceproduktion zum Ausdruck bringen lassen, falls die Beziehungen zwischen den Metriken nicht genau bekannt sind (siehe hierzu das Metrikabhängigkeitsmodell in [Hu07]).

Grundsätzlich wird durch das *CIM* die technische Sicht auf Infrastrukturelemente beschrieben, wenngleich durch den angesprochenen Objekt-orientierten Ansatz Erweiterungen in Richtung der Beschreibung von *Service-Level-Agreements* vorgenommen werden können [DK03], wenngleich hier eigentlich nur technisch-realisierbare Parameter in die Formulierung der SLA's einfließen. Im Rahmen eines integrierten Dienstmanagementansatzes interessieren jedoch auch nicht-technische Parameter, daher kommt die Frage auf, wie die technische Sicht mit der nicht-technischen Aspekten verbunden werden kann. Hierdurch wird somit letztlich eine Brücke zwischen dem reinen technischen Management von Infrastrukturressourcen und dem Management von IT-Diensten geschlagen.

2.3 Strukturierende Beschreibungsansätze

Mit dem *Common Information Model* wird die technische Sicht auf eine Infrastruktur beschrieben. Hier stehen vor allem technische Parameter zur Überwachung und Steuerung der technischen IT-Ressourcen im Vordergrund. Demgegenüber interessiert im Hinblick auf die voranschreitende Ausrichtung am Begriff IT-Dienst auch die Sicht eines Nutzers auf den angeforderten Dienst, um letztlich die zugesicherte Diensterbringung am *Service-Access-Point* beim Service-Provider überwachen zu können. Um einen einheitlichen Bezug zwischen den *angeforderten* Leistungen (seitens eines Dienstnehmers) und den *angebotenen* Leistungen (seitens eines Dienstleisters) festzusetzten, werden SLA's eingesetzt. SLA's sind demnach ebenso wie IT-Ressourcen Gegenstand des Managements, gehören jedoch aufgrund ihres nicht zwangsläufig technischen Charakters nicht zum unmittelbaren technischen Management und werden somit nicht unmittelbar vom technischen Informationsmodell erfasst. Problematisch erscheint hierbei, das viele Ansätze bei der Formulierung von SLA's lediglich an technisch-orientierten Parametern ausgerichtet sind (beispielsweise in [DK03]).

Um die vom technischen Management gewonnene Information über Systemgrenzen hinweg austauschen zu können, müssen abstrahierende Beschreibungsansätze eingesetzt werden, um die vorhandene Information unabhängig vom eigentlichen Informationsgehalt austauschen zu können. Hierdurch wird letztlich eine Trennung zwischen reinem technischem Management, also jenen Aufgaben, die sich mit der Verwaltung, Überwachung und Steuerung der technischen Infrastrukturelemente befasst, und dem *Service Level Management*, also jenen Aufgaben, die sich mit der Verwaltung, Überwachung und Steuerung der Dienstleistungsvereinbarungen befasst, erreicht.

Diese Trennung macht deswegen Sinn, weil sich die unterschiedlichen Fachbereiche auf ihren jeweiligen Kernarbeitsbereich konzentrieren können, funktioniert aber nur dann zufriedenstellend, wenn die einzelnen Managementaspekte sich in einen holistischen Ansatz integrieren lassen. Dazu sind zum einen Standards im Bereich der technischen Informationsmodelle (beispielsweise *CIM*) notwendig, aber auch Standards im Bereich der höherwertigen strukturierenden Beschreibungsansätze, um die Information aus den technischen Modellen untereinander austauschen zu können.

Diese Forderungen können auf der einen Seite mit dem Einsatz der *XML* als Sprache zum Auszeichnen der Managementdaten, und auf der anderen Seite mit dem Einsatz des auf der *XML* aufbauenden Standards zur Beschreibung von Informationsressourcen, *RDF*, bedient werden.

2.3.1 Extensible Markup Language (XML)

Die *extensible Markup Language* (*XML*) des *W3C* (*world wide web consortium*, [W3Cc]) beschreibt eine Klasse von Datenobjekten, sogenannte *XML-Dokumente*, teilweise aber auch das Verhalten von Programmen, die diese Datenobjekte verarbeiten [Du99]. Der Zweck ist hierbei, Informationen *strukturiert* beschreiben zu können.

Die Elemente der Sprache haben zunächst keinen Bezug zum eigentlichen Inhalt, auch wird keine Semantik, also keine Regeln für die Bedeutung der Sprachelement, definiert. Durch die Sprache *XML* wird somit vom eigentlichen Inhalt eines Dokuments abstrahiert, und nicht die Bedeutung der beschriebenen Information (die Semantik) definiert, sondern lediglich die Grundlagen für Interpretationsregeln beim Domänen-übergreifenden *Informationsaustausch* gelegt.

Die zu beschreibende Information wird dabei über sogenannte *tags* (*engl*. Auszeichner) markiert und benannt. Die *tags* werden wiederum strukturiert verschachtelt, so dass sich eine Baumstruktur bildet. Bei der Beschreibung eines Serversystems *server1*, bestehend aus einer CPU *cpu1*, wird die eigentliche Information *server1* bzw. *cpu1* durch die *tags* <server> bzw. <cpu> geschachtelt dargestellt.

```
<server system=„linux“>
        <name>Server1</name>
        <cpu  currentLoad=„12.8“>
                <name>CPU1</name>
        </cpu>
</server>
```

Abbildung 13 Server-CPU Beispiel in XML

Zunächst definiert die *XML* lediglich grundsätzliche Elemente der Sprache. Um Regeln zu definieren, die auf die Gültigkeit des Aufbaus des Dokuments zielen, müssen die unterschiedlichen eingesetzten tags und deren Aufbau beschrieben werden. Diese Beschreibung findet entweder in *Document Type Definitions* (*DTD*) oder in *XML-Schema* (*XMLS*) statt. Beides sind standardisierte Ansätze, um gültige Regeln für den semantisch-korrekten Aufbau von Beschreibungen zu erhalten, die durch *XML*-Dokumente definiert werden.

Die erzielbare Baumstruktur ist ausgerichtet auf die hierarchische Beschreibung von Modellierungselementen, was jedoch nicht immer optimal ist. Aufgrund der Tatsache, dass lediglich Sprachelemente definiert werden, bzw. Regeln für den syntaktisch korrekten Aufbau, ist es schwierig mit diesen Mitteln einen semantischen Bezug innerhalb der Dokumentstruktur zu erreichen. Im Beispiel aus Abbildung 13 wird die semantische Bedeutung der "*besteht aus*"-Beziehung (*contains*) implizit über den hierarchischen Aufbau der Beschreibung definiert, weshalb eine Modellierung als eigenständiges Modellelement, wie in Abbildung 14 ersichtlich, vorgenommen werden kann ist.

```
<server system=„linux“>
        <name>Server1</name>
</server>

<cpu  currentLoad=„12.8“>
        <name>CPU1</name>
</cpu>

<contains>
        <father>Server1</father>
        <child>CPU1</child>
</contains>
```

Abbildung 14 Server-CPU Beispiel mit Modellierung der besteht-aus Beziehung

Im Rahmen der Fragestellung, wie Abhängigkeiten zwischen IT-Ressourcen untereinander oder im Bezug auf den Kontext eines IT-Dienstes automatisierbar erkannt und verarbeitet werden können, spielt die *XML* eine grundlegende Rolle:

(1) Als Möglichkeit, Information strukturiert, anwendungsunabhängig und frei von Semantik zu beschreiben, um Management-relevante Information unterschiedlicher Managementsysteme in Bezug zueinander setzen zu können. Dies setzt dann voraus, dass die entsprechende Logik in den Managementsystemen implementiert wird.
(2) Als Grundlage für das *Resource Description Framework* (*RDF*) und Ontologie-Sprachen, um durch maschinengestützte Verfahren Management-relevante Information automatisiert zu erzeugen (*Reasoning*)
(3) Als Grundlage für die *Data Center Markup Language* (*DCML*), um die Integration unterschiedlicher Datenbestände von Managementsystemen zu erleichtern.
(4) Als Verfahren im Kommunikationsprotokoll von *WBEM*-basierten Management-Systemen

2.3.2 Ressource Description Framework (RDF)

Das *Resource Description Framework* (*RDF*) ist ein Framework, um Information im *World Wide Web* (*WWW*) zu repräsentieren [W3C]. Eine Ressource im Sinne des *RDF* ist dabei alles, was identifiziert werden kann [VWZ01]. Einer Ressource werden Attribute (*properties*) und Werte (*values*) zugeordnet. Die Notwendigkeit für die Arbeit an diesem Standard lag darin begründet, dass der wachsenden Flut an Information durch die Akzeptanz des *WWW* Ende des letzten Jahrtausends das Problem gegenüberstand, in dieser riesigen Menge an vernetzten Datenbeständen bestimmte Informationen gezielt aufzufinden. Abhilfe schafft hierbei die Definition von Meta-Information, um vom eigentlichen Informationsgehalt eines Datums zu abstrahieren und einen strukturierten Zugriff darauf zu ermöglichen. Meta-Information definiert also ähnlich einem Meta-Modell ein Regelsystem, um die eigentliche Information, ähnlich dem eigentlichen Modell, formal erfassen zu können. Hierdurch kann die semantische Bedeutung einer Information bezüglich eines Wissensbereiches beschrieben werden.

Aufbau

Das *RDF* definiert eine abstrakte Syntax, um ein einfaches Graphen-basiertes Datenmodell umzusetzen. Es wird ein *Entity-Relations*-basierter Ansatz verfolgt, während *XML* einen hierarchisch-strukturierten Modellierungsansatz zu Verfügung stellt.

Die grundlegenden Konzepte sind hierbei unter anderem folgende [W3C]:
(1) Graphen-basiertes Daten Model
(2) *Uniform Resource Identifier* (URI) - basiertes Vokabular
(3) Definition einer XML-Syntax

Graphen-basiertes Datenmodell
Grundlegende Ausdruckweise im *RDF* stellt das *RDF-Statement* <*S*,*P*,*O*> dar, wobei *S Subjekt*, *P Prädikat* und *O Objekt* heißt. Dies bringt zum Ausdruck, das ein gewisser Bezug zwischen dem *Subjekt S* und dem *Objekt O* einer Aussage besteht, welcher durch das *Prädikat P* beschrieben ist. Subjekte sind demnach die Dinge, die identifizierbar sind, Prädikate deren Attribute und Objekte die Werte. Ein solches Triple heißt *RDF-Graph*, und kann wie in nachfolgender Abbildung zum Ausdruck gebracht werden.

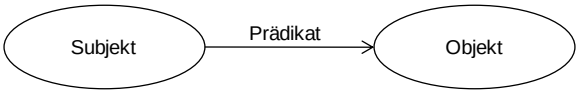

Abbildung 15 RDF-Graph

Die Elemente eines *RDF-Statements* sind in der Definition der abstrakten *RDF-Syntax* in [W3C] genau festgelegt.

Ein *RDF-Statement* bildet wiederum eine Resource, die somit ihrerseits durch weitere *RDF-Statements* verknüpft werden kann. Hierdurch können komplexe Bezüge aus einfachen Tatsachen hergestellt werden. Dementsprechend wird ein Objekt eines *RDF-Statements* wiederum zum Subjekt des verknüpften *RDF-Statements*.

Um das Beispiel aus Abbildung 13 aufzugreifen, kann die Aussage "Der Server besteht aus der CPU" in *RDF* durch Zuweisung des *Subjekts S*="Server" und *Objekts O*="CPU" sowie dem Prädikat *P*="besteht aus" und somit dem *RDF*-Tripel <"Server", "CPU", "besteht aus"> zum Ausdruck gebracht werden. In Abbildung 16 ist dieser Sachverhalt graphisch dargestellt. Auffällig hierbei ist die *URI*-Schreibweise der Bezeichner von Subjekt, Prädikat und Objekt, wobei der Präfix *http://example.org* nach gängiger Konvention bei hypothetischen Beispielen eingesetzt wird.

Abbildung 16 Server-CPU Beispiel als RDF Graph

URI-basiertes Vokabular
Durch die eigentliche Ausrichtung des *RDF* auf die Verknüpfung von Web-Inhalten, ist die Lokalisierung der Elemente über *Uniform Resource Identifier* (*URI*) vorgesehen. Hierdurch kann eine einfache Verknüpfung von unterschiedlichen Resourcen im *WWW* erreicht werden. Daher sind für Subjekte, Objekte und Prädikate ausschließlich folgende Belegungen zugelassen:

- Subjekt: URI oder leerer Knoten (zur Formulierung von Existenzaussagen)
- Prädikat: URI
- Objekt: URI, leerer Knoten oder Literale (einfache Datenwerte ohne aktive Rolle)

Definition einer XML-Syntax
XML ist die Grundlage für die Kodierung von *RDF*-Graphen. Hierzu werden die Elemente, Attribute, Namen und Inhalte durch *XML*-Ausdrücke wie in [W3Ca] definiert.

Abbildung 17 zeigt das Beispiel aus Abbildung 16 in *XML*-Notation.

```
1:  <?xml version="1.0"?>
2:  <rdf:RDF  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
              xmlns:ex="http://example.org/Syntax#">
3:  <rdf:Description rdf:about="http://example.org/server1">
4:  <ex:contains>
5:  <rdf:Description rdf:about="http://example.org/cpu1">
6:  </rdf:Description>
7:  </ex:contains>
8:  </rdf:Description>
9:  </rdf:RDF>
```

Abbildung 17 XML-Kodierung des RDF-Beispiels

Durch Einsatz von *XML* zur Kodierung für den Datenaustausch wird das *RDF* universell einsetzbar.

RDF Schema

Ähnlich der *XML* definiert das *RDF* nur eine Syntax von Sprachelementen, jedoch noch keine Regeln für den korrekten Aufbau oder die Interpretation der Information bezüglich einer bestimmten Domäne. Dies ist jedoch wichtig, wenn *RDF*-Aussagen bezüglich deren Korrektheit im Sinne der vorgesehen Semantik untersucht werden sollen.

Um die Korrektheit von *RDF-Statements* und den daraus resultierenden Aussagen überprüfen zu können, kann mit dem *RDF-Schema* [W3Cb] ein Vokabular bezüglich einer bestimmten Domäne definiert werden, um die in einer Domäne vorhandenen Ressourcen, deren Eigenschaften und Beziehungen untereinander anhand formaler Regeln zu beschreiben. Hierdurch kann letztlich eine nachvollziehbare Definition der *Semantik* bezüglich der durch die durch *RDF* beschriebenen Ressourcen erreicht werden, was als Grundlage für automatisierte Verfahren zur Erkennung und Verarbeitung von Beziehungen dienen kann. Durch diesen Ansatz werden nicht die eigentlichen Ressourcen, sondern deren Klassifizierung und der damit verbundenen Beziehungen beschrieben.

Ohne Definition der Semantik der *"besteht-aus"*-Beziehung aus obigem Beispiel könnte beispielsweise auch folgendes falsche *RDF-Statement* formuliert werden:

Abbildung 18 Syntaktisch korrekter, semantisch falscher Einsatz der *contains*-Beziehung

Syntaktisch ist dies zwar korrekt, jedoch ist eine CPU ein Teil eines Server-Systems, nicht umgekehrt, da für den Bezug der "besteht-aus"-Beziehung keine Festlegung innerhalb des RDF-Dokuments gemacht wurde.

Bei der Beschreibung eines *RDF Schemas* wird eine Klassifizierung der zu beschreibenden Ressourcen vorgenommen. Gültige Ausdrücke im *RDF* Dokument sind demnach Instanzen der Ausrücke im *RDF Schema* Dokument, wobei *RDF Schema* wiederum selber in *RDF* formuliert ist.

Folgende Beschreibungselemente stehen zur Verfügung [W3Cb]:

(1) Classes
- *a. rdfs:Class* : Abstrakte Oberklasse für alle Elemente in RDF Schema
- *b. rdfs:Resource* : Oberklasse, um alle durch RDF beschreibbaren Ressourcen zu erfassen
- *c. rdfs:Literal* : Oberklasse zur Beschreibung der atomaren Basistypen
- *d. rdfs:Datatype* : Oberklasse zur Beschreibung von komplexen Datentypen
- *e. rdf:XMLLiteral* : Oberklasse, um direkt auf XML-Ausdrücke referenzieren zu können.
- *f. rdf:Property* : Oberklasse zur Beschreibung von Properties

(2) Properties
- *a. rdfs:Domain* : bezieht sich auf den Ursprung einer Property
- *b. rdfs:Range* : bezieht sich auf das Ziel einer Property.
- *c. rdf:type* : gibt an, von welchem Typ (Klasse) eine Ressource in RDF ist.
- *d. rdf:subProperty, rdf:subClass* : Hiermit können innerhalb der Klassifizierung Vererbungsmechanismen eingebaut werden.

Neben diesen wichtigen Beschreibungselementen stehen noch eine Vielzahl weiterer Elemente zur Verfügung, um beispielsweise geordnete oder ungeordnete Listen definieren zu können [W3Cb].

Im Beispiel aus Abbildung 16 wird die *contains*-Beziehung mit dem Subjekt `contains_schema`, den beiden Properties `domain` und `range` mit den jeweiligen Objekten `Server_schema` bzw. `Cpu_schema` definiert. Hierdurch wird ausgedrückt, das zwischen der Klassifizierung eines Servers und der Klassifizierung einer CPU die contains-Beziehung in der Art besteht, das der Server als Ausgangspunkt betrachtet wird, und das Element CPU enthält.

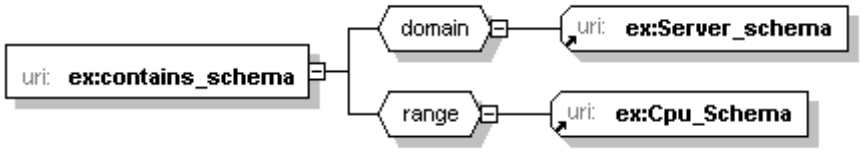

Abbildung 19 Graphische Darstellung des RDF-Schemas der *contains*-Beziehung

Bewertung

Das *RDF* wurde entwickelt, um auf einfache Weise Inhalte im *WWW* mit Meta-Information zu versehen, um Beziehungen zwischen diesen Inhalten darstellen zu können und für maschinelle Verarbeitung zugänglich zu machen. Durch die Definition der Syntax in *XML* wird das *RDF* universell einsetzbar und kann für den einfachen Austausch von Modellinformation über unterschiedliche Anwendungen und Systeme hinweg eingesetzt werden.

Eine Ressource ist im Sinne von *RDF* nicht das genaue Abbild der technischen Repräsentation einer IT-Ressource, sondern verbindet mit dem Begriff "Ressource" die abstrakte Beschreibung einer Informationsquelle. So ist das *RDF* nicht dafür konzipiert, Ressourcen an sich zu beschreiben, sondern die durch den *Subjekt-Prädikat-Objekt* basierten Ansatz möglichen Beziehungen zwischen Ressourcen.

Durch den Ansatz, Beziehungen zwischen Ressourcen beschreiben zu können, qualifiziert sich *RDF* potentiell als Lösungsansatz im Rahmen der in dieser Arbeit behandelten Fragestellungen. In [VWZ01] wird beispielsweise ein Ansatz beschrieben, *CIM* Modelle nach *XML* zu transformieren, wodurch die Möglichkeit entsteht, die entsprechenden *XML*-Elemente mittels *RDF* zu verknüpfen. Dabei wird ein *RDF*-Schema definiert, durch welches die *CIM* Modellelemente zum Ausdruck gebracht werden können. Dieses Vorgehen findet in der Verwaltung von Energie-Systemen (EVS, Energieversorgungssysteme) verstärkt großen Zuspruch, da hier durch die voranschreitende Privatisierung der ehemals staatlichen Unternehmen und der damit einhergehenden Notwendigkeit, interoperabel über verschiedene Energiesysteme hinweg zu bleiben, der Druck auf die Festlegung eines einheitlichen Standards zum Austausch von Managementinformation gewachsen ist [VMZ01]. Vergleichbare Ansätze, aus den Informationen des technischen Informationsmodells heraus, *RDF*-Ausdrücke zu formulieren, können beispielsweise in [QAW+04] gefunden werden. Siehe hierzu auch Kapitel 3.2.2.

2.4 Managementarchitektur

Generell kann zwischen *isolierten*, *koordinierten* und *integrierten* Managementansätzen unterschieden werden [HAN99]. Da in heutigen Infrastrukturen eine Vielzahl unterschiedlicher Komponenten von unterschiedlichen Herstellern eingesetzt wird, entsteht hierdurch ein Höchstmaß an Heterogenität. Im gleichen Zug werden oftmals eine Reihe unterschiedlicher spezialisierter Tools eingesetzt, um dieser Heterogenität begegnen zu können.

Beim isolierten Ansatz arbeiten die eingesetzten Werkzeuge unabhängig voneinander. Dabei werden sowohl Funktionen wie auch Datenbestände der einzelnen Werkzeuge separat von einander in Anspruch genommen und verwaltet. Im Hintergrund der steigenden Komplexität der Infrastruktur auf der einen und wachsendem Kostendruck auf der anderen Seite, ist dieser Ansatz nicht mehr zu rechtfertigen [HAN99].

Beim koordinierten Ansatz werden die isoliert voneinander stehen Werkzeuge in ihrer Funktionserbringung untereinander koordiniert [HAN99]. Die Kooperation wird in der Regel durch Steuerungsprozesse (Skripte) untereinander realisiert. Der Aufwand ist in der Regel geringer als beim isolierten Ansatz, jedoch erweist sich die Koordination mithilfe der

Steuerungsprozesse gerade auch im Hinblick der steigenden Vernetzung und damit der steigenden Komplexität als schwerwiegende Fehlerquelle, falls hier nicht zusätzlicher Aufwand betrieben wird.

Im Rahmen eines integrierten Managementansatzes werden oftmals eine Vielzahl unterschiedlicher Tools, Standards und Protokolle eingesetzt. Eine übergreifende Architektur garantiert eine einheitliche Sicht auf die eingesetzten Managementanwendungen und ermöglicht ein herstellerübergreifendes Zusammenspiel. Eine Managementarchitektur definiert somit ein Rahmenwerk für das Management relevante Aufgaben.

Um ein herstellerübergreifendes Zusammenspiel der unterschiedlichen Komponenten zu gewährleisten, lassen sich vier Teilmodelle innerhalb des Rahmenwerks identifizieren.

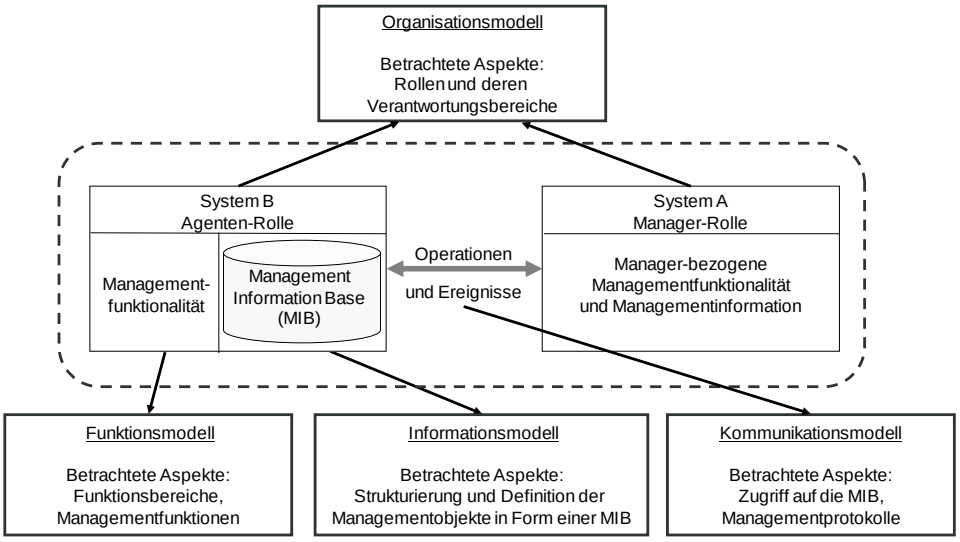

Abbildung 20 Teilmodelle einer Managementarchitektur nach [Ga07]

Nachfolgend werden die vier Teilmodelle kurz erläutert. Für eine detaillierte Beschreibung sei auf [HAN99, Me06] verwiesen. Anschließend wird der Standard WBEM im Hinblick auf das Managementreferenzmodell untersucht.

Informationsmodell

Im Informationsmodell werden die zu managenden Ressourcen und deren Beziehungen untereinander beschrieben. Somit kann im Informationsmodell Wissen über Abhängigkeiten gespeichert werden. Eine Ressource wird durch ein Managementobjekt (*managed object*, MO) dargestellt, welches somit eine abstrakte Repräsentation der Ressource innerhalb des Informationsmodells ist. Das Informationsmodell ist somit frei von konkreten Anwendungen, Plattformen oder Protokollen und stellt eine Abstraktion der zur verwaltenden Entitäten der zu managenden Umgebung dar.

Dem Informationsmodell kommt eine zentrale Bedeutung innerhalb der Managementarchitektur zu, definiert es doch die zu verwaltenden Objekte und deren Beziehungen zueinander. Daher ist die Mächtigkeit einer Managementanwendung, basierend auf der generischen Managementarchitektur, von der Mächtigkeit des zugrunde liegenden Informationsmodells abhängig.

Im Rahmen der in dieser Arbeit entwickelten Referenzarchitektur (Kapitel 6) wird das *Common Information Model* als zugrunde liegendes Informationsmodell eingesetzt (siehe hierzu auch 2.2)

Funktionsmodell

Das Funktionsmodell definiert einzelne funktionale Bereiche die sich aus den Eigenschaften der Managementobjekte des Informationsmodells ergeben. Die Funktionen werden dabei generisch beschrieben und lassen sich in die fünf Managementdisziplinen gemäß FCAPS eingliedern

Organisationsmodell

Im Organisationsmodell wird der organisatorische Aufbau der einzelnen Komponenten des Managementsystems beschrieben. Dabei wird definiert, welche Rollen existieren und wie einzelne Komponenten in diese Rollen eingeordnet werden.

Kommunikationsmodell

In diesem Modell werden die Protokolle zum Austausch von Managementinformation zwischen den einzelnen Komponenten des Managementsystems definiert.

Das Kommunikationsmodell unterstützt drei Aspekte:

(1) Managementinformation wird passiv von einem Managementobjekt gesammelt (*get*)
(2) Auf ein Managementobjekt wird steuernd zugegriffen (*set*)
(3) Ein Managementobjekt versendet von sich aus Managementinformation (*alarm/trap*)

Konkrete Managementarchitektur: Web-Based Enterprise Management (WBEM)

Basierend auf der generellen Managementarchitektur hat die *DMTF* eine Sammlung von Protokollen und Spezifikationen unter dem Begriff *Web-Based Enterprise Management* (*WBEM*) zusammengefasst, um so eine einheitliche und offene Plattform für das Management von verteilten Umgebungen zu schaffen. Die Fortschritte in den vier Teilmodellen sind unterschiedlich stark ausgeprägt, auch gibt es in bestimmten Bereichen keine verbindlichen Vorschriften.

Im Kern der *WBEM*-Architektur steht das *Common Information Model*, das eine Entwicklung zur Integration bestehender Managementansätze wie *SNMP* [RFC1157] oder *DMI* [DMTF03a] darstellt (siehe auch 2.2).

Folgende Abbildung gibt eine Übersicht über eine typische WBEM Architektur.

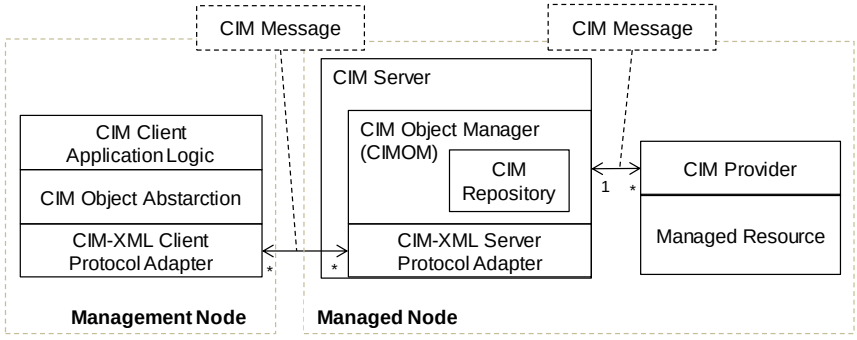

Abbildung 21 WBEM Architektur nach [DMTF]

Es lässt sich eine grundlegende Unterteilung zwischen *Management Node* und *Managed Node* feststellen (entspricht einem *Manager-Agent* Aufbau im Organisationsmodell). Weiterhin wird auf dem *Managed Node* zwischen dem *CIM Server* und *CIM Providern* unterschieden. Ein *CIM Provider* stellt eine Schnittstelle zwischen den spezifischen Eigenschaften der *Managed Resource* und dem plattformunabhängigen CIM-Kommunikationsprotoll dar. Daher können auf einem *Managed Node* mehrere *CIM Provider* einem *CIM Server zugeordnet* werden.

Zentraler Bestandteil des *Managed Nodes* ist neben angepassten *CIM Providern* der *CIM Object Manager* (CIMOM). Er verwaltet die konkreten Instanzen der Management Objekte, welche durch Daten der *CIM Provider* gebildet werden. Im *CIM Repository* sind die gültigen Beschreibungen der Managementobjekte der aktuell eingesetzten CIM-Version abgelegt.

Die Kommunikation zwischen dem *CIM Client* (*Management Node*) und dem *CIM Server* (*Managed Node*) wird über eine Kapselung der definierten CIM-Operationen in *CIM-Messages* durchgeführt. Dabei werden die gewünschten Operationen in *XML* transformiert und über http transportiert.

3 STAND DER TECHNIK

Nach der thematischen Einführung und der Vermittlung grundlegender Begriffe aus dem Bereich IT-Management werden nun weitere Ansätze untersucht, die sich weitestgehend mit der Frage beschäftigen, wie Abhängigkeiten zwischen Ressourcen einer Infrastruktur dargestellt und erfasst werden können. Desweiteren muss die Frage geklärt werden, wie sich Abhängigkeiten *überhaupt* charakterisieren lassen. Dabei werden sowohl die Vor- wie auch die Nachteile jedes Ansatzes herausgearbeitet um die Motivation für den Ansatz in dieser Arbeit nachvollziehen zu können.

3.1 Analyse des Begriffs Abhängigkeit

Im Rahmen dieser Arbeit wird ein grundlegendes Verständnis für den Begriff Abhängigkeit in Bezug auf die Produktion eines IT-Dienstes, entwickelt. Dies ist notwendig, da in bestehenden Arbeiten [KBK00, KK01, CDS01, HSM+06, DLS05] aber auch im CIM dieser Schritt nicht oder nicht vollständig vollzogen wird. Gerade aber eine präzise Erfassung von Abhängigkeiten und deren Eigenschaften, losgelöst von speziellen semantischen Bezügen wie beispielsweise Abhängigkeiten im Rahmen von Fehlermanagement oder die Erfassung von qualitativen Auswirken von abhängigen Komponenten, ermöglicht den Vergleich von unterschiedlichen Ansätzen in diesem Bereich. Dabei werden Arbeiten aus unterschiedlichen Bereichen betrachtet. Es stellt sich heraus, das auf der einen Seite eine Übereinstimmung dabei herrscht, Abhängigkeiten formal verstehen zu müssen, die hierbei notwendigen formalen Ableitungen werden jedoch zumeist nur innerhalb einer definierten Problemdomäne entwickelt.

Zunächst werden unterschiedliche Aspekte von Abhängigkeiten untersucht, um über die Defizite der betrachteten Ansätze eine Motivation für das im vierten Kapitel vorgestellte generische Vorgehen begründen zu können.

Statische und Dynamische Zusammenhänge

Generell können zwei wesentliche Unterschiede bei der Sicht auf eine Infrastruktur, und damit auf die IT-Service erbringenden IT-Ressourcen, festgemacht werden: Den Zustand zum Zeitpunkt des Entwurfs eines verteilten IT-Dienstes und den aktuellen Zustand zum Zeitpunkt der Produktion einer IT-Dienstleistung. Während ersterer Aspekt eine *statische* Sicht auf *generelle* Zusammenhänge [KBK00] vermittelt, gibt die Sicht zur Laufzeit einen Blick auf die dynamischen Zusammenhänge.

Hierdurch kann eine grobe Einordnung von Abhängigkeiten anhand differenzieller Modelle durchgenommen werden, die sich grob am Lebenszyklus eines IT-Dienstes (*IT-Service Lifecycle*) orientieren. [KBK00] führt hierzu zwei Modelle ein, welche in [KK01] um ein weiteres Modell erweitert und verfeinert wird.

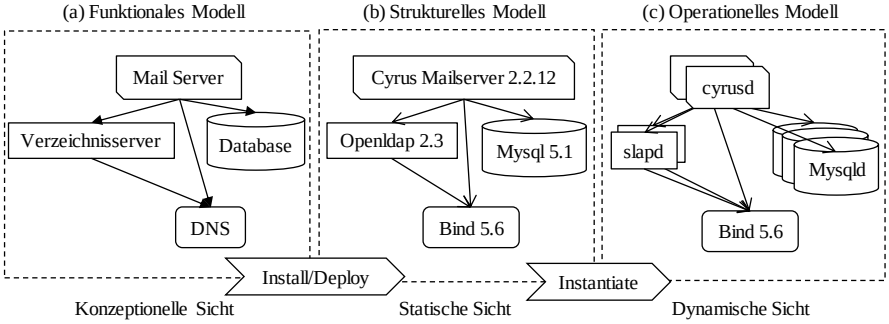

Abbildung 22 Abhängigkeitsmodelle am Service-Lifecycle orientiert

Die unterschiedlichen Modelle lassen sich wie folgt beschreiben:

(a) *Funktionales Modell*: In diesem Modell werden grundlegende Zusammenhänge beschrieben. Die Beschreibung orientiert sich hauptsächlich an generellen Beziehungen, die sich zwischen den verschiedenen (Teil-)Diensten einer IT-Infrastruktur im Hinblick auf einen konkreten IT-Service ergeben. Hierbei wird sowohl von konkreten IT-Ressourcen wie auch von Details abgesehen. Das funktionale Modell kann somit als Serviceplan aufgefasst werden, aus dem ersichtlich ist, welche Komponenten in einer Beziehung zueinander stehen, ohne Details der Art der Beziehung anzugeben. Abhängigkeiten werden in diesem Modell durch gerichtete Pfeile zwischen den unterschiedlichen Komponenten dargestellt.

(b) *Strukturelles Modell*: In diesem Modell werden die grundlegenden Zusammenhänge zwischen konkreten eingesetzten (Software-)Komponenten einer IT-Infrastruktur dargestellt. Dieses Modell ist eine Verfeinerung des funktionales Modells in der Hinsicht, das sowohl keine weiteren Teildienste/Komponenten wie auch keine weiteren Abhängigkeiten hinzugefügt werden, sondern die im funktionalen Modell identifizierten Komponenten spezifiziert werden.

Sowohl Funktionales wie auch strukturelles Modell stellen statische Sichtweisen auf eine IT-Infrastruktur und deren Zusammenhänge dar. Die Information in diesen Modellen kann beispielsweise wie in [KBK00, HSM+03, DAS+06, Li03, LHY06, SJS05] bereits vor der Laufzeit eines Systems bestimmt werden.

(c) *Operationelles Modell*: Dieses Modell gibt eine Sicht zur Produktionszeit eines IT-Dienstes wieder. Das Modell ist wiederum eine Verfeinerung des strukturellen Modells und unterliegt daher der gleichen Prämisse wie dieses auch. Die Verfeinerung des Modells geht dabei dahingehend soweit, das die im strukturellen Modell spezifizierten (Software-)Komponenten durch einzelne Prozessinstanzen ausgedrückt werden, wodurch eine Verknüpfung der unterschiedlichen Komponenten auf einer sehr konkreten Ebene stattfindet.

Während funktionales und strukturelles Modell eine statische Sicht zur Entwurfszeit eines IT-Dienstes wiedergeben, kann mit dem operationellem Modell die Dynamik innerhalb einer Infrastruktur wiedergegeben werden. [KK01] schlägt dabei eine schrittweise Verfeinerung vom funktionalen Modell hin zum operationellen Modell vor. Diese Sicht ist insofern von entscheidender Bedeutung, da sich bestimmte Zusammenhänge erst zur Laufzeit ergeben.

Dabei stellen sich eine Reihe neuer Fragen:

(a) Wie wird das Wissen um Zusammenhänge im funktionalen Modell, bzw. im strukturellen Modell generiert? [KBK00] schlägt hierzu eine auf den Softwarerepositories der eingesetzten Betriebssysteme aufbauendes Verfahren vor, [KK01] erweitert dieses Vorgehen um eine auf *XML*/*RDF*-basierende abstrakte Beschreibungsschicht aufgesetzte Methodik. Dabei wird nicht geklärt, ob ein formales Konzept für diese Vorgehensweise abgeleitet werden kann.
[Li03] untersucht Abhängigkeiten im Bereich der komponentenbasierten Softwareentwicklung (*CBSE, component based software engineering* siehe auch [LHY06], [SJS05]). Dabei werden Abhängigkeiten betrachtet, die sich schon zur Zeit des Entwurfes ausprägen – beispielsweise inter-Modul Funktionsaufrufe, Datenabhängigkeiten, Steuerflussabhängigkeiten. Es werden acht unterschiedliche Typen von Abhängigkeiten definiert, die sich hauptsächlich an den möglichen Problembereichen innerhalb der Softwareentwicklung orientieren: *Data Dependency, Control Dependency, Interface Dependency, Time Dependency, State Dependency, Cause Dependency, Input/Output Dependency, Context Dependency.*

(b) Die schrittweise Verfeinerung von einer abstrakten Sicht hinab zu den konkreten (softwarebasierten) IT-Ressourcen garantiert keine vollständige Erfassung aller in einen IT-Dienst eingebundenen IT-Ressourcen. Die Modellierung in [KBK00, KK01] bleibt zum einen auf reine softwarebasierte IT-Ressourcen beschränkt, zum anderen findet keine Unterscheidung zwischen IT-Dienst und IT-Ressource statt. Somit stellt sich die Frage, wie ausgehend von einer gegebenen Menge von IT-Ressourcen der Kontext zu einem bestimmten IT-Dienst hergestellt werden kann. Dies stellt besonders im Hintergrund der wachsenden Dynamisierung im Bereich Service-orientierten Architekturen (SOA) eine besonders starke Prämisse dar.

(c) Die Dynamik innerhalb eines Systems wird zwar durch das operationelle Modell ausgedrückt, jedoch wird die Modellierungsmöglichkeit auf reine funktionale Zusammenhänge beschränkt. Dies ist zwar schlüssig innerhalb der logischen Herangehensweise, vom funktionalen Modell über eine schrittweise Verfeinerung hinab zu den operationellen Zusammenhängen zu argumentieren, hierdurch wird jedoch die Möglichkeit genommen, detailierte Information bezüglich der Auswirkungen der IT-Ressourcen untereinander während der Produktion eines IT-Dienstes zu integrieren.

Die Einordnung anhand statischer bzw. dynamischer Aspekte macht dahingehend Sinn, das durch die Orientierung am Lebenszyklus eines IT-Dienstes (Entwurf, Anpassung, Betrieb) auch die unterschiedlichen Aspekte im Management dieses Dienstes betrachtet werden. Beispielsweise eignet sich die strukturelle Sicht auf eine Infrastruktur für die Unterstützung des Change-Managements während das funktionale Modell einen generellen Überblick über die Infrastruktur verschafft. Sobald jedoch Aspekte des operationellen Betriebes untersucht werden müssen (Failure-, Performancemanagement), muss eine detailierte Sicht auf die Infrastruktur und damit eine detailierte Sicht auf die einzelnen Komponenten dieser Infrastruktur vorliegen.

Charakterisierung von Abhängigkeiten anhand von Attributen

Während [KBK00, KK01] einen ersten groben Ansatz liefert, um Abhängigkeiten anhand statischer bzw. dynamischer Ausprägungen zu unterscheiden, gehen [KBK00, HSM+06, CDS01] weiter und differenzieren Abhängigkeiten anhand von Attributen, die an den Abhängigkeiten ausgemacht werden können.

[KBK00] unterscheidet zwischen folgenden sechs Attributen:

a) **Space/Locality/Domain**: gibt an, wie weit (nah) eine Komponente von deren abhängiger Komponente innerhalb eines Systems lokalisiert ist.
   Mögliche Werte: *inter-domain, intra-domain, inter-system, intra-system, intra-package*
b) **Component Type**: gibt an, welcher wirklicher Typ der Komponente zugeordnet werden kann
   Mögliche Werte: *hardware, software, logical entity*
c) **Component Activity**: gibt an, ob die Komponente aktiv instrumentiert werden kann (Hardware, Systemprozess, …) oder passiv ist und nicht direkt instrumentiert werden kann (Konfigurationsdatei)
   Mögliche Werte: *active, passive*
d) **Dependency Strength**: gibt an, wie stark eine Komponente von einer anderen abhängt.
   Mögliche Werte: *none, optional, mandatory*
e) **Dependency Formalization**: gibt an, welchen Abstraktionsgrad die Abhängigkeit inne hat
   Mögliche Werte: *low, high*
f) **Dependency Criticality**: gibt an, welche Anstrengungen unternommen werden müssen, um diese Abhängigkeit zu erfüllen
   Mögliche Werte: *prerequisite, co requisite, exrequisite*

Zusätzlich zu diesen sechs grundlegenden Eigenschaften lassen sich folgende Charakteristika von Abhängigkeiten untersuchen, die nicht direkt mit obigen Attributen in Verbindung stehen und somit allgemeiner Natur sind:

a) **Time**: Abhängigkeiten lassen sich einem bestimmten Zustand eines System zuordnen, welcher wiederum von der Zeit abhängt. Beispiele hierfür sind temporär benötigte Speicherkapazitäten während der Installation einer Softwarekomponente
b) **Dependency Lifecycle**: gibt den Zustand im Lebenszyklus einer Abhängigkeit an. Es wird unterschieden zwischen funktionalen und strukturellen Abhängigkeiten. Dabei ändert sich dieser Typ im Laufe der Zeit von einer reinen funktionalen Abhängigkeit hin zu einer strukturellen mit detaillierter Statusinformation.

Auf [KBK00] aufbauend unterscheidet [CDS01] zwischen folgenden sechs Attributen:

(a) **Importance:** In [KBK00] entspricht dies dem Attribut „criticality“. Es gibt an, wie wichtig diese Abhängigkeit für die abhängige Entität in Bezug auf Erfüllung ihrer Funktionalität ist.
Mögliche Werte: *not applicable, high, medium, low*
(b) **Strength**: definiert ein Maß für die Häufigkeit aus Sicht einer abhängigen Entität, wie oft diese Abhängigkeit innerhalb einer bestimmten Zeitperiode erfüllt sein muss. Obwohl dieses Attribut den gleichen Namen wie in [KBK00] trägt, kommt ihm hier eine andere semantische Bedeutung zu.
Mögliche Werte: *value/time period*
(c) **Sensitivity**: wie stark ist diese Abhängigkeit anfällig für Fehlverhalten
Mögliche Werte: *fragile, moderate, robust*
(d) **Stability**: gibt an, in welchen Zeitperioden diese Abhängigkeit anfällig für Fehlverhalten ist. Dieses Attribut entspricht in etwa dem Attribut „time“ in [KBK00]
Mögliche Werte: *extremely stable, infrequent, periodic, defined time*
(e) **Need**: Dies definiert ein höherwertiges semantisches Attribut, mit dem angegeben wird, welche Bedürfnisse die Vorgängerentität (*antecedent*) der abhängigen Entität (*dependent*) erfüllt.
Mögliche Werte: *beliebige Information*
(f) **Impact**: gibt an, welche Auswirkung eine Verletzung der Abhängigkeitserfüllung auf die funktionale Erbringung der abhängigen Entität diese Abhängigkeit hat.
Mögliche Werte: *none, mission compromised, performance degraded, information unreliable, Corruption of information/communication*

Ebenfalls auf [KBK00] aufbauend unterscheidet [HSM+06] zwischen folgenden Attributen, wobei festgestellt werden kann, das die definierten Attribute in eine Typisierung von Abhängigkeiten in Richtung der Unterscheidung zwischen IT-Service und IT-Ressource ermöglichen sollen:

(a) **type of dependency**: es wird unterschieden zwischen *inter-service, service-resource* und *inter-resource* Abhängigkeiten.
(b) **level of detail**: Dieser Aspekt richtet sich an Abhängigkeiten innerhalb einer Servicebeziehung und gibt an, ob ein Service als Gesamtes oder vielmehr einzelne Funktionalitäten einer Dienstleistung untersucht werden.
(c) **redundancy**: gibt an, ob die betrachtete Abhängigkeit isoliert untersucht werden kann oder Teil einer redundanten Infrastruktur ist und somit in eine ganzheitliche Betrachtung mit weiteren Abhängigkeiten gestellt werden muss. Mögliche Ausprägungen hier sind *isolierte* und *zusammengesetzte* Abhängigkeiten
(d) **dynamic**: eine Abhängigkeit kann charakterisiert werden nach deren zeitlichen Verhalten. Daher kann zwischen *statischen* und *dynamischen* Abhängigkeiten unterschieden werden
(e) **service lifecycle**: gibt an, in welchem Lebensabschnitt eines IT-Dienstes diese Abhängigkeit auftaucht
(f) **degree of formalization**: gibt an, wie diese Abhängigkeit beschrieben werden kann. Beispiele hierfür sind *formal model, pseudo code* und *no formalization*.

(g) **characteristic**: Es wird erkannt, das Abhängigkeiten nicht nur *funktionaler*, sondern auch *organisatorischer* Natur sein können.

Es stellt sich die Frage, woraus sich die definierten Attribute motivieren, bzw. ob sich ein Ansatz auf einen weiteren reduzieren lässt. Außerdem ist ersichtlich, dass die Attribute „*critically*“ und „*strength*“ in [KBK00] und [CDS01] unterschiedlich interpretiert werden. [DLS05] führt das Attribut „*strength*“ im Hinblick auf die Überschneidung von Fehlermengen ein, wodurch eine weitere begriffliche Interpretationsmöglichkeit geschaffen wird. Nachfolgende Tabelle gibt eine Übersicht der unterschiedlichen Attribute der unterschiedlichen Ansätze.

[KBK00]	**[CDS01]**	**[HSM+06]**
Dependency Strength	Strength	Type of dependency
Dependency Criticality	Importance	level of detail
Dependency Formalization	Sensitivity	Redundancy
Space/Locality/Domain	Stability	Dynamic
Component Type	Need	service lifecycle
Component Activity	Impact	degree of formalization
Time		characteristic
Dependency Lifecycle		

Tabelle 1 Vergleich der Attribute von Abhängigkeiten in [KBK00,CDS01,HSM+06]

Aufgrund der Tatsche, das die Charakterisierung von Abhängigkeiten nicht eindeutig ist, stellt sich die Frage, inwiefern eine automatisierte Erfassung von Abhängigkeiten schließlich korrekt ist, oder vielmehr nur denjenigen Teil von Abhängigkeiten erfasst, der sich unter dem Wirkbereich der jeweiligen Problemdomäne ergibt. Eine Einordnung anhand von Eigenschaften, die bestimmte Abhängigkeiten unter einem gewünschten Blickwinkel aufweisen macht dann Sinn, wenn ein abgeschlossenes Managementsystem für eine bestimmte Zielsetzung gefordert wird, nicht jedoch wenn nach einer integrierten Lösung gesucht wird.

Formalisierung von Abhängigkeiten

Wie die Defizite in [KBK00, CDS01, HSM+06] und weiterhin in [DLS05, KK01] aufzeigen, besteht ein Bedürfnis, den Begriff Abhängigkeit formal herzuleiten. [CDS01] kommt dieser Forderung am ehesten nach, die formale Herangehensweise in [DLS05] bezieht sich auf die Fragestellung, wie sich Fehler in verteilten Systemen über Fehlermengen beschreiben und die Auswirkungen auf die Funktionalität damit verknüpfter Komponenten erfassen lassen.

In [CDS01] wird versucht, den Begriff Abhängigkeit über die Begriffe *Entität* (Entity) und *Veränderung* (Change) generell zu definieren. Eine Entität ist demnach ein Element einer Menge, wobei eine Menge definiert ist als „*a set can be a set of anything“*.. Dadurch entsteht die Möglichkeit, den Begriff Entität an beliebige Gegenstände der zu modellierenden Realität zu heften, insbesondere auch immaterielle Dinge wie beispielsweise ein Objekt, ein Konzept eine Organisation oder einen abstrakten IT-Dienst. Weiterhin wird angenommen, das Entitäten nicht statischer Natur sind, sondern sich vielmehr in ihrer Ausprägung ändern. Mit Ausprägung wird hierbei der messbare Zustand in der Gesamtheit aller Eigenschaften (Attribute) verstanden. Diese Veränderung wird nicht genauer definiert, auch wird keine Verbindung dazu hergestellt, woran diese Veränderung festgemacht werden kann.

Als verbindende Komponente wird der Begriff *Relation* eingeführt, wobei angenommen wird, das eine Relation $\Upsilon(A, B, C, D, \dots)$ zwischen den Entitäten A, B, C, D existiert. Eine

Abhängigkeit χ wird schliesslich definiert als Element der Relationsmenge $\chi \in \Upsilon$ wobei die Relation gilt, falls sie so beschaffen ist, das die Änderung von einer Entität $\alpha \in \chi$ eine Änderung einer weiteren Entität $\beta \in \chi$ bewirkt.

Grafisch kann diese generelle Abhängigkeit durch Knoten-Kanten-Diagramme zum Ausdruck gebracht werden. Dabei sind die Kanten des Grafen ungerichtet, den Knoten werden entweder Entitäten zugewiesen, oder Abhängigkeiten (*Dependency*), welche Entitäten miteinander verbinden. Bei generellen, ungerichteten Abhängigkeiten spricht man auch von *bi-direktionalen Abhängigkeiten.*

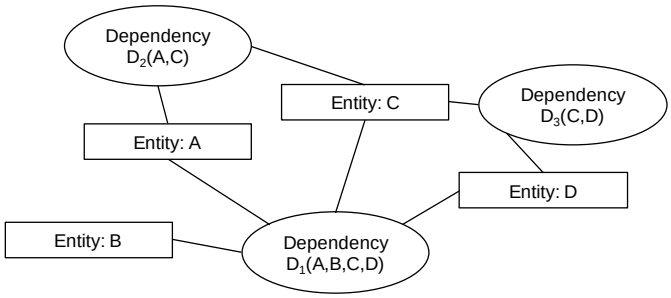

Abbildung 23 Grafische Darstellung von Entitäten und Dependencies

Die generelle Abhängigkeit lässt sich vereinfachen, ohne dass eine Untersuchung dieser vereinfachten Abhängigkeit an Bedeutung verliert. Dazu wird die generelle *n-stellige Abhängigkeit* auf eine zweistellige Abhängigkeit reduziert, also eine Abhängigkeit die nur zwischen zwei Entitäten A, B besteht. Für den Fall, das eine Entität mit sich selbst in Abhängigkeit steht, kann $B = A$ gesetzt werden, womit $\chi(A, A)$ gilt.

Diese bi-direktionalen Abhängigkeiten können weiter vereinfacht werden, indem die Richtung der Abhängigkeit in die Betrachtung mit einbezogen wird Die Abhängigkeit $\chi_1(A, B)$ gibt an, dass die Abhängigkeit nur von A nach B besteht, genauer, dass A von B abhängig ist. Eine Veränderung von B bewirkt somit eine Veränderung von A. Dabei werden die bezeichnenden Begriffe *dependent* für die abhängige Entität A und *antecedent* für die vorangehende Entität B verwendet.

Dem ungerichteten Grafen in Abbildung 23 entspricht diese Situation in einem Beispiel mit den zwei Entitäten A, B die Situation in Abbildung 24:

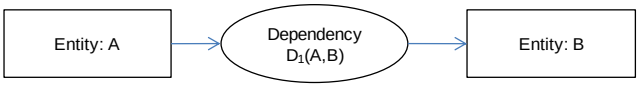

Abbildung 24 Gerichtete Entity/Dependency

Innerhalb dieser Sicht kann eine ungerichtete Abhängigkeit $\chi(A, B)$ durch zwei gerichtete Abhängigkeiten $\chi_1(A, B)$ und $X_2(B, A)$ zum Ausdruck gebracht werden. Analog zur Ausdrucksweise bi-direktionale Abhängigkeit für eine ungerichtete Beziehung wird die Bezeichnung *uni-direktionale Abhängigkeit* für eine gerichtete Beziehung angebracht.

Obwohl die Autoren in [CDS01] eine formale Herangehensweise an die Definition des Begriffes Abhängigkeit anstreben, wird eine Umsetzung nicht bis in letzter Konsequenz vorangetrieben. So bleibt die Betrachtung der Entitäten auf die Tatsache beschränkt, das sich der Zustand einer Entität ändert, und daher eine Abhängigkeit bestehen muss. Der Gedanke kann aber an dieser Stelle vertieft werden um den Begriff Abhängigkeit an den jeweiligen Attributen, die untereinander in Beziehung stehen, aufzuhängen.

Im Kern des Formalismus in [DLS05] steht der Begriff der Fehlersemantik. Dazu wird ein Verständnis für Fehlerklassen geschaffen um anschließend den Fokus auf die speziellen Bedingungen im Rahmen von Echtzeitfähigen Kontrollsystemen zu lenken. Oftmals stehen bei diesen Systemen Betrachtungen im Kontext von „Zeit“ im Mittelpunkt, wobei sich auch unterschiedliches Fehlverhalten in diesem Bereich definieren lässt.

Die Fehlerklassen können nun als Grundlage für die weitere Betrachtung dahingehend erweitern werden, das durch Parametrisierung ein anwendungsspezifisches Fehlerverhalten erfasst werden kann. Dazu wird eine Mengendefinition angeführt, die eine betrachtete Komponente *A* mit deren Fehlerklasse *S* in Verbindung bringt, bezeichnet durch *spezifische Fehlersemantik*FS_A^S. Demnach ist eine *Fehlersemantik* einer Komponente A die Menge bezeichnet durch FS_A.

Die mengentheoretische Definition des Begriffes Fehlersemantik erlaubt nun, ebenso einfach den Begriff Abhängigkeit über eine mengentheoretische Beschreibung zu erfassen, jedoch mit dem Hintergrund, das spezielle Abhängigkeiten – Abhängigkeiten die sich durch Fehlerfortpflanzung ergeben – untersucht werden.

Dazu werden zwei gedankliche Komponenten *A* und *B* betrachtet, die sich dadurch auszeichnen, dass in einer Kommunikationsbeziehung zwischen diesen beiden Komponenten ein Datenaustausch stattfindet. Komponente *A* erwartet für die Erfüllung einer angebotenen Funktion Daten von einer Komponente *B.* Durch die Annahme, das ein Fehler in Komponente B eine Nichterfüllung des angebotenen Funktionsdienstes nach sich zieht, kann angenommen werden, das dieser Fehler auch in der Nichterfüllung der Aufgabe von Komponente A feststellbar ist. Übertragen auf die Idee, Fehler über deren Fehlersemantiken mengentheoretisch zu beschreiben, kann dieser Umstand dadurch ausgedrückt werden, das die entsprechende Fehlersemantik von Komponente FS_B^S in die Fehlersemantik der Komponente FS_A^S gemappt wird:

$$map\left(FS_B^S, B, A\right) = FS_A^{S'}.$$

Die grundlegende Annahme dabei ist, das ein möglicher Fehler in Komponente B dadurch festgestellt werden kann, das dieser Fehler in der Fehlersemantik von Komponente A auftaucht. Durch diese Annahme kann schließlich der Begriff Abhängigkeit intuitiv definiert werden, indem Abhängigkeiten dort existieren, wo im Sinne von Fehlerabhängigkeiten Fehlersemantiken einer vorangehenden Komponente eine Teilmenge von Fehlersemantiken von einer abhängigen Komponente darstellen.

In diesem Sinne kann somit auch die Gewichtung einer Abhängigkeit dadurch definiert werden, das eine Aussage über die größte Teilmenge FS_B^S der Fehlersemantik von Komponente B FS_B gemacht wird, so dass gilt: $map\left(FS_B^S, B, A\right) = \emptyset$.

Weiterhin werden totale Abhängigkeiten und partielle Abhängigkeiten unterschieden in dem Sinne, das totale Abhängigkeiten (*TDep(A,B)*) in jedem Fall einen Folgefehler in einer abhängigen Komponente nach sich ziehen, während bei partiellen Abhängigkeiten (*PDep(A,B)*)die Feststellung gemacht wird, das die Fehlersemantiken einer Vorläuferkomponente nicht vollständig in der Fehlermenge einer abhängigen Komponente enthalten sein muss, ein Fehler in der Vorläuferkomponente somit nicht zwangsläufig zu einem Fehler in der abhängigen Komponente führen muss. Die Begriffe totale bzw. partielle Abhängigkeit können durch folgende Feststellung verbunden werden:

$$Dep(A, B) = TDep(A, B) \vee PDep(A, B)$$

Eine Abhängigkeit ist somit entweder partiell oder total.

Eine direkt Anwendung der Möglichkeit, Gewichtungen von Abhängigkeiten bestimmen zu können, kann darin gesehen werden, hochkritische Komponenten einer Infrastruktur zu identifizieren. Mit diesem Wissen kann die Konstruktion eines Systems dahingehend beeinflusst werden, das hochkritische Komponenten zur Erfüllung ihrer gedachten Aufgaben nicht von minder-kritischen Komponenten abhängig gemacht werden.

Modellierung von Abhängigkeiten

Es existieren eine Reihe unterschiedlicher Ansätze, um Abhängigkeiten zu modellieren. Im einfachsten Fall können gerichtete Kanten/Knoten-Graphen eingesetzt werden, falls klar ist, wie der Begriff Abhängigkeit im jeweiligen Kontext aufzufassen ist.

[CDS01] beschreibt die Modellierung von Abhängigkeiten anhand von konzeptionellen Graphen (sieht auch Abbildung 24). Nachteilig hierbei ist jedoch die Tatsache, das solche visuellen Modellen schwieriger maschinell bearbeitbar sind, da in der Regel ausführbare Meta-Modelle fehlen.

Innerhalb *UML* [OMG06] wird der Begriff Abhängigkeit (*Dependency*) über die Änderung der *Definition* eines Elements (*antecedent, server*) und der damit verbundenen Änderung der Definition eines abhängigen Elements (*dependent, client*) aufgespannt:

> „*A relationship between parts of an UML model drawn as a dashed line with an arrow head that indicates that if one thing changes then the change may effect the other thing. Often one thing uses the other thing*." [CSCI].

Eine Abhängigkeit definiert somit eine (statische) Beziehung, die zur Modellierungszeit zwischen zwei Elementen ausgemacht werden kann.

Im Common Information Model (*CIM*, siehe auch Kapitel 2) werden Abhängigkeiten durch Assoziationsklassen zwischen den einzelnen Elementen der *CIM*-Schemata definiert. Die Abhängigkeit besteht dann, wenn die Elemente in einer besonderen Beziehung zueinander stehen – wie diese besondere Beziehung aussieht, wird nicht definiert. In [DMTF05] taucht hierbei der Hinweis auf, das Abhängigkeiten entweder funktionaler oder existentieller Natur sind. Diese Definition lässt somit ein gewünschtes Maß an Abstraktion vermissen. [HSM+06] greift den Ansatz auf, die Assoziationsklassen vom *CIM* durch Einführung einer hierarchischen Struktur zu untergliedern, die angesprochenen Schwachstellen können so jedoch auch nicht behoben werden.

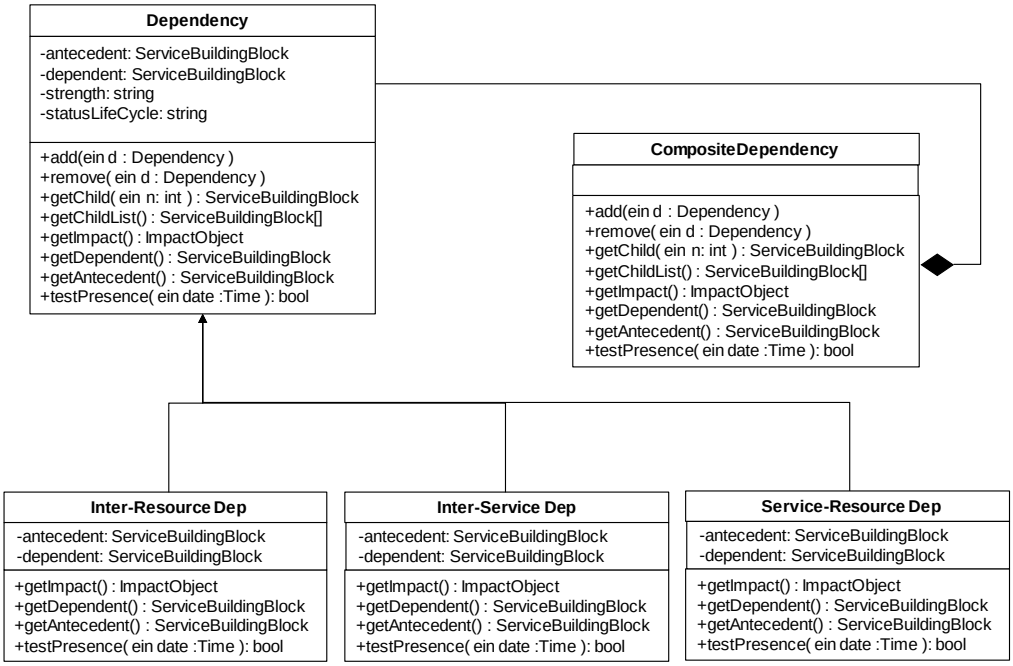

Abbildung 25 Erweiterung von *CIM_Dependency*

Dabei wird das Softwaredesignpattern „*composite*“ wird umgesetzt, um die abstrakte Klasse *CIM_Dependency* in Richtung der Möglichkeit einer Hierarchiebildung zu erweitern. Dazu werden die Klassen *Inter_Resource_Dependency*, *Inter_Service_Dependency* und *Service_Resource_Dependency* mittels der Klasse *CompositeDependency* an die Klasse *CIM_Dependency* gekoppelt.

Motivation für die Formulierung eines generischen Ansatzes

[KBK00, KK01] führen eine grobe Unterscheidung von Abhängigkeitsmodellen an der Orientierung des *Service-Lifecycles* ein. Dabei wird eine schrittweise Verfeinerung vom abstrakten funktionalen Modell hinab zur detailierten Sicht auf instanziierte Komponenten im operationellen Modell angestrebt. Bei einer gegeben Menge an IT-Ressourcen wird zumeist nach den Abhängigkeiten von IT-Ressourcen untereinander und damit die Auswirkungen dieser Abhängigkeiten im Bezug auf eine IT-Dienstleistung gefragt. Daher ist eine entgegengesetzte Sicht anzustreben, die zunächst die Dynamik einer Infrastruktur in den Vordergrund der Betrachtung stellt, um daraus zum einen auf aktuelle Entwicklungen innerhalb der Infrastruktur reagieren zu können, und zum anderen über die Betrachtung von dynamischen Zusammenhängen auf statische Sichtweisen schließen zu können.

Die in [KBK00, CDS01, HSM+06] vorgeschlagene Charakterisierung von Abhängigkeiten anhand von Attributen, welche dem Konzept Abhängigkeit im jeweiligen Kontext zugeordnet werden kann, ist insofern nicht schlüssig, da ein formales Herangehen an die Problemstellung fehlt. Die definierten Attribute lassen sich nicht schlüssig begründen, zumal in [KBK00] und [CDS01] sowie in [DLS05] das Attribute „*strength*“ unterschiedlich interpretieren lässt. Die Motivation für eine formale Herangehensweise, gerade im Hinblick auf die automatisierte Erfassung von Abhängigkeiten, die sich erst zur Laufzeit ergeben, kann somit einfach begründet werden.

[CDS01] führt eine formale Betrachtungsweise des Begriffes Abhängigkeit über die Begriffe Entität und Veränderung (von Zuständen von Entitäten) ein, ohne jedoch den Formalismus bis auf die Ebene der Attribute von Entitäten durchzuführen. Somit ist der Versuch, Abhängigkeiten zwischen Objekten einer IT-Infrastruktur anhand einer abstrakteren Betrachtung des Begriffes Entität und Attribut(-Wert) letztlich nicht konsequent bis zu Ende

geführt. Der Formalismus in [DLS05] motiviert sich aus der Problemstellung heraus, Auswirkungen von Fehlern über eine mengentheoretische Betrachtung von Überschneidungen von Fehlermengen, durchzuführen. Das im vierten Kapitel vorgestellte generische Modell verknüpft die Ansätze von [CDS01] und [DLS05] dahingehend, das versucht wird, Abhängigkeiten, die sich zur Laufzeit eines Systems über die Änderung von Zuständen von Objekten einer Infrastruktur und somit im gedanklichen Modell über die Änderung von Wertemengen von Entitäten manifestieren, erfassen zu können.

Mit der Möglichkeit, Abhängigkeiten im *CIM* zu beschreiben kann ein bestehendes Informationsmodell herangezogen werden, um einheitliche Managementanwendungen zu entwickeln. Da im *CIM* durch die Ableitung an der abstrakten Klasse *CIM_Dependency* und der Realisierung von Abhängigkeiten durch Assoziationsklassen ein flacher, unstrukturierter Raum aufgespannt wird, muss eine Möglichkeit geschaffen werden, Abhängigkeiten hierarchisch zu gliedern. [HSM+06] stellt hierzu einen Ansatz vor, jedoch wird nicht untersucht, inwiefern der dynamische Aspekt des operationellen Modells unterstützt werden kann. Genau aber dieser Aspekt interessiert im Rahmen von Betrachtungen der Auswirkung von Abhängigkeiten in die Produktion eines IT-Dienstes.

Schlussfolgerung

Der in [CDS01] eingeführte Ansatz, Abhängigkeiten nicht direkt an den Ressourcen zu definieren sondern über die abstrahierte Betrachtung der Begriffe Entität und Veränderung (der Entitätszustände) einen Zusammenhang herzustellen, wird im vierten Kapitel aufgegriffen und verfeinert. Dabei werden die in [CDS01, DLS05] zugrunde liegenden Ansätze erweitert, um Abhängigkeiten, die sich zur Laufzeit ergeben (operationelles Modell in [KK01]), an den Merkmalen (Werte von Attributen) einer Entität, und damit innerhalb der zu modellierenden Infrastruktur, feststellen zu können. Mit der in [HSM+06] vorgeschlagenen Erweiterung der abstrakten Klasse *CIM_Dependency* besteht die Möglichkeit, ein bestehendes Informationsmodell als Basis für eine Modellierung zu verwenden.

3.2 Analyse strukturierender Ansätze

Neben der reinen technischen Sicht, die durch das CIM in dieser Arbeit eingenommen wird, interessiert im Rahmen der Fragestellung, wie Managementinformation über unterschiedliche Wissensdomänen hinweg in Beziehung zueinander gesetzt werden kann.

Diese Fragestellung interessiert im Hinblick auf die Zielsetzung, das Management von IT-Diensten durch Wissen aus der IT-Infrastruktur zu verbessern in dem Sinne, das sowohl technische wie auch nicht-technische Parameter zusammengenommen bei der Formulierung von SLA's herangezogen werden können. Eine Abstrahierung von den technischen Modellelementen wird demnach notwendig.

3.2.1 Datacenter Markup Language (DCML)

Oftmals werden eine Reihe unterschiedlicher Managementanwendungen für spezialisierte Aufgabenbereiche eingesetzt. Während die einzelnen Anwendungen unabhängig voneinander über System- und Managementgrenzen hinweg für einen kleinen Aufgabenbereich spezialisiert sind, fehlt oftmals neben einem einheitlichen Zugang auch eine Integrationsmöglichkeit, die angefallenen relevanten Managementdaten in einem verbindenden Kontext zueinander zu betrachten.

Die *Organization for the Advancement of Structured Information Standards* (*OASIS*, [OASIS]) definiert mit der *Datacenter Markup Language* (*DCML*, [DCMLa]) eine *XML*-basierte Beschreibungssprache, um Managementinformation unabhängig von unterschiedlichen Managementtools und deren Informationsmodellen beschreiben zu können. Durch diesen Datenorientierten Ansatz kann Managementrelevante Information bezüglich der

Infrastrukturelemente, der IT-Dienste aber auch der IT-Dienst-basierten Geschäftsprozesse einheitlich erfasst werden und in Beziehung zueinander gestellt werden.

In Abbildung 26 ist beispielhaft der Einsatz der *DCML* als Beschreibungssprache dargestellt, um Managementinformation aus einem Asset-Managementdatensystem zu exportieren und in ein Monitoring-System zu importieren. Die Übersetzung der Quellinformation erfolgt dabei durch einen *exporter*, die Übersetzung der Zielinformation durch einen *importer*, der Austausch zwischen *exporter* und *importer* wird durch *DCML*-Dokumente geregelt.

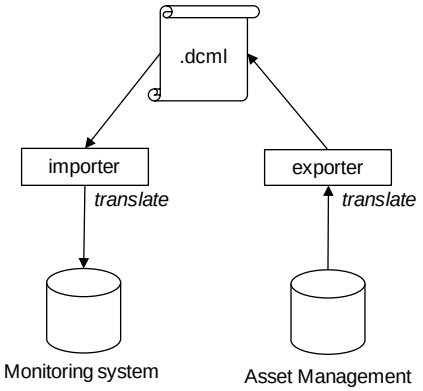

Abbildung 26 Beispielhafter Austausch von Information mittels DCML nach [DCMLa]

Grundlage hierfür ist ein einheitliches Verständnis über die beschriebenen Elemente. Die *DCML* definiert hierfür die Elemente, Schlüsselkonzepte, semantische Regeln, Kodierungsinformation und Verarbeitungsrichtlinien in Form von *XML*-Dokumenten. Durch diesen Ansatz können weiterführende Technologien eingesetzt werden wie beispielsweise *RDF* (*Resource Description Framework* [W3C]) und *OWL* (*Web Ontology Language* [W3Cd]) um aus den beschriebenen Elementen heraus Beziehungen über Datenbestandsgrenzen hinweg zu verarbeiten.

Im Gegensatz zum *CIM* konzentriert sich die *DCML* nicht auf die detaillierte Beschreibung der eigentlichen *Managed Objects* innerhalb einer Infrastruktur, sondern auf die Beschreibung der *Ansätze*, wie zwischen diesen Elementen über unterschiedliche Managementanwendungen hinweg Beziehungen definiert werden können, um eine einheitliche Sicht auf die eingesetzten Managementsysteme und hierdurch letztlich eine einheitliche Sicht auf die Infrastruktur und deren bereitgestellte Dienste zu bekommen.

Daher entsteht die Möglichkeit, nicht nur den Zustand einer Infrastruktur zu beschreiben, sondern vielmehr auch Regeln angeben zu können, wie diese Infrastruktur planmäßig rekonstruiert werden kann ("*Blueprints*"). In der Tat ist einer der Hauptanforderungen hinter der Entwicklung der *DCML* nicht nur der einheitliche Austausch von Managementinformation gewesen, sondern auch die Automatisierung in der Planung und Implementierung von wiederkehrenden Rechenzentrendesigns zu vereinfachen.

Einsatzmöglichkeit

Im Rahmen der Fragestellung, wie Beziehungen zwischen IT-Ressourcen im Kontext eines IT-Dienstes erfasst, beschrieben und verarbeitet werden können spielt die *DCML* im Bezug auf die im Kapitel 5 vorgestellte Referenzarchitektur eine Rolle. Um IT-Dienste mit festgelegten Qualitätsniveaus anbieten zu können, muss der Aufbau der produzierenden Infrastruktur bekannt sein, um Beziehungen zwischen IT-Ressourcen untereinander und in Bezug auf den IT-Dienst in das Management der Infrastruktur integrieren zu können. Die Daten von bestehenden unterschiedlichen spezialisierten Managementtools sollen dabei die Grundlage für einen

integrierten Ansatz bilden, in dem sowohl Wissen aus dem Entwurf und der Implementierung der Dienste wie auch Laufzeitinformation zur Verifikation der Planung anfällt. Die *DCML* kann hierbei den Prozess der Integration vorhandenen Wissens aus den Datenbeständen unterschiedlicher Tools unterstützen.

Aufbau

Ähnlich den Anforderungen an das *CIM* wird vom Konzept der *DCML* aufgrund deren Zielsetzung gefordert, dahingehend hinreichend abstrakt zu sein, sowohl die Elemente der Infrastruktur wie auch die Regeln, mit denen Beziehungen untereinander hergestellt werden können, zu beschreiben.

Das konzeptionelle Modell wird daher in zwei Schichten eingeteilt [DCMLa]:

- *Core schema*: enthält Information über die zu beschreibenden Elemente (*Managed Objects*); Information, wie Pläne ("*Blueprints*") einer idealisierten Infrastruktur anzulegen sind; Regeln, mit denen Veränderungen in der Umgebung beschrieben werden können.
- *Extension schema*: Hier wird domain-spezifisches Wissen wie beispielsweise Konfigurationsparameter spezieller *Managed Objects* definiert.

Die *core* und *extension schemes* sind in *RDF*/*XML* Dokumenten kodiert [DCMLa]. Dazu definiert das *DCML Meta-Model* die Syntax und die Regeln, nach welche gültige *DCML* Schemas aufgebaut sind. Unter anderem wird im *DCML Meta-Model* festgelegt, wie ein *DCML* Dokument strukturiert ist, welche gewöhnlichen Elemente in allen *DCML* Dokumenten auftauchen wie die Modellelemente *Class* und *Property* definiert sind und wie DCML-Modellelemente grafisch repräsentiert werden.

Im *core schema* werden folgende für die *extension schemes* grundlegende Elemente in *XML* spezifiziert [DCMLa]:

- *Header Class*: Erweiterung der *OWL Ontology* Klasse. Hiermit kann Information bezüglich des Schemadokuments abgelegt werden.
- *Entity Class*: Eine *Entity* stellt den Basistyp für die in den Extension Schemas beschrieben Elemente dar.
- *NonDCMLEntity Class*: definiert einen Wrapper, um Information, die nicht in *RDF* kodiert ist, einzubinden. Die hierdurch beschriebene Information wird nicht durch DCML-Prozessoren verarbeitet.
- *CIMEntity Class*: definiert ein Modellelement, um CIM-Klassen in *DCML* abbilden zu können.
- *Relationship*: Beziehungen zwischen Entitäten werden in *DCML* durch OWL Properties beschrieben.

Beispiel

Beispiehlhaft wird gezeigt, wie ein *DCML*-Dokument aufgebaut ist, um die Klasse *CIM_OperatingSystem* in *DCML* zu beschreiben. Die Klasse *CIM_OperatingSystem* wird dabei innerhalb des DCML-Dokuments durch die Klasse *LogicalServer* definiert. Neben den eigentlichen Klassenelementen *CIM_OperatingSystem* und *LogicalServe*r muss auch die verbindende Assoziation zum Ausdruck gebracht werden.

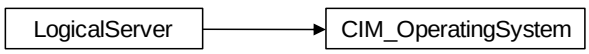

Abbildung 27 Grafische Repräsentation der Modellierung einer CIM-Klasse in DCML

Dabei werden die einzelnen Elemente wie folgt in RDF/XML notiert:

XML-Namespace Deklarationen

```
<! DOCTYPE rdf:RDF [
      <! ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
      <! ENTITY owlrdf "http://www.w3.org/2002/07/owl#">1999/02/22-rdf-syntax-ns#">
      <! ENTITY dcmlrdfs "http://www.dcmlw3.org/ns/dcml/1/core2000/01/rdf-schema#">
      <! ENTITY owl "http://www.w3.org/2002/07/owl#">
      <! ENTITY dcml "http://www.dcml.org/ns/dcml/1/core#">
]>

<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" = "&rdf;"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" = "&rdfs;"
      xmlns:owl="http://www.w3.org/2002/07/ = "&owl#";"
      xmlns:dcml="http://www. = "&dcml.org/ns/dcml/1/core#";"
      xmlns="http://www.dcml.org/ex/dcml/1/exampleSchema#"1/extension#"
      xml:base="http://www.dcml.org/ex/dcml/1/exampleSchema1/exaension">
```

Abbildung 28 RDF/XML Namespace Deklaration für das Beispiel

CIMOperatingSystem
Dieses Element wird als Unterklasse der Elemente *Environment* und *CIMEntity* definiert. Außerdem wird die benötige Assoziation definiert und mit der Kardinalität 1 versehen.

```
<owl:Class rdf: ID="CIMOperatingSystem">
      <rdfs:comment>
            An Operating System resource defined by CIM.
      </rdfs:comment>
      <rdfs:subClassOf rdf:resource="&dcml;Environment"/>>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#os"/>
                  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
                        1
                  </owl:cardinality>
            </owl:Restriction>
      </rdfs:subClassOf rdf:resource="&dcml;CIMEntity"/>>
</owl:Class>
```

Abbildung 29 RDF/XML Notation der Klasse CIMOperatingSystem

LogicalServer
Der *LogicalServer* ist eine Erweiterung des Elements *Resource*, außerdem wird die Assoziation mit der Kardinalität 1 definiert.

```
<owl:Class rdf:ID="LogicalServer">
      <rdfs:comment>
            A DCML logical server.
      </rdfs:comment>
      <rdfs:subClassOf rdf:resource="&dcml;Resource"/>
      <rdfs:subClassOf>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#os"/>
                  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
                        1
                  </owl:cardinality>
            </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

Abbildung 30 RDF/XML Notation der Klasse LogicalServer

Bewertung

Die *DCML* zielt auf den Austausch von Informationen zwischen unterschiedlichen Managementsystemen. Durch die Wahl, *RDF*/*XML*-basierte Beschreibungs- und Verarbeitungsverfahren einzusetzen, kann einfach ein Bezug zwischen unterschiedlichen *Managed Elements* aus unterschiedlichen Managementsystemen hergestellt werden. Durch die Verankerung der *CIMEntity Class* als Modellierungselement im *core schema* kann einfach ein Mapping zu CIM-basierten Managementsystemen hergestellt werden. Die *DCML* kann beispielsweise als Sprache eingesetzt werden, um den Zugriff auf unterschiedliche

Datenbestände im Hinblick auf einen integrierten Managementansatz zu regeln (siehe auch 6.1 hierzu).

Aus diesen Gründen wird im Rahmen dieser Arbeit zunächst ein Ansatz entworfen, um die technischen Modellierungselemente aus dem *CIM* heraus durch *RDF* zu beschreiben, um weitergehende strukturierende Ansätze, wie beispielsweise den Einsatz der *DCML*, zu unterstützen.

3.2.2 CIM Modellaustausch

Neben der *DCML* existieren weitere Ansätze, um das Wissen aus dem technischen Management über unterschiedliche Managementbereiche hinweg auszutauschen. In [VWZ01] wird beispielsweise ein Ansatz diskutiert, die *CIM* Modelle in *RDF* zu transformieren, um hierdurch eine Wissensbasis für einen Austausch von Managementinformation unabhängig vom technischen Bezug (Semantik) der Information zu erhalten. Das *CIM* ist in diesem Kontext ebenfalls mit dem Synonym *Common Information Modell* verbunden, bezeichnet jedoch einen von der Energieversorgerbranche spezifizierten Ansatz, die technischen Elemente im Bereich der Energieversorgung strukturiert zu modellieren in dem Sinne, wie auch die *DMTF* das *CIM* für den Bereich der IT-Infrastrukturen entwickelt hat.

Unterschiedliche Ansätze sind hierbei denkbar:

(1) Transformation auf Meta-Modellebene
Hierbei werden die Konstrukte des Meta-Modells betrachtet und eine Abbildung zwischen Meta-Modellelementen erreicht. Dies ist Vergleichbar mit dem *recast-mapping* (siehe auch Kapitel 4.4.2. Der Vorteil ist, das mit der Umsetzung des Meta-Modells eine Automatisierung bei der Umsetzung der von dem Meta-Modell abgeleiteten Instanzen erfolgen kann.

(2) Transformation auf Instanz-Ebene
Hierbei wird eine Abbildung von Instanzen von Modellelementen des technischen Modells in Modellelemente des strukturierenden Ansatzes vorgenommen. Die Abbildung wird dabei für jede einzelne Instanz durchgeführt, wobei durch dieses Vorgehen die Bedeutung von Information gewahrt bleiben kann. Dieser Ansatz ist jedoch auf spezifische Modellelemente beschränkt, eine Automatisierung kann nicht vorgenommen werden. [VVB02].

Im Rahmen der Transformation der technischen Managementinformation in einen strukturierenden Ansatz, der von der Semantik der technischen Sicht frei ist, wird in Kapitel 5.2.3 eine Transformation auf Instanz-Ebene angestrebt, weshalb ein *RDF Schema* spezifiziert wird, um syntaktisch und semantisch korrekte Transformationen der *CIM*-Modellelemente des generischen Ansatzes durchführen zu können.

4 ENTWURF EINES KONZEPTS ZUR MODELLIERUNG VON RESSOURCENABHÄNGIGKEITEN

In diesem Kapitel wird der Begriff „Abhängigkeit“ in Bezug auf Laufzeitumgebungen untersucht. Hierzu wird zunächst ein generisches Infrastrukturmodell betrachtet, um anschließend im zugehörigen Meta-Modell den Begriff Abhängigkeit anhand des Laufzeitverhaltens von Infrastruktur-Komponenten zu beschreiben. Somit wird der Begriff Abhängigkeit zwar innerhalb einer bestimmten Problemdomäne (Laufzeitumgebungen) untersucht, jedoch ist der hier vorgestellte Ansatz generisch und kann demnach auch auf andere Szenarien, beispielsweise die Betrachtung von Fehlerfortpflanzungen wie in [DLS05], übertragen werden. Der generische Ansatz erlaubt es weiterhin, neben den Beziehungen der funktionalen und strukturellen Modelle auch die operationelle Sicht zu integrieren, um demnach eine umfassendere Sicht auf IT-Infrastrukturen zu erhalten.

Bei dem hier vorgestellten Konzept handelt sich um Überlegungen bezüglich notwendigen Modellierungselementen in einem Informationsmodell. Zunächst wird untersucht, welche Modellierungselemente grundlegend notwendig sind, um einen Laufzeitaspekt bei der Modellierung einer Infrastruktur zu integrieren.

Es sind zwei Vorgehensweisen denkbar:

(1) *Top-down Ansatz*: bekannte, strukturelle und funktionale Zusammenhänge, die vorhersehbare dynamische Abhängigkeiten nach sich ziehen, können modelliert werden.

(2) *Bottom-up Ansatz*: zunächst unbekannte Zusammenhänge zwischen Infrastruktur-Komponenten, die sich erst aufgrund des Laufzeitverhaltens während der Produktion eines IT-Dienstes ergeben, sollen identifiziert und modelliert werden können. Der generische Ansatz bezieht sich hierbei auf die Datensicht und beschreibt noch keine konkreten Algorithmen, um aus Messwerten Abhängigkeiten korrelieren zu können. Durch den *Bottom-up* Ansatz lassen sich bestehende Modelle verfeinern bzw. verifizieren und der Realität der gegeben Infrastruktur in Bezug zu den angebotenen IT-Diensten anpassen.

Der hier vorgestellte generische Ansatz wird weiterhin in ein bestehendes Informationsmodell integriert, damit die Informationen für bestehende Managementapplikationen nutzbar werden. Diese Integration erfolgt im Rahmen der Formulierung eines einfachen *CIM-Extension Schemas*, um somit beispielsweise für eine *WBEM*-basierte Managementanwendung zur Verfügung zu stehen.

4.1 Motivation und Vorgehen

Das Verhalten eines Dienstes manifestiert sich zum Zeitpunkt des Dienstaufrufes in einer wahrnehmbaren Veränderung der Zustände der beteiligten Ressourcen und damit auch eine Veränderung der Beziehungen der beteiligten Ressourcen untereinander. Dies ist intuitiv verständlich, da ein IT-Dienst von den IT-Ressourcen einer Infrastruktur produziert wird. Der Zustand einer Ressource ist hierbei die Gesamtsumme der von außen messbaren Parameter in Form von *Attribut-Attributwert*-Paaren. Während funktionale und strukturelle Modelle die aus dem Entwurf und der Implementierung einer Infrastruktur bekannten *Relationen* beschreiben, wird der Begriff *Abhängigkeit* in den Kontext von Laufzeitumgebungen gestellt und damit mit der Dynamik innerhalb einer Infrastruktur verknüpft. Dies soll im Folgenden nun detailierter erläutert werden.

Zur Verdeutlichung wird zunächst folgendes vereinfachtes Szenario (Abbildung 31) betrachtet (siehe auch hierzu 1.5).

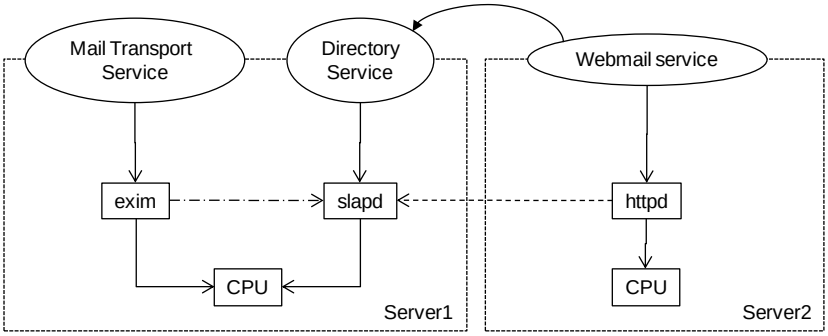

Abbildung 31 Vereinfachtes Szenario

Dargestellt sind für zwei Serversysteme *Server1* und *Server2* die Dienstzugangspunkte *Mail Transport Service*, *Directory Service* und *Webmail Service*, sowie die Systemprozesse *exim*, *slapd* und *httpd*. Als einzige physikalische Ressource wird die *CPU* betrachtet. Ein Benutzer kommt nur mit dem *Webmail* S*ervice* bzw. dem *Mail Transport Service* direkt in Kontakt.

Die Nutzung der Webmailschnittstelle erfordert eine Authentifizierung eines Benutzers am Verzeichnisdienst. Dies geschieht beispielsweise bei der Anmeldung, aber auch beim Senden oder Betrachten von Mails. Die daraus resultierende Kommunikationsbeziehung zwischen *httpd* und *slapd* zieht neben der steigenden CPU-Last auf *Server2* auch eine steigende CPU-Last auf *Server1* nach sich, weshalb dem ebenfalls auf *Server1* lokalisierten Systemprozess *exim* für dessen Produktionsanteil am *Mail Transport Service* weniger Rechenzeit zur Verfügung steht.

Weiterhin benötigt der *exim* für die Verifikation von zustell baren E-Mailadressen ebenso den *slapd*, was sich wiederum in sinkenden Bearbeitungsraten der Anforderungen der Webmailschnittstelle wiederspiegelt. Abbildung 32 gibt eine Darstellung des Zusammenhangs einer Messung über einen Zeitraum von 5 Stunden der CPU-Last und der Mail-Warteschlangen wieder.

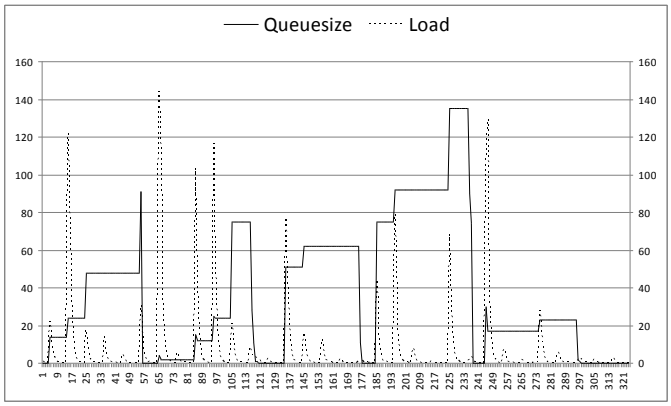

Abbildung 32 Auslastungen von CPU-Last und Mail-Warteschlangen

Dargestellt ist die Auslastung der CPU (*Load*) durch die gestrichelte Linie, sowie die Größe der Mailwarteschlange (Queuesize) durch die durchgezogene Linie. Erkennbar ist ein Anwachsen der Warteschlangenlänge in Korrelation zu steigender CPU Last, die periodisch durch eine rechenleistungsintensive Anforderung gestellt wurde. Die Auslastung der IT-Ressource Queuesize steht also offenbar in Wechselwirkung zur Last auf der beteiligten CPU. Dieser triviale Zusammenhang soll hier nicht weiter vertieft werden, verdeutlich aber die Beobachtung, das Wechselwirkungen zwischen IT-Ressourcen erst zur Laufzeit quantifiziert werden können, weshalb Ansätze für einen ganzheitlichen Blick auf die Zusammenhänge innerhalb einer Infrastruktur neben den bekannten strukturellen und funktionalen auch die operationellen Aspekte integrieren muss. Da sich aber gerade im operationellen Aspekt die Auswirkung einer Wechselwirkung zwischen IT-Ressourcen bemerkbar macht, nämlich wenn Aufgrund von

Überlastsituationen Verletzungen von *Service Level Agreements* wahrscheinlich werden, wird der Begriff Abhängigkeit in den Kontext von Laufzeitumgebungen, und damit in Bezug zu den messbaren Zuständen, von IT-Ressourcen gestellt. Diese messbaren Zustände spiegeln sich wie bereits festgestellt, in Attribut-Attributwert-Paaren wieder.

Definition Abhängigkeit

Für die weitere Arbeit ist daher folgende Definition des Begriffes Abhängigkeit gültig:

> Abhängigkeiten zwischen IT-Ressourcen in Bezug auf die Produktion eines IT-Dienstes sind Abhängigkeiten zwischen Attributwerten der entsprechenden Attribute der betrachteten IT-Ressourcen.

Eingrenzung

Die Überlegungen in diesem Kapitel sind wie bereits erwähnt im Kontext eines Informationsmodells angesiedelt. Wie genau mathematisch-funktionale Abhängigkeiten zwischen Attributwerten (beispielsweise mit Hilfe von Korrelationsverfahren) festgestellt werden können, sind nicht Gegenstand der vorliegenden Arbeit. Zwar wird im Rahmen einer Managementreferenzarchitektur in Kapitel 6 eine Korrelationskomponente beschrieben und prototypisch umgesetzt, jedoch ist die Umsetzung einfach gehalten und soll nur die Tragfähigkeit des Ansatzes demonstrieren.

Motivation

Im CIM existiert zwar die Möglichkeit, Assoziationen und Abhängigkeiten zu modellieren, jedoch wird im Metaschema festgelegt, das Assoziationen zwischen Klassen, nicht jedoch zwischen den Elementen der Klassen – den Attributen - bestehen. Diese Modellierung ist jedoch zu grobgranular, will man genau angeben, von welcher Art und Ausprägung eine Abhängigkeit zwischen Komponenten der Infrastruktur besteht (Parametrisierung der Abhängigkeiten). Der Zustand einer Komponente wird durch die Definition ihrer Eigenschaften (Attribute und deren Werte) beschrieben, weshalb auch die Betrachtung einer Beziehung zwischen Komponenten an der Betrachtung der Beziehung der entsprechenden Attribute ausgerichtet werden sollte. Schließlich spiegelt sich die Produktion eines IT-Dienstes innerhalb der messbaren Zustandsänderung von den produzierenden IT-Ressourcen wieder.

Weiterhin bekannte Ansätze geben keine zufriedenstellende Definition des Begriffes Abhängigkeit wieder, und sind eher auf die Integration in ein Informationsmodell bei bekannter Semantik ausgelegt:.

(1) In der UML [OMG06] wird der Begriff Abhängigkeit (*Dependency*) über die Änderung der *Definition* eines Elements (*antecedent*, *server*) und der damit verbundenen Änderung der *Definition* eines abhängigen Elements (*dependent*, *client*) aufgespannt. Eine Abhängigkeit definiert somit eine (statische) Beziehung, die zur *Modellierungszeit* zwischen zwei Elementen ausgemacht werden kann. [CSCI]

(2) [HSM+06] greift den Ansatz auf, die Assoziationsklassen des CIM durch Einführung einer hierarchischen Struktur zu untergliedern, die angesprochenen Schwachstellen können so jedoch nicht behoben werden. Weiterhin wird die Abhängigkeit zwischen Komponenten und dadurch als Assoziation zwischen Klassenelement definiert. Jedoch zeigt sich zur Laufzeit, das über die Änderung von Attributwerten eine Abhängigkeit zwischen Attributen von Element besteht. Im CIM ist es nicht möglich, diesen Sachverhalt zu modellieren.

(3) Ansätze, über *Ontologien* Zusammenhänge, und damit auch Abhängigkeiten ausmachen zu können, bleiben auf die statische Beschreibung (zur Entwurfszeit) beschränkt, da ein dynamischer Aspekt schwer zu integrieren ist Jedoch kann über eine statische Analyse ein Hinweis auf eine mögliche Abhängigkeit erkannt werden, die sich zur Laufzeit ausbildet. Weiterhin kann über diese Technik der Service-Kontext einer IT-Ressource

ermittelt werden. So wird im folgenden Kapitel 5 untersucht, die mittels RDF eine strukturierende Beschreibung der technischen Managementinformation vorgenommen werden kann.

Bestehende Ansätze, Abhängigkeiten zu analysieren, sind aus folgenden Gründen ungeeignet, damit sowohl strukturelle und funktionale wie auch die angesprochenen operationellen Aspekte zu modellieren:

(1) In [DLS05] werden beispielsweise Abhängigkeiten über die Menge der Fehlerzustände einer Teilkomponente eines Echtzeitsteuerungssystems definiert.
(2) in [CDS01] werden Abhängigkeiten über die Auswirkung von Zustandsänderungen einer Ressource im Vergleich zu einer weiteren Ressource erfasst. Jedoch bezieht die Formalisierung des Begriffes Abhängigkeit die beteiligten Attribute und deren Werte nicht unmittelbar in die Beschreibung ein.
(3) In [LHY06] wird unter Zuhilfenahme von Metadaten (Komponentenabhängigkeiten aus dem Quellcode einer komponentenbasierten Anwendung) eine Abhängigkeitsmatrix erstellt. Dieser Ansatz ist jedoch auf die Problemdomäne der Softwareentwicklung ausgelegt und kann für bestehende Anwendungen, die nicht im Quellcode vorliegen, schwer adaptiert werden.
(4) In [KBK01] findet eine Charakterisierung von Abhängigkeiten anhand deren Eigenschaften statt, jedoch ist nicht ersichtlich, woraus sich die definierten Attribute motivieren.

Wie in [CDS01] angedeutet, fehlt bisher ein formales Verständnis für den Begriff Abhängigkeit: „ … *What is needed is a formal characterization oft the concept of dependency along a more formal and unified approach to dependency analysis* …“.
In [KBK00], [CDS01] wie auch in [HSM+06] wird versucht, Abhängigkeiten anhand deren Eigenschaften zu klassifizieren, eine formale Entwicklung dieses Begriffes fehlt jedoch. Alle drei Publikationen kommen schließlich zu unterschiedlichen Ergebnissen in der Art, wie Abhängigkeiten zu beschreiben und interpretieren sind, jedoch wird in keiner bekannten Publikation ein direkter Bezug zum dynamischen Aspekt eingegangen. Der Ansatz in [CDS01] über konzeptionelle Graphen zunächst losgelöst von konkreten technischen Modellen Abhängigkeiten zu beschreiben, ist dabei generischer als die weiterhin genannten Arbeiten und ist daher die Grundlage für das im Folgenden nun vorgestellte Modell

Vorgehen

In Abbildung 33 ist das Vorgehen in diesem Kapitel verdeutlicht. Um eine *reale* Infrastruktur modellieren zu können, muss ein geeignetes Infrastruktur*modell* definiert werden. Durch das *Metamodell* werden die Form der Elemente und die Regeln für den syntaktisch-korrekten Aufbau des Infrastrukturmodells bestimmt, indem dort die Elemente formal beschrieben werden.

Bei der Formulierung des Infrastrukturmodells wird hier von bestehenden Ansätzen wie beispielsweise dem CIM zunächst Abstand genommen, da durch die Fokussierung auf die wesentlichen Elemente des Infrastrukturmodells der Begriff der Abhängigkeit einfacher untersucht werden kann, und sich letztlich der dynamische Aspekt, der durch die operationelle Sicht gebildet wird, besser erfassen lässt. Anschließend wird im Metamodell der Begriff der Relation an den statischen und der Begriff der Abhängigkeit an den dynamischen Zusammenhängen festgemacht.

Das vorgestellte Metamodell sowie das Infrastrukturmodell bilden daher zusammengenommen einen generischen Ansatz zur Modellierung auf Basis der Definition von dynamischer Abhängigkeit, mit dessen Hilfe die Relationen und Abhängigkeiten innerhalb der funktionalen, strukturellen und operationellen Sichten generisch beschrieben werden können Später soll dieser generische Ansatz dann in ein bestehendes Informationsmodell integriert werden.

Abbildung 33 verdeutlicht den Bezug des Metamodells zum Infrastrukturmodell, um im Rahmen der Erfassung der Dynamik einer Infrastruktur vom Monitoring zum Management zu gelangen.

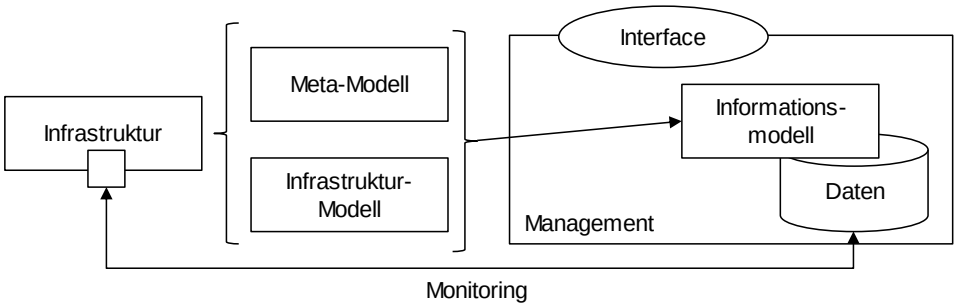

Abbildung 33 Motivation und Vorgehen zur Definition des generischen Ansatzes

Zunächst wird das generische Infrastrukturmodell definiert, um hierdurch eine einfache Argumentationsbasis für das darauffolgende Meta-Modell zu bekommen. In letzterem wird die Form der Elemente des Infrastrukturmodells, und somit über das Verhalten des Infrastrukturmodells eine Erklärung für die Auffassung des Begriffes der Abhängigkeit geliefert.

Das hierdurch gewonnene Verständnis für den dynamischen Aspekt von Abhängigkeiten wird im weiteren Verlauf dieses Kapitels innerhalb eines Erweiterungsschemas für das *CIM* eingebracht, welches in Kapitel 6 im Rahmen eine *WBEM*-basierten Management-Referenzarchitektur integriert wird.

4.2 Generisches Infrastrukturmodell

Aus Sicht des Managements kann eine IT-Infrastruktur durch die Ebenen Netzwerk-, System-, Systemprozess- (Anwendungs-) und Serviceebene beschrieben werden (siehe 2.1.1). Dabei ist der IT-Dienst das immaterielle Ergebnis des Produktionsprozesses, der durch die beteiligten IT-Infrastrukturressourcen vollführt wird. Demnach spiegelt sich im Ergebnis der Serviceproduktion die (vorhergehende) Dynamik der zugrunde liegenden IT-Infrastruktur wieder.

IT-Service, IT-Ressource

Das Modell einer IT-Infrastruktur kann grob in die beiden Infrastrukturelemente *IT-Service* und *IT-Ressource* geteilt werden. Eine IT-Ressource steht stellvertretend für die zu modellierenden generischen IT-Infrastrukturkomponenten und kann weiterhin durch eine Spezialisierung durch die Begriffe *virtuell* und *physikalisch* verfeinert werden. IT-Ressourcen werden im Modell durch ihre *Attribute* und *Attributwerte* beschrieben. Ein IT-Service (im Modell) entspricht dem technischen Aspekt des immateriellen Produkt, das durch die IT-Ressourcen erbracht wird. Diesem Modellelement können ebenso wie den IT-Ressourcen Attribute und Attributwerte zugeordnet werden und beschreiben die Charakteristika eines IT-Dienstes.

Relation

Durch die Produktion eines IT-Dienstes treten die beteiligten Infrastrukturkomponenten in Interaktion zueinander, um gemeinsam das Dienstleistungsprodukt zu erstellen. Dabei kommen in der Regel eine Reihe von Wechselwirkungen zu Tage, die teilweise schon zur Entwurfszeit bekannt sind (funktionale/strukturelle Sicht), teilweise aber erst zur Laufzeit konkret parametrisiert werden können (operationelle Aspekte).

Die statischen Beziehungen werden im generischen Infrastrukturmodell durch den Begriff *Relation* modelliert. Dahinter verbergen sich beispielsweise Beziehungen zwischen Infrastrukturkomponenten, die sich bereits aus dem Softwareentwurf ergeben (funktionale Zusammenhänge zwischen Softwaremodulen) oder durch die Installation von Softwarekomponenten aus den Softwarerepositories (strukturelle Zusammenhänge zwischen Softwarekomponenten).

Relationen zeichnen sich dadurch aus, das sie bereits zur Entwurfszeit bekannt sind jedoch keinen unmittelbaren zwingenden dynamischen Aspekt mit sich bringen. So bestehen bei vielen Softwarekomponenten bestimmte Installationsbedingungen, die die Existenz weiterer Software-Komponenten regelt. Diese Bedingungen sind jedoch unabhängig davon, ob eine Software auch zur Ausführung gebracht wird oder nicht. Somit sind diese Bedingungen auch unabhängig von den konkreten Instanzen, die sich zur Laufzeit aus dem strukturellen Modell ergeben.

Abhängigkeit (Dependency)

Die Wechselwirkungen, die sich zur Laufzeit ergeben, unterscheiden sich demnach von den Relationen, die bereits zur Entwurfszeit bekannt sind. Diese dynamischen Zusammenhänge beziehen sich auf konkrete Instanzen der Modellierungselemente und bestehen immer dann, wenn auch eine Relation besteht. Sie beziehen sich demnach auf die Dynamik innerhalb der Infrastruktur, welche letztlich durch die messbaren Zustände der Infrastrukturkomponenten und deshalb über die Attributwerte im Infrastrukturmodell beschrieben werden.

In Anlehnung an den mathematischen Begriff der Abhängigkeit, der zwei Werte über eine mathematische Funktion in Zusammenhang stellt, wird diese dynamische Wechselwirkung zwischen den Komponenten einer Infrastruktur ebenfalls mit dem Begriff Abhängigkeit beschrieben. Schwächer noch als die mathematische (Funktionale-) Abhängigkeit kann eine Wechselwirkung zwischen Attributwerten durch den Begriff der Korrelation beschrieben werden.

Durch einen Klassen- / Instanz orientierten Ansatz in der Modellierung wird einem Attribut ein konkreter Wert zugeordnet, um die Instanzen einer Klasse zu beschreiben. Hierdurch kann jedoch auf der Ebene des Infrastrukturmodells nicht entschieden werden, zwischen welchen Attributen eine Relation, bzw. zwischen welchen Werten eine Abhängigkeit besteht.

Folgende Abbildung 34 zeigt das bisher vorgestellte generische Infrastrukturmodell.

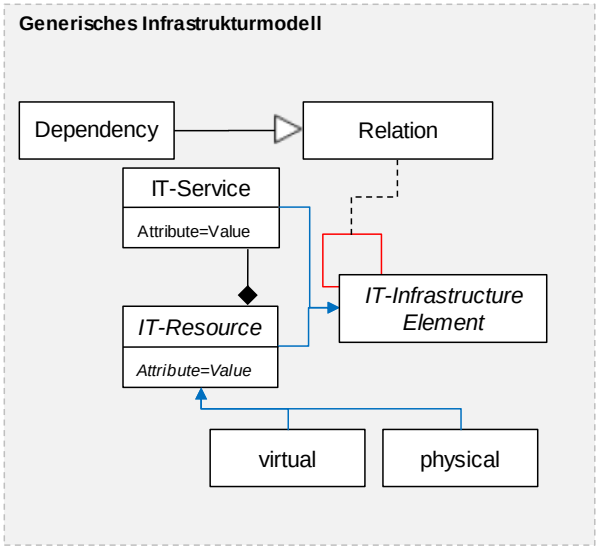

Abbildung 34 generisches Infrastrukturmodell

Mit dem vorgestellten generischen Infrastrukturmodell können beliebige IT-Infrastrukturen und deren Elemente wie IT-Ressourcen und IT-Dienste modelliert werden. Desweiteren kann dieser einfache generische Ansatz in den Kontext auf bestehende Arbeiten wie beispielsweise dem *CIM* gestellt werden.

4.3 Metamodell zum generischen Infrastrukturmodell

Während im Infrastrukturmodell die Elemente der zu modellierenden Realität (der Infrastruktur) beschrieben werden, wird im Meta-Modell zum Infrastrukturmodell die Form der Infrastrukturmodellelemente, deren Verhalten und Beziehungen untereinander beschrieben. Im Meta-Modell wird demnach ein formales Verständnis für die beschriebenen Begriffe Relation und Abhängigkeit geschaffen. Weiterhin legt es die Regeln fest, nach denen ein der Realität entsprechendes gültiges Infrastrukturmodell spezifiziert werden kann.

Grundlegend wird hier zwischen der statischen Sicht (Entwurfszeit) und der dynamischen Sicht (Laufzeit) unterschieden. Zur Entwurfszeit zählen die in [KK01] definierte funktionale und strukturelle Sichten, zur Laufzeit zählt die operationelle Sicht. Im Meta-Modell wird durch die Trennung der Beschreibung der unterschiedlichen Sichten ein generischer Beschreibungsansatz geschaffen, mit dem die unterschiedlichen Aspekte auf eine einheitliche Weise beschrieben werden können. Hierzu werden die Elemente Entität, Entitätstyp, Entitätsinstanzen, Attribut, Attributwert, Relation und Abhängigkeit als modellierende Elemente eingeführt.

Grundlegendes Konzept hierbei ist die Trennung der Beschreibung eines Gegenstandes (Entität) von dessen Eigenschaft (Attribut) und der aktuellen Ausprägung dieses Attributs (Attributwert). Dieser Ansatz kann beispielsweise auch in der Definition des grafischen Datenmodelles des *RDF* wiedergefunden werden. Dort werden die Beschreibungen der *RDF-Properties* von den eigentlichen *RDF*-Klassen getrennt beschrieben. Abbildung 35 verdeutlicht in Analogie zu 2.3.2 die Beziehung zwischen *Entität*, *Attribut* und *Attributwert*.

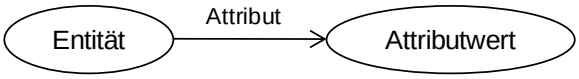

Abbildung 35 Konzept Entität, Attribut und Attributwert in RDF-Graphennotation

Nachfolgende Abbildung gibt eine Übersicht über den Aufbau des Meta-Modells wieder. Anschließend werden die Modellierungselemente erläutert.

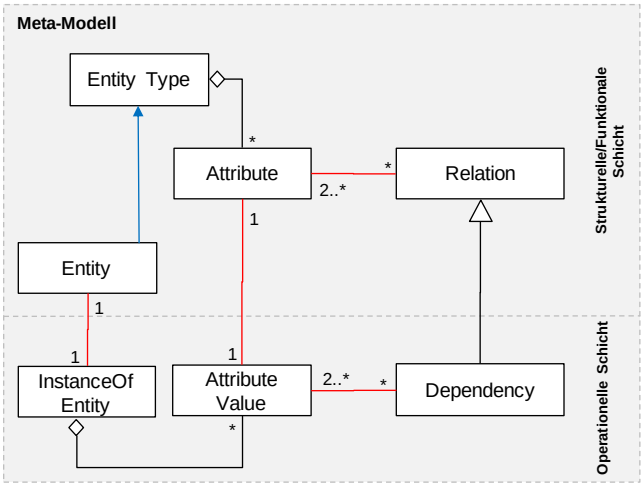

Abbildung 36 Meta-Modell zum Infrastrukturmodell

Entität, Entitätsinstanz

Als Entität wird in der Informatik ein eindeutig bestimmbares Objekt bezeichnet, dem Information zugeordnet werden kann. Information ist in diesem Sinne ein Datum mit einem klaren semantischen Bezug. Objekte können sowohl *materieller* (fassbarer) wie auch *immaterieller* (abstrakter) Natur sein, wodurch die Verwendung des Begriffes Entität eine Abstraktion von der eigentlichen Beschaffenheit des Objektes erreicht. Der Vorteil, den Begriff Entität zu verwenden, liegt hierbei darin begründet, dass keine Mehrdeutigkeiten durch falsche Wortbindungen entstehen können.

Die Objekte, die durch Entitäten beschrieben werden, entsprechen genau jenen IT-Ressourcen, die im Kontext eines IT-Dienstes eine (funktionale) Beteiligung an der Serviceerbringung haben. Eine IT-Ressource ist dabei eine aktive Teilkomponente einer IT-Infrastruktur, die nur beschränkt verfügbar ist (in der Anzahl). Im allgemeinen Fall sind IT-Ressourcen beispielsweise Netzwerkkomponenten, Systemprozesse (Mailserverprozesse, Verzeichnisdienst, DNS, ….) , Stromversorgung oder die physikalischen Komponenten eines Serversystems (CPU, Hauptspeicher, Festplatten, Netzwerkkarten, …).

Objekte der Realität manifestieren sich durch ihre wahrnehmbaren Ausprägungen. Der Satz „Dieser Systemprozess heißt exim“ bezeichnet ein Objekt (Systemprozess) mit der wahrnehmbaren Eigenschaft „Prozessname:= exim“. Zunächst ist es unerheblich, welche wahrnehmbaren Eigenschaften in die Beschreibung eines Objektes aufgenommen werden.

Die Trennung von statischem Aspekt (funktionale und strukturelle Sicht) und dynamischem Aspekt (operationelle Sicht) wird durch Trennung der Definition von Entitäten mit deren Attributen und Relationen sowie Instanzen dieser Entitäten mit deren Attributwerten und Abhängigkeiten erreicht.

Attribut, Attributwert

Eine Entität (im generischen Modell) als Abstraktion eines Objekts (in der Realität) zeichnet sich ebenso durch Merkmale (*Attribute*) aus, denen bestimmte Eigenschaften in Form von *Attributwerten (Attribute Values)* zugesprochen werden können. Wie diese Attribute genau aussehen, ist nicht festgeschrieben, vielmehr entsteht hierdurch die Möglichkeit, Information einzubringen, die zur unmittelbaren Lösung einer Problemstellung dient. Ebenso ist nicht vorgeschrieben, welche Attribute der Realität in die Modellierung eines Objektes aufzunehmen sind. Eine Entität beschreibt demnach immer eine an die Problemlösung angepasste Sicht auf ein Objekt. Weiterhin sei angemerkt, das ein Teil der konkreten Werte von Attributen erst zur Laufzeit ermittelt werden können, einige Werte sind jedoch schon zur Entwurfs-/Installationszeit bekannt.

Beispielsweise ist der Name einer Komponente zur Entwurfszeit bekannt, während der Füllstand eines Puffers sich erst zur Laufzeit ermitteln lässt.

Entitätstyp

Durch die Unterscheidung zwischen Attribut und Attributwert kann eine Klassenbildung erreicht werden. Wird einem Attribut ein konkreter Wert zugewiesen, entsteht aus der abstrakten Beschreibung der Objektstruktur eine konkrete *Instanz* dieses Objektes. Diese Auffassung kann beispielsweise auch in der objektorientierten Softwareentwicklung beobachtet werden, wo durch die Trennung der Begriffe *Klasse* und *Objekt* eine Trennung zwischen struktureller Beschreibung eines Gegenstandes und dessen tatsächlicher Ausprägung erreicht wird.

Die Klassenbildung im Beispielsatz von oben wird durch die Beschreibung des Attributs „Prozessname“ erreicht, die konkrete Instanz entsteht durch die Zuweisung des Wertes „exim“. Innerhalb der formalen Sicht entspricht dem Begriff der Klasse der Begriff Entitätstyp.

Hierdurch können Entitäten gleichen Typs gruppiert werden, um so grundlegende Gemeinsamkeiten von Managementobjekten zu definieren.

Zumeist wird in der gedanklichen Abstraktion eine Entität nicht nur durch ein einzelnes Attribut, sondern durch eine Menge an Attributen beschrieben. Diese Menge wird als *Attributmenge* (*Attribute Set*) bezeichnet, bzw. die Menge der konkreten Werte der einzelnen Attribute als *Wertemenge (Attribute Value Set)*.

Mit den Elementen des Meta-Modells kann nunmehr eine vollständige Sicht auf der Infrastrukturmodell beschrieben werden. Dabei ist das Element *IT-Infrastructure Element* als Instanz des Meta-Modellelements *Entity*, das Element Relation als Instanz des Elements Relation und das Element *Dependeny* als Instanz des Elements *Dependency* aufzufassen. Folgende Abbildung 37 verdeutlicht die Verknüpfungen des generischen Infrastrukturmodells mit den Elementen des zugrunde liegenden Meta-Modells.

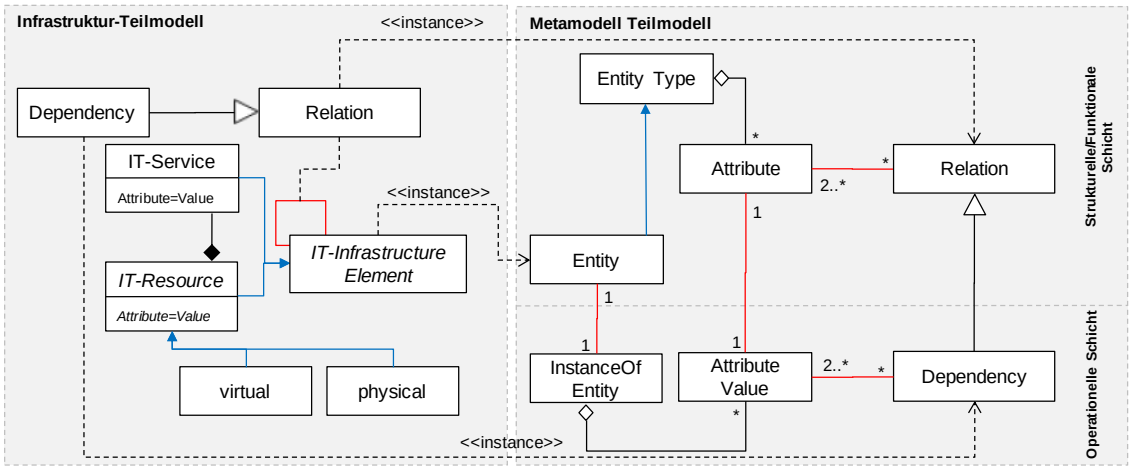

Abbildung 37 vollständiger generischer Ansatz

Statischer und dynamischer Aspekt

Während bisher die zu modellierenden Objekte einer Infrastruktur getrennt voneinander betrachtet wurden, erfolgt im nächsten Schritt eine Verknüpfung über die Attribute der Entitätstypen bzw. über deren Attributwerte.

Dabei kann grundlegend zwischen Beziehungen unterschieden werden, die bereits zur Entwurfszeit feststehen (siehe [Li03], [DLS05]) und Beziehungen, die sich während der Laufzeit eines Systems konkret manifestieren (siehe [LN99],[LN01],[PG+00]). In [KK01] wird hierzu zwischen einem funktionalen Modell (für den statischen Aspekt) und einem operationellem Modell (für den dynamischen Aspekt) unterschieden. [En01] führt hierzu die Begriffe *environmental model* (für das Laufzeitmodell) und *abstract models* (für das Entwurfsmodell) ein.
Es lassen sich zunächst zwei Sichtweisen ableiten:

1. Erfassung der Relationen zwischen den Attributen von Entitätstypen (funktionaler Aspekt) und damit der Relationen zwischen Entitätstypen bzw. Entitäten
2. Erfassung der Abhängigkeiten zwischen den Attributwerten von Entitäten (operationeller Aspekt) und damit der Abhängigkeiten zwischen Entitäten.

Zu (1)
In der statischen Sicht interessiert zunächst, welche Komponenten generell in die Erbringung einer Dienstleistung eingebunden werden müssen. Beziehungen auf dieser Ebene haben Auswirkungen bei Änderungen von Komponenten (*Change Management*) oder der Ermittlung von Auswirkung von Fehlern (*Incident-/Problem Management*). Da diese Beziehungen

schon zur Entwurfszeit (Planung) eines Systems bekannt werden, können beispielsweise folgende bestehende Techniken erfolgreich eingesetzt:

- Abhängigkeitsinformation von Paketmanagern [KBK00]
- Abhängigkeitsinformation aus dem Quelltext [Li03]
- Abhängigkeitsinformation aus XML-Beschreibungen [KBK00]

Zu (2)
Während statische Beziehungen bereits zur Entwurfszeit eines IT-Dienstes bekannt sind, prägen sich die wahrnehmbaren Eigenschaften von Komponenten erst zur Laufzeit eins Systems aus. Hierbei ist entscheidend, das die Anforderungen an eine angebotene IT-Dienstleistung zumeist nicht konstant sind, sondern vielmehr stark schwanken in der Zeit. Die Anforderungen an den IT-Dienst werden letztendlich von den in den IT-Dienst eingebundenen IT-Ressourcen erbracht, weshalb der IT-Dienst in letzter Konsequenz auch erst zur Laufzeit qualitativ erfasst werden kann. Abbildung 38 zeigt die Anforderungen des E-Mail-Teildienstes „E-Mail-Senden“ in einer 24 Stunden Ansicht. Auffällig hierbei sind neben dem flachen tageszeitbedingten Anstieg von Mailaufkommen die starken Peaks, welche sich weder einer Tageszeit noch einem innerhalb der Infrastruktur korrelierendem Ereignis zuordnen lassen.

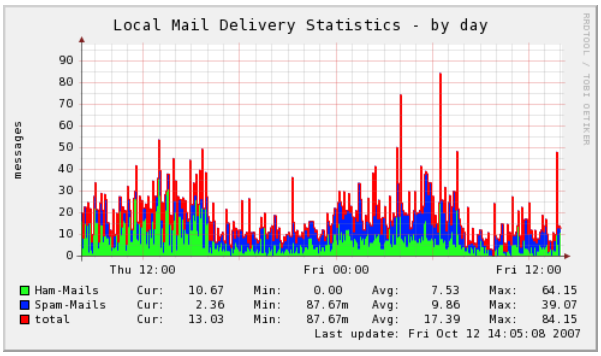

Abbildung 38 Anforderungsverlauf an den Mail-Teildienst "E-Mail-Senden"

Innerhalb des generischen Ansatzes werden die Entitäten durch die mit konkreten Attributwerten besetzten Attribute repräsentiert. Die zugeordneten Attributwerte entsprechen den wahrnehmbaren Eigenschaften der abgebildeten IT-Ressourcen. Ändern sich die wahrnehmbaren Eigenschaften durch Auswirkungen von Aktivitäten von Ressourcen, ändern sich somit auch die entsprechenden Attributwerte. Die Dynamik innerhalb der Infrastruktur wird somit durch eine Dynamik innerhalb des generischen Modells abgebildet.

Mit dem Wissen um dynamische Beziehungen kann bei geeigneter Historie eine Sicht auf die statischen Zusammenhänge generiert werden. Die Betrachtungen im Folgenden konzentrieren sich somit auf die Erfassung der Beziehungen, die sich zur Laufzeit ergeben.

Beispiel: Über die Beobachtung, das der Wert der zugestellten Mails inkrementiert wurde und ein Lastunterschied an der entsprechenden CPU festgestellt wurde, kann eine Abhängigkeit zwischen den Attributwerten, eine Relation zwischen den Attributen und schließlich eine Abhängigkeit der Komponenten innerhalb der Infrastruktur modelliert werden.

Zustand

Der Zustand einer Entität ist die Gesamtheit der messbaren einzelnen Eigenschaften, wobei keine Gewichtung von unterschiedlichen Attributen gemacht wird. Jeder Attributwert trägt somit zum Gesamtzustand gleichermaßen bei. Somit können Entitäten anhand deren Zustands unterschieden werden. Da der Zustand sich aus den messbaren Eigenschaften, den Attributwerten und somit aus zur Laufzeit instanziierten Entitäten ergibt, kann ein Zustand auch nur zur Laufzeit zugeordnet werden.

Eine Änderung des Zustandes bedeutet, dass sich mindestens ein Attributwert der zugrundeliegenden Attributmenge geändert hat. Somit kann eine Abhängigkeit zwischen Entitäten letztlich durch die Beobachtung von Zustandsänderungen festgestellt werden.

Dienstkontext

Eine grundlegende Fragestellung im Rahmen dieser Arbeit ist die Frage nach dem Kontext, in den eine IT-Ressource in einen IT-Service eingebunden ist (Dienstkontext). Während dies auf Basis von funktionalen bzw. strukturellen Modellen einfach durch Verarbeitung der entsprechenden Attribute möglich ist, kann bei Beziehungen, die sich erst zur Laufzeit ergeben, diese Frage nicht mehr *ad hoc* beantwortet werden. Hierzu zählen beispielsweise Störungen einer Softwarekomponente, die sich durch die Ausführung in Bezug auf die zugrunde liegende Hardware ergeben. So kann die Abarbeitung eines Softwaremoduls durch erhöhte CPU-Aktivität festgestellt werden, die CPU kann demnach weniger Rechenzeit in die Abarbeitung weiterer Softwarekomponenten investieren. Durch Analyse von Überwachungsdaten, und Anwendung des generischen Ansatzes können die vorhandenen funktionalen und strukturellen Modelle durch die Beobachtungen zur Laufzeit validiert und ergänzt werden. Der Dienstkontext einer Ressource wird demnach nicht nur über Relationen, sondern auch über dynamische Abhängigkeiten bestimmt.

Erste Anhaltspunkte für Abhängigkeiten ergeben sich demnach aus der Betrachtung von bekannten funktionalen Zusammenhängen. Durch *Monitoring* innerhalb der Infrastruktur und Anwendung des generischen Modells kann im Umkehrschluss das funktionale Modell dahingehend verfeinert werden, das die beobachteten dynamischen Abhängigkeiten durch funktionale Beziehungen zum Ausdruck gebracht werden. So kann letztlich der Dienstkontext einer IT-Ressource wiederum über die Definition von sinnvollen Attributen im generischen Modell erfolgen, wodurch sich der funktionale Kontext einer Ressource zum erbrachten Dienst ausdrücken lässt.

4.4 Konkretisierung des generischen Ansatzes

Um die Allgemeinheit des vorgestellten Ansatzes aufzuzeigen, wird das generische Modell in Beziehung zu verschiedenen ausgewählten Arbeiten gestellt, um somit zum einen den unmittelbaren Mehrwert der abstrakten Betrachtung zu verdeutlichen, zum anderen einen praktischen Bezug herzustellen.

4.4.1 Modellierung von Fehlerzuständen

In [DLS05] werden Abhängigkeiten formal über die Betrachtung von Schnittmengen von Fehlerzuständen beliebiger Komponenten. Hierbei können die Komponenten als Entitäten aufgefasst werden, die Zustände werden durch Attribute ausgedrückt. Hierbei werden zwischen den Zuständen „OK“ und „Fehler“ unterschieden.

Mithilfe des generischen Modells kann dieser Sachverhalt wie in folgender Abbildung ausgedrückt werden.

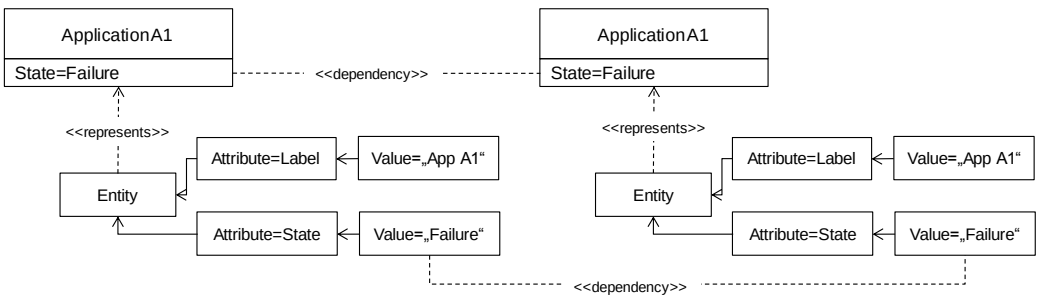

Abbildung 39 Abhängigkeiten über Fehlerzustände nach [DLS05]

Dargestellt sind zwei Applikationen, die durch einen Datenaustausch eine Kommunikationsbeziehung unterhalten. Durch den Fehler in einer Applikation wird ein Fehlerzustand im Kommunikationspartner ausgelöst – der Fehler pflanzt sich demnach fort. Wird der Zustand der Komponente als Attribut definiert und der entsprechende Wert überwacht, kann durch die Zustandsänderung in beiden Komponenten eine Abhängigkeit beobachtet werden. Durch hinzunehmen von zeitlichen Parametern wird außerdem eine Möglichkeit geschaffen, kausale Bedingungen und damit die Richtung des Abhängigkeitspfeiles zu spezifizieren.

4.4.2 Implementierung im CIM

Die Bedeutung des *Common Information Model* wurde bereits in 2.2 beschrieben. Nachteilig wurde festgestellt, das Assoziationen (und damit auch deren Spezialisierung Abhängigkeiten) innerhalb des *CIM* zwischen beliebigen Elementen des Modells festgemacht werden können, ohne die Notwendigkeit für eine Assoziation formal belegen zu müssen. Dies ist hilfreich bei der Erfassung der funktionalen und strukturellen Zusammenhänge innerhalb einer Infrastruktur, jedoch bleibt der Laufzeitaspekt mit den dynamischen Änderungen von Attributwerten hierbei außen vor.

Das Metamodell des generischen Modells beschreibt, wie das Infrastrukturmodell aufgebaut ist. Das Infrastrukturmodell ist dabei bewusst einfach gehalten, da an dieser Stelle mit dem *CIM* ein umfassendes und standardisiertes Informationsmodell zur Verfügung steht. Eine Verknüpfung des generischen Modells und des *CIM* wird dabei über eine Verknüpfung der entsprechenden Metamodelle erreicht. [DMTF05] sieht hierfür mögliche Abbildungen (engl. *Mappings*) zwischen den Metamodellen vor, welche kurz erläutert werden.

Domain Mapping

Beim *domain mapping* werden die Instanzen von Elementen des Quellmodells durch Instanzen von Elementen des Zielmodells zur Grundlage für die Beschreibung des Zielmodells herangenommen [DMTF05]. Hierdurch kann zwar semantische Information gewahrt bleiben jedoch kann die Übersetzung von Quell- in Zielmodell nicht automatisiert geschehen [VVB02].

Recast Mapping

Das *recast mapping* übersetzt die Meta-Modellelemente des Ausgangsmodells in Meta-Modellelemente des Zielmodells. [DMTF05]. Durch diesen Ansatz kann relativ leicht gesehen werden, dass das generische Modell in der Formulierung des Begriffes Abhängigkeit präziser ist als der Ansatz im *CIM*.

Im generischen Modell wird eine Abhängigkeit über Werte von Attributen definiert, während im *CIM* Abhängigkeiten als Spezialisierung von Assoziationen definiert werden. Das *CIM* müsste demnach im Metaschema um eine Präzisierung bezüglich der Definition der Assoziation erweitert werden, wobei Assoziationen nicht zwischen Klassen, sondern zwischen den Attributen von Klassen definiert werden.

Technique Mapping

Beim *technique mapping* werden die Elemente eines Quellmodells als Metamodell für die Elemente eines Zielmodells aufgefasst. Konkret am Beispiel des generischen Ansatzes bedeutet dies, dass das *CIM* Metamodell als Metamodell für die Formulierung des generischen Ansatzes als *CIM* Modell herangezogen wird. Hierdurch kann einfach ein Modell entwickelt werden, das in Bezug zu bestehenden *CIM* Modellen gebracht werden kann.

Bei diesem Ansatz wird das Generische Modell als Verfeinerung von Elementen des *Core Schemas* sowie des *Metrics Models* (siehe 2.2) aufgefasst in dem Sinne, das die Umsetzung

des generischen Ansatzes eigenständige *CIM* Klassen umsetzt, um Relationen und Dependencies im Sinne von Managed Objects zu modellieren.

Die Modellelemente *Entity*, *Attribute*, *Attributevalue*, *Instance of Entity*, *Relation* und *Dependency* werden durch das Metamodellelement Class des *CIM* Metamodell ausgedrückt. Die Klassenattribute, welche die *CIM* Klassen beschreiben (beispielsweise *Caption* für die Benennung einer *CIM*-Klasse) werden durch das Metamodellelement *Property* ausgedürckt. Die Assoziationen im generischen Ansatz sind gemäß dem *CIM* Metaschema durch eigenständige Assoziationsklassen modelliert.

Bei der Instanziierung des Generischen Ansatzes wird das Element Entitätstyp nicht explizit betrachtet, da durch die Generalisierungsbeziehung in *UML* eine Entität auch ein Entitätstyp ist, die Aggregation Attribut – Entitätstyp daher auch direkt an die Entität festgemacht werden kann.

Attribute und Attributwerte werden durch die im *Metrics Model* definierten Klassen `BaseMetricDefinition` (für Attribut) und `BaseMetricValue` (für Attributwert) spezialisiert. Hierdurch können gemäß dem Metamodell des generischen Modells Relationen bzw. Abhängigkeiten an der Klasse `GA_Attribute` bzw. `GA_AttributeValue` modelliert werden.

Eine weitere Verfeinerung des vorgeschlagenen *Extension Schema* bzgl. der im *CIM* definierten Abhängigkeiten stellt die Ableitung der Klasse `GA_Relation` von der abstrakten Wurzelklasse `CIM_ManagedElement` dar. Hierdurch wird dem Konzept der Beziehung (`GA_Relation`) bzw. der Abhängigkeit (`GA_Dependency`) der Status eines *Managed Element* zugeordnet.

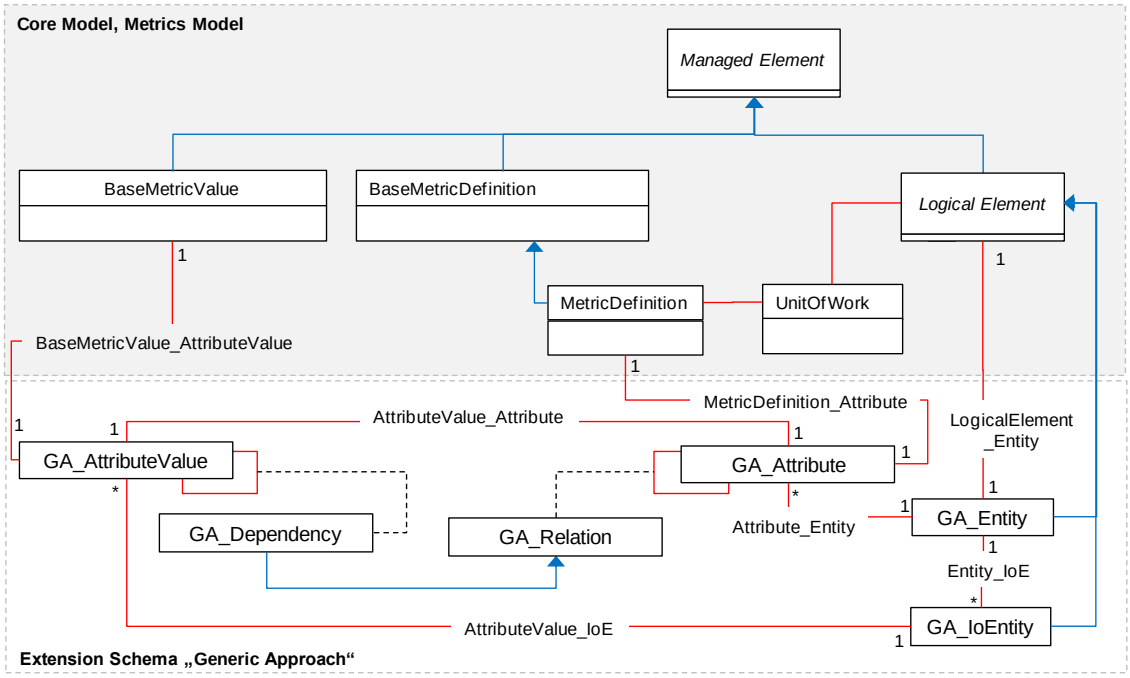

Abbildung 40 Technique Mapping durch Formulierung eines extension schemas

Die Klassenelemente des generischen Schemas werden dabei um folgende Klassenattribute erweitert. *Key=(True|False)* gibt an, ob es sich um ein Schlüsselattribut handelt, *Type=(type)* gibt den Typ des Klassenattributs an. *Name=(name)* gibt den Name des Klassenattributs an.

Tabelle 2 Klassenattribute der CIM Klassen des generischen Ansatzes

GA_Entity	Key=true, Type=String, Name=Identifier
GA_Attribute	Key=true, Type=String, Name=Identifier
GA_AttributeValue	Key=true, Type=String, Name=Identifier Key=false, Type=String, Name=Value
GA_Relation	Key=true, Type=String, Name=Partner1 Key=true, Type=String, Name=Partner2
GA_Dependency	Key=true, Type=String, Name=Antecedent Key=true, Type=String, Name=Dependent
GA_IoEntity	Key=true, Type=String, Name=Identifier

Im Anhang an die vorliegende Arbeit ist die zugehörige *MOF*-Datei (*Managed Object Format*) des Extension Schemas zu finden. Hierdurch kann das Extension Schema einfach in bestehende *CIM*-basierte Managementumgebungen nachgeladen werden. Basis für die Formulierung ist das *CIM* in der aktuellen Version 2.17.

Im Rahmen des in dieser Arbeit entwickelten prototypischen Demonstrators wird dieses Extension Schema eingesetzt, um die dynamischen Beziehungen zwischen IT-Infrastrukturkomponenten zu erfassen und dem Management strukturiert zur Verfügung zu stellen.

4.5 Zusammenfassung

Bestehende Arbeiten unterteilen die Sicht auf eine Infrastruktur in drei unterschiedliche Ebenen mit Beziehungen unterschiedlicher Natur - funktionale, strukturelle und operationelle Zusammenhänge. Während für erstere beiden Sichten Ansätze existieren, um Beziehungen erkennen und verarbeiten zu können, fehlt für den operationellen Aspekt ein Modell, der zum einen die Ressourcen mit deren Eigenschaften in den Vordergrund rückt, zum anderen hinreichend generisch ist, um in bestehende Managementansätze integriert zu werden.

Mit dem in diesem Kapitel vorgestellten generischen Ansatz können alle drei Sichtweisen auf eine Infrastruktur integriert betrachtet werden:

- Für den statischen Aspekt wird der Begriff Relation an den Attributen von Entitäten (IT-Infrastrukturelementen) definiert. Hierdurch entsteht die Möglichkeit, unabhängig von konkreten dynamischen Situationen Zusammenhänge innerhalb einer Infrastruktur zu beschreiben.
- Für den operationellen Aspekt wird der Begriff Abhängigkeit an der Änderung von Ressourcenattributwerten definiert. Hierdurch entsteht die Möglichkeit, die Objekte einer Infrastruktur mit deren Eigenschaften hinreichend genau zu beschreiben (durch Entitäten und Attribute) und die Auswirkung, die die Produktion eines IT-Dienstes auf die zugeordneten IT-Ressourcen hat, zu erfassen und beschreiben zu können.

Dieser generische Ansatz ist somit Teil des Konzepts zur Erfassung der Wechselbeziehungen zwischen IT-Ressourcen, da sowohl statische wie dynamische Aspekte vereint werden und im Rahmen eines Informationsmodells für eine Managementarchitektur eingesetzt werden können. Daher wurde eine Erweiterung auf Basis des *CIM Core Models* und *CIM Metrics Models* definiert, die im Rahmen des prototypischen Demonstrators zum Einsatz kommt.

5 MODELLIERUNG UND ANALYSE VON BEZIEHUNGEN UND ABHÄNGIGKEITEN

Es wird in Anlehnung an das reale Szenario innerhalb der *ATIS* wiederum nachfolgendes vereinfachtes Szenario betrachtet (siehe auch 1.5 und 4.1). Ziel ist es, die bisherigen Ansätze zur Modellierung von IT-Ressourcen und IT-Diensten hinsichtlich der Möglichkeiten der (automatisierten) Erfassung und Beschreibung von Abhängigkeiten, und damit verbunden auch die Frage nach dem Service Kontext einer IT-Ressource, zusammenfassen zu können. Aus diesen Erkenntnissen kann eine Strategie abgeleitet werden, um die gestellten Fragestellungen hinsichtlich des Managements von IT-Ressourcen im Kontext eines IT-Dienstes beantworten zu können.

Darauf aufbauend wird im nachfolgenden Kapitel eine Referenzarchitekur entwickelt, die das Management von Abhängigkeiten im Hinblick auf einen integrierten Managementansatz unterstützt.

5.1 Szenario

Betrachtet wird wiederum der in 1.5 eingeführte Maildienst mit folgendem Aufbau, wie in Abbildung 41 dargestellt.

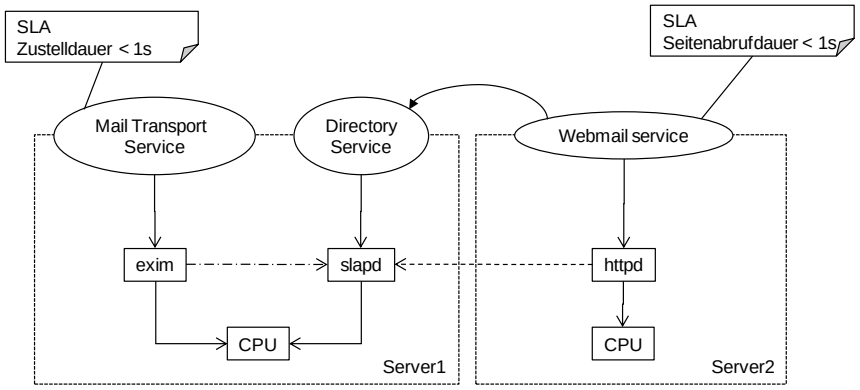

Abbildung 41 vereinfachtes Szenario

Im Rahmen des Maildienstes lassen sich die Funktionalitäten *MailTransport* und *Webmail Zugriff* als eigenständige Teildienste kapseln. Die Schnittstellen zum *Service User* sollen nun mit Dienstleistungsvereinbarungen (*SLA's*) versehen werden, beispielsweise soll der Transport einer E-Mail vom Empfang am Dienstzugangspunkt bis zum Ablegen im Postfach nicht länger als eine Sekunden dauern, ebenso soll die Auslieferungsdauer einer Seite an der Webmail-Benutzerschnittstelle nicht länger als eine Sekunde dauern.

Durch die zusätzliche Betrachtung der qualitativen Leistungsparameter rücken die dynamischen Abhängigkeiten, die sich erst zur Laufzeit konkret manifestieren, in den Vordergrund. Um eine Dienstleistung mit garantiertem Leistungsniveau zusichern zu können, müssen die entsprechenden Kapazitäten innerhalb der Infrastruktur zur Laufzeit bereitstehen.

Konkret lassen sich folgende zu modellierende IT-Ressourcen erfassen:

- Serversystem *Server1*
- Physikalische IT-Ressource *CPU*
- Logische IT-Ressource Systemprozess *exim*
- Dienstzugangspunkt zum IT-Dienst *Mailtransport Service*
- SLA zum *Mailtransport Service*
- Systemprozess *slapd*

- Dienstzugangspunkt zum IT-Dienst *Directory Service*
- Serversystem *Server2*
- Physikalische IT-Ressource *CPU*
- Logische IT-Ressource Systemprozess *httpd*
- SLA zum *Webmail Service*

5.2 Modellierungsansätze für Ressourcen und Abhängigkeiten

Im Folgenden werden nun die beiden Ansätze (*CIM* als Modellierungsansatz für das technische Informationsmodell, *RDF* als darüber hinausgehend strukturierender Beschreibungsansatz) detailierter untersucht.

Die Motivation für den zweigeteilten Ansatz lässt sich wie folgt begründen:

(1) *CIM* als Modellierungsgrundlage für die technische Sicht (mit Integration des statischen und dynamischen Aspekts) ist aufgrund der technischen Ausrichtung der definierten CIM-Elemente geeignet, technische Details der zu verwaltenden Elemente zu beschreiben. Es existieren eine Vielzahl von Tools, die zum einen *CIM* als grundlegendes Informationsmodell einsetzten, zum anderen darauf aufbauen *WBEM*-basierten Zugriff auf Managementinformation anbieten. Beispielsweise existiert mit *WMI* (*Windows Management Instrumentation*) eine Möglichkeit, Microsoft Windows basierte Systeme zu verwalten. Weiterhin wird eine *WBEM*-basierte Managementarchitektur beschrieben, die im Rahmen des prototypischen Demonstrators am Beispiel des vorgestellten Szenarios zur Evaluation des generischen Ansatzes implementiert wird.

(2) Über die technische Sicht hinaus lassen sich Bereiche finden, deren Ressourcen zur qualitativ-gesicherten Produktion von IT-Diensten notwendig sind, die durch technische Aspekte eines technischen Informationsmodells jedoch nicht abgedeckt werden können. Hierzu zählen nicht-technische Parameter wie beispielsweise die Arbeitszeiten von Supportmitarbeiten, aber auch die Sicht eines Dienstnehmers auf einen IT-Dienst. Diese Anforderungen sind im Rahmen des Service Level Managements von Bedeutung; daher muss die gesammelte Managementinformation der technischen Ebene strukturiert präsentiert werden, um so letztlich im Rahmen eines integrierten Managementansatzes mit Information (aus weiteren Systemen) verknüpft werden zu können. Weiterhin entsteht durch die Notwendigkeit, Managementinformation über unterschiedliche Systemgrenzen hinweg austauschen zu können das Bedürfnis, von der Semantik der eigentlichen Information zu abstrahieren und vielmehr den Aufbau der Information an sich zu beschreiben (Loslösung von der technischen Ebene des Informationsmodells).

Gerade letztgenannter Punkt wird bedeutend in der Fragestellung, wie sich die durch die technische Sicht gesammelte Information in einen von der Technik unabhängigen Kontext integrieren lässt.

Abbildung 42 verdeutlicht die Position der Ansätze mit entsprechenden Abstraktionsniveaus.

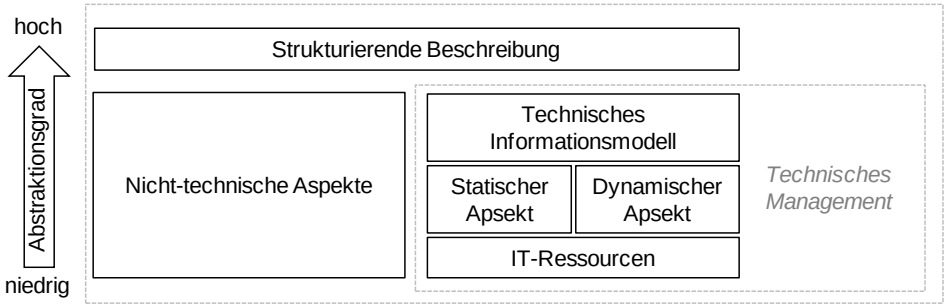

Abbildung 42 Trennung der beiden Ansätze

5.2.1 Vorgehensmodell

Zunächst wird die technische Ebene betrachtet. Hierbei spielen statische und dynamische Aspekte eine wesentliche Rolle. Während die statischen Aspekte bereits aus dem Entwurf bekannt sind und in die Modellierung einfließen können, kann der dynamische Aspekt erst zur eigentlichen Laufzeit, also zur Produktionszeit eines IT-Dienstes, auf den beteiligten IT-Ressourcen der Infrastruktur erkannt werden.

Zur Erfassung der Abhängigkeiten im Rahmen der Fragestellung wird daher folgendes rekursives Vorgehen angesetzt:

(1) Identifizieren der an der Service-Produktion beteiligten IT-Ressourcen
Zunächst wird ein grobes Modell der Infrastrukturkomponenten entworfen. Erste Anhaltspunkte für die notwendigen Modellierungselemente sind die physikalischen Komponenten von Infrastrukturbauteilen (CPU, Festplatte, Netzwerkkarte) und damit verbunden deren wahrnehmbare Eigenschaften wie CPU-Auslastung (*load*), Festplattennutzung (*usage*), Netzwerkkartendurchsatz (*bandwith*). Die Infrastrukturkomponenten werden durch die entsprechenden Klassenlemente des *CIM* und durch *GA_Entity* modlliert, die zugehörigen Eigenschaften durch *GA_Attribute*. Dieser Schritt ist notwendig, da das Managementsystem Initial mit den notwenigen Elementen gefüllt werden muss.

(2) Festlegen der statischen Beziehungen
Die aus dem Entwurf und der Implementierung vorab bekannten Beziehungen können in Form von Relationen im *CIM* bzw. durch *GA_Relationen* zwischen den entsprechenden Attributen unabhängig von der Laufzeit eines Systems modelliert werden. Dieser Schritt kann zunächst entfallen und durch Verfeinern der statischen Modelle mit Wissen aus der Laufzeit später nachgeholt werden, falls die Beziehungen zwischen Infrastrukturelementen anfangs nicht klar sind, jedoch existiert oftmals ein detailiertes Wissen aus dem Entwurf von Softwarekomponenten (beispielsweise durch Auswertung von Funktionsreferenzen zwischen Softwaremodulen oder durch Untersuchung von Softwarerepositories wie in [KK01]), was hierdurch nicht doppelt erreicht werden müsste.

(3) Erfassen der Laufzeitausprägungen
Nachdem die statischen Elemente identifiziert und vorab mit bekannten Relationen verknüpft wurden, werden die dynamischen Aspekte integriert. Konkret sind dies Attributwerte von Attributen, ermittelt durch Messungen (Monitoring) während der Produktion eines IT-Dienstes. Offen bleibt hierbei die Frage nach der Zuordnung des Kontextes eines Attributwerts zu einer konkreten Dienstoperation. Diese Fragen wurde untersucht in [Ga07], der dort beschriebene Ansatz über die Auswertungen von *SOAP*-Nachrichten einen Kontext zwischen einem konkreten Messwert und der bezüglichen Dienstoperation herzustellen ist jedoch auf *SOA*-basierte Ansätze beschränkt. Alternativ wird ein aktives Monitoring mit Zeitstempeln (*timestamps*) durchgeführt, wobei die Zuordnung zwischen Messwert und Dienstoperation von der Genauigkeit der eingesetzten Systemuhren abhängt und somit ebenfalls nicht generisch eingesetzt werden kann.

(4) *Verfeinern der Modelle aus* (1)
Mit mathematischen Korrelationsmethoden können die erfassten Messwerte in Bezug zueinander gestellt werden und demnach gemäß dem generischen Ansatz dynamische Abhängigkeiten zwischen Attributwerten (*GA_Dependency*) und hierdurch letztlich statische Relationen zwischen Attributen (*GA_Relation*) erkannt werden. Das Ergebnis dieser Verfeinerung kann wiederum als Eingabe für den Schritt (1) dienen und somit eine rekursive Betrachtung der Infrastruktur im Bezug auf die erbrachten

Dienstleistungen unterstützen. Dies ist letztlich am Lebenszyklus eines IT-Dienstes angelehnt (siehe 2.1.1).

Neben der reinen technischen Sicht ist weiterhin von Interesse, wie sich dieses Wissen strukturiert zur Verfügung stellen lässt, und somit unabhängig von konkreten Problemdomänen ist. Durch den in dieser Arbeit vorgestellten generischen Ansatz und der Entscheidung, sowohl *CIM* als Informationsmodell für die technische wie auch *RDF* als Beschreibungsansatz für die strukturierende Sicht einzusetzen, kann zu jedem Schritt aus den vorhandenen *CIM*-Instanzen die jeweilige *RDF*-Ressource generiert werden. Zur semantischen Verifikation wurde hierzu ein *RDF-Schema* entwickelt, das als Grundlage für einen Transformation der *CIM*-Elemente in *RDF*-Elemente dient. Details hierzu in 5.2.3 und 7.1.

Im Folgenden werden nun die ersten beiden Schritte des vorgeschlagenen Vorgehens am Beispiel von Infrastrukturelementen des Szenarios durchgeführt. Die Betrachtungen beziehen sich zunächst auf die technische Sicht, anschließend wird die Betrachtung auf die strukturierende Sicht ausgedehnt.

5.2.2 CIM und Generischer Ansatz

Bei der Modellierung der IT-Ressourcen und IT-Dienste mittels des *Common Information Models* steht die Modellierung der technischen Gegebenheiten der Infrastrukturelemente im Vordergrund. Dadurch kann leicht ein Bezug zwischen der technischen Sicht eines IT-Dienstes, wie er sich zur Laufzeit präsentiert, und dem Verhalten der darin eingebundenen Ressourcen hergestellt werden. Zunächst werden jedoch die bekannten Beziehungen der Ressourcen modelliert, um die Modelle sukzessive zu erweitern.

Der bessern Übersicht halber werden die einzelnen Teilmodelle der Ressourcen separat erläutert. Das Modell des *server2* ist der Einfachheit halber analog zu dem Modell des *server1* aufgebaut und wird daher an dieser Stelle nicht explizit betrachtet.

Zunächst wird die Ressource exim näher untersucht. Folgende Abbildung gibt einen grundlegenden Bezug zwischen dem Systemprozess und dem modellierten Serversystem wieder.

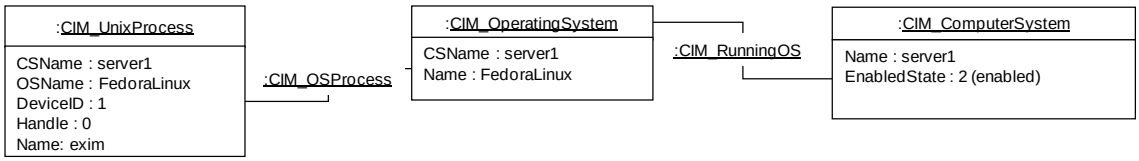

Abbildung 43 CIM Model des Systemprozesses exim

Abbildung 43 zeigt das *CIM*-Modell des Systemprozesses *exim*. Der Bezug zum Serversystem *server1* erfolgt über das Element *CIM_OperatingSystem*. Die Modelle der Systemprozesse *slapd* und *httpd* sind analog aufgebaut und werden ebenfalls an dieser Stelle nicht explizit angegeben.

Um einen Bezug zwischen den Ressourcen einer IT-Infrastruktur und den davon erbrachten IT-Diensten herstellen zu können, stehen im *CIM Core Model* die Klassen *CIM_Service*, *CIM_ServiceAccessPoint* und *CIM_ProtocolEndpoint* zur Verfügung. Abbildung 44 verdeutlicht den Zusammenhang der bekannten Beziehungen zwischen de, Systemprozess exim, dem Serversystem server1 sowie den entsprechenden Elementen der Netzebene.

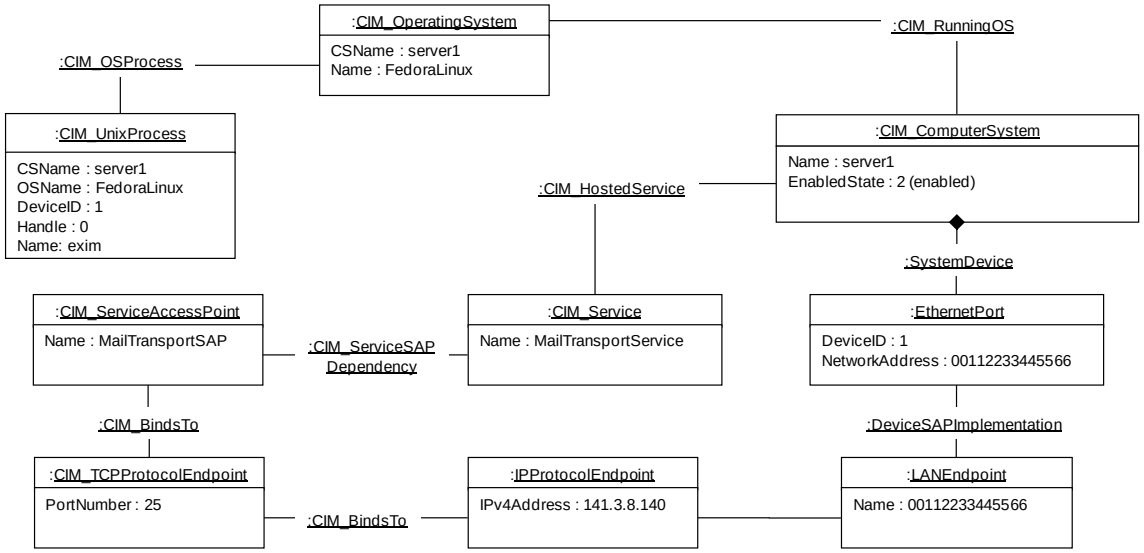

Abbildung 44 CIM Modell des exims mit Modellierung des Service Access Points

Während bisher lediglich Modellelemente des *CIM* Verwendung fanden, erfolgt nun die Hinzunahme der Modellelemente des generischen Ansatzes. Hierzu werden die einzelnen IT-Ressourcen durch die Klassen `GA_Entity` und `GA_Attribute` referenziert. Vorab bekannte Beziehungen zwischen Infrastrukturelementen können durch das Modellelemente `GA_Relation` festgelegt werden.

Dieser Schritt, die Modellelemente des generischen Ansatzes in Bezug zu Modellelementen der *CIM Core* und *Common Models* zu setzen kann auch algorithmisch erfolgen. Folgende Ansätez sind hierbei denkbar:

(1) Einfache Enumeration aller vorhandenen *CIM Instanzen* und Generieren einer *GA_Entity* für jedes von *CIM_LogicalElement* abgeleitete *Managed Object* sowie Generieren von GA_Attribute-Instanzen für jedes Klassenattribut. Dieser Ansatz setzt auf Anwendungslogik-Ebene auf und erfordert vollständig funktionale *CIM*-Klientanwendungen.

(2) *XSLT*-Transformation der in *XML* serialisierten *CIM*-Instanzen. Dieser Ansatz setzt auf der Datentransportebene auf, da keine vollständige CIM-Klientanwendungen implementiert werden müssen, sondern lediglich *XSLT*-Stylesheets, um die gewünschten Instanzen von GA_Entity-Objekten zu ausgewählten *CIM*-Instanzen zu generieren.

Abbildung 45 stellt zunächst das Modell einer CPU (*CIM_Processor*) dar, verknüpft mit Instanzen der Klassen `GA_Entity`, `GA_Attribute` und `GA_AttributeValue`.

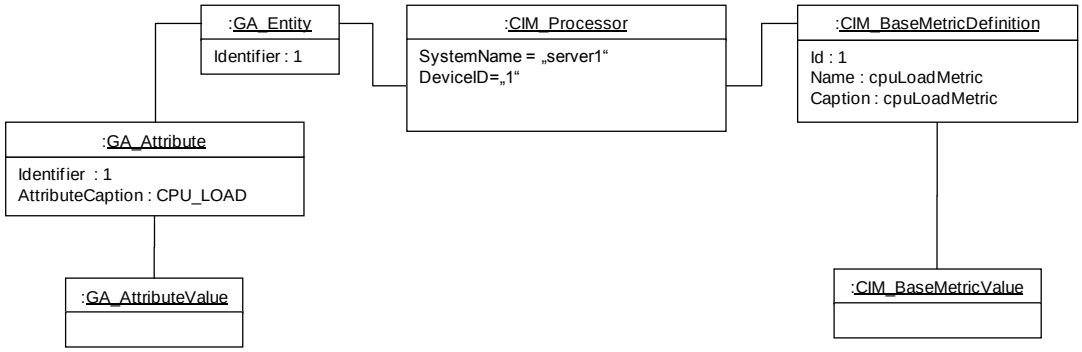

Abbildung 45 Erweiterung durch das Extension Schema des generischen Ansatzes

Durch das *CIM* wird, wie bereits argumentiert, die technische Ebene beschrieben. Mit dem vorgestellten Erweiterungsschema kann dabei sowohl der statische wie auch der dynamische Aspekt integriert werden, wodurch diese Modellierung als Grundlage für weitergehende strukturierende Beschreibungsansätze herangezogen werden kann.

Relationen werden im generischen Ansatz zwischen Attributen von Entitäten modelliert, Abhängigkeiten zwischen Attributwerten. Folgende Abbildung 46 veranschaulicht den Einsatz von Relationen und Abhängigkeiten (anhand hypothetischer Attributwerte). Betrachtet wird hierbei die Relation zwischen der *MailQueueSize* und der *CPULoad*; bei der Modellierung der Abhängigkeit wurde in diesem Beispiel angenommen, dass eine steigende CPU-Last (durch die Nutzung der Webmail-Schnittstelle und damit verbunden eine Nutzung des Verzeichnisdienstes) ein ansteigen der *MailQueueSize* nach sich zieht, da die CPU weniger Zeit in die Abarbeitung der wartenden E-Mails investieren kann.

Diese Annahme ist in der Tat vereinfacht und zieht eine sehr genaue Instrumentierung der einzelnen Entitäten (und deren Attribute) nach sich, ist aber nicht Gegenstand der vorliegenden Arbeit. Ansätze hierfür können beispielsweise in [Ga07] gefunden werden.

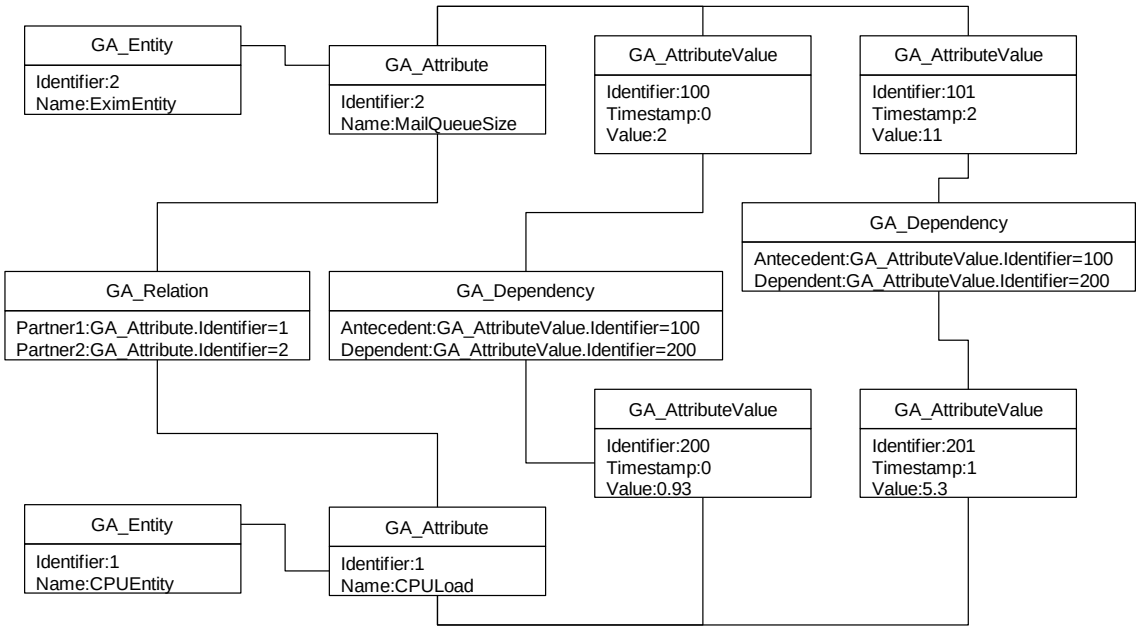

Abbildung 46 Modell mit Relation und Abhängigkeit

5.2.3 RDF/XML

Um den strukturierten Zugriff auf die Instanzen der *CIM* Elemente zu ermöglichen, werden die einzelnen *CIM Instanzen* in gültige *RDF Ressourcen*, die entsprechenden *CIM Assoziationen* in gültige *RDF Properties* übersetzt. Die Regeln für gültige RDF Ausdrücke werden hierbei von einem *RDF Schema* definiert, welches die Modellformen einer *RDF* Ressource im Bezug auf entsprechende *CIM* Instanzen definiert.

Vorgehen

Hierbei sind prinzipiell folgende zwei Ansätze möglich:

(1) Formulierung eines *RDF Schemas*, um die Elemente des *CIM Meta Schemas* in *RDF* zu überführen (Abbildung auf Meta-Modell Ebene). Dies hat zum Vorteil, dass alle dem *CIM Meta Schema* entsprechenden Klassen mittels des formulierten *RDF* Schemas in gültige *RDF* Ausdrücke umgeformt werden können. Dieser Ansatz entspricht dem *recast mapping*, bei dem das *CIM Meta Schema* durch ein entsprechendes *RDF* Schema

ausgedrückt wird. Dieser Ansatz ist jedoch aufwändig und geht nicht auf die Semantik des Domänen-spezifischen generischen Ansatzes ein.

(2) Formulierung eines *RDF Schemas*, um die Elemente eines konkreten *CIM* Modells in *RDF* zu überführen (Abbildung auf Modellebene). Obwohl dies bedeutet, für unterschiedliche *CIM* Modelle unterschiedliche *RDF* Schemata definieren zu müssen, führt dies aufgrund der geringen Komplexität schneller zum Ziel, auch kann so auf Domänen-spezifische Semantik eines *CIM* Modells besser eingegangen werden.

Der Ansatz, ein *RDF Schema* für ein konkretes *CIM* Modell (genauer für das *CIM* Modell des generischen Ansatzes) wird im Folgenden genauer untersucht. Die notwendigen Schritte, um von den *CIM* Klassen über die instanziierten Objekte im *CIM* Server hinzu *RDF* Ausdrücken der entsprechenden *CIM* Instanzen zu gelangen ist in folgender Abbildung 47 dargestellt. Zu Grunde gelegt wird hierbei eine *WBEM*-basierte Architektur.

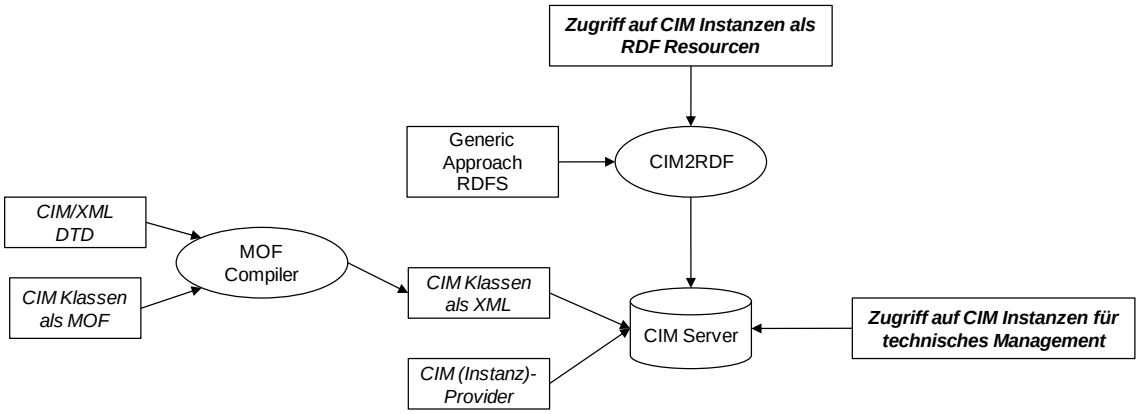

Abbildung 47 Notwendige Schritte zum erzeugen von RDF Ressourcen aus CIM Instanzen

Ein *MOF*-Compiler übersetzt aufgrund der syntaktischen Definitionen in der *CIM*/*XML DTD* [DMTF07a] die *CIM* Klassen in *XML*, um diese in einen *CIM* Server laden zu können. Der CIM Server überprüft mit Hilfe der *CIM* Klassendefinitionen den gültigen Aufbau von instanziierten Objekten, die von *CIM* Providern angelegt werden. Der Zugriff auf die *CIM* Instanzen für das technische Management erfolgt aus den entsprechenden Managementanwendungen direkt, die Übersetzung in *RDF* Ausdrücke erfolgt indirekt. Ein entsprechender Compiler übersetzt die *CIM* Instanzen in einem *RDF* Schema entsprechend gültige *RDF* Ausdrücke.

RDF Schema für den generischen Ansatz

Ein *RDF Schema*, um die *CIM* Modellelemente des generischen Ansatzes in *RDF* Ausdrücke zu übersetzten, ist wie folgt aufgebaut:

(1) Die *CIM* Klassen als *RDF* Element vom Typ `rdf:Resource` mit entsprechender Benennung definiert. Hierdurch können in einem *RDF* Dokument *RDF* Ressourcen instanziiert werden, die Subjekte von *RDF* Statements sein können.

Tabelle 3 Abbildung der CIM Klassen auf RDF Ressourcen

CIM Klasse	*RDF Resource*
`GA_Entity`	`Entity`
`GA_Attribute`	`Attribute`
`GA_AttributeValue`	`AttributeValue`
`GA_Relation`	`Relation`
`GA_Dependency`	`Dependency`
`GA_InstanceOfEntity`	`Instance`

(2) Die Klassenattribute der *CIM* Klassen des generischen Ansatzes sind durch *RDF* Properties vom Typ `rdfs:Property` mit Range `rdf:Literal` wie folgt abgebildet :

Tabelle 4 Abbildung der CIM Klassenattribute auf RDF Properties

CIM Klassenattribut	*RDF Property*
`GA_Entity.Caption`	`CaptionEntity`
`GA_Attribute.Caption`	`CaptionAttribute`
`GA_AttributeValue.AttributeValue`	`Value`
`GA_AttributeValue.Timestamp`	`Timestamp`
`GA_Relation.Partner1`	`Rel_p1`
`GA_Relation.Partner2`	`Rel_p2`
`GA_Dependency.Antecedent`	`Antecedent`
`GA_Dependency.Dependent`	`Dependent`

(3) Zusätzlich ist eine *RDF Property* `attrPartRel` definiert, die bei der *RDF* Ressource `Attribut` einen Verweis auf die Relation enthält, an dem das Attribut teilnimmt.

(4) Die Assoziationen zwischen den *CIM* Klassen des *Extension Schemas* sind ebenfalls durch *RDF Properties* definiert, wobei *Range* und *Domain* auf die jeweiligen Ausgangs- bzw. Ziel *RDF*-Ressourcen zeigen:

Tabelle 5 Abbildung der CIM Assoziationen auf RDF Properties

CIM Assoziation	*RDF Property*
`GA_AttributeValue_Attribute`	`CurrentValue`
`GA_Attribute_Entity`	`Consists`
`GA_Entity_IoE`	`Instantiierung`
`GA_AttriubuteValue_IoE`	`InstanceValue`

Somit kann die Konzeption des *RDF Schemas* zur Transformation der *CIM* Instanzen des Extension Schemas für den Generischen Ansatz wie in Abbildung 48 ersichtlich in *RDF* Graphen-Darstellung angegeben werden. Die entsprechende *XML*-Datei ist im Anhang enthalten. Der abgebildete Graph wurde mit dem *RDF Schema* Validator unter [DFKI] aus dem zugrunde liegenden *RDF*-Dokument automatisch erstellt.

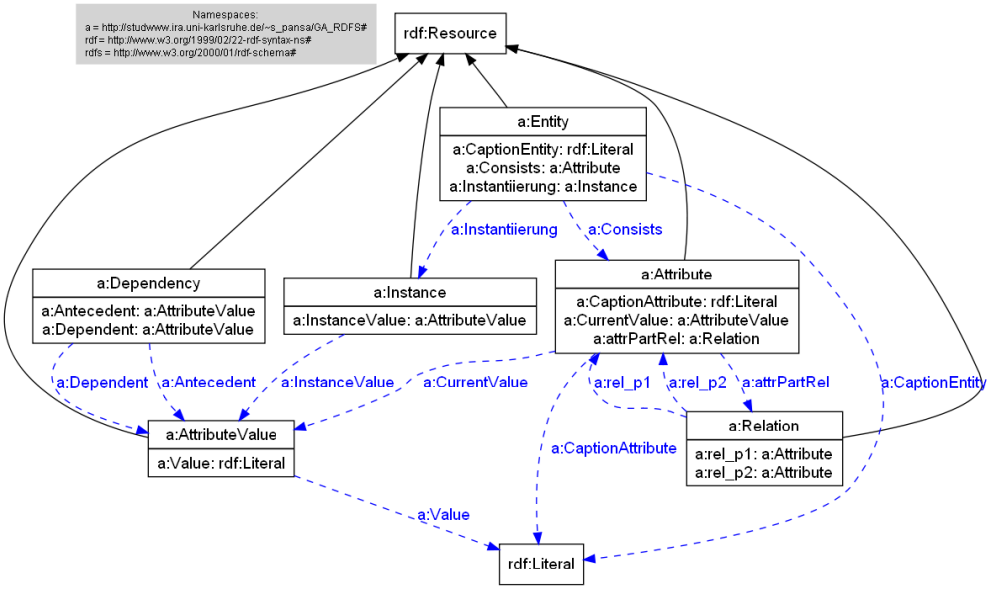

Abbildung 48 RDF Schema für den generischen Ansatz als RDF Graph

Ein prototypischer Compiler für die Übersetzung der *CIM* Instanzen in *RDF* Ressourcen, inklusive der Erstellung der entsprechenden Properties wird im Rahmen des prototypischen Demonstrators im siebten Kapitel vorgestellt.

5.3 Zusammenfassung

In diesem Kapitel wurde zum einen der Schritt motiviert, über die Sicht auf die technische Ebene des Managements hinausgehend die gesammelte Managementinformation strukturierend beschreiben zu können, zum anderen wurde neben der Analyse des Szenarios ein *RDF Schema* entwickelt, mit dem die *CIM* Instanzen des Extension Schemas des Generischen Ansatzes mittels *RDF* dargestellt werden können.

Durch diesen Ansatz kann die Information des technischen Managements zum einen direkt für darauf spezialisierte Anwendungen zur Verfügung gestellt werden, zum anderen kann außerhalb des eigentlichen technischen Managements auf Instanzen der Klassen *GA_Relation* bzw. *GA_Dependency* strukturiert zugegriffen werden. Dieser Ansatz unterstützt demnach sowohl das unmittelbare technische Management (Sammeln von Attributwerten, Überwachung von technischen Parametern, Durchführen von konkreten Aktionen) wie auch darauf aufbauende Managementprozesse wie beispielsweise das Service Level Management, das nicht die technischen Parameter der IT-Ressourcen zum Fokus hat.

Das *RDF Schema* für den generischen Ansatz ist demnach ebenso Teil des Konzepts zur Erfassung von Wechselbeziehungen zwischen IT-Ressourcen, wie das vorgestellte vier-teilige Vorgehensmodell.

6 ENTWURF EINER MANAGEMENTARCHITEKTUR

Im Rahmen der Integration des vorgestellten generischen Ansatzes in bestehende Managementinfrastrukturen wird im vorliegenden Kapitel eine Referenz für eine Managementarchitektur entwickelt. Im Kern dieser Architektur steht hierbei mit dem standardisierten *Common Information Model* ein erweiterbares Informationsmodell, das als Basis für einen integrierten Managementansatz dient. Die Details der Implementierung des Erweiterungsschemas können im Anhang eingesehen werden.

Zunächst wird das Vorgehen motiviert und die Bedeutung des Informationsmodells im Kontext von integrierten Managementlösungen verdeutlicht. Nach der Formulierung von grundlegenden Anforderungen an eine Managementarchitektur zur Integration von statischen und dynamischen Aspekten wird auf den Aufbau der einzelnen Komponenten detailierter eingegangen.

6.1 Motivation

Da in vielen Bereichen des IT-Managements bereits erfolgreich unterschiedliche spezialisierte Tools (isolierte bzw. kooperierende Ansätze [HAN99]) eingesetzt werden, ist im Rahmen der Verbesserung der Managementansätze nach einer integrierten Lösung zu suchen, die die bestehenden Ansätze vereint und somit deren unterschiedliche Stärken verbindet. Im Fokus ist hierbei besonders die Integration von statischen und dynamischen Aspekten, um eine holistische Sicht auf die eingesetzten Infrastrukturelemente zu bekommen, um damit letztlich qualitative Leistungszusicherungen von IT-Diensten garantieren zu können. Dabei müssen jedoch eine Reihe von Anforderungen untersucht werden, um im Hinblick auf ein am IT-Dienst ausgerichtetes IT-Management zu realisieren.

Die vorliegende Arbeit unterstützt einen integrierten Managementansatz dahingehend, das mit dem generischen Ansatz aus Kapitel 4 die Möglichkeit besteht, die Dynamik der Element einer IT-Infrastruktur zu erfassen und somit das *Service Level Management* um eine operationelle Komponente zu erweitern, letztlich aber auch die notwendige statische Sicht zu integrieren. Hierzu kann wiederum Information aus der Datenhaltung von verschiedenen bestehenden Tools eingesetzt werden. Dem Informationsmodell der Referenzarchitektur kommt demnach eine zentrale Stellung zu, da es vom eigentlichen Informationsgehalt eines Datums abstrahiert und somit eine Integration der Daten aus unterschiedlichen Tools erlaubt.

Der Ausdruck *Referenzarchitektur* wird innerhalb der vorliegen Arbeit dabei der Architektur einer Managementanwendung zugeordnet, mittels welcher der erwünschte integrierte Managementansatz erreicht werden kann. Wesentliches Merkmal soll hierbei die in der Einleitung angesprochene Orientierung an den technischen Parametern eines IT-Dienstes sein und sowohl statische Aspekte wie auch dynamische Aspekte unterstützen.

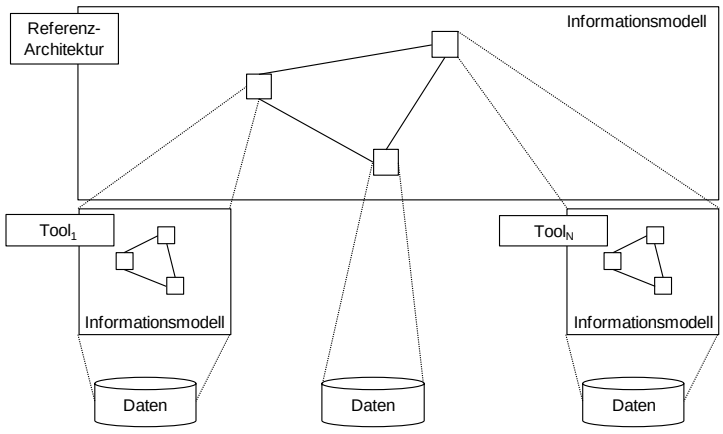

Abbildung 49 Informationsmodell als zentrales Mittel zur Integration verschiedener Systeme

Abbildung 49 verdeutlicht den Zusammenhang zwischen dem Informationsmodell der Referenzarchitektur und den unterschiedlichen (vorhandenen) Tools. Über die Abstrahierung der eigentlichen relevanten Daten durch ein Informationsmodell entsteht die Möglichkeit, Daten unterschiedlicher Systeme automatisierbar zueinander in Verbindung zu bringen. Wesentlicher Vorteil dieser Herangehensweise ist die Tatsache, dass die funktionalen Stärken der einzelnen (vorhandenen) Tools beibehalten wird können, gesammelte Managementdaten jedoch durch eine holistische Sicht erreichbar werden. Weiterhin können nur die für die unmittelbare Produktion eines Dienstes wichtigen Daten aus den einzelnen Datenbeständen herangezogen werden, um so bedarfsgerecht auf die jeweilige Anforderung an das Management reagieren zu können.

Eine Abbildung zwischen verschiedenen Informationsmodellen kann beispielsweise durch folgende Konzepte realisiert werden:

- *DCML* (siehe 3.2.1)
- *CIM Mappings* (2.2)
- *XML Mappings* auf Basis von *XSLT*
- Direkte Modelltransformationen wie beispielsweise durch die Komponente *CIM2RDF* aus der prototypischen Implementierung der Managementarchitektur

6.2 Anforderungen

Wie bereits angesprochen, bringt der Einsatz von modernen informationstechnologischen Verfahren ein hohes Maß an Heterogenität mit sich. Im Gegensatz dazu sollte das Management dieser Infrastrukturen jedoch eine einheitliche Sicht auf die Infrastrukturelemente ermöglichen, wobei der IT-Dienst als zentrale Komponente im Vordergrund stehen sollte. Im vorigen Kapitel wurde hierzu neben dem grundlegenden Verständnis für operationelle Zusammenhänge eine Erweiterung für ein konkretes Informationsmodell formuliert, um zum einen die Dynamik während der Produktion eines IT-Dienstes modellieren zu können, zum anderen aber auch mit Hilfe der Möglichkeit der Modellierung bestehende Managementsystem erweitern zu können. Diese Erweiterung soll in den Entwurf einer Referenzarchitekur fließen, um sowohl statische wie auch dynamische Aspekte zu integrieren.

Diese grundlegenden Anforderungen schlagen sich im Entwurf einer Managementarchitektur nieder. Während üblicherweise verschiedene spezialisierte Managementsysteme im Rahmen von isolierten oder kooperierenden Ansätzen [HAN99] eingesetzten werden, soll im Rahmen der Aufgabenstellung eine integrierte Lösung formuliert werden, die das Management von IT-Dienstleistungen unterstützt, gleichzeitig aber auch die Einbeziehung der Infrastrukturelemente, in Abhängigkeit eines konkreten Dienstes, ermöglicht.

Hieraus folgt bereits, dass die Managementarchitektur sowohl eine Verwaltung von *IT-Diensten*, genauer den funktionalen Schnittstellen, wie auch eine Verwaltung von *IT-Ressourcen*, also jener Infrastrukturelemente, die hinter einer funktionalen Schnittstelle zusammengefasst werden, unterstützen muss. Damit verknüpft ist aber auch eine Verwaltung von Service Level Agreements, also jener Objekte, die die Bindung zwischen einem *Service User/Service Konsument* und einem *Service Provider* definieren und die vereinbarten Leistungen eines IT-Dienstes an dessen funktionaler Schnittstelle festschreiben.
Da neben dem statischen Aspekt im Hinblick auf die korrekte Einhaltung von zugesicherten Service Level Agreements vor allem aber auch der dynamische Aspekt interessiert, muss die reine Verwaltung durch Überwachungskomponenten ergänzt werden.

Im Rahmen der Erfassung des Verhaltens von IT-Ressourcen während der Produktion eines IT-Dienstes stehen zunächst Messwerte im Vordergrund. Über die Messwerte (Attributwerte) können gemäß dem generischen Ansatz dynamische Abhängigkeiten erfasst werden. Daher ist die hier vorgestellte Referenzarchitektur auf den reinen Monitoring-Aspekt beschränkt.

Aufgaben, die einen statischen Charakter haben und somit unabhängig von der Laufzeit der Infrastruktur sind, sind beispielsweise:

- Hinzufügen oder Entfernen einer IT-Ressource, eines IT-Dienstes oder eines Service Level Agreements in die Managementanwendung
- Hinzufügen oder Entfernen einer vorab bekannten Relation zwischen IT-Ressourcen, beispielsweise durch Wissen aus Design und/oder Implementierungsphasen
- Modifizieren der abgelegten Information bezüglich der verwalteten Infrastrukturelemente und den Relationen untereinander. Hierbei wird vor allem gefordert, Schnittstellen zur Datenhaltung anzubieten, mit denen gezielt Relationen zwischen Infrastrukturelementen definiert werden können.
- Generieren von Plänen, sogenannte *Topologie-Karten*, auf Basis der abgelegten Informationen. Dieser wichtige Punkt unterstützt zum Beispiel Planungs- oder Änderungsprozesse und hilft dem Service Provider, eine Übersicht über eingesetzte Komponenten und deren statische/funktionale Beziehungen untereinander erkennen zu können.

Dagegen werden zur Überwachung der Einhaltung von zugesicherten Qualitätsniveaus im Kontext der Laufzeit einer Infrastruktur beispielsweise folgende Aufgaben anfallen:

- Sammeln von Messwerten oder Metriken
- Korrelieren von Messwerten, um Abhängigkeiten erkennen oder verifizieren zu können
- Verifizieren und Evaluieren der statischen Topologie-Karten durch Laufzeitinformation. Obwohl dieser Punkt eigentlich unabhängig von der Laufzeit eines Systems ist, ist dieser Anwendungsfall von den Messwerten der Laufzeit abhängig.
- Generieren von Berichten, um beispielsweise Verletzungen von vereinbarten Qualitätsniveaus melden zu können

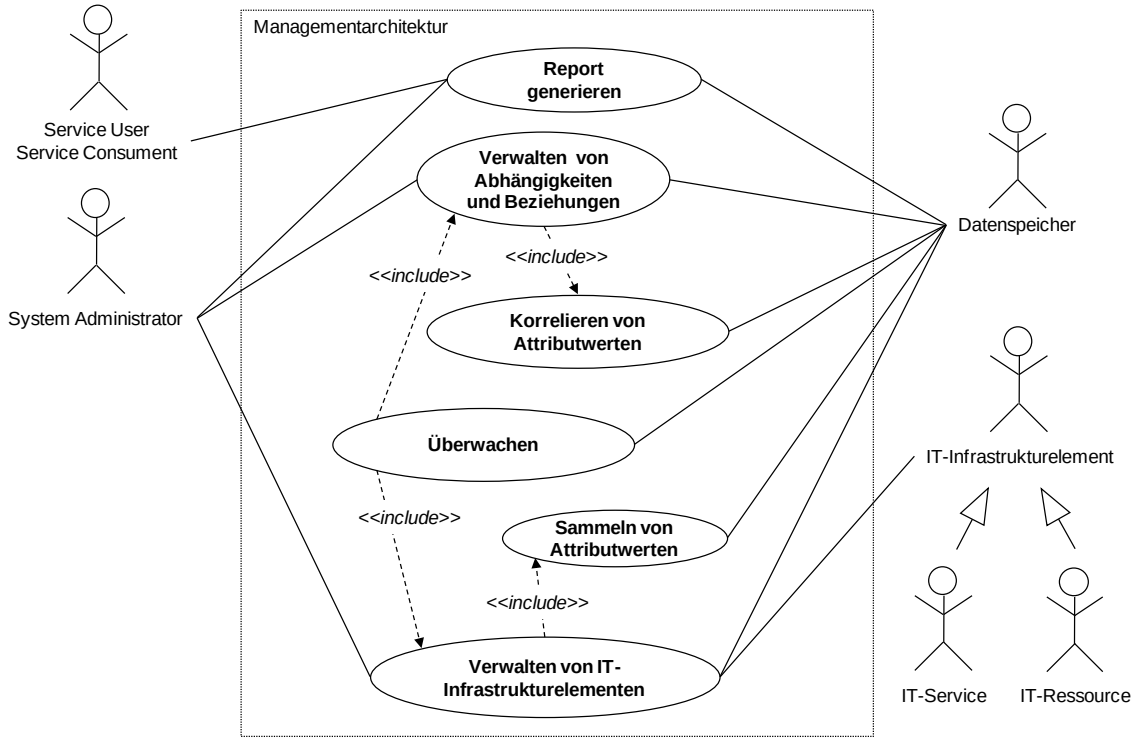

Abbildung 50 Funktionale Anforderungen und beteiligte Akteure

Abbildung 50 zeigt in einem Anwendungsfalldiagramm die funktionalen Aufgaben und die Beziehungen der einzelnen Anwendungsfälle zueinander, sowie die beteiligten Akteure.

Im Folgenden werden die identifizierten Anwendungsfälle und die einzelnen Akteure detailierter beschrieben:

- *Service User/Service Consumer (Akteur)*
  Der Akteur *Service User/Service Consumer* kann sowohl eine menschliche Person wie auch eine maschinenbasierte Komponente darstellen, welche eine funktionale Schnittstelle zu einer Infrastruktur benutzt. Dabei werden qualitative Parameter der Dienstleistung im SLA festgeschrieben, weshalb dieser Akteur im Falle einer Verletzung dieser Vereinbarung davon unterrichtet werden sollte. Dieser Akteur hat nur einen passiven Zugriff auf die Managementarchitektur, indem er lediglich Berichte empfängt.
- *System Administrator (Akteur)*
  Der Akteur *System Administrator* stellt eine menschliche Person dar, welche die Managementarchitektur zur Verwaltung und Überwachung der einzelnen IT-Infrastrukturelemente einsetzt. Dabei wird entweder über eine grafische Benutzerschnittstelle oder über eine Sammlung von Kommandozeilentools auf die Managementanwendung zugegriffen
- *IT-Infrastrukturelement (Akteur)*
  Der Akteur *IT-Infrastrukturelement* stellt die einzelnen Komponenten einer IT-Infrastruktur dar, wie sie beispielsweise im generischen Informationsmodell in Kapitel 4.2 unterschieden werden. Es lässt sich eine Spezialisierung in die Akteure *IT-Ressource* und *IT-Service* definieren.
- *Datenspeicher (Akteur)*
  Der Akteuer Datenspeicher stellt eine *technische* Komponente im Rahmen der Managementarchitektur dar, da in ihr sowohl statische Information über den Aufbau der Infrastruktur und den Relationen der einzelnen Elemente im Kontext eines IT-Dienstes abgelegt werden, wie auch Messwerte, die sich zur Laufzeit ergeben und somit dynamische Abhängigkeiten beschreiben. Nicht notwendigerweise muss hierzu eine einzelne zentrale Datenbank eingesetzt werden.
- *Verwalten von IT-Infrastrukturelementen (Anwendungsfall)*
  Eine zentrale Aufgabe in der Arbeit bezüglich einer Managementanwendung besteht darin, Managementobjekte hinzuzufügen oder zu entfernen. Diese Aufgabe kehrt im Rahmen der Optimierung von IT-Infrastrukturen (zur Optimierung von IT-Diensten) wieder und muss daher über dedizierte Schnittstellen automatisierbar werden. Weiterhin wird an dieser Stelle eine integrierte Sicht erreicht, indem Information aus unterschiedlichen Datenbeständen referenziert werden kann.
- *Sammeln von Attributwerten (Anwendungsfall)*
  Zur Überwachung der Einhaltung von zugesicherten Qualitätsniveaus an einer Dienstschnittstelle muss der Dienstanbieter eine detaillierte Sicht auf die im Kontext einer Dienstleistung eingebundenen Ressourcen haben. Da die Produktion eines IT-Dienstes sich in der messbaren Änderung von Zuständen der Ressourcen manifestiert, muss diese Dynamik erfasst werden, um bei eventuellen Verletzungen eines Leistungsvertrages die schuldige Ressource identifizieren zu können. Das Sammeln von Attributwerten ist demnach stark in den dynamischen Aspekt integriert und unmittelbarer Bestandteil beim Management von Dienstinfrastrukturen.
- *Verwalten von Beziehungen und Abhängigkeiten (Anwendungsfall)*
  Die bekannten statischen Beziehungen wie auch bekannte Abhängigkeiten müssen definiert werden können und als Managed Objects in die Datenhaltung der Managementanwendung integriert werden können. Weiterhin müssen dynamische Abhängigkeiten, die sich erst zur Laufzeit ergeben und daher nicht vorab bekannt sind, über eine automatisiert ansprechbare Schnittstelle in die Datenhaltung integriert werden können.

- *Korrelieren von Attributwerten (Anwendungsfall)*
  Zur Erkennung von dynamischen Abhängigkeiten in Bezug auf die Produktion eines IT-Dienstes werden gemäß dem generischen Ansatz Abhängigkeiten über die Änderung von Attributwerten von Infrastrukturelementen definiert.
- *Überwachen*
  Im Rahmen der Überwachung der Einhaltung der vereinbarten Qualitätsniveaus an der Dienstschnittstelle werden die gesammelten Attributwerte gegenüber Schwellwerten verglichen, um so frühzeitig von Problemen von einzelnen Ressourcen informiert zu werden. Weiterhin können auch Abhängigkeiten zwischen Attributwerten überwacht werden, wobei nicht mehr ein einzelner Wert im Vordergrund steht, sondern der gesamte Kontext betrachtet wird.
- *Report generieren (Anwendungsfall)*
  Sowohl *Service User* / *Service Consumer* wie auch der *System Administrator* können über den Zustand von einzelnen Ressourcen informiert werden. Unter diesen Anwendungsfall fallen auch die Erstellung von Übersichten und Statusplänen.

6.3 Modell einer Managementarchitektur

Nachdem die funktionalen Anforderungen an die Managementarchitektur formuliert sind, wird im folgenden Teilkapitel der Aufbau der Architektur genauer untersucht (Organisationsmodell). Ziel dieses Abschnittes ist es, durch die Beschreibung von Komponenten und deren Schnittstellen die Vorlage für eine Referenzarchitektur zu erhalten, die im Hinblick auf die gestellten Anforderungen als Grundlage für die Implementierung einer Managementanwendung herangezogen werden kann. Dabei werden im wesentlichen die Anwendungsfälle umgesetzt, die auf bestehenden Managementinformationen bezüglich den Modellierungselementen von Infrastrukturkomponenten aufsetzen, Komponenten, die eine Schnittstelle bieten, um Infrastrukturelemente in das Managementsystem aufzunehmen werden nicht weiter untersucht..

Aufgrund der Heterogenität und der Notwendigkeit, eine integrierte Sicht auf eine IT-Ressourcen basierte Infrastruktur im Kontext von Servicemanagement zu erlangen, wird eine verteilte Managementarchitektur entworfen, bei der eine Trennung in die Kernkomponenten *Management Node* (mit der Rolle *Manager*) und *Managed Node* (mit der Rolle *Agent*) vorgenommen wird. Die modellierten Objekte der Infrastrukturelemente, die *Managed Objects*, sind dabei direkt auf den *Managed Nodes* implementiert. Der Vorteil bei dieser Architektur liegt darin begründet, dass der Gesamtdatenbestand des Managementsystems flexibel um beliebige *Managed Nodes* erweitert werden kann. Der Austausch von Managementinformation wird demnach durch ein verteiltes System realisiert.

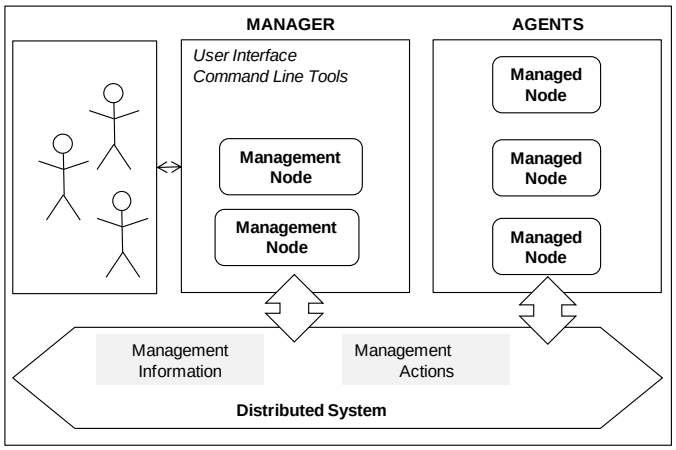

Abbildung 51 Grundlegender organisatorischer Aufbau der Referenzarchitektur

Durch die verteilte Implementierung der *Managed Objects* auf den einzelnen *Managed Nodes* müssen diese innerhalb des Managementsystems lokalisiert werden. Beispielweise können die

einzelnen *Managed Nodes* hinter *Web Services* gekapselt werden und mittels einem *Service Location Protocol* (WS-Management, [DMTF06a]) lokalisiert werden. Der Einfachheit halber wird jedoch des prototypischen Charakters der Umsetzung wegen ein hybrider Ansatz aus dezentraler Datenhaltung (auf den einzelnen Managed Nodes) und zentralem Lokalisierungsdienst eingesetzt.

6.3.1 Komponentenmodell

Zur Realisierung der gestellten Anforderungen wird eine *Manager*/*Agent*-basierte Managementarchitektur umgesetzt. Es wird angenommen, dass lediglich lesend auf Managementinformation zugegriffen wird. Ein Schreibzugriff, um den Zustand von *Managed Objects* im Rahmen von Steuerungseingriffen zu verändern, ist zunächst nicht vorgesehen. Zur Lokalisierung der *Managed Nodes* wird ein Verzeichnis-basiertes System zu Grunde gelegt, die grundlegenden Anforderungen im Rahmen der Fragestellung werden durch eine Korrelationskomponente zur Erkennung von Abhängigkeiten aufgrund von Attributwertänderungen (`Correlator`) und einer Komponente zur Generierung von *RDF*-Modellen aus der technischen Managementinformation des *CIM* erfüllt (`CIM2RDF`).

Dieser Aufbau wird gewählt, da mit der Kernforderung der Erkennung und Verifikation von Abhängigkeiten innerhalb einer IT-Infrastruktur die Korrelationskomponente unabhängig von den übrigen Komponenten ist und somit auch unabhängig davon implementiert und erweitert werden kann. Die Korrelationskomponete soll Aufgrund unterschiedlicher Korrelations-Algorithmen über die Verarbeitung von Attributwerten Abhängigkeiten im Sinne von funktionalen Änderungen von Zahlwerten feststellen und letztlich dynamische Abhängigkeiten zwischen IT-Ressourcen während der Produktion eines IT-Dienstes erkennen, aber auch bekannte Relationen aus dem statischen Entwurf validieren können. Dies ist notwendig, da mit dem generischen Ansatz lediglich ein Informationsmodell definiert ist, in dem prinzipielle Zusammenhänge modelliert werden.

Nachfolgendes Schaubild gibt eine Übersicht über die Referenzarchitekur in Form eines Komponentenmodells wieder. Der Akteur *Service User* / *Service Consumer* wird aus den folgenden Betrachtungen ausgeklammert, da im Rahmen des reinen Monitoringaspekts kein prinzipieller Unterschied zum Akteur *System Administrator* besteht. Die einzelnen Komponenten *Management Node*, *Managed Node*, *Correlator* und *Managed Objects Directory* werden anschließend detaillierter beschrieben.

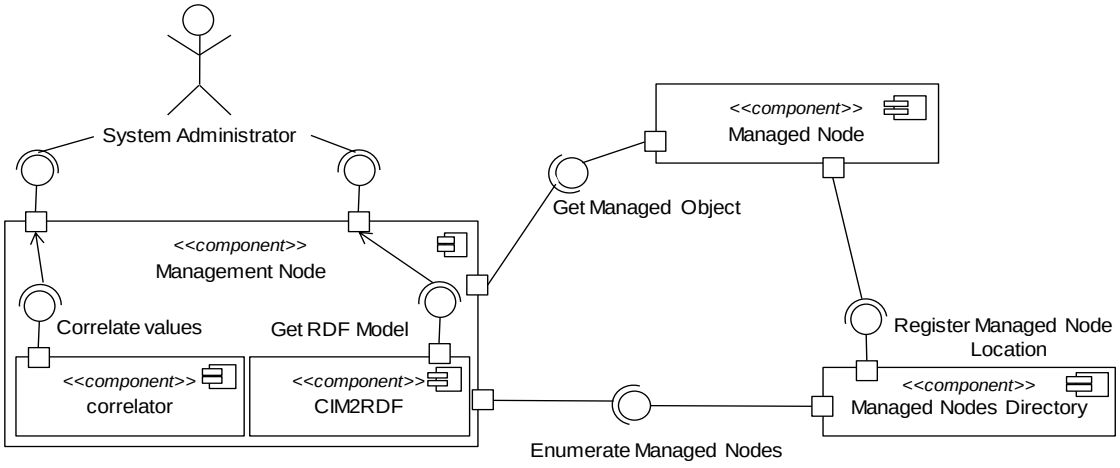

Abbildung 52 Organisationsmodell der Referenzarchitektur

Sowohl *Service User*/*Service Consumer* wie auch *System Administratoren* als menschliche Akteure kommen nur mit den *Management Nodes* direkt in Kontakt. Der Akteur *Datenspeicher* wird durch eine verteilte Datenbank und einen zentralen Verzeichnisserver zur Lokalisierung

der *Managed Nodes* implementiert. Die Idee ist hierbei, die anfallenden Messwertinformation direkt auf den *Managed Nodes* – sofern möglich – zu speichern und erst bei Bedarf, beispielsweise durch einen *Management Node* oder durch die Korrelationskomponente, abzurufen. Es wird angenommen, dass jeder Managed Node einen Agenten im Sinne von Abbildung 51 ausführen kann.

Das *Managed Nodes Directory* ermöglicht die Lokalisierung der einzelnen *Managed Nodes*. Hierzu kann über eine Registrierungsfunktion eine Referenz zu einem Lokalisierungspfad angelegt werden, und über eine Lokalisierungsfunktion der Pfad zu einem *Managed Node* aufgelöst werden. Unter einem Pfad ist hierbei ein Identifier gemeint, der eine eindeutige Lokalisierung des Managed Nodes innerhalb des verteilten Systems ermöglicht.

6.3.2 Management Node

Der Management Node ist diejenige Komponente, die eine direkte Benutzerschnittstelle zum Managementsystem bietet.

Am *Management Node* fallen folgende Aufgaben an:

- Modifizieren, Hinzufügen oder Entfernen von *Managed Objects.*
- Erkennen von Beziehungen zwischen Ressourcen, sowohl statischer wie auch dynamischer Natur (`Correlator`)
- Generieren von strukturierten Modellen zur Überwachung von Ressourcen, Beziehungen aber auch um Pläne der Infrastruktur zu erhalten. (`CIM2RDF`)

Diese grundlegenden Aufgaben spiegeln den statischen und dynamischen Aspekt von Beziehungen und Abhängigkeiten im Rahmen des Lebenszyklus eines IT-Dienstes wieder und unterstützen das Management dabei, sowohl vorhandenes Wissen in die Management-Anwendung zu integrieren wie auch dieses Wissen durch Laufzeitbeobachtungen schrittweise zu validieren, um letztlich wiederum Konsequenzen für Planungs- und Entwurfsprozesse zu erhalten.

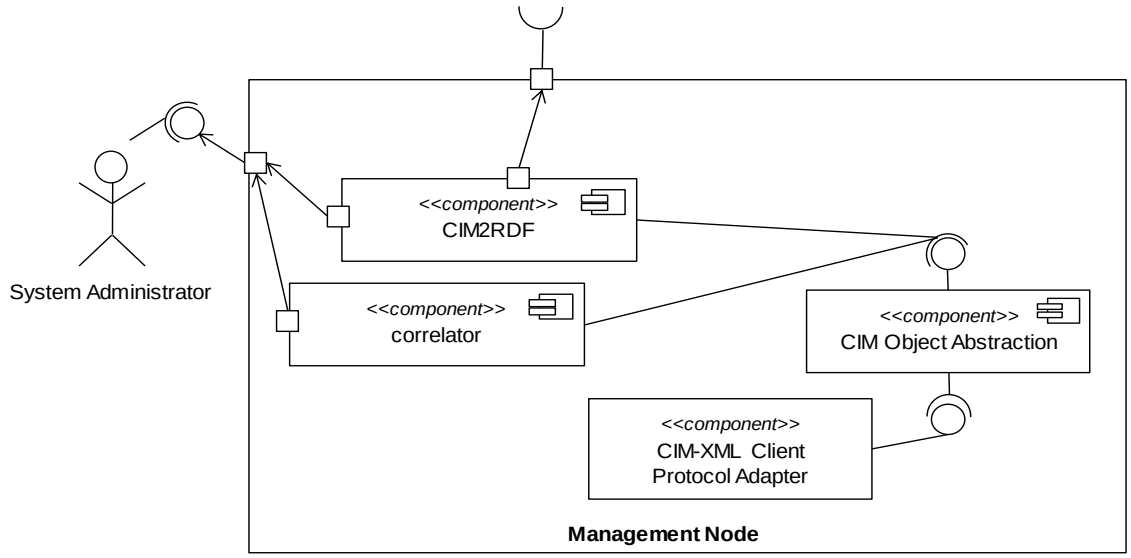

Abbildung 53 Komponenten eines Management Nodes mit CIM-relevantem Teil

Abbildung 53 gibt eine Übersicht über die benötigten Komponenten, um die anfallenden Aufgaben gemäß den Anforderungen zu erfüllen. Die Kommunikation zu den restlichen Komponenten der Managementarchitektur wird durch die Komponente `CIM/XML Protocol Adapter` umgesetzt, welche den Transport der *CIM* Objekte durch Serialisierung in *XML* und anschließend den eigentlichen Datentransport über *http* realisiert [DMTF07b].

Correlator

Die Komponente `Correlator` wird als Teilkomponente des Management Nodes entworfen. Der `Correlator` erkennt aufgrund mathematischer Korrelationsalgorithmen Abhängigkeiten zwischen Komponenten durch Operation auf den entsprechenden Attributwerten von Attributen.

Während durch den generischen Ansatz lediglich die Möglichkeit geschaffen wird, die Information "Abhängigkeit zwischen Attributwerten" in ein Informationsmodell zu integrieren, muss darüber gehend hinaus untersucht werden, wie diese Abhängigkeit zwischen zwei Attributwertpaaren in einer beliebigen Menge an Attributwerten erkannt werden kann. Der `Correlator` implementiert demnach eine Anwendungslogik, die auf den unmittelbaren Modellierungsmöglichkeiten durch den generischen Ansatz im Informationsmodell aufsetzt.

Im Rahmen dieser Arbeit steht die Untersuchung der notwendigen Modellierungsansätze im Vordergrund, um Wechselbeziehungen *beschreiben* zu können und Wissen über bekannte Wechselbeziehungen im Rahmen eines integrierten Managementansatzes zur Verfügung zu stellen, daher wird die Komponente `Correlator` an dieser Stelle nur rudimentär untersucht. Der Idee bei der Umsetzung der Korrelationskomponente liegt die Auffassung zugrunde, das sich ändernde Attributwerte in Abhängigkeit zueinander stehen müssen. Über die Änderung eines Attributwertes eines Attributs und der Feststellung, das sich ein Attributwert eines weiteren Attributs im Rahmen einer einstellbaren Quantifikation ändert, wird eine reale Abhängigkeit zwischen Attributwerten erfasst.

Folgender einfacher Algorithmus könnte auf Basis eines einstellbaren Schwellwertes Änderungen zwischen Attributwerten erkennen und demnach Abhängigkeiten zwischen Attributwerten erfassen.

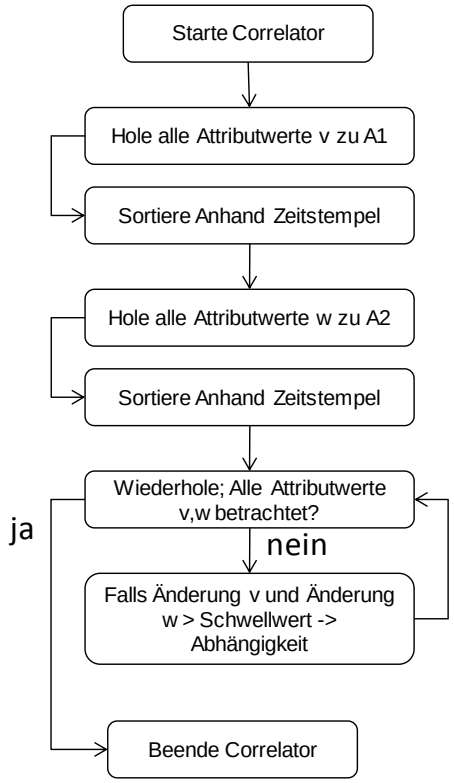

Abbildung 54 Ablauf des einfachen Korrelationsalgorithmus

Für eine detaillierte Untersuchung von Korrelationskomponenten können beispielsweise Ansätze wie in [En01, GJ+06] verfolgt werden.

CIM2RDF

Die Komponente `CIM2RDF` stellt Funktionalität bereit, um aus den technischen Managementdaten eine strukturierte Sicht auf die Managementinformation zu bekommen. Grundlage hierfür ist das *RDF Schema* aus Kapitel 5.2.3. Details der Implementierung im Rahmen der prototypischen Umsetzung können in Kapitel 7 eingesehen werden.

6.3.3 Managed Node

Am Managed Node fallen folgende Aufgaben ab:

- Hinzufügen, Entfernen oder Modifizieren eines Managed Objects.
- Speichern und Abfragen des Zustandes ein Managed Objects. In Bezug auf die Umsetzung des generischen Ansatzes bedeutet dies, Attributwerte

Managed Nodes sind diejenigen Komponenten, die im Rahmen der Betrachtung eines IT-Dienstes an dessen Produktion beteiligt sind und von der Managementanwendung erfasst sind. *Managed Nodes* stellen demnach die Agenten innerhalb einer verteilten Managementanwendung dar und implementieren die *Managed Objects*.

Nachfolgende Abbildung gibt eine detailierte Übersicht über die zur Bewältigung dieser im Rahmen einer *WBEM*-basierten Architektur anfallenden Aufgaben wieder. Die Schnittstelle `AccessManagedNode` stellt *CIM*/*XML*-basierte Zugriffsfunktionen bereit.

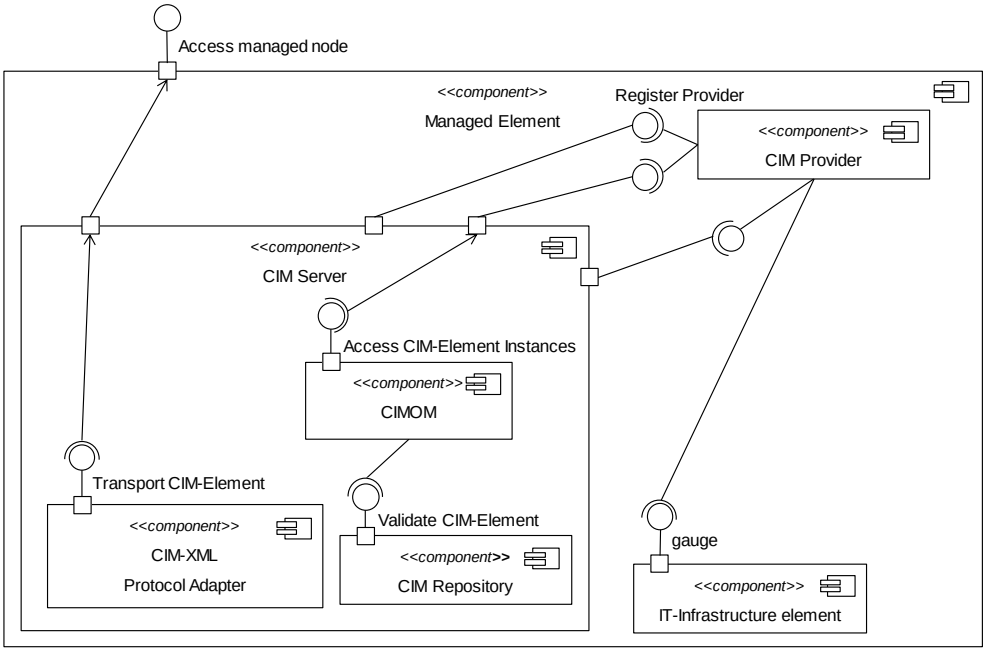

Abbildung 55 Komponenten eines *Managed Nodes* mit CIM-relevantem Teil

Der Aufbau der Komponenten `CIM Server` und `CIM Provider` entspricht den gleichnamigen Komponenten einer *WBEM*-Architektur (siehe 2.4). Hierbei sei angemerkt, das *CIM* Provider nicht zwangsläufig nur Messwerte von Infrastrukturelementen zur Verfügung stellen müssen, sondern Instanzen für beliebige *CIM* Klassen anbieten können. Die Komponente `IT-Infrastructrue element` steht stellvertretend für die zu modellierenden Infrastruktur-Elemente, müssen aber nicht zwangsläufig physikalischer Natur sein.

Am Infrastrukturelement fallen Messwerte ab, dies kann entweder durch aktive Überwachung seitens des `CIM Providers` geschehen, oder durch Push-Mechanismen seitens des Infrastrukturelements. Über die Schnittstelle `Access CIM-Element Instances` in der Teilkomponente `CIM Server` werden dem `CIMOM` (`CIM Object Manager`) neue Instanzen

bekannt gegeben, die ein `CIM Provider` direkt verwaltet, oder neue Instanzen angelegt, die durch den `CIMOM` selbst verwaltet werden.

6.3.4 Managed Nodes Directory

Die Komponente *Managed Nodes Directory* bietet bedingt durch den verteilten Ansatz der Managementarchitektur eine Möglichkeit, *Managed Nodes* innerhalb des Managementsystems zu lokalisieren. *Managed Objects* werden direkt auf den jeweiligen *Managed Nodes* implementiert und nicht an zentraler Stelle gehalten, weshalb eine Lokalisierung erforderlich wird. Die Lokalisierung erfolgt hierbei in zwei Schritten:

(1) *Lokalisierung eines Managed Nodes*: Beispielsweise durch auflösen einer Name-IP Adresse Bindung
(2) *Lokalisierung eines Managed Objects*: Beispielsweise durch Enumeration aller *CIM* Instanzen und anschließender Filterung.

Hierzu werden zwei Schnittstellen angeboten, die es zum einen ermöglichen, einen Lokalisierungspfad für ein *Managed Node* festzulegen (`RegisterManagedNode`), zum anderen den Lokalisierungspfad eines *Managed Nodes* aufzulösen (`LookupManagedNode`).

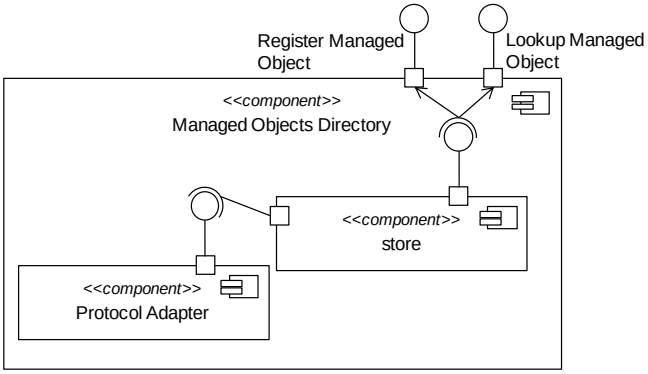

Abbildung 56 Komponentenmodell des Managed Object Directory

Die Teilkomponente `store` ermöglicht eine persistente Datenablage, die Komponente `Protocol Adapter` realisiert die Kommunikation mit den Komponenten `Managed Node`, `Management Node` und `Correlator`.

6.4 Zusammenfassung

In diesem Kapitel wurde eine Managementarchitekur basierend auf einem verteilten Manager/Agent-basierten Ansatz vorgestellt. Die vorgeschlagene Architektur realisiert einen *WBEM*-basierten Ansatz für verteiltes Management. Der Entwurf der Architektur konzentriert sich auf die Umsetzung des generischen Ansatzes um sowohl statische Beziehungen wie auch Abhängigkeiten aufgrund von Laufzeitbeobachtungen erfassen und verwalten zu können.

Die hier vorgestellte Managementarchitekur setzt dabei die in den beiden vorhergehenden Kapiteln betrachteten Aspekte um:

(1) Die technische Sicht auf die Elemente einer IT-Infrastruktur wird durch das *Common Information Model* mit dem Erweiterungsschema für den generischen Ansatz beschrieben. Auf den einzelnen Managed Nodes werden Messwerte abgegriffen und diesen den durch das Informationsmodell festgelegten Entitäten und Attributen in Form von Instanzen der Klasse `GA_AttributeValue` zugewiesen. Die Komponente `Correlator` findet dabei auf Basis von mathematischer Korrelationsanalyse Beziehungen zwischen Attributwerten unterschiedlicher Attribute in dem Sinne, das hierdurch die beschriebene Dynamik während der Produktion einer IT-Dienstleistung in

der Veränderung von Attributwerten zu den zugehörigen IT-Ressourcen erfasst werden kann.

(2) Um die gesammelten Managementdaten im Rahmen eines integrierten Management-Ansatzes weiterverarbeiten zu können, werden die im Managementsystem vorhandenen Instanzen der *CIM* Klassen (*Managed Objects*) in *RDF*-Ausdrücke überführt. Hierdurch wird die technisch-bezogene Ebene verlassen und es entsteht die Möglichkeit, die technische Managementinformation in den Kontext weiterer Managementinformation zu stellen, die nicht durch rein-technische Ansätze beschrieben werden können.

7 PROTOTYPISCHE IMPLEMENTIERUNG, EVALUATION UND ERGEBNISSE

In diesem Kapitel wird eine prototypische Implementierung des Szenarios und Teilen der Managementarchitektur vorgestellt. Zunächst wird von einigen bekannten Zusammenhängen innerhalb der Infrastruktur ausgegangen, um den Schritt der rekursiven Verfeinerung anhand von Anwendung des generischen Ansatzes auf reale Messdaten zu demonstrieren. Die gewonnen Informationen sowie eine kurze Diskussion der Tragfähigkeit des vorgestellten Konzeptes wird im Anschluss durchgeführt.

7.1 Details der Implementierung

7.1.1 Einschränkungen im Rahmen eines Prototyps

Bei der Implementierung des prototypischen Demonstrators wurden folgende begründete Einschränkungen vorgenommen:

(1) Die Modellierung der technischen Aspekte konzentriert sich auf die Umsetzung der *CIM*-Elemente des vorgestellten *Extension Schemas*, beim Prototyp wird keine detailgetreue Modellierung wie in Kapitel 5, durchgeführt. Dies stellt keine Einschränkung in Bezug auf den Nachweis der Tragfähigkeit des Ansatzes dar, da die möglichst detailgetreue Nachbildung einer Infrastruktur mittels *CIM* nicht im eigentlichen Fokus dieser Arbeit liegt.

(2) Die Umsetzung der Service Level Agreements bezüglich der einzuhaltenden Qualitätsniveaus wird nicht weiter untersucht, da dies beispielsweise in [DK03] betrachtet wird. Durch den Ansatz, die Information der technischen Ebene strukturiert zur Verfügung zu stellen, entsteht die Möglichkeit, die technischen Parameter und deren messbaren Charakteristika im Rahmen des Service Level Managements mit weiteren Parametern zu verbinden und so letztlich den IT-Dienst in seiner Gesamtheit zu beschreiben.

(3) Die Implementierung der Komponente Provider wurde der Einfachheit halber nicht wie in einer *WBEM*-basierten Architektur eigentlich vorgesehen durch in den *CIMOM* eines *CIM* Servers nachladbare Programmmodule realisiert, sondern durch externe Programme, die Messwerte sammeln und die entsprechenden *CIM* Instanzen im *CIMOM* direkt anlegen. Dies stellt keine Einschränkung dar, da die detailierte Implementierung eines *CIM* Servers nicht im Fokus dieser Arbeit liegt.

(4) Gänzlich unbeachtet blieb im Rahmen dieser Arbeit die Frage der Instrumentierung der einzelnen Infrastrukturressourcen. So wurde zur mangels Alternative zur Messung der *CPULoad* auf Information des Betriebssystems zurückgegriffen, welche jedoch nicht den korrekten Wert der CPU Last zum Zeitpunkt des Aufrufes zurück liefert, sondern einen Mittelwert aus unterschiedlichen Zeitintervallen.

7.1.2 Implementierung des Szenario

Um den generischen Ansatz am Beispiel des vorgestellten Szenarios demonstrieren zu können, wurde die im Szenario angedachte Infrastruktur bestehend aus Webmaildienst, Verzeichnisdienst und Mailtransportdienst umgesetzt.

Um die Auswirkungen der Laufzeitaspekte zu unterstreichen, wird zusätzlich die Entität `WMService` betrachtet, welche die Dienstschnittstelle zum Webmaildienst modelliert. Über die Messung des Attributs `DelieveryDuration` kann die Qualität des Servicelevels an der Nutzerschnittstelle in Bezug zu den Attributwerten der übrigen Entitäten gestellt werden. Weiterhin wird angenommen, dass die Relationen zu den übrigen Infrastrukturelementen noch nicht bekannt sind. Durch die Anwendung des generischen Ansatzes soll über die Abhängigkeit von Attributwerten schließlich ein Bezug des `DelieveryDuration` Attributs zu den übrigen Attributen beobachtet werden.

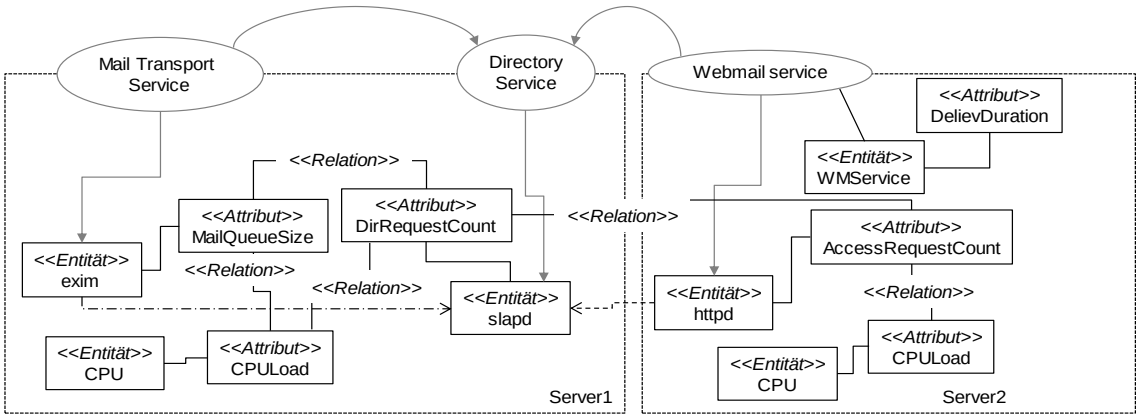

Abbildung 57 Elemente des Szenarios mit Modellelementen des generischen Ansatzes

Die Serversysteme `Server1` und `Server2` wurden durch virtuelle Maschinen (*virtual machines*, *VM's*) umgesetzt. Die Software Komponenten der Dienste `Webmail Service`, `Mail Transport Service` und `Directory Service` wurden wie folgt umgesetzt:

(1) `Mail Transport Service`
Den Mailtransportdienst übernimmt der *Mail Transfer Agent* (MTA) *exim*. Die Zieladressen von eingehenden E-Mails werden durch den Verzeichnisdienst authentifiziert. Dieser Mechanismus entspricht der realen Umsetzung im Rahmen des Maildienstes der ATIS und verdeutlich die Notwendigkeit, Abhängigkeiten zwischen IT-Ressourcen auch bezüglich deren Laufzeitverhalten bewerten zu können. Hierbei interessiert das Attribut `MailQueueSize` bezüglich der Entität exim, da dieses Attribut einen Hinweis darauf gibt, wie schnell eingehende E-Mails die Warteschlange verlassen können und daher innerhalb der vereinbarten *Service Level Agreements* zugestellt werden können.

(2) `Directory Service`
Der Verzeichnisdienst wird durch eine Instanz des ldap-Servers *slapd* übernommen. In die Datenbank des Verzeichnisdiensts wurden 10.000 Benutzer eingefügt, welches in etwa auch der Größenordnung von angelegten Einträgen im zentralen Verzeichnisdienst der ATIS entspricht. Bei der Entität *slapd* interessiert die Anzahl der durchgeführten Verzeichnisdienstanfragen pro Zeiteinheit (`DirRequestCount`), da dies ein Indikator für den Durchsatz an Anfragen ist und dadurch im Rahmen der Beurteilung der Servicequalität der betrachteten Dienstschnittstellen eine Rolle spielt.

(3) `Webmail Service`
Der Webmail Service wird durch den *http*-Server *apache* erbracht, wobei die Authentifikation an der Webmail-Schnittstelle durch eine Passwortgeschützte `index.html` simuliert wird. Als Attribut wird die Anzahl der Anfragen pro Zeiteinheit betrachtet (`AccessRequestCount`), da dies wie beim Directory Service auch ein Indikator für das Qualitätsniveau der entsprechenden Dienstschnittstelle darstellt.

Während die betrachteten Attribute streng genommen noch keinen direkten Aufschluss auf die Güte des Dienstes darstellen, in dessen Kontext die zugehörige Entität eingebunden ist, stellen sie dennoch wichtige technische Parameter bei der Beurteilung der möglichen Güte der zugehörigen Dienste dar.

Die Entität *CPU* wird durch das Attribut `CPULoad` charakterisiert, da dieser Wert die aktuelle Auslastung der *CPU* wiedergibt und Aufgrund der zentralen Bedeutung der *CPU* als Ressource in einem Serversystem eine wichtige Charakteristik bei der Untersuchung von Qualitätsniveaus bezüglich systembasierten IT-Diensten ist. Die Relation zwischen den Attributen `MailQueueSize` und `CPULoad` bzw. `DirRequestCount` und `CPULoad` (auf `Server1`) und ebenso `AccessRequestCount` und `CPULoad` (auf `Server2`) steht offensichtlich schon zur Entwurfszeit fest und kann demnach in Form einer (statischen) Relation durch das

Modellelement `GA_Relation` des generischen Ansatzes beschrieben werden. Die Relation zwischen den Attributen der Entitäten *exim* und *slapd* bzw. *slapd* und *httpd* stehen ebenfalls zur Entwurfszeit fest, da eine funktionale Beziehung aufgrund der Notwendigkeit der Authentifikation besteht.

Insgesamt werden demnach jeweils sechs Entitäten mit zugehörigen Attributen im Szenario betrachtet, wobei insgesamt fünf Relationen zwischen Attributen vorab aus dem Entwurf der Infrastruktur bekannt sind. Die bekannten funktionalen Beziehungen sollen gemäß dem in 5.2.1 diskutierten Vorgehensmodell verifiziert werden, eine bisher unbekannte Beziehung (zwischen `DelieveryDuration` und `CPULoad` von CPU1) im Rahmen der Anwendung des generischen Ansatzes entdeckt werden.

7.1.3 Implementierungen von Managementkomponenten

Teile der vorgestellten Managementarchitektur wurden im Rahmen des Demonstrators mit Java 1.6 implementiert. Im Rahmen der *WBEM*-basierten Umsetzung der Managementarchitektur wurde für die Implementierung auf den *CIM* Server der opensource-Lösung *WBEMServices* [WBEM] zurück gegriffen, da obwohl seit 2004 nicht mehr aktiv weiterentwickelt mit der vorliegenden Version 1.0.2 hinreichend stabil und vollständig in der Umsetzung der von der *DMTF* definierten *CIM/XML* Operationen ist.

Managed Nodes Directory

Während für die *Managed Nodes* die verfügbare Lösung *WBEMServices* eingesetzt wurde, entstand im Rahmen des Entwurfs einer verteilten Managementlösung die Notwendigkeit, die einzelnen *Managed Nodes* lokalisieren zu können, um Managementinformation zu sammeln. Diese Funktionalität der Lokalisierung wird durch das *Managed Nodes Directory* zur Verfügung gestellt (siehe auch 6.3.4) und im Rahmen der prototypischen Umsetzung implementiert.

Das `Managed Nodes Directory` wurde hierbei als TCP-basierter Serverprozess implementiert, der folgende Schnittstellenfunktionen zur Verfügung stellt:

(1) `registerManagedNode( )`
Ein *CIM* Server kann beim *Managed Nodes Directory* mit einem Namen und einer IP-Addresse registriert werden.

(2) `enumerateManagedNode( )`
Die lookup-Funktion zur Auffindung eines *Managed Nodes* wird durch Aufzählung aller registrierten *Managed Nodes* erreicht.

Provider

Die Komponente Provider implementiert eine Methodik, um gesammelte Messwerte von Sensoren bezüglich der untersuchten Entitäten und deren Attribute in den CIM Server zu laden. Konkret geschieht dies, indem die Komponente Provider zu einem bestimmten Attribut-Identifier den vom Aufruf der Provider-Komponente übergeben Parameter als Attribut einer Instanz der Klasse `GA_AttributValue` hinzufügt, und die Assoziation zwischen dieser `GA_AttributValue` Instanz und dem zugehörigen Attribute in Form einer Instanz der Assoziationsklasse `GA_AttributValue_Attribute` hinzufügt.

Zur konkreten Implementierung der Komponente Provider wurden Funktionen der im Rahmen der *WBEMServices* ausgelieferten Funktionsbibliothek *wbem.jar* herangezogen. Folgendes Codebeispiel gibt einen Ausschnitt aus der Provider-Komponente wieder. Ersichtlich ist neben dem Initiieren einer Verbindung zum *CIM* Server die notwendigen Schritte, um eine Instanz der Klasse `GA_AttributeValue` dem *CIM* Server hinzuzufügen.

```
// connecte zum CIM Server
CIMNameSpace cns = new CIMNameSpace("//"+localhost+"/"+NameSpace);
UserPrincipal up = new UserPrincipal(User);
PasswordCredential pc = new PasswordCredential(Passwd);
cimClient = new CIMClient(cns, up, pc);
//+++

// erstelle Instanzen von GA_AttributeValue
opAttributeValue = new CIMObjectPath("GA_AttributeValue");
cimClass = cimClient.getClass(opAttributeValue, false );
instAttributeValue = cimClass.newInstance();
instAttributeValue.setProperty("Identifier", new CIMValue(identifier));
instAttributeValue.setProperty("AttributeValue", new CIMValue(givenValue));
instAttributeValue.setProperty("Timestamp", new CIMValuetimestamp));
opAttributeValue.addKey("Identifier", new CIMValue(identifier));
cimClient.createInstance(opAttributeValue, instAttributeValue);
//++++
```

Abbildung 58 Codeauszug aus der Komponente `Provider`

Die einzelnen Messwerte werden dabei mit vom Betriebssystem zur Verfügung gestellten Mechanismen gesammelt:

(1) Für die Messung der *CPULoad* wird Information aus dem *proc*-Dateisystem des *linux*-Kernels abgefragt:

```
CURRENTLOAD=`cat /proc/loadavg | cut -d ' ' -f 1`
```

(2) Für die Messung der Warteschlangengröße der Mailqueue wird das Kommando `` `mailq -v` `` eingesetzt.

(3) Für die Messung der Zugriffe auf den *slapd* bzw. *httpd* werden die entsprechenden log-Dateien nach Einträgen untersucht.

Correlator

Im generischen Ansatz wird das Ziel verfolgt, Wechselbeziehungen zwischen Infrastrukturelementen aufgrund von Abhängigkeiten zwischen Attributwerten von Attributen von Entitäten zu erfassen.

Dies bedeutet, das wenn zu Attribut a_1 der Wert des Attributs (Attributwert) $f_{a_1}(t_1) \neq f_{a_1}(t_0)$ ist und wenn zu Attribut a_2 der Wert des Attributes $f_{a_2}(t_1) \neq f_{a_2}(t_o)$ ist, eine Abhängigkeit in dem Sinne zwischen den Attributwerten besteht, das eine Änderung des Zustandes einer Ressource (Änderung eines Attributwerts) eine Änderung des Zustandes einer weiteren Ressource (Änderung eines Attributwerts) nach sich zieht.

Die Implementierung dieser Komponente erfolgte nur skizzenhaft und wurde lediglich dazu eingesetzt, unterschiedliche Attributwerte anhand deren Zeitstempel zu sortieren, eine Korrelation auf Attributebene wurde nicht durchgeführt. Für eine tiefergehende Untersuchung der Korrelationskomponenten wären auch Ansätze wie in [En01,GJ+06] denkbar.

CIM2RDF

Die Komponente zur Umsetzung der technischen Managementinformation aus CIM heraus stellt den Kern der prototypischen Implementierung im Rahmen der Fragestellung, wie die gesammelte Managementinformation der technischen Ebene strukturiert präsentiert werden kann, dar.

Zur Erzeugung von RDF-Ausdrücken wurde auf das java-basierte Framework *jena* [Jena] in Version 2.5.4 zurückgegriffen, das eine komfortable Schnittstelle zu unterschiedlichen Aspekten bei der Erstellung von gültigen RDF-Dokumenten bietet.

Die Funktionalität der Komponente `CIM2RDF` ist auf fünf Klassen aufgeteilt. Im Kern steht die Klasse `CIM2RDFFactory`, welche im Prinzip das Software-Designpattern Fabrik (engl. Factory) umsetzt. Die Erzeugung der RDF-Ausdrücke zu den unterschiedlichen Elementen der CIM Instanzen des Extension Schemas wird von der Factory `CIM2RDFFactory` mit der Schnittstellenfunktion `Model getModel( Model model, CIMInstance instance)` realisiert. Abhängig der der Art der parametrisierten CIM Instanz wird ein entsprechender RDF Ausdruck realisiert und mittels der Funktion `model.addResource()` als *RDF Resource* bzw. `model.addProperty()` als *RDF Property* direkt in das bestehende RDF Modell eingefügt.

Abbildung 59 zeigt beispielhaft einen Codeauschnitt aus der Klasse `CIM2RDFFactory`, welche bei Eingabe einer CIM Instanz mit passender Klassenbezeichnung `GA_Entity` die Methode `getEntityModel`() aufruft. Als Parameter werden der Funktion dabei die Parameter der öffentlichen Fabrikmethode `getModel()` durchgereicht.

```
public static Model getModel( Model model, CIMInstance instance ) throws Exception {
  String cimClassName = instance.getClassName();

  if( cimClassName.equals( "GA_Entity" )){
    return CIM2RDFFactory.getEntityModel(model, instance);
  } else if( cimClassName.equals( "GA_Attribute" ) ) {
    return CIM2RDFFactory.getAttributeModel(model, instance);
  } else ……

  ……
}

private static Model getEntityModel( Model model, CIMInstance inst ) {
  String value = cutQuotes(inst.getProperty("Identifier").getValue().toString());
  Resource entity = model.createResource( nsPrefix
                                          + inst.getClassName()
                                          + "."
                                          + "Identifier"
                                          +"="+ value, GASchema.Entity );
  ntity.addProperty(GASchema.CaptionEntity,
                    cutQuotes(inst.getProperty("name").getValue().toString()));
  return model;
}
```

Abbildung 59 Codeauszug der Fabrikmethode `getEntityModel()`

Das vollständige Code-Listing ist im Anhang an die vorliegende Arbeit enthalten.

Die weiteren Klassen implementieren folgende Funktionalität:

Tabelle 6 Klassen der Komponente CIM2RDF und deren Funktionalität

MObjectsEnum	Implementiert ein Auswahlobjekt, das mit hinterlegter Parameterliste eine Enumeration alles *CIM* Instanzen eines *CIM* Servers auflistet
MNodesEnum	Implementiert ein Auswahlobjekt, das eine Enumeration aller IP-Adressen aller registrierten an einem Managed Nodes Directory registrierten Managed Nodes enthält
GASchema	Implementiert das entwickelte *RDF Schema*, das bei der Produktion der *RDF* Modellobjekte in der *CIM2RDFFactory* als Grundlage zur syntaktischen Verifikation durch das *jena*-Framework dient
CIM2RDF	Implementiert die *main*-Funktion und traversiert mit Hilfe einer *MOEnum*-Instanz durch die verfügbaren *CIM* Instanzen, um diese von der *CIM2RDFFactory* in *RDF* Ausdrücke transformieren zu lassen. Die Ausgabe wird in einer Text-Datei gespeichert.

7.2 Evaluation

Zur Demonstration des Ansatzes und zum Nachweis der Tragfähigkeit des vorgestellten Konzepts werden die Ergebnisse der Korrelationskomponente sowie die automatisiert erstellten Modelle mit der Komponente *CIM2RDF* diskutiert. Hierbei sei nochmals angemerkt, das die Korrelationskomponente bewusst einfach gehalten wurde, da mit dem Tragfähigkeitsnachweis vor allem die Umsetzung des generischen Ansatzes und darüber hinausgehend die Möglichkeiten der strukturierenden Beschreibung diskutiert werden sollen.

Weiterhin wurde festgestellt, das die eingesetzte Instrumentierung der IT-Ressourcen mit Mittel der zugrundeliegenden Betriebssysteme schnell an ihre Grenzen stößt.

Es wurden unterschiedliche Lastsituationen simuliert, und die Attributwerte zu den Attributen der Entitäten gemessen. Weiterhin wurde zu den unterschiedlichen Situationen das technische *SLA* am Webmailinterface betrachtet, um die Auswirkungen der Produktion des E-Mail-Teildienstes Mail-Transport auf die Qualität des zugesicherten Service-Levels an der Webmail Schnittstelle zu bekommen.

Die Simulation bestand auf Seiten des Mailtransport Teildienstes durch Zustellung von E-Mails an 10.000 zufällige Nutzer, die im Verzeichnisdienst angelegt sind. Auf Seiten der Webmail-Schnittstelle wurde die Authentifizierung durch eine gesicherte Startseite simuliert, wobei die Authentifizierungsinformation aus dem gleichen ldap-Datenbestand kommt wie für den E-Mail-Transport Dienst. Der technischen Qualitätslevelparameter "Auslieferungszeit" wurde an einem dritten virtuellen Rechner innerhalb der gleichen *VMWare*-Applikation gemessen, so dass Effekte durch den Netzwerktransport vernachlässigt werden können. Die Attribute *DirRequestCount* und *AccessRequestCount* geben Aufschluss über Anforderungsraten an den Verzeichnisdienst bzw. an den Webserver.

Das Lastszenario war wie folgt aufgebaut:

(1) Zunächst wurden die Messwerte während eines Leerlaufs der IT-Ressourcen abgegriffen, weder der E-Mail-Dienst noch der Webmail-Dienst wurden angefordert.
(2) Der E-Mail-Dienst wurde 1000 mal angefordert (jeweils zufällig-unterschiedliche Nutzer), anschließend wurde wieder die Leerlaufsituation beobachtet.
(3) Der Webmaildienst wird 200 mal jeweils von zufälligen Nutzern angefordert, anschließend wieder Leerlauf
(4) Sowohl der E-Mail-Dienst wie auch der Webmail-Dienst werden angefordert.

Abbildung 60 gibt den Verlauf der absoluten Attributwerte der jeweiligen Entitäten in einem Diagramm wieder. Dargestellt sind die Attributwerte der Entitäten CPU1 mit dem Attribut `CPULoad` (durchgezogene Linie), der CPU2 ebenfalls mit dem Attribut `CPULoad` (gestrichelte Linie), des exim mit dem Attribut `MailQueueSize` (gepunktete Linie) und die Zeiten der Auslieferungsdauer an der Webmailschnittstelle (Attribut `DelievDuration`, markierte Linie). Die Messwerte wurden in den CIM-Servern auf den Systemen Server1 bzw. Server2 durch die entsprechenden Provider gespeichert und zur Analyse durch die `Correlator`-Komponente geordnet bezüglich der Zeitstempel der Messwerte ausgegeben.

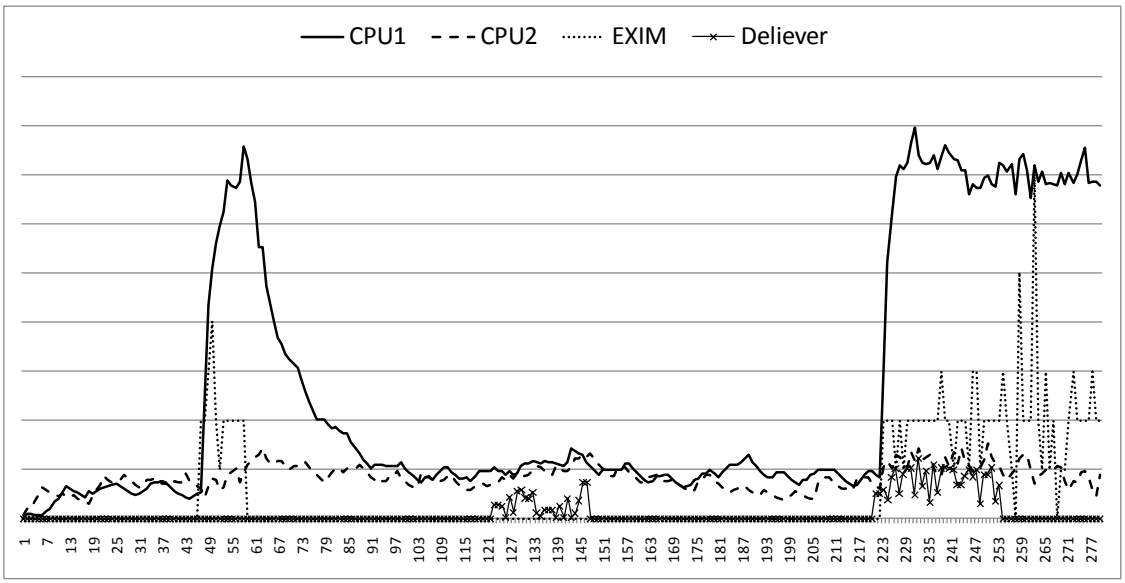

Abbildung 60 Verlauf der Attributwerte im Lastszenario

Folgende Situationen können beobachtet werden:

(1) Die Anforderung an den Mail-Teildienst schlägt sich in einer höheren Last der CPU1 nieder.
(2) Die alleinige Nutzung der Webmail-Schnittstelle erzeugt keine nennenswerte Last auf sowohl CPU1 wie auch auf CPU2.
(3) Die Nutzung von sowohl E-Mail-Dienst wie auch der Webmailschnittstelle gleichzeitig wirkt sich in einer erhöhten Last von sowohl CPU1 wie auch CPU2 aus, außerdem steigt durch die gleichzeitige Nutzung die Auslieferungsdauer auf Seiten der Webmail-Schnittstelle um etwa das doppelte.

Als Grundlage für die Formulierung des generischen Ansatzes wurde die Annahme getroffen, das Abhängigkeiten zwischen IT-Ressourcen sich in einer Veränderung des Zustandes der betreffenden Ressourcen während der Produktion eines IT-Dienstes bemerkbar machen. Im Szenario entspricht dies genau den Beobachtungen (1)-(3). Der generische Ansatz mit der Trennung von Entitäten, Attributen und Attributwerten durch eigenständige Modellierungselemente kann demnach ein geeignetes Mittel sein, um Attributwerte (Messwerte) unterschiedlicher Entitäten (IT-Ressourcen) miteinander in Bezug zu setzen. Der letzte Schritt, mithilfe der Korrelationskomponente Attributwerte über die Änderungen deren Werte untereinander in Beziehung zu setzen um im Kontext der Produktion eines IT-Dienstes die Wechselwirkungen der IT-Ressourcen untereinander zu erfassen konnte nicht mehr vollständig umgesetzt werden.

Neben der Evaluation des generischen Ansatzes bezüglich der technischen Sicht mit der Umsetzung in einer *WBEM*-basierten Managementanwendung konnte die Implementierung der Komponente *CIM2RDF* erfolgreich dazu eingesetzt werden, die Managementinformation strukturiert zur Verfügung zu stellen.

Folgender Ausschnitt aus dem mit Hilfe der Komponente *CIM2RDF* generierten *RDF*-Dokument verdeutlicht die Umsetzung der technischen Sicht in eine strukturierende Sicht, die frei von der technischen Semantik der Elemente des *CIM* ist.

Dargestellt sind drei *RDF-Statements* bezüglich der Entität 1 (entspricht der CPU1), der Entität 2 (entspricht dem exim) und der Relation zwischen den Attributen der beiden Entitäten.

```
<rdf:Description rdf:about="http://example.org/GA_Entity.Identifier=1">
        <gas:Consists rdf:resource="http://example.org/GA_Attribute.Identifier=1"/>
```

```
        <gas:CaptionEntity>CPU Entity</gas:CaptionEntity>
        <rdf:type rdf:resource="http://studwww.ira.uni-karlsruhe.de/~s_pansa/GA_RDFS#Entity"/>
</rdf:Description>

<rdf:Description rdf:about="http://example.org/GA_Entity.Identifier=2">
        <gas:Consists rdf:resource="http://example.org/GA_Attribute.Identifier=2"/>
        <gas:CaptionEntity>Exim Entity</gas:CaptionEntity>
        <rdf:type rdf:resource="http://studwww.ira.uni-karlsruhe.de/~s_pansa/GA_RDFS#Entity"/>
</rdf:Description>

<rdf:Description rdf:about="http://example.org/GA_Relation -
                                GA_Attribute.Identifier=1,GA_Attribute.Identifier=2">
        <gas:rel_p2 rdf:resource="http://example.org/GA_Attribute.Identifier=2"/>
        <gas:rel_p1 rdf:resource="http://example.org/GA_Attribute.Identifier=1"/>
        <rdf:type rdf:resource="http://studwww.ira.uni-karlsruhe.de/~s_pansa/GA_RDFS#Relation"/>
</rdf:Description>
```

Somit kann auf einfache Weise die technische Managementinformation durch einen strukturierenden Ansatz beschrieben werden.

7.3 Zusammenfassung

Mit der prototypischen Umsetzung des Szenarios konnte die praktische Anwendung des generischen Ansatzes demonstriert werden. In einer Simulation wurden dabei unterschiedliche Lastsituationen bezüglich der Anforderung der im Szenario betrachteten IT-Dienste aufgezeigt. Dabei ändern die beteiligten IT-Ressourcen ihre Zustände in dem Sinne, dass die Attributwerte zu den Attributen der entsprechenden Entitäten ihre Ausprägungen ändern.

Hierdurch können schließlich bekannte Zusammenhänge aus dem Entwurf (im generischen Ansatz durch die Relation zweier Attribute) verifiziert werden. Weiterhin kann durch die Anwendung des generischen Ansatzes eine Wechselwirkung zwischen einer erhöhten CPU-Last auf *Server1* und eine Erhöhung der Auslieferungszeiten an der Webmail-Schnittstelle auf Server2 beobachtet werden. Durch Analyse der entsprechenden einzelnen Attributwerte können letztlich Abhängigkeiten auf Ebene der Attributwerte festgestellt werden. Dieses Wissen kann schlussendlich dazu eingesetzt werden, im Rahmen des *Service Level Managements* die Auswirkung der Nutzung der Webmailschnittstelle anhand der Beobachtungen während der Produktionszeit des IT-Dienstes besser zu verstehen. Durch die Erfassung der Abhängigkeit zwischen den Attributwerten und Modellierung als eigenständiges *Managed Object* (in Form des Elements *GA_Dependency*) kann die Abhängigkeit als Gegenstand des Managements bestehenden Managementanwendungen zur Verfügung gestellt werden.

Um weitergehende Ansätze zu unterstützen, die direkt auf den Managementdaten der technischen Ebene aufsetzten, konnte erfolgreich automatisiert ein *RDF*-Dokument erstellt werden, das Anhand des in 5.2.3 entwickelten *RDF Schemas* ein syntaktisch und semantisch korrektes Abbild der technischen Managementdaten ist, ohne jedoch die technischen Details zu integrieren. Dieses Dokument kann dann im Rahmen eines integrierten Managementansatzes dazu eingesetzt werden, nicht-technische Parameter (wie beispielsweise die Arbeitszeit von Supportmitarbeitern) in Bezug zu den strukturierten Information der technischen Parameter zu stellen, um das Management von IT-Diensten durch *SLA's* zu verbessern.

8 ZUSAMMENFASSUNG UND AUSBLICK

8.1 Zusammenfassung

Die Produktion eines IT-Dienstes spiegelt sich zur Laufzeit einer IT-Infrastruktur in den Änderungen der Zustände der beteiligten IT-Ressourcen wieder. Diese Zustandsänderungen der IT-Ressourcen lassen Rückschlüsse auf die erzielbare Qualität an der Dienstschnittstelle zu, weshalb das Management von IT-Ressourcen nicht zuletzt aus Gründen wie *Kosteneffizienz*, *Outsourcing* und *BDIM* aus Sicht eines IT Service-Providers von bedeutendem Interesse ist.

Während in modernen vernetzten Kommunikationssystemen oftmals eine Vielzahl unterschiedlicher IT-Ressourcen an der Produktion eines IT-Dienstes beteiligt sind, haben die Änderungen der Zustände oftmals Auswirkungen auf weitere Ressourcen, die Aufgrund der Lokalisation (zwei Netz-basierte Systemprozesse konkurrieren um die gleiche physikalische Ressource) in weitere IT-Dienste eingebunden sind. Diese Beziehungen zwischen IT-Ressourcen, die sich erst zur Produktionszeit eines IT-Dienstes manifestieren, müssen vom Management ebenso betrachtet werden, wie die schon zur Entwurfszeit bekannten statischen Modelle. Eine Analyse von Metriken um Beziehungen ableiten zu können, setzt voraus, das der Aufbau der Metriken bekannt ist.

Im Verlauf dieser Arbeit wurde ein Konzept entwickelt, das Beziehungen zwischen IT-Ressourcen, die an der Produktion eines IT-Dienstes beteiligt sind, genauer untersucht. Im Kern dieses Konzepts stehen die Formulierung eines generischen Beschreibungsansatzes, der strukturierenden Transformation der Elemente der technischen Ausrichtung des generischen Beschreibungsansatzes und der Entwurf einer Managementarchitektur, das sowohl die technische Sicht auf die Elemente einer IT-Infrastruktur unterstützt, wie auch darüber hinausgehend die Möglichkeit bietet, auf die Information der technischen Ebene strukturiert zugreifen zu können. Im Kern des generischen Ansatzes steht die Vereinigung von statischen und dynamischen Aspekten.

Der statische Aspekt wurde hierbei mit dem Begriff der Relation in Verbindung gebracht und beschreibt Beziehungen, die aus dem Entwurf und der Implementierung von IT-Ressourcen bekannt sind, während mit dem dynamischen Aspekt der Begriff der Abhängigkeit verbunden wird, da hier konkrete Abhängigkeiten zwischen Messwerten betrachtet werden. Im generischen Ansatz werden hierzu die Infrastrukturelemente durch den Begriff Entität, die Eigenschaften von Entitäten durch den Begriff Attribut und konkrete Ausprägungen in Form von Messwerten durch den Begriff Attributwert beschrieben. Relationen bestehen zwischen Attributen von Entitäten, Abhängigkeiten dagegen zwischen konkreten Attributwerten. Da es sich bei dem vorgestellten generischen Ansatz um ein Modellierungskonzept auf Ebene eines Informationsmodells handelt, wird hier die Frage, wie konkrete Abhängigkeiten zwischen Attributwerten *erkannt* werden können, zunächst ausgeklammert. Um den generischen Ansatz in ein bestehendes Informationsmodell integrieren zu können, wurde ein Erweiterungsschema für das *Common Information Model* definiert, das die Modellelemente des generischen Ansatzes in CIM integriert.

Die Analyse bestehender Arbeiten hat gezeigt, das die Ansätze nicht dafür ausgelegt wurden, einen holistischen Ansatz (statische und dynamische Aspekte in einem Ansatz vereint) zu begehen. Zwar wurde in [KK01] eine Trennung in strukturelle, funktionale und operationelle Modelle vorgenommen, jedoch ist dieser Ansatz auf die Beziehungen zwischen IT-Diensten ausgelegt. Im *Common Information Model* existiert mit dem *Metrics Model* zwar ein komplexes Beschreibungsmodell, um Metriken und deren Beziehungen zu Infrastrukturelementen beschreiben zu können, jedoch besteht bislang keine Möglichkeit, Abhängigkeiten zwischen Messwerten in Bezug auf zugehörige Eigenschaften von Ressourcen zu erfassen. Der Ansatz in [Hu07] bezieht sich auf bekannte funktionale Zusammenhänge innerhalb einer *SOA*, welche in

einer realen System-basierten IT-Infrastruktur nicht immer genau bekannt sind. Die Erkenntnisse des generischen Ansatzes können demnach einen Ansatz wie in [Hu07] unterstützen.

Um einen ganzheitlichen Ansatz bezüglich einer integrierten Sicht auf die zu einem IT-Dienst gehörigen IT-Ressourcen erreichen zu können, muss die gesammelte Managementinformation der technischen Managementebene strukturiert zur Verfügung gestellt werden können. Während der vorgestellte generische Modellierungsansatz sich auf die technische Sicht von Infrastrukturelementen bezieht, wurde mit der Formulierung eines *RDF* Schemas ein Schritt in Richtung der strukturierenden Beschreibung der im *CIM* Erweiterungsschema zum generischen Ansatz vorhandenen Modellelemente vorgenommen.

Dazu wurde im Rahmen des Entwurfs einer Managementarchitektur ein *WBEM*-basierter verteilter Managementansatz entworfen, der mit dem Common Information Model ein standardisiertes und umfassendes Managementinformationsmodell für die technische Sicht auf IT-Infrastrukturelemente bietet. Weiterhin wurde der Entwurf einer Komponente diskutiert, die die angesprochene Umsetzung der technischen Managementinformation der *CIM* Instanzen in *RDF*-Ausdrücke ermöglicht. Abbildung 61 fasst die Beziehungen der einzelnen Teile der in dieser Arbeit untersuchten Aspekte zusammen.

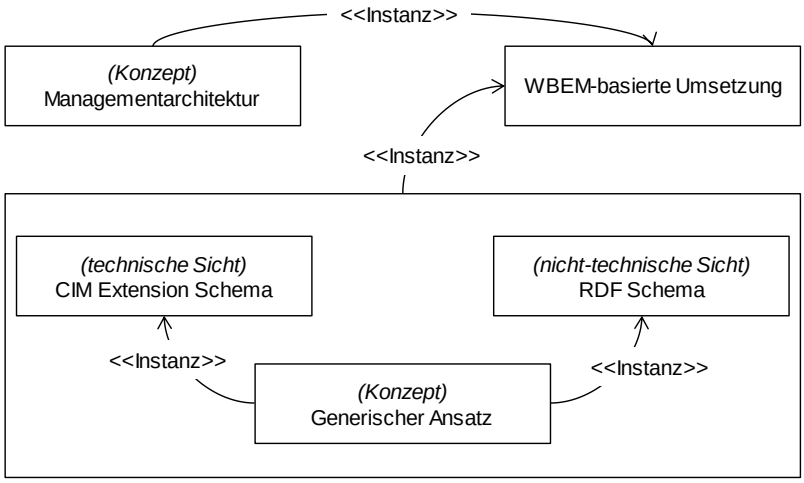

Abbildung 61 Beziehungen der einzelnen Aspekte der vorliegenden Arbeit

Im Rahmen eines prototypischen Demonstrators wurde die vorgestellte Managementarchitekur umgesetzt. Hierbei kam zum einen der entwickelte generische Ansatz in Form des CIM Erweiterungsschemas zum Einsatz, zum anderen wurde RDF dazu eingesetzt, strukturiert auf die in den CIM Servern vorhandene Managementinformation zuzugreifen. Durch die Überwachung von ausgewählten IT-Ressourcen des Beispielszenarios wurde das Konzept der Abhängigkeiten zwischen Attributwerten demonstriert.

8.2 Ausblick

Bei der Untersuchung der Fragestellung, wie Wechselbeziehungen zwischen Elementen einer IT-Infrastruktur erfasst werden können, wurde einige Aspekte bewusst einfach gehalten, um sich zum einen zunächst auf die wesentlichen Kernfragestellungen im Rahmen eines integrierten Managementansatzes konzentrieren zu können, zum anderen, um den Rahmen dieser Arbeit nicht zu sprengen.

Folgende weiterführenden Untersuchungen können direkt auf der vorliegenden Arbeit aufbauen:

- Die Umsetzung der Korrelationskomponente wurde aus den oben genannten Gründen bewusst einfach gehalten, da die Fragestellungen zunächst in Richtung Informationsmodell gelenkt wurde. In [En01] ist ein Ansatz beschrieben, mittels neuronaler Netze Zusammenhänge im Muster von Werteänderungen feststellen zu können. Dieser Ansatz könnte bei der Korrelationskomponente dazu eingesetzt werden, sehr viel präziser Abhängigkeiten zwischen Attributwerten zu finden, als dies mit dem einfachen Ansatz über den Vergleich von Werteänderungen von Attributwerten zu zwei Zeitpunkten t_0 und t_1 geschieht.
- Durch den Ansatz, den Begriff Abhängigkeit im Kontext von Laufzeitumgebungen als eigenständiges *Managed Object* zur Verfügung zu stellen, entstehen vielfältige Möglichkeiten, Abhängigkeiten in das aktive Management zu übernehmen. Dadurch kann dem *Managed Object* "Abhängigkeit" Funktionalität zugeordnet werden, die eigentlich in Anwendungslogik implementiert ist, beispielsweise werden Schwellwerte für Messungen an Infrastrukturressourcen heutzutage direkt den betreffenden Metriken der Infrastrukturelemente zugeordnet, während ein *Managed Object* "Abhängigkeit" um Funktionalität erweitert werden kann, das nicht nur den einzelnen Messwert und dessen Schwellwert betrachtet, sondern auch davon abhängige Werte. Im Beispiel des Szenarios könnte die Abhängigkeit zwischen *CPULoad*-Attributwerten und MailQueueSize-Attributwerten als *Managed Object* dahingehend ausgelegt werden, das eine hohe *CPULoad* zwar einen Schwellwert verletzt, wenn die davon abhängige MailQueueSize jedoch im Rahmen der notwendigen Bedingen bleibt, um keine SLA-Verletzungen wahrscheinlich zu machen, muss auch kein Alarm ausgelöst werden.
- In [KBK00, KK01] wurde versucht, Abhängigkeiten anhand deren Charakteristika zu Klassifizieren. Wie in 3.1 festgestellt wurde, ist jedoch nicht offensichtlich, woraus sich diese Charakteristika motivieren. Mit dem generischen Ansatz und dem erlangten Verständnis über die Wechselbeziehungen zwischen IT-Ressourcen während der Produktion einer IT-Dienstleistung, kann auf dieser Arbeit aufbauend ein Versuch unternommen werden, Abhängigkeiten im Kontext von Laufzeitumgebungen zu charakterisieren und letztlich so dem Ziel, Abhängigkeiten als *Managed Objects* vollständig in das Management zu integrieren, näher zu kommen.

ANHÄNGE

Abkürzungen und Glossar

Abkürzung oder Begriff	Langbezeichnung und/oder Begriffserklärung
Abhängigkeit	Eine Abhängigkeit bezeichnet eine besondere Beziehung zwischen zwei oder mehreren Elementen eines Modells, bei der die besondere Beziehung aufgrund von Domänen-spezifischen Wissens gegeben ist. So ist dieser Begriff in unterschiedlichen Problemdomänen unterschiedlich belegt. Im Rahmen dieser Arbeit werden Wechselbeziehungen zwischen IT-Ressourcen während der Produktion eines IT-Dienstes untersucht, weshalb der Begriff Abhängigkeit mit der Dynamik während der Produktion des IT-Dienstes verbunden ist und sich auf die Änderung von Zuständen (Änderung von Attributwerten) von IT-Ressourcen (Entitäten und deren Attribute) bezieht.
ATIS	*Abteilung technische Infrastruktur* Einrichtung der *Universität Karlsruhe* zur Betreuung der IT-Infrastruktur der Informatik-Institute
Attribut	Ein Attribut bezeichnet eine Merkmal (Charakteristik) einer Entität.
Attributwert	Ein Attributwert bezeichnet eine konkrete Ausprägung eines Attributs während eines bestimmten Zeitpunktes.
BDIM	*Business-Driven IT Management* Managementansatz, bei dem der Fokus auf die Geschäftsprozesse, und damit auf die Notwendigkeit, den Einfluss von Ressourcen auf den oder die entsprechenden Geschäftsprozesse sichtbar zu machen, gelegt wird.
C&M	*Cooperation* & *Management* Name der an der Universität Karlsruhe (TH) angesiedelten Forschungsgruppe.
CIM	*Common Information Model* Standard der *DMTF*. Bezeichnet ein Hersteller- und Technologie-unabhängiges Informationsmodell als Grundlage für eine verteilte Managamentarchitektur nach *WBEM*
DCML	*Data Center Markup Language* Standard der *OASIS*. Bezeichnet eine Auszeichnungssprache zur Integration verschiedener Managementansätze und zum Austausch von Managementinformation über Anwendungsgrenzen hinweg.
Dienstleister	Ein Dienstleister ist diejenige Instanz, die das immaterielle Gut IT-Dienst produziert und an einen Dienstnutzer / Dienstkonsument ausliefert.
Dienstleistungs-vereinbarungen	Eine Dienstleistungsvereinbarung (DLV) ist eine vertragliche Fixierung zwischen einem Dienstnehmer/Dienstnutzer
DMTF	Die *Distributed Management Task Force* ist ein Zusammenschluss mehrerer führender IT-Unternehmen, um einheitliche

	Managementstandards bezüglich Infrastrukturmanagement zu schaffen. Unter anderem betreut die DMTF die Entwicklung von CIM und WBEM.
Entität	Eine Entität ist ein eindeutig bestimmbares Objekt innerhalb eines Modells dem bestimmte Information zugeordnet werden kann.
Entwurfszeit	Bezeichnet die Zeitspanne von der Planung über das Design, Entwurf und Implementierung bis hin zur Inbetriebnahme eines Systems. Die Entwurfszeit wird im Bezug auf die Produktion einer IT-Dienstleistung mit einem statischen Aspekt in Verbindung gebracht.
Exim	*Exim* ist ein Mail-Transfer-Agent (MTA) zum Transport von E-Mail im Internet (und in lokalen Netzen).
FCAPS	Abkürzung der englischen Bezeichnungen der fünf Management-Funktionen. FCAPS steht für Failure-, Configuration, Accounting-, Performance-, und Security Management.
Funktionaler Zusammenhang	
Funktionsmodell	Das Funktionsmodell als Teilmodell der generischen Management-Architektur definiert die funktionalen Aspekte, die die einzelnen Komponenten der Architektur umsetzten.
Informationsmodell	Das Informationsmodell als zentrale Komponente einer Management-Architektur beschreibt den Aufbau der Modellelemente zur Abbildung der Realität (Infrastrukturelemente) in das Modell (Managed Objects).
IT-Dienst	Ein Dienst (Dienstleistung) ist das immaterielle Gegenstück zur Produktion einer Ware. Ein IT-Dienst ist dementsprechend ein Dienst, der von den IT-Ressourcen einer IT-Infrastruktur produziert wird. IT-Dienste lassen sich nach [Ol06] wie folgt beschreiben: *(1) Immateriell* *(2) Unteilbar* (3) *Zeitlich begrenzt* *(4) Individuell* *(5) Standortbezogen* (6) *Nicht zurückrufbar* Der IT-Dienst stellt aus Sicht eines Kunden den Kontaktpunkt zum Dienstproduzenten dar, während der IT-Dienstleister darüber gehend hinaus auch die technischen Komponenten zur Realisierung dieses Dienstes im Blickpunkt hat. Im Rahmen dieser Arbeit wird der Begriff IT-Dienst mit der Sicht des Dienstleisters verbunden und rückt daher die technische Ausprägung eines Dienstes in den Vordergrund (technische Parameter). IT-Dienste werden dabei von Netz-basierten Systemprozessen erbracht.
ITIL	Die Information Technology Infrastructure Library ist eine Sammlung von Dokumenten, die Lösungsansätze zur Bereitstellung von Dienstleistungen und den damit verbundenen Managementprozessen beschreibt. Die Beschreibung ist dabei weniger formal als vielmehr an sog. *best practices* orientiert, also an Lösungsansätzen, die sich in der Vergangenheit bewährt haben.
IT-Infrastruktur	Im eigentlichen Sinne bezeichnet IT-Infrastruktur „die baulich-technischen

	Gegebenheiten wie Gebäude, Räume und Schutzschränke, die für den IT-Einsatz benötigt werden“ [BSI]. Im weiteren Sinne gehören hierzu auch die vernetzten Komponenten, die darauf laufenden Systemprozesse und Anwendungen wie auch die Verkabelung selbst dazu.
IT-Management	Das IT-Management umfasst alle Maßnahmen, die für einen unternehmenszielorientierten, effektiven und effizienten Betrieb eines verteilten IT-Systems und seiner Ressourcen erforderlich ist [HAN99].
IT-Ressource	Eine IT-Ressource ist eine virtuelle (z.B.: ein Systemprozess) oder physikalische (z.B.: eine CPU) Komponente, die als Betriebsmittel Teil einer IT-Infrastruktur ist und damit in die Produktion einer IT-Dienstleistung eingebunden wird.
ITSM	Das IT Service Management beschreibt die Verfahren und Prozesse, die nötig sind, um das Management von IT-Diensten nachvollziehbar zu ermöglichen. Als prozessorientiertes Rahmenwerk ist die ITIL Standard im Bereich der IT-Dienstleister.
Kommunikations-Modell	Das Kommunikationsmodell beschreibt als Teilmodell die Komunikationsbeziehungen, Protokolle und Abläufe der einzelnen Komponenten einer verteilten Managementarchitektur.
Laufzeit	Bezeichnet die Zeitspanne, in der ein System aktiv an der Produktion von IT-basierten IT-Diensten teilnimmt, oder in Bereitschaft für die Produktion steht. Die Laufzeit wird dagegen mit einem dynamischen Aspekt in Verbindung gebracht
Metrik	Eine Metrik bezeichnet eine (atomare oder aggregierte) Maßzahl, um eine quantifizierbare Charakterisk beschreiben zu können.
Modell	Ein Modell ist die vereinfachte Sicht der Realität. Ein Modell beschreibt demnach ein Abbild von realen Gegenständen und/oder Konzepten. Um Modelle auf formale Korrektheit hin überprüfen zu können, müssen Regeln vereinbart werden, die die Elemente eines Modells definieren. Diese Sammlung von Regeln bezüglich eines Modells heißen Meta-Modell.
MOF	Das *Managed Object Format* ist die Definition einer kontextfreien Grammatik zur Beschreibung der CIM Modellierungselemente
Organisations-Modell	Das Organisationsmodell beschreibt als Teilmodell der generischen Managementarchitektur den Aufbau und die Verteilung der Rollen der unterschiedlichen Komponenten in einem verteilten Managementansatz.
RDF	Das *Resource Description Framework* ist ein Standard des W3C, um die Beziehung zwischen (Web-)Ressourcen (=Information) untereinander zu beschreiben. Hierdurch kann die Bedeutung einer Information in Bezug auf einen Wissenskontext beschrieben werden.
Relation	Eine Relation ist eine (statische) Beziehung, die aus dem Entwurf/Implementierung eines Infrastrukturelements bekannt ist und daher einen statischen Charakter hat.
Service Level	Das Service Level Management umfasst diejenigen Verfahren und

Management	Teilprozesse, um die zugesicherte Dienstgüte eines IT-Dienstes an der Schnittstelle einzuhalten.
Service Livecycle	Bezeichnet den Lebenszyklus von der Planung über den Entwurf/Implementierung und der Überführung in den laufenden Betrieb bis hin zur Verbesserung und damit wieder in die Planung hinein.
SLA	Siehe *Dienstleistungsvereinbarung*
SLM	Siehe *Service Level Management*
SOA	*Service Orientierte Architektur* Ansatz für Design, Entwurf und Management einer Systemarchitektur, die aus schwach gekoppelten Komponenten besteht.
W3C	*World Wide Web Consortium* Das W3C steht als treibende Kraft hinter dem Entwurf und der Standardisierung der World Wide Web Protokolle.
WBEM	*Web-Based Enterprise Management* Standard der *DMTF*. Bezeichnet einen verteilten Managementansatz für das Management von IT-Infrastrukturen. Grundlage einer WBEM-Managementarchitektur ist das Informationsmodell *CIM*
XML	Die Extensible Markup Language ist ein Sprachstandard des W3C, um die Struktur und den Aufbau von Daten und Dokumenten zu beschreiben, um darauf strukturiert zugreifen zu können.
XSLT	Extensible Stylesheet Language Transformation Turing-vollständige Sprache zur Umformung von XML Dokumenten.
Zusammenhang	Ein Zusammenhang zwischen zwei oder mehr Komponenten (einer IT-Infrastruktur) ergibt sich dann, wenn eine besondere Beziehung zwischen diesen Komponenten in der Art und Weise besteht, das diese besondere Beziehung anhand von formal nachvollziehbaren Gegebenheiten eingegrenzt werden kann. Prinzipiell können diese Beziehungen am Zeitpunkt ihres Auftretens innerhalb einer IT-Infrastruktur festgemacht werden: Zur Entwurfszeit des Systems (oder einer Komponente) und zur Laufzeit des Systems (oder einer Komponente): (1) Entwurfszeit - statischer Zusammenhang: zur Entwurfszeit bekannt; Vermutung, das sich zur Laufzeit eine Beziehung einstellt, die detaillierter charakterisiert werden kann. Bsp.: Funktionsaufruf, Kontrollflußbedingungen (2) Laufzeit - dynamischer Zusammenhang: eine sich zur Laufzeit manifestierende (erkennbare, feststellbare) Beziehung zweier oder mehr Komponenten;

Abbildungsverzeichnis

Abbildung 1 Service als Filterkriterium 9
Abbildung 2 Motivation für den Ansatz 10
Abbildung 3 Betreibersicht auf den E-Mail Dienst (Übersicht) 13
Abbildung 4 Vereinfachtes Szenario 13
Abbildung 5 Unterteilung der Managementebenen 17
Abbildung 6 SLA als Vereinbarungsgegenstand zwischen Service User und Service Provider 20

Abbildung 7 Aggregation der Metriken über Managementebenen hinweg 21
Abbildung 8 Übersicht CIM 25
Abbildung 9 Übersicht über das CIM Meta-Schema (nach [DMTF05a]) 26
Abbildung 10 Übersicht über den Kernaufbau des Core Models (nach [DMTF00]) 27
Abbildung 11 Übersicht der definierten Dependency-Klassen im Core Modell 28
Abbildung 12 CIM Metrics Model [DMTF03,DMTF06] 29
Abbildung 13 Server-CPU Beispiel in XML 31
Abbildung 14 Server-CPU Beispiel mit Modellierung der besteht-aus Beziehung 31
Abbildung 15 RDF-Graph 32
Abbildung 16 Server-CPU Beispiel als RDF Graph 33
Abbildung 17 XML-Kodierung des RDF-Beispiels 33
Abbildung 18 Syntaktisch korrekter, semantisch falscher Einsatz der *contains*-Beziehung 34
Abbildung 19 Graphische Darstellung des RDF-Schemas der *contains*-Beziehung 35
Abbildung 20 Teilmodelle einer Managementarchitektur nach [Ga07] 36
Abbildung 21 WBEM Architektur nach [DMTF] 37
Abbildung 22 Abhängigkeitsmodelle am Service-Lifecycle orientiert 39
Abbildung 23 Grafische Darstellung von Entitäten und Dependencies 44
Abbildung 24 Gerichtete Entity/Dependency 44
Abbildung 25 Erweiterung von *CIM_Dependency* 47
Abbildung 26 Beispielhafter Austausch von Information mittels DCML nach [DCMLa] 49
Abbildung 27 Grafische Repräsentation der Modellierung einer CIM-Klasse in DCML 50
Abbildung 28 RDF/XML Namespace Deklaration für das Beispiel 51
Abbildung 29 RDF/XML Notation der Klasse CIMOperatingSystem 51
Abbildung 30 RDF/XML Notation der Klasse LogicalServer 51
Abbildung 31 Vereinfachtes Szenario 54
Abbildung 32 Auslastungen von CPU-Last und Mail-Warteschlangen 54
Abbildung 33 Motivation und Vorgehen zur Definition des generischen Ansatzes 57
Abbildung 34 generisches Infrastrukturmodell 58
Abbildung 35 Konzept Entität, Attribut und Attributwert in RDF-Graphennotation 59
Abbildung 36 Meta-Modell zum Infrastrukturmodell 59
Abbildung 37 vollständiger generischer Ansatz 61
Abbildung 38 Anforderungsverlauf an den Mail-Teildienst "E-Mail-Senden" 62
Abbildung 39 Abhängigkeiten über Fehlerzustände nach [DLS05] 63
Abbildung 40 Technique Mapping durch Formulierung eines extension schemas 65
Abbildung 41 vereinfachtes Szenario 67
Abbildung 42 Trennung der beiden Ansätze 68
Abbildung 43 CIM Model des Systemprozesses exim 70
Abbildung 44 CIM Modell des exims mit Modellierung des Service Access Points 71
Abbildung 45 Erweiterung durch das Extension Schema des generischen Ansatzes 71
Abbildung 46 Modell mit Relation und Abhängigkeit 72
Abbildung 47 Notwendige Schritte zum erzeugen von RDF Ressourcen aus CIM Instanzen ... 73
Abbildung 48 RDF Schema für den generischen Ansatz als RDF Graph 74
Abbildung 49 Informationsmodell als zentrales Mittel zur Integration verschiedener Systeme. 76
Abbildung 50 Funktionale Anforderungen und beteiligte Akteure 78
Abbildung 51 Grundlegender organisatorischer Aufbau der Referenzarchitektur 80
Abbildung 52 Organisationsmodell der Referenzarchitektur 81
Abbildung 53 Komponenten eines Management Nodes mit CIM-relevantem Teil 82
Abbildung 54 Ablauf des einfachen Korrelationsalgorithmus 83
Abbildung 55 Komponenten eines *Managed Nodes* mit CIM-relevantem Teil 84
Abbildung 56 Komponentenmodell des Managed Object Directory 85
Abbildung 57 Elemente des Szenarios mit Modellelementen des generischen Ansatzes 88
Abbildung 58 Codeauszug aus der Komponente `Provider` 90
Abbildung 59 Codeauszug der Fabrikmethode `getEntityModel()` 91
Abbildung 60 Verlauf der Attributwerte im Lastszenario 93
Abbildung 61 Beziehungen der einzelnen Aspekte der vorliegenden Arbeit 96

Literaturanalysen

[MHG+07]	Definition Of Metric Dependencies For Monitoring The Impact Of Quality Of Services On Quality Of Processes

Inhalte
Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) es soll die Auswirkung von IT-Services auf die Qualität von IT-Prozessen untersucht werden. Dazu wird ein formaler Zusammenhang abgeleitet, der von der funktionalen Abhängigkeit *IT-Prozess - IT-Service* auf die funktionale Abhängigkeit der beteiligten Metriken leitet.

Defizite
Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) BDIM stellt den IT-Management in den Fokus von Geschäftszielen. Die Maskierung von funktionalen Einheiten hinter dem Begriff IT-Service führt zu der Notwendigkeit, diese Services unter dem Aspekt BDIM überwachen zu können. Eine Definition des Begriffes *Metric Dependency* fehlt bisher.

Prämissen
Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) SOA, CIM, QoS-UML

Lösungen
Was sind die eigenen Lösungen?

(L1) Formales Metric-Dependency Model

(L2) Monitoring Architektur

(L3) Erweiterung des QoS-UML-Profils

Nachweise
Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) Der eingeführte Begriff *metric dependency* wird formal abgeleitet.

(N2) In einem Demonstrator wird der ToR-Process bestehend aus 3 Webservices instrumentiert und das *metric dependency model* wird evaluiert

Offene Fragen
Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) Der Demonstrator sieht keine Parallelen Anfragen vor.

(O2) Die funktionalen Zusammenhänge müssen klar und bekannt sein, um die Metrik Abhängigkeiten darstellen zu können. Hieraus lässt sich aber die Motivation ableiten, wie dieser

Zusammenhang auf die Ressourcenebene abgebildet werden kann, wo zum einen der Zusammenhang nicht immer bekannt ist, zum anderen die Zusammenhänge hinreichend komplex werden, um diese nicht mehr formulieren zu können.

[KK01]	Determining Service Dependencies in Distributed Systems

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) Die Erfassung von Abhängigkeiten wird im Hinblick auf voranschreitendes Outsourcing von IT-Diensten immer wichtiger. Um ein durchgehendes End-to-End Fehlermanagement erreichen zu können, ist es wichtig, sowohl die Abhängigkeiten zwischen IT-Diensten im Kontext eines Geschäftsprozesses zu kennen, wie auch die Abhängigkeiten der diensterbringenden IT-Ressourcen zu erfassen. Schließlich will man nicht nur die Symptome eines Problems bekämpfen (Incident Management), sondern vielmehr die Ursache lösen, um weiteres Fehlverhalten auszuschließen (root cause analysis). Abhängigkeiten werden weder von IT-Ressourcen explizit formuliert, noch werden Mechanismen von Seiten der Managementschnittstellen angeboten, diese Abhängigkeiten zu erfassen.

(I2) Unterschiedliche Managementsysteme existieren und verwalten Information über Komponenten, aus denen Abhängigkeitsinformation generiert werden kann. Ein generischer Ansatz, der sowohl unterschiedliche Abhängigkeitsmodelle integriert, wie auch dynamische Abhängigkeiten, die sich erst zur Laufzeit ergeben, ist daher von besonderem Interesse.

(I3) Es werden unterschiedliche Methoden aufgeführt, wie Information generiert werden, die Abhängigkeitsinformation generiert:

a) Erweitern der Anwendungen und IT-Dienste um Instrumentierungspunkte; jedoch verfügt keine Anwendung bis zum derzeitigen Stand über eine generische Schnittstelle, um sowohl funktionale, wie auch strukturelle als auch operationelle Abhängigkeitsinformationen generieren zu können.
b) Instrumentierung des Protokollstapels und dynamischer Bibliotheken
c) Verwendung von Information aus Softwarerepositories; die Nachteile hierbei wurden schon in der Analyse von [KBK00] gelistet.
d) Aktives Einschleusen von Fehlern in Komponenten, um durch die Beobachtung abhängige Komponenten identifizieren zu können.
e) Neuronale Netze, die mit Ereignissen trainiert werden und anhand von Mustern auf Abhängigkeiten schließen lassen.
f) Im *CIM* wird die Klasse *CIM_Dependency* definiert, womit sich Abhängigkeiten modellieren lassen. Wie in [KBK00] bemerkt, können diese Abhängigkeiten jedoch keinen dynamischen Aspekt wiedergeben und sind auf die Modellierung zur Entwurfszeit beschränkt.

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) Die Arbeiten im Bereich Systems-Management konzentrierten sich hauptsächlich auf den Bereich der Eventkorrelation. Die Beschreibungen hierzu liegen jedoch zumeist in einem proprietären Format vor, da sich deren Nutzen zumeist auf die Anwendung in einem einzelnen dedizierten Managementtool konzentrierte. Da aber gerade im Fehlermanagementbereich unterschiedliche Anwendungen basierend auf unterschiedlichen Managementmodellen

eingesetzt werden, ist es notwendig, eine anwendungsübergreifende Beschreibung von Abhängigkeiten anzustreben.

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) Es wird eine Abgrenzung zu Abhängigkeiten gemacht, die sich aus dem internen Wissen über den Aufbau von Softwarekomponenten ergeben. Es wird unterschieden zwischen application debugging, also jenem Prozess, Information aus dem internen Verhalten einer Anwendung zu ziehen, und Service Level Management, also jenen Beobachtungen, die sich an der (funktionalen) Schnittstelle zu einer Anwendung ergeben.

(P2) Es werden Abhängigkeiten innerhalb des Servicelevel untersucht, also nur zwischen einem vorangehenden (antecedent) Service und einem abhängigen (dependend) Service.

Lösungen

Was sind die eigenen Lösungen?

(L1) In Erweiterung zu [KBK00] lässt sich die Aufteilung der unterschiedlichen Abhängigkeitsmodelle um ein operationelles Modell erweitern. Dem funktionalen Modell kommt die gleiche Aufgabe zu wie in [KBK00], während das strukturelle Modell nicht mehr den genauen Zustand eines Dienstes zur Laufzeit wiedergibt, sondern den Zustand des Systems nach der Installation der einzelnen Komponenten. Das operationelle Modell schließlich nimmt die Details der installierten Komponenten zur Laufzeit auf und trägt somit detailliertes Wissen der Interna der eingesetzten Softwarekomponenten.

(L2) Es wird eine dreigeteilte Architektur vorgestellt, in deren Mittelpunkt XML-Beschreibungen von (managed) IT-Ressourcen stehen. Die Architektur besteht aus den Komponenten *Management System, Management Services* und *Managed Resources.*
Das Management System kann hierbei generisch betrachtet werden, da die XML-Beschreibungen durch eine *Dependency Query Facility* an das Managment System gekoppelt werden. Die XML Beschreibungen werden durch RDF realisiert.

Nachweise

Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) In einer prototypischen Implementierung wird eine vereinfachte typische e-commerce Umgebung bestehend aus Applikationsserver und Datenbankservern. Im ersten Schritt wurden die bestehenden Softwarerepositories der eingesetzen Systeme (AIX und WindowsNT) abgefragt, um daraus XML-Dokumente der betrachteten Komponenten generieren zu können, um letztlich hiermit mittels XPath Abhängigkeitsinformation zu generieren.

Offene Fragen

Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) Wie sehen die konkreten Algorithmen aus, um die angesprochenen XML Dokumente zu generieren? Dies ist ein entscheidender Schwachpunkt, denn letztlich muss hier auch ein einheitliches Verfahren entworfen werden.

[CDS01]	Dependency Analysis Using Conceptual Graphs

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) Die Erfassung und Analyse von Abhängigkeiten, die sich zwischen Objekten einer IT-Infrastruktur ergeben, ist für viele Anwendungen von entscheidender Bedeutung. Hierbei müssen Abhängigkeiten betrachtet werden, die sich zwischen den unterschiedlichen Typen von IT-Ressourcen ergeben. Daher muss nach einer formalen Beschreibungsmöglichkeit gesucht werden, anhand derer sich Abhängigkeiten aus einer gegebenen Menge an Objekten erkennen lassen.

(I2) Die Visualisierung von Abhängigkeiten kann neben dem klassischen Darstellen durch gerichtete Knoten-/Kantengraphen auch konzeptuelle Graphen erfolgen. Dabei werden konkrete Objekte durch Rechtecke, und Konzepte, die diese Objekte verbinden, durch Ovale dargestellt. Die Leserichtung der Beziehung wird schließlich ebenfalls durch gerichtete Pfeile dargestellt.

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) Ein Großteil der bestehenden Literatur im Bereich Abhängigkeitsmanagement (und IT-Management im weiteren Sinne) nimmt den Begriff „Abhängigkeit“ als gegeben hin. Wenn in einer Publikation der Versuch unternommen wird, diesen Begriff zu definieren, geschieht dies zumeist unter dem Wirkbereich einer ausgewählten Problemdomäne, weshalb der Begriff Abhängigkeit im strengeren Sinne zwischen unterschiedlichen Lösungsansätzen nicht austauschbar ist, diese Lösungsansätze somit nicht vergleichbar sind.

(D2) Folgende Erklärungsansätze mit jeweiligen Defiziten für den Begriff Abhängigkeit werden aufgeführt:

a) UML: Repräsentation durch einen gestrichelten Pfeil; Nachteil: Abhängigkeit bezieht sich auf die Definition der Klassen und ist nicht für den Einsatz zur Laufzeitmodellierung gedacht.
b) OSD (OpenSoftwareDesciption): Ansatz aus dem Bereich der Softwareentwicklung, um Abhängigkeiten zwischen Softwarepaketen zu beschreiben. Dieser Ansatz ist in das strukturelle Modell in [KK01] einzuordnen und daher ebenfalls nicht dazu geeignet, Abhängigkeiten zur Laufzeit zu erfassen.
c) Natürliche Sprache: Untersucht Abhängigkeiten zwischen Wörtern in der natürlichen Sprache; Nachteil: auf den Bereich der Sprachforschung beschränkt.
d) Logik erster Ordnung, Logik höherer Ordnung, statistische/probabilistische Ansätze, Klient/Server-Beziehungen: Ansätze aus der Mathematik, die jedoch keine genaue Definition des Begriffes Abhängigkeit liefern.

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) Der Typenbasierte Ansatz muss noch ausgebaut werden, um alle wichtigen Typen von Abhängigkeiten erfassen zu können.

Lösungen

Was sind die eigenen Lösungen?

(L1) Definition der Begriffe Entität und Veränderung im Kontext von Infrastrukturmodellierung. Demnach werden Entitäten durch Attribute, und Abhängigkeiten

durch die Zustandsänderung von Entitäten (und damit implizit die Änderung von Attributwerten) erfasst.

(L2) Charakterisierung von Abhängigkeiten anhand folgender Attribute (siehe hierzu auch [KBK00, HSM+06]):

(a) Importance**:** In [KBK00] entspricht dies dem Attribut „criticality“. Es gibt an, wie wichtig diese Abhängigkeit für die abhängige Entität in Bezug auf Erfüllung ihrer Funktionalität ist.
Mögliche Werte: *not applicable, high, medium, low*

(b) Strength: definiert ein Maß für die Häufigkeit aus Sicht einer abhängigen Entität, wie oft diese Abhängigkeit innerhalb einer bestimmten Zeitperiode erfüllt sein muss. Obwohl dieses Attribut den gleichen Namen wie in [KBK00] trägt, kommt ihm hier eine andere semantische Bedeutung zu.
Mögliche Werte: *value/timeperiod*

(c) Sensitivity: wie stark ist diese Abhängigkeit anfällig für Fehlverhalten?
Mögliche Werte: *fragile, moderate, robust*

(d) Stability: gibt an, in welchen Zeitperioden diese Abhängigkeit anfällig für Fehlverhalten ist. Dieses Attribut entspricht in etwa dem Attribut „time“ in [KBK00]
Mögliche Werte: *extremly stable, infrequent, periodic, defined time*

(e) Need: Dies definiert ein höherwertiges semantisches Attribut, mit dem angegeben wird, welche Bedürfnisse die Vorgängerentität (antecedent) der abhängigen Entität (dependent) erfüllt.
Mögliche Werte: *beliebige Information*

(f) Impact: gibt an, welche Auswirkung eine Verletzung der Abhängigkeitserfüllung auf die funktionale Erbringung der abhängigen Entität diese Abhängigkeit hat.
Mögliche Werte: *none, mission compromised, performance degraded, information unreliable,. Corruption of information/communication*

(L3) Abhängigkeiten werden mithilfe von Konzeptgraphen dargestellt. Dies ist gleichzeitig ein wichtiges Hilfsmittel, um bei einer gegebenen Infrastruktur Abhängigkeiten anhand der Modellierung gemäß der Definition der Konzeptgraphen (Rechtecke entsprechen den Objekten, Ovale den verbindenden Konzepten) durchzuführen.

Nachweise

Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) Es wird kein Demonstrator vorgestellt, die Idee wird jedoch anhand zweier einfacher Beispiele in gedanklichen Experimenten durchgeführt.

Offene Fragen

Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) Wie werden die Abhängigkeiten in letzter Konsequenz über die Attribute definiert? [CDS01] geht zwar einen ersten Schritt in die Richtung, jedoch wird nicht vollständig begründet, wie die den Entitäten zugesprochenen Attribute im Kontext zum Konzept der Abhängigkeit stehen. Dieses Defizit führt schließlich zur Formulierung des im Rahmen der vorliegenden Arbeit entwickelten generischen Modells

[DAS+06]	The Dependency Management Framework: A Case Study of the ION CubeSat

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) Vorstellung des DMF (Dependency Management Framework) und Beschreibung von unterschiedlichen Anwendungsmöglichkeiten im Beispielszenario ION CubeSat

(I2) Hintergrund ist die Notwendigkeit an einem bestehenden Real Time System verteilte Komponenten entwickeln zu können, ohne jedoch den globalen View auf die Abhängigkeiten der Softwarekomponenten über Teamgrenzen hinweg zu haben. Zentrales Mittel hierzu ist die Idee, das jeder Entwickler/Team seine SW-Komponente annotiert (mithilfe der im Framework entwickelten Beschreibungssprache) um dann mittels Dependency-Queries Abhängigkeiten ausmachen zu können

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) Normalerweise wird dieser Prozess mittels Software Fault Tree (SFT) realisiert, nachteilig hierbei ist jedoch die Tatsache, das SFT manuell von Hand erstellt werden müssen

(D2) Baut auf einem eigenen früher vorgestellten Ansatz (Dependency Algebra) auf und erweitert diesen

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) Evaluation im Rahmen des studentischen ION CubeSat Projektes; SW-System mit starkem Real time Bezug; kein genereller Ansatz im Bereich IT-Management

Lösungen

Was sind die eigenen Lösungen?

(L1) Entwicklung eines Frameworks mit eigener Beschreibungs- und Abfragesyntax

Nachweise

Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) Es werden keine konkreten Beweise geliefert, diese sind im Vorgängerpaper Dependency Algebra zu suchen

Offene Fragen

Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) Wie werden die Annotation automatisiert entworfen? Existieren hier anwendbare Standards? Wo werden diese Annotation abgelegt? Wie lassen sich die Annotation wieder verwenden? Lassen sich die Annotation über Teamgrenzen hinweg austauschen?

(O2) Orientiert sich die Abfragesprache an einem Standard?

(O3) Lassen sich Teile des Ansatzes aus dem Bereich Real time isolieren und in einen generellen IT-Managementansatz einbinden?

Sonstiges

(S1) Grundlage für das DMF: Dependency Algebra - A tool for designing robust real-time systems. In Proceedings of the 26th IEEE Real-Time Systems Symposium

(S2) viele der Referenzen werden im Rahmen der Arbeit untersucht

[KBK00]	Classification and Computation of Dependencies for Distributed Management

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) Die Identifizierung von Abhängigkeiten wird immer wichtiger, weil Anwendungen und IT-Dienste auf einer Vielzahl unterstützender IT-Dienste basieren. Diese unterstützenden Dienste können in der Regel auch bei Outsourcing Dienstleistern untergebracht sein. Hierdurch wird die Fehlerbehandlung erschwert, oftmals ist nicht klar, welche Abhängigkeiten bestehen.

(I2) In der Regel werden Anwendungen ohne unterstützende Managamentschnittstellen ausgeliefert. Hierdurch steigt die Schwierigkeit, über Managementschnittstellen Abhängigkeiten erfassen zu können, dies gilt sowohl für Abhängigkeiten zwischen Anwendungen, wie auch für Abhängigkeiten zwischen Anwendungen und IT-Diensten, wodurch letztlich auch Abhängigkeiten zwischen IT-Diensten und deren unterstützende Anwendungen nicht erfasst werden können. Es wird somit nach einer Möglichkeit gesucht, Abhängigkeitsmodelle erstellen zu können, ohne den Quellcode von Anwendungen im Kontext von IP-basierten IT-Diensten verändern zu müssen. Hierzu werden Abhängigkeiten anhand deren Eigenschaften quantifiziert, um somit managementrelevante Information generieren zu können.

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) Die OpenGroup Distributed Software Administration (XDSA) spezifiziert Mechanismen, um Software zu verteilen, auszufahren und zu installieren. Es werden drei Typen von Abhängigkeiten in diesem Kontext erkannt: *prerequesite*, *exrequisite*, *corequisite*. Die XDSA betrachtet jedoch lediglich Entwurfsszenarien von Systemen und nicht konkrete Instanzen von Komponenten eines Systems, weshalb der Laufzeitaspekt ausgeblendet wird. Somit können keine Abhängigkeiten erfasst werden, die sich erst zur Laufzeit ergeben.

(D2) Im CIM besteht die Möglichkeit, Abhängigkeiten über Assoziationsklassen zwischen den CIM Klassen zu modellieren. Dabei kommen die Vorteile des Klassenmodell zum tragen wodurch Abhängigkeiten in eine Vererbungshierarchie eingegliedert werden. Jedoch besteht auch hier wiederum keine Möglichkeit, Abhängigkeiten zu erfassen, die sich zur Laufzeit ergeben. Desweiteren sind bis dato (2000) keine zufriedenstellende Managementapplikationen auf dem Markt verfügbar, die Abhängigkeiten erfassen und verwalten.

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) Es wird ein pragmatischer Ansatz vorgestellt, bei dem zwar keine Erweiterung der IT-Dienst unterstützenden Anwendungen hinsichtlich Managementinformation gemacht werden muss, der jedoch durch eine Klassifikation von Abhängigkeiten anhand deren Attribute vornimmt. Dazu muss ein statisches Wissen von Abhängigkeiten vorhanden sein, um die

unterschiedlichen Typen von Abhängigkeiten richtig einordnen zu können, um letztlich dadurch einen Mehrwert für die Gewinnung von Managementinformation zu erreichen.

(P2) Die Betrachtungen werden auf funktionale Beziehungen und damit auf Consumer/Provider Beziehungen reduziert.

Lösungen

Was sind die eigenen Lösungen?

(L1) Es werden zwei grundlegende Typen von Abhängigkeitsmodellen definiert

a) Functional model: definiert generelle Abhängigkeiten zwischen Diensten (Datenbankdienst, Namensdienst, Verzeichnisdienst, etc.)
b) Structural model: enthält die detailierten Informationen bzgl. Eines Aufbaus eines Dienstes und verfeinert daher das funktionale Modell dahingehend, das die generellen Abhängigkeiten im strukturellen Modell spezifiziert werden. Das strukturelle Modell bezieht sich somit auf die konkreten Instanzen von Anwendungen, die einen IT-Dienst erbringen.

(L2) Der pragmatische Ansatz verfolgt das Ziel Abhängigkeiten anhand deren Eigenschaften zu klassifizieren, um dadurch Managementinformation generieren zu können. Die Autoren definieren sechs unterschiedliche Typen von Attributen einer Abhängigkeit:

a) Space/Locality/Domain: gibt an, wie weit (nah) eine Komponente von deren abhängiger Komponente innerhalb eines Systems lokalisiert ist.
   Mögliche Werte: *inter-domain, intra-domain, inter-system, intra-system, intra-package*
b) Component Type: gibt an, welcher wirklicher Typ der Komponente zugeordnet werden kann
   Mögliche Werte: *hardware, software, logical entity*
c) Component Activity: gibt an, ob die Komponente aktiv instrumentiert werden kann (Hardware, Systemprozess, …) oder passiv ist und nicht direkt instrumentiert werden kann (Konfigurationsdatei)
   Mögliche Werte: *active, passive*
d) Dependency Strength: gibt an, wie stark eine Komponente von einer anderen abhängt.
   Mögliche Werte: *none, optional, mandatory*
e) Dependency Formalization: gibt an, welchen Abstraktionsgrad die Abhängigkeit inne hat
   Mögliche Werte: *low, high*
f) Dependency Criticality: gibt an, welche Anstrengungen unternommen werden müssen, um diese Abhängigkeit zu erfüllen
   Mögliche Werte: *prerequisite, co requisite, exrequisite*

Zusätzlich zu diesen sechs grundlegenden Eigenschaften lassen sich folgende Charakteristika von Abhängigkeiten untersuchen, die nicht direkt mit obigen Attributen in Verbindung stehen und somit allgemeiner Natur sind:

a) Time: Abhängigkeiten lassen sich einem bestimmten Zustand eines System zuordnen, welcher wiederum von der Zeit abhängt. Beispiele hierfür sind temporär benötigte Speicherkapazitäten während der Installation einer Softwarekomponente
b) Dependency Lifecycle: gibt den Lebenszyklus einer Abhängigkeit an. Es wird unterschieden zwischen funktionalen und strukturellen Abhängigkeiten. Dabei ändert sich dieser Typ im Laufe der Zeit von einer reinen funktionalen Abhängigkeit hin zu einer strukturellen mit detaillierter Statusinformation.

Nachweise

Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) Es werden keine Ergebnisse eines konkret implementierten Demonstrators vorgestellt, jedoch angeführt, das das funktionale Modell mit Information aus Software-Repositories gefüllt werden kann, und das strukturelle Modell durch Laufzeitinformation ergänzt werden kann, die im funktionalen Modell vorgeben wird. Es werden jedoch keine genauen Angaben gemacht, wie dieser Lösungsansatz in einem generischen Modell definiert werden kann. Außerdem wird angenommen, dass jedes Betriebssystem ein Softwarerepository bereithält, das vollständige funktionale Abhängigkeitsbeschreibungen beinhaltet. Da unterschiedliche Betriebssysteme jedoch unterschiedliche Paketverwaltungssysteme verwenden, ist es fraglich, ob ein System-übergreifendes Verfahren definiert werden kann.

Offene Fragen

Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) Abhängigkeiten werden eingeschränkt auf Consumer/Provider Beziehungen. Dies impliziert zwangsläufig, dass nur funktionale Abhängigkeiten untersucht werden, in der Publikation wird aber nicht darauf eingegangen, ob bzw. welche weiteren Eigenschaften von Abhängigkeiten ausgemacht werden können

(O2) Wie auch in anderen Arbeiten wird hier nicht versucht, den Begriff Abhängigkeit formal zu erfassen und zu beschreiben, vielmehr nähern sich die Autoren von einer analytischen Richtung, getrieben durch die Notwendigkeit, ausgelagerte IT-Dienste zu untersuchen. Gerade aber ein formales Verständnis für den Begriff Abhängigkeit ist von grundlegender Bedeutung.

(O3) Es werden keinerlei Anstrengungen unternommen, neben den Softwarekomponenten eines IT-Dienstes auch dessen zu Grunde liegende IT-Infrastrukturressourcen in ein Abhängigkeitsmodell zu integrieren. Vielmehr wird ausgesagt, dass dies zu kompliziert wird und mit den vorhandenen Methoden nicht tragbar sei. Ein formales Verständnis des Begriffes Abhängigkeit könnte aber genau hier ansetzten und versuchen, eine generelle Architektur zu formulieren.

(O4) Es wird keine Anstrengung unternommen, den Begriff Abhängigkeit zu formalisieren. Dies wäre jedoch Notwendig, um die Motivation für die definierten Attribute offen zu legen.

[HSM+06]	Specification of Dependencies for IT Service Fault Management

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) Um das Fehlermanagement im Servicelevel unterstützen zu können, muss die Fehlerbehandlung auf die den IT-Service unterstützende IT-Ressourcenschicht ausgedehnt werden – das Wissen um Abhängigkeiten ist hierbei von entscheidender Bedeutung.

(I2) Die im CIM vorhandenen Assoziationsklassen um Abhängigkeiten zu modellieren stammen alle von *CIM_Depenency* erlauben keine hierarchische Strukturierung der Abhängigkeiten. Durch eine Erweiterung in Form des Designpatterns „composite“ wird zwischen Inter-Resource, Inter-Service und Service-Resource Abhängigkeiten unterschieden und hierdurch die Möglichkeit erschaffen, einen hierarchischen Abhängigkeitsbaum zu erstellen.

(I3) Die Publikation entstammt aus dem Jahre 2006; die Autoren betrachten das Forschungsfeld der Erfassung und Modellierung von Abhängigkeiten bis dato als offenes Gebiet.

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) Bestehende Arbeiten aus dem Bereich Abhängigkeitsmanagement (und IT-Management im speziellen) definieren den Begriff „Abhängigkeit“ nicht oder nicht vollständig.

(D2) Das CIM kann zwar dafür eingesetzt werden, die Entitäten einer IT-Infrastruktur (IT-Ressourcen, IT-Services) zu modellieren, die Möglichkeiten Abhängigkeiten zu beschreiben sind jedoch beschränkt.

(D3) Das Shared Information/Data (SID) Modell der NGOSS führt ähnlich zu CIM einen Objekt-orientierten Ansatz in der Modellierung von Infrastrukturen ein. Die Arbeit befindet sich in einem frühen Entwicklungsstadium und konzentriert sich momentan auf die Definition der Kernaussagen. Die Entwicklung Richtung Abhängigkeitserfassung bleibt jedoch zu beobachten.

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) Die Notwendigkeit für eine präzise Erfassung der Abhängigkeiten einer Infrastruktur wird hauptsächlich im Fehlermanagement gesehen.

Lösungen

Was sind die eigenen Lösungen?

(L1) Zunächst wird eine Einordnung von Abhängigkeiten nach deren Attribute durchgeführt (siehe auch [KBK00, CDS01]):

(a) type of dependency: es wird unterschieden zwischen *inter-service*, *service-resource* und *inter-resource* Abhängigkeiten.
(b) level of detail: Dieser Aspekt richtet sich an Abhängigkeiten innerhalb einer Servicebeziehung und gibt an, ob ein Service als Gesamtes oder vielmehr einzelne Funktionalitäten einer Dienstleistung untersucht werden.
(c) redundancy: gibt an, ob die betrachtete Abhängigkeit isoliert untersucht werden kann oder Teil einer redundanten Infrastruktur ist und somit in eine gesamtheitliche Betrachtung mit weiteren Abhängigkeiten gestellt werden muss. Mögliche Ausprägungen hier sind *isolierte* und *zusammengesetzte* Abhängigkeiten
(d) dynamic: eine Abhängigkeit kann charakterisiert werden nach deren zeitlichen Verhalten. Daher kann zwischen *statischen* und *dynamischen* Abhängigkeiten unterschieden werden
(e) service lifecycle: gibt an, in welchem Lebensabschnitt eines IT-Dienstes diese Abhängigkeit auftaucht
(f) degree of formalization: gibt an, wie diese Abhängigkeit beschrieben werden kann. Beispiele hierfür sind *formal model*, *pseudocode* und *no formalization*.
(g) characteristic: Es wird erkannt, das Abhängigkeiten nicht nur *funktionaler*, sondern auch *organisatorischer* Natur sein können.

(L2) Das Softwaredesignpattern „composite“ wird eingesetzt, um die abstrakte Klasse CIM_*Dependency* in Richtung der Möglichkeit einer Hierarchiebildung zu erweitern. Dazu werden die Klassen *Inter_Resource_Dependency*, *Inter_Service_Dependency* und *Service_Resource_Dependency* mittels der Klasse *CompositeDependency* an die Klasse *CIM_Dependency* gekoppelt.

Nachweise

Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) keine

Offene Fragen

Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) Wie auch schon in [KBK00, CDS01] stellt sich die Frage, wodurch sich die definierten Attribute motivieren?

(O2) Wird durch den Einsatz von CIM hauptsächliche ein statischer Aspekt bedient? Kann der Ansatz hierdurch eventuell nur in das strukturelle Modell aus [KK01] eingeordnet werden?

[DLS05]	Dependency Algebra: A Theroretical Framework for Dependency Management in Real-Time Control Systems

Inhalte

Was sind die zentralen Inhalte (Themen, interessante Aussagen, Botschaften, Fragestellungen), die in der Arbeit (d.h., in der analysierten Literatur) behandelt werden?

(I1) Abhängigkeiten sind der Grund für Fehlverhalten in Softwarekomponenten

(I2) Eine Algebra kann aus der Erfassung von Fehlersemantiken abgeleitet werden

Defizite

Welche Defizite bestehender Arbeiten und Lösungen werden als Motivation der eigenen Lösungen genannt?

(D1) Abhängigkeiten werden ad hoc vorausgesetzt

(D2) Bekannte Ansätze gehen keinen formalen Weg

Prämissen

Welche Einschränkungen und Vorgaben werden hinsichtlich der eigenen Lösungen gemacht?

(P1) Real-Time Control Systems, Software Komponenten

(P2) Es werden nur Fehlerabhängigkeiten betrachtet; die Formalisierung geschieht mit dem Hintergrund von Fehlersemantiken

Lösungen

Was sind die eigenen Lösungen?

(L1) Erweiterung des Begriffes „Fehlersemantik“

(L2) Einführung einer Algebra, die auf der Idee beruht, das sich Fehlerabhängigkeiten dadurch bemessen lassen, dass alle mögliche Fehlerzustände zweier Komponenten verglichen werden und die Größe der Schnittmenge dieser Mengen als Indikator für die Gewichtung der Abhängigkeit definiert wird.

Nachweise

Welche Nachweise (*Evidence*) werden hinsichtlich der Tragfähigkeit der eigenen Lösungen geliefert?

(N1) Die Algebra wird auf ein Evaluationssystem namens Simplex abgebildet um daran theoretisch die Möglichkeiten aufzeigen zu können

Offene Fragen

Welche Fragen sind noch ungelöst geblieben bzw. stellen sich dem Leser?

(O1) es wird kein funktionierender Demonstrator beschrieben

(O2) es stellt sich die Frage, ob der Ansatz verallgemeinert werden kann, insbesondere um nicht nur Fehlerfortpflanzungsabhängigkeiten feststellen zu können

Literatur

[AGR+05] *Asaf Adi, Dagan Gilat, Royi Ronen, Ron Rothblum, Guy Sharon, Inna Skarbovsky*, Modeling and Monitoring Dynamic Dependency Environments, 2005
http://ieeexplore.ieee.org/iel5/10249/32669/01531256.pdf?arnumber=1531256

[ATIS07a] ATIS Dienstübersicht, 2007
http://www.atis.uka.de/808.php

[ATIS07b] ATIS Mitarbeiter, 2007
http://www.atis.uka.de/413.php

[BSI] *Bundesamt für Informationssicherheit*, BSI Schulung IT-Grundschutz – Glossar
https://ncc.uni-mannheim.de/bsi-webkurs/gsschul/gskurs/seiten/glossar/gloss_io.htm

[BSIa] *Bundesamt für Informationssicherheit*, BSI-Standard 100-1: Managementsysteme für Informationssicherheit (ISMS)
Version 1.0
http://www.bsi.de/literat/bsi_standard/standard_1001.pdf

[C&M-TA-AW] Cooperation & Management: C&M-ARBEITSWEISE, http://www.cm-tm.uka.de, "Teamarbeit bei C&M -> C&M-Arbeitsweise", Universität Karlsruhe (TH), C&M (Prof. Abeck).

[CA07] *CA*, The Unified Service Model, 2007
http://ca.com/de/press/release.aspx?cid=143008

[CDS01] *Lisa Cox, Dr. Harry S. Delugach*, Dependency Analysis Using Conceptual Graphs, 2000
http://www.brc2.com/techlib/CoxDelugachSkipper2001.pdf

[CFK05] *Karl Czajkowski, Ian Foster, Carl Kesselmann*, Agreement-Based Resource Management, 2005
http://ieeexplore.ieee.org/iel5/5/30407/01398016.pdf

[CSCI] *Department of Computer Sciene and Engineering – California State University San Bernardino*, UML Glossary
http://www.csci.csusb.edu/dick/samples/uml.glossary.html#dependency

[DAS+06] *Hui Ding, Leon Arber, Lui Sha, Marco Caccamo*, The Dependency Management Framework: A Case Study of the ION CubeSat, 2006
http://doi.ieeecomputersociety.org/10.1109/ECRTS.2006.28

[DCMLa] *Dcml Inc.* Data Center Markup Langauge Framework Specification, 2004
http://www.oasis-open.org/committees/download.php/10266/DCML%20Framework%20v13.pdf

[DFKI] The Frodo RDFSViz Tool
http://www.dfki.uni-kl.de/frodo/RDFSViz/

[DLS05] *Hui Ding, Kihwal Lee, Lui Sha*, Dependency Algebra: A Theoretical Framework for Dependency Management in Real-Time Control Systems, 2005
http://ieeexplore.ieee.org/iel5/9677/30561/01409900.pdf

[DK03] *Markus Debusmann, Alecanxer Keller*, SLA-Driven Management of Distributed Systems Using the Common Information Model, 2003
http://citeseer.ist.psu.edu/595934.html

[DMTF00] *Distributed Management Task Force*, Common Information Model – Core Model, 2000

http://www.dmtf.org/standards/published_documents/DSP0111.pdf

[DMTF03] *Distributed Management Task Force*, Common Information Model – Metrics Schema, 2003
http://www.dmtf.org/standards/documents/CIM/DSP0141.pdf

[DMTF03a] *Distributed Management Task Force*, Desktop Management InterfaceSpecification, 2003
http://www.dmtf.org/standards/documents/DMI/DSP0005.pdf

[DMTF05] *Distributed Management Task Force*, Common Information Model Infrastruktur Specification, 2005
http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf

[DMTF06] *Distributed Management Task Force*, CIM Metrics Schema V2.15, 2006
http://www.dmtf.org/standards/cim/cim_schema_v216/CIM_Metrics.pdf

[DMTF06a] *Distributed Management Task Force*, Web Services for Management (WS-Management), 2006
http://www.dmtf.org/standards/published_documents/DSP0226.pdf

[DMTF07] *Distributed Management Task Force*, Common Information Model v2.15, 2007
http://www.dmtf.org/standards/cim/cim_schema_v215/

[DMTF07a] *Distributed Management Task Force*, XML Document Type Definition (DTD), 2007
http://www.dmtf.org/standards/published_documents/DSP0203_2.3.0.dtd

[DMTF07b] *Distributed Management Task Force*, CIM Operations over HTTP, 2007
http://www.dmtf.org/standards/published_documents/DSP0200_1.3.0.pdf

[DNS] Domain Name System, RFC103, http://tools.ietf.org/html/rfc103

[DSB06] *Deutsches Statistisches Bundesamt*, Informationstechnologie in Unternehmen und Haushaltem 2006, 2006
https://www-ec.destatis.de/csp/shop/sfg/bpm.html.cms.cBroker.cls?cmspath=struktur,vollanzeige.csp&ID=1019869

[Du99] *Bob DuCharme*, XML – The Annotated Specification, 1999
ISBN 0-13-082676-6

[En01] *Christian Ensel*, New Approach for Automated Generation of Service Dependency Models, 2001
http://citeseer.ist.psu.edu/ensel01new.html

[En99] *Christian Ensel*, Automated generation of dependency models for service management, 1999
http://citeseer.ist.psu.edu/207869.html

[En01] *Christian Ensel*, New Approach for automated generation of service dependency models, 2001
http://citeseer.ist.psu.edu/cache/papers/cs/22972/http:zSzzSzwww.nm.informatik.uni-muenchen.dezSzLiteraturzSzMNMPubzSzPublikationenzSzense01azSzPostscript-VersionzSzense01a.pdf/ensel01new.pdf

[Ga07] *Jens-Uwe Gaspar*, Instrumentierung und Überwachung von serviceorientierten Architekturen, 2007

[GJ+06] *Zhen Guo, Guofei Jiang, Haifeng Chen, Kenji Yoshihira*, Tracking Probabilistic Correaltion of Monitoring Data for Fault Detection in Complex Systems, 2006
http://ieeexplore.ieee.org/iel5/10881/34248/01633515.pdf

[HAN99] *Heinz-Gerd Hegering ; Sebastian Abeck ; Bernhard Neumair*, Integriertes Management vernetzter Systeme, 1999
ISBN 3-932588-16-9

[Horde] Horde Groupware Webmail, http://www.horde.org/webmail

[HSM+06] *Andreas Hanemann, David Schmitz, Patricia Marcu, Martin Sailer,* Specification of Dependencies for IT service Fault Management, 2006
http://www.mnm-team.org/pub/Publikationen/hmss06/PDF-Version/hmss06.pdf

[Hu07] *Kai.Moritz Hüner*, Überwachung von prozessorientierten und anwendungsbasierten Diensten, 2007

[IMAP] Internet Mail Access Protocol, http://tools.ietf.org/html/rfc3501

[iSCSI] Internet Small Computer System Interface, http://tools.ietf.org/html/rfc3720

[ISO89] International Standard Organisation (ISO), Open Systems Interconnection – Basic Reference Model – Part 4: Management Framework, 1989

[ITIL] *Office of Governance Commerce*, Information Technology Infrastructure Library V2/V3, 2007
www.itil.org

[ITILa] *Office of Governance Commerce*, Information Technology Infrastructure Library – ITIL V3 Service Life Cycle
http://www.itil.org/de/itilv3-servicelifecycle/index.php

[Jä03] *Klaus Jähnke*, Management verteilter Systeme und Anwendungen mit dem Common Informations Model, Diplomarbeit, 2003
http://kj.uue.org/papers/cim/cim.html

[Jena] *Jena – A semantic Web Framework for Java*
http://jena.sourceforge.net/

[KBK00] *Alexander Keller, Uri Blumenthal, Gautam Kar,* Classification And Computation Of Dependencies For Distributed Management, 2000
http://ieeexplore.ieee.org/iel5/6942/18650/00860604.pdf?arnumber=860604

[KK01] *Alexander Keller, Gautam Kar,* Determining Service Dependencies in Distributed Systems, 2001
http://ieeexplore.ieee.org/iel5/7452/20279/00937026.pdf

[LDAP] Openldap, http://www.openldap.org/

[LHY06] *Ma Liangli, Wang Houxiang, Li Yongjie,* Using Component Metadat based on Dependency Relationships Matrix to improve the Testability of Component-based Software, 2006
http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=4221857&arnumber=4221860&count=85&index=2

[Li03] *Bixin Li,* Managing Dependencies in Component-Based Systems Based on Matrix Model, 2003
http://citeseer.ist.psu.edu/650086.html

[Me06] *Oliver Mehl,* Modellgetriebene Entwicklung managementfähiger Anwendungssysteme, 2006
ISBN 978-3-86624-248-7

[MHG+07] *Christian Mayerl, Kai Moritz Hüner, Jens-Uwe Gaspar, Christof Momm, Sebastian Abeck,* Definition of Metric Dependencies for Monitoring the Impact of Quality of Services

on Quality of Processes, 2007
http://www.mayerl.de/mayerl/public/quellen/2007bdim/monitoring_of_dependencies_between_qos_and_qop_07-03-21-huener.pdf

[MS+03] *Vijay Machiraju, Akhil Sahai, Aad van Moorsel,* Web Services Management Network: An Overlay Network for Federated Service Management, 2002
http://citeseer.ist.psu.edu/machiraju02web.html

[OASIS] Organization for the Advancement of Structured Information Standards
http://www.oasis-open.org/who/

[OCL06] *T.M.Ong, L.T. Chia, B.S. Lee,* A Repository Adapter for Resource Management, 2006
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1630881

[OMG06] *Object Management Group (OMG),* UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, 2006
http://www.omg.org/docs/formal/06-05-02.pdf

[OMG07] *Object Management Group (OMG),* OMG IDL Syntax and Semantics
ftp://ftp.omg.org/pub/docs/formal/02-06-07.pdf

[Openssh] OpenSSH, Freiverfügbare Secure Shell Implementierung
http://www.openssh.com

[Pa07] *Ingo Pansa,* Evaluation eines Nachfolgesystems für das System- und Netzwerkmanagement in der ATIS, 2007
http://www.atis.uka.de/download/2007_SA_Pansa.pdf

[POP] Post Office Protocol
http://tools.ietf.org/html/rfc1939

[QAW+04] *Stephen Quirolgico, Pedro Assis, Andrea Westerinen, Micahel Baskey, Ellen Stokes,* Toward a Formal Common Information Model Ontology, 2004
http://ieeexplore.ieee.org/iel5/8567/27121/01204674.pdf?arnumber=1204674

[Se00] *Bran Selic,* A Generic Framework for Modeling Resources with UML, 2000
http://ieeexplore.ieee.org/iel5/2/18367/00846320.pdf?arnumber=846320

[Ser07] *Serview – The Buissness IT-Alignment Consultants,* 2007
http://www.serview.de/content/unternehmen1/unternehmen/business_it_alignment/view

[SJS05] *Neeraj Sangal, Ev Jordan, Vineet Sinha, Daniel Jackson,* Using Dependency Models to Manage Complex Software Architecture, 2005
http://portal.acm.org/citation.cfm?id=1103845.1094824

[SM+02] *Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Li Jie Jin, Fabio Casati,* Automated SLA Monitoring for Web Services, 2002
http://citeseer.ist.psu.edu/sahai02automated.html

[SMS+06] *Jacques Sauvé, Antão Moura, Marcus Sampaio, João Jornada, Eduardo Radziuk,* An Introductory Overview and Survey of Business-Driven IT Management, 2006
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1649205

[SO+01] *Akhil Sahai, Jinsong Ouyang, Vijay Machiraju, Klaus Wurster,* Specifying and Guarenteeing Quality of Service for Web Services through Real Time Measurement and Adaptive Control, 2001
http://www.hpl.hp.com/techreports/2001/HPL-2001-134.html

[Sö06] *Thomas Söbing,* Handbuch Outsourcing : Recht, Strategien, Prozesse, IT, Steuern samt Business-Process- Outsourcing, 2006
ISBN 3-8114-3320-2
ISBN 978-3-8114-3320-5

[SS+03] *Mehmet Sayal, Akhil Sahai, Vijay Machiraju, Fabio Casati,* Semantic Analysis of E-Business Operations, 2002
http://www.hpl.hp.com/techreports/2002/HPL-2002-176.pdf

[VVB02] *Vergara, Vollagra, Berrocal,* Semantic Management: advantages of using an ontology based management information meta-model
management information meta-model
http://jungla.dit.upm.es/~jlopez/publicaciones/hpovua02.pdf

[RFC1157] *Internet Engineering Task Force (IETF),* A Simple Network Manageemnt Protocol, 1990
http://www.ietf.org/rfc/rfc1157.txt

[VWZ01] *A.de Vos, S.E. Widergren, J. Zhu,* XML for CIM Model Exchange, 2001
http://www.langdale.com.au/PICA/CIMXML.pdf

[W3C] *World Wide Web Consortium,* Resource Description Framework(RDF) Concepts and Abstract Syntax, 2004
http://www.w3.org/TR/rdf-concepts/

[W3Ca] *World Wide Web Consortium,* RDF/XML Syntax Specification(Revised), 2004
http://www.w3.org/TR/rdf-syntax-grammar/

[W3Cb] *World Wide Web Consortium,* RDF Vocabulary Description Language 1.0: RDF Schema, 2004
http://www.w3.org/TR/rdf-schema/

[W3Cc] *World Wide Web Consortium,* Extensible Markup Language (XML) 1.1 (Second Edition), 2006
http://www.w3.org/TR/2006/REC-xml11-20060816/

[W3Cd] *World Wide Web Consortium,* Web Ontology Language (OWL)
http://www.w3.org/2003/08/owlfaq.html

[WBEM] *WBEM Services,* Java Web Based Enterprise Management
http://wbemservices.sourceforge.net/

MOF-Datei der CIM-Extension

```
//===========================================================================
// GenericApproach mof file
// 11.12.07    initial draft
//===========================================================================

class GA_Entity : CIM_LogicalElement {
       [Key, Description(
              "GA weiter eindeutiger Identifier.")]
              String Identifier;
};

class GA_InstanceOfEntity : CIM_LogicalElement {
    [Key, Description(
        "GA weiter eindeutiger Identifier.")]
        String Identifier;
};

class GA_Attribute : CIM_LogicalElement{
    [Key, Description(
        "GA weiter eindeutiger Identifier.")]
        String Identifier;
       [Description(
              "Bezeichner dieses Attributes.")]
              String AttributeCaption;
       [Description(
              "Caption.")]
              String Caption;
};

class GA_AttributeValue : CIM_LogicalElement{
    [Key, Description(
        "GA weiter eindeutiger Identifier.")]
        String Identifier;
       [Description(
              "AttributeWert.")]
              String AttributeValue;

       [Description(
              "AttributeValueCaption.")]
              String Caption;
};

class GA_Relation : CIM_ManagedElement{
       [Key,Description("Partner1 für diese Relation")]
              String Partner1;
       [Key,Description("Partner2 für diese Relation")]
              String Partner2;
};

class GA_Dependency : CIM_ ManagedElement {
    [Key,Description("Abhängendes Element innerhal dieser Dep")]
        String Antecedent;
       [Key,Description("Abhängiges Element innerhal dieser Dep")]
        String Dependent;
};

// Association Classes

[Association]
class GA_LogicalElement_Entity {
       [Key,Description("Verweisst auf das LE")]
        CIM_LogicalElement REF LE;
       [Key,Description("Verweisst auf die Entity")]
        GA_Entity REF Entity;
};

[Association]
class GA_BaseMetricValue_AttributeValue {
    [Key,Description("Verweisst auf den BaseMetricValue aus dem MetricsModel")]
        CIM_BaseMetricValue REF BaseMetricValue;
       [Key,Description("Verweisst auf den AttributeValue aus dem generischen Ansatz")]
        GA_AttributeValue REF AttributeValue;
};
```

```
[Association]
class GA_AttributeValue_Attribute {
    [Key,Description("Verweisst auf den AttributeValue")]
        GA_AttributeValue REF AttributeValue;
       [Key,Description("Verweisst auf das Attribute")]
        GA_Attribute REF Attribute;
};

[Association]
class GA_MetricDefinition_Attribute {
    [Key,Description("Verweisst auf die MetricDefinition aus dem Metrics Model")]
        CIM_MetricDefinition REF MetricDefinition;
       [Key,Description("Verweisst auf das Attribute")]
        GA_Attribute REF Attribute;
};

[Association]
class GA_AttributeValue_IoE {
    [Key,Description("Verweisst auf den AttributeValue")]
        GA_AttributeValue REF AttributeValue;
       [Key,Description("Verweisst auf die Instanz einer Entität")]
        GA_InstanceOfEntity REF IoE;
};

[Association]
class GA_Attribute_Entity {
       [Key,Description("Verweisst auf das Attribute")]
        GA_Attribute REF Attribute;
       [Key,Description("Verweisst auf die Entity")]
        GA_Entity REF Entity;
};

[Association]
class GA_Entity_IoE {
       [Key,Description("Verweisst auf die Entity")]
        GA_Entity REF Entity;
       [Key,Description("Verweisst auf die Instanz einer Entity")]
        GA_InstanceOfEntity REF IoE;
};
```

RDF-Schema zum generischen Ansatz

```
<?xml version="1.0"?>
<rdf:RDF
xml:base="http://studwww.ira.uni-karlsruhe.de/~s_pansa/.public_html/GA_RDFS#"
xmlns:prefix1="http://studwww.ira.uni-karlsruhe.de/~s_pansa/.public_html/GA_RDFS_v02.rdfs#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
        <rdfs:Class rdf:ID="Entity">
                <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
        </rdfs:Class>
        <rdfs:Class rdf:ID="Attribute">
                <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
        </rdfs:Class>
        <rdfs:Class rdf:ID="AttributeValue">
                <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
        </rdfs:Class>
        <rdfs:Class rdf:ID="Instance">
                <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
        </rdfs:Class>
        <rdfs:Class rdf:ID="Relation">
                <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
        </rdfs:Class>
        <rdfs:Class rdf:ID="Dependency">
                <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource"/>
        </rdfs:Class>
        <rdf:Property rdf:ID="CaptionEntity">
                <rdfs:comment>Caption for the Entity</rdfs:comment>
                <rdfs:domain rdf:resource="#Entity"/>
                <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
        </rdf:Property>
        <rdf:Property rdf:ID="CaptionAttribute">
                <rdfs:comment>Caption for the Attribute</rdfs:comment>
                <rdfs:domain rdf:resource="#Attribute"/>
                <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
        </rdf:Property>
        <rdf:Property rdf:ID="Value">
                <rdfs:comment>Value for AttributeValue</rdfs:comment>
                <rdfs:domain rdf:resource="#AttributeValue"/>
                <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
        </rdf:Property>
        <rdf:Property rdf:ID="Instantiierung">
                <rdfs:comment>Beziehung zwischen Entity und Instance of Entity</rdfs:comment>
                <rdfs:domain rdf:resource="#Entity"/>
                <rdfs:range rdf:resource="#Instance"/>
        </rdf:Property>
        <rdf:Property rdf:ID="Consists">
                <rdfs:comment>Beziehung zwischen EntityType und Attribute</rdfs:comment>
                <rdfs:domain rdf:resource="#Entity"/>
                <rdfs:range rdf:resource="#Attribute"/>
        </rdf:Property>
        <rdf:Property rdf:ID="CurrentValue">
                <rdfs:comment>Beziehung zwischen Attribute und Attributwert</rdfs:comment>
                <rdfs:domain rdf:resource="#Attribute"/>
                <rdfs:range rdf:resource="#AttributeValue"/>
        </rdf:Property>
        <rdf:Property rdf:ID="rel_p1">
                <rdfs:comment>Partner 1 der Relation</rdfs:comment>
                <rdfs:domain rdf:resource="#Relation"/>
                <rdfs:range rdf:resource="#Attribute"/>
        </rdf:Property>
        <rdf:Property rdf:ID="rel_p2">
                <rdfs:comment>Partner 2 der Relation</rdfs:comment>
                <rdfs:domain rdf:resource="#Relation"/>
                <rdfs:range rdf:resource="#Attribute"/>
        </rdf:Property>
        <rdf:Property rdf:ID="attrPartRel">
                <rdfs:comment>Attribut ist Teil einer Relation</rdfs:comment>
                <rdfs:domain rdf:resource="#Attribute"/>
                <rdfs:range rdf:resource="#Relation"/>
        </rdf:Property>
        <rdf:Property rdf:ID="Antecedent">
                <rdfs:comment>Antecedent</rdfs:comment>
                <rdfs:domain rdf:resource="#Dependency"/>
                <rdfs:range rdf:resource="#AttributeValue"/>
        </rdf:Property>
```

```
        <rdf:Property rdf:ID="Dependent">
                <rdfs:comment>Dependent</rdfs:comment>
                <rdfs:domain rdf:resource="#Dependency"/>
                <rdfs:range rdf:resource="#AttributeValue"/>
        </rdf:Property>
        <rdf:Property rdf:ID="InstanceValue">
                <rdfs:comment>Dependent</rdfs:comment>
                <rdfs:domain rdf:resource="#Instance"/>
                <rdfs:range rdf:resource="#AttributeValue"/>
        </rdf:Property>
</rdf:RDF>
```

RDF-Schema zum Beispiel aus Abbildung 16

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:ex="http://example.org/Syntax#"
xmlns:prefix1="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <rdf:Description rdf:about="http://example.org/server1">
                <ex:contains>
                        <rdf:Description rdf:about="http://example.org/cpu1">
                                <rdf:type>
                                        <rdf:Description rdf:about="http://example.org/Syntax#Cpu_Schema"/>
                                </rdf:type>
                        </rdf:Description>
                </ex:contains>
                <rdf:type>
                        <rdf:Description rdf:about="http://example.org/Syntax#Server_schema"/>
                </rdf:type>
        </rdf:Description>
        <rdf:Description rdf:about="http://example.org/Syntax#contains_schema">
                <rdf:type>
                        <rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
                </rdf:type>
                <prefix1:domain>
                        <rdf:Description rdf:about="http://example.org/Syntax#Server_schema"/>
                </prefix1:domain>
                <prefix1:range>
                        <rdf:Description rdf:about="http://example.org/Syntax#Cpu_Schema"/>
                </prefix1:range>
        </rdf:Description>
        <rdf:Description rdf:about="http://example.org/Syntax#Cpu_Schema">
                <rdf:type>
                        <rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/>
                </rdf:type>
        </rdf:Description>
        <rdf:Description rdf:about="http://example.org/Syntax#Server_schema">
                <rdf:type>
                        <rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-schema#Class"/>
                </rdf:type>
        </rdf:Description>
</rdf:RDF>
```

Java-Quellcode der Komponente CIM2RDF

Klasse CIM2RDF

```
package mntNode;
import java.io.FileWriter;

public class CIM2RDF {
  public static void main(String[] args) throws Exception {
    MObjectsEnum moenum = null;
    int counter = 0;

    CIM2RDFFactory rdfFactory = null;

    if( args.length != 2 ) {
      System.out.println("usage: Tester SERVERADDR PORT");
      System.exit(1);
    }

    MNodesEnum enumerMNodes = new MNodesEnum( args[0], Integer.parseInt(args[1]));
    rdfFactory = new CIM2RDFFactory();

    while(enumerMNodes.hasMoreElements()) {
      moenum = new MObjectsEnum( enumerMNodes.nextElement() );

      while( moenum.hasMoreElements()) {
        CIM2RDFFactory.getModel(rdfFactory.getModel(),moenum.nextElement());
        counter ++;
      }

    }

    FileWriter fw = new FileWriter( "MyModel.rdf" );
    System.out.println( rdfFactory.getModel().write(fw, "RDF/XML"));
    System.out.println( " counted " + counter + " CIM Instanzen" );
  }

}
```

Klasse CIM2RDFFactory

```
package mntNode;

import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMProperty;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.vocabulary.RDF;
import com.hp.hpl.jena.vocabulary.RDFS;

public class CIM2RDFFactory {
  private Model myModel;
  private static String nsPrefix = "http://example.org/";

  public Model getModel() { return this.myModel; };

  public CIM2RDFFactory() {
    this.myModel = ModelFactory.createDefaultModel();
    this.myModel.setNsPrefix("gas", GASchema.getUri());
    this.myModel.setNsPrefix("ex", nsPrefix);
  }

  private static Model getEntityModel( Model model, CIMInstance inst ) {
    String value = cutQuotes(inst.getProperty("Identifier").getValue().toString());
    Resource entity = model.createResource( nsPrefix + inst.getClassName() + "."
        + "Identifier" +"=" + value, GASchema.Entity );
        entity.addProperty(GASchema.CaptionEntity,
                cutQuotes(inst.getProperty("name").getValue().toString()));
    return model;
  }

  private static Model getAttributeModel( Model model, CIMInstance inst ) {
    String value = cutQuotes(inst.getProperty("Identifier").getValue().toString());
    Resource attr = model.createResource( nsPrefix + inst.getClassName() + "."
```

```
        + "Identifier" +"="+ value, GASchema.Attribute );
      attr.addProperty(GASchema.CaptionAttribute,
        cutQuotes(inst.getProperty("Caption").getValue().toString()));
      return model;
    }

    private static Model getAttributeValueModel( Model model, CIMInstance inst ) {
      String value = cutQuotes(inst.getProperty("Identifier").getValue().toString());
      Resource attr = model.createResource( nsPrefix + inst.getClassName() + "."
        + "Identifier" +"="+ value, GASchema.AttributeValue );
      attr.addProperty(GASchema.Value,
        cutQuotes(inst.getProperty("AttributeValue").getValue().toString()));
      attr.addProperty(GASchema.Timestamp,
        cutQuotes(inst.getProperty("Timestamp").getValue().toString()));
      return model;
    }

    private static Model getRelationModel( Model model, CIMInstance inst ) {
      String partner1 =  cutSlashes(
                cutQuotes(inst.getProperty("Partner1").getValue().toString()));
      String partner2 = cutSlashes(
cutQuotes(inst.getProperty("Partner2").getValue().toString()));
      Resource partner1Res = model.getResource(nsPrefix+partner1);
      Resource partner2Res = model.getResource(nsPrefix+partner2);

      Resource rel = model.createResource( nsPrefix +
                inst.getClassName()+"-"+partner1+","+partner2, GASchema.Relation);

      rel.addProperty(GASchema.rel_p1, partner1Res );
      rel.addProperty(GASchema.rel_p2, partner2Res );

      partner1Res.addProperty(GASchema.attrPartRel, rel);
      partner2Res.addProperty(GASchema.attrPartRel, rel);

      return model;
    }

    private static Model getDependencyModel( Model model, CIMInstance inst ) {
      String partner1 = cutSlashes(
        cutQuotes(inst.getProperty("Antecedent").getValue().toString()));
      String partner2 = cutSlashes(cutQuotes(
        inst.getProperty("Dependent").getValue().toString()));
      Resource partner1Res = model.getResource(nsPrefix+partner1);
      Resource partner2Res = model.getResource(nsPrefix+partner2);

      Resource rel = model.createResource( nsPrefix + inst.getClassName()+"-"
        + partner1 + "," + partner2, GASchema.Dependency);

      rel.addProperty(GASchema.Antecedent, partner1Res );
      rel.addProperty(GASchema.Dependent, partner2Res );
      return model;
    }

    private static Model getAttributeEntityModel( Model model, CIMInstance inst ) {
      String domain = cutSlashes(
        cutQuotes(inst.getProperty("Entity").getValue().toString()));
      String range = cutSlashes(
        cutQuotes(inst.getProperty("Attribute").getValue().toString()));

      Resource domRes = model.getResource(nsPrefix+domain);
      Resource ranRes = model.getResource(nsPrefix+range);

      Property prop = model.createProperty(GASchema.getUri(),"Consists");
      domRes.addProperty(prop,ranRes);
      return model;
    }

    private static Model getAttributeValueAttributeModel( Model model, CIMInstance inst )
{
      String domain = cutSlashes(
        cutQuotes(inst.getProperty("Attribute").getValue().toString()));
      String range = cutSlashes(
        cutQuotes(inst.getProperty("AttributeValue").getValue().toString()));
      String sub = cutPlus(cutDot(range.substring(range.indexOf("=")+1)));

      Resource domRes = model.getResource(nsPrefix+domain);
      Resource ranRes = model.getResource(nsPrefix+range);
```

```
    Property prop = model.createProperty(GASchema.getUri(),"CurrentValue"+sub);
    domRes.addProperty(prop,ranRes);

    return model;
  }

  private static Model unimplemented( Model model, CIMInstance inst ) throws Exception {
    throw new Exception("unimplemented");
  }

  public static Model getModel( Model model, CIMInstance instance ) throws Exception {
    String cimClassName = instance.getClassName();
    if( cimClassName.equals( "GA_Entity" )){
      return CIM2RDFFactory.getEntityModel(model, instance);
    } else if( cimClassName.equals( "GA_Attribute" ) ) {
      return CIM2RDFFactory.getAttributeModel(model, instance);
    } else if( cimClassName.equals( "GA_AttributeValue" ) ) {
      return CIM2RDFFactory.getAttributeValueModel(model, instance);
    } else if( cimClassName.equals( "GA_Relation" ) ) {
      return CIM2RDFFactory.getRelationModel(model, instance);
    } else if( cimClassName.equals( "GA_Dependency" ) ) {
      return CIM2RDFFactory.getDependencyModel(model, instance);
    } else if( cimClassName.equals( "GA_InstanceOfEntity" ) ) {
      return CIM2RDFFactory.unimplemented(model, instance);
    } else if( cimClassName.equals( "GA_Attribute_Entity" ) ) {
      return CIM2RDFFactory.getAttributeEntityModel(model, instance);
    } else if( cimClassName.equals( "GA_AttributeValue_Attribute" ) ) {
      return CIM2RDFFactory.getAttributeValueAttributeModel(model, instance);
    } else if( cimClassName.equals( "GA_AttributeValue_IoE" ) ) {
      return CIM2RDFFactory.unimplemented(model, instance);
    } else if( cimClassName.equals( "GA_BaseMetricValue_AttributeValue" ) ) {
      return CIM2RDFFactory.unimplemented(model, instance);
    } else if( cimClassName.equals( "GA_Entity_IoE" ) ) {
      return CIM2RDFFactory.unimplemented(model, instance);
    } else if( cimClassName.equals( "GA_LogicalElement_Entity" ) ) {
      return CIM2RDFFactory.unimplemented(model, instance);
    } else if( cimClassName.equals( "GA_MetricDefinition_Attribute" ) ) {
      return CIM2RDFFactory.unimplemented(model, instance);
    }
    return null;
  }

  private static String cutQuotes(String in) { return in.substring(1, in.length()-1); }
  private static String cutDot( String in ) { return in.replace(".", "");}
  private static String cutPlus( String in ) { return in.replace("+", "");
  private static String cutSlashes(String in) {
    String s = in.replace("\"", "");
    s = s.replace("\\","");
    s = s.substring(s.indexOf(":")+1);
    return s;
  }
}
```

Klasse GASchema

```
package mntNode;

import com.hp.hpl.jena.rdf.model.Property;
import com.hp.hpl.jena.rdf.model.Resource;
import com.hp.hpl.jena.rdf.model.ResourceFactory;


public class GASchema {

  protected static final String uri
              = "http://studwww.ira.uni-karlsruhe.de/~s_pansa/GA_RDFS#";

  public static String getUri()
  { return uri; }


  protected static final Resource resource( String local )
  { return ResourceFactory.createResource( uri + local ); }

  protected static final Property property( String local )
  { return ResourceFactory.createProperty( uri, local ); }
```

```
  public static final Resource Entity = resource ( "Entity" );
  public static final Resource Attribute = resource( "Attribute");
  public static final Resource AttributeValue = resource( "AttributeValue");
  public static final Resource Instance = resource( "Instance" );
  public static final Resource Dependency = resource( "Dependency" );
  public static final Resource Relation = resource( "Relation" );

  public static final Property InstanceValue = property( "InstanceValue" );
  public static final Property CaptionEntity = property( "CaptionEntity" );
  public static final Property CaptionAttribute = property( "CaptionAttribute" );
  public static final Property Value = property( "Value" );
  public static final Property Timestamp = property( "Timestamp" );
  public static final Property Instantiierung = property( "Instantiierung" );
  public static final Property Consists = property( "Consists" );
  public static final String ConsistsURI = uri + "Consists/";
  public static final Property CurrentValue = property( "CurrentValue" );
  public static final Property rel_p1 = property( "rel_p1" );
  public static final Property rel_p2 = property( "rel_p2" );
  public static final Property attrPartRel = property( "attrPartRel" );
  public static final Property Antecedent = property( "Antecedent" );
  public static final Property Dependent = property( "Dependent" );
}
```

Klasse MNodesEnum

```
package mntNode;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.*;

import com.MOCommand;
import com.MOData;
import com.MOMessage;

/*
 * erstellt intern einen Vector, der eine Sammlung aller IP Adressen enthaelt,
 * die bei einem ManagedNodesDirectoy registriert sind
 */
public class MNodesEnum implements Enumeration<String> {

  private Vector<String> mnodesVec = new Vector<String>();
  private Enumeration<String> enu;

  public MNodesEnum( ) throws Exception {
    throw new Exception( "unsupported" );
  }

  public MNodesEnum( String serverAddr, int serverPort ) throws Exception {
    MOMessage msgReqEnum = new MOMessage( new MOCommand(MOCommand.ENUM), new MOData(  )
);
    MOMessage msgReply;
    String line = null;
    String addr = null;
    boolean exit = false;

    try {
      Socket socket = new Socket(serverAddr, serverPort);

      PrintStream outStream = new PrintStream(socket.getOutputStream());
      BufferedReader inStream = new BufferedReader(new
               InputStreamReader(socket.getInputStream()));
      outStream.println(msgReqEnum.toString());

      while( !exit ) {
        line = inStream.readLine();

        msgReply = new MOMessage(line);

        if(msgReply.getCommand().getCMD().equals(MOCommand.ENUMEND)) {
          exit=true;
        } else {
          addr= msgReply.toString().substring(
                    msgReply.toString().indexOf("<url>")+"<url>".length(),
```

```
                    msgReply.toString().indexOf("</url>")
                    );
            mnodesVec.add(addr);
          }
        }
      } catch ( Exception exc ) {
        System.out.println("Fehler beim Verbinden zum Server");
        exc.printStackTrace();
      }

      enu = mnodesVec.elements();
    }

    public boolean hasMoreElements() {
      return enu.hasMoreElements();
    }

    public String nextElement() {
      return enu.nextElement();
    }
}
```

Klasse MObjectsEnum

```
package mntNode;

import java.util.Enumeration;
import java.util.Vector;

import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.client.CIMClient;
import javax.wbem.client.PasswordCredential;
import javax.wbem.client.UserPrincipal;

public class MObjectsEnum implements Enumeration<CIMInstance>{
  private CIMNameSpace cns = null;
  private CIMClient cimClient = null;
  private Vector<CIMInstance> instanceVector = new Vector<CIMInstance>();

  private Enumeration<CIMInstance> instanceEnumeration;

  /*
   * MOEnumTargets enthaelt eine Liste mit allen GA_* Klassen,
   * die als moegliche Instanzen in Frage kommen
   */
  private String[] MOEnumTargets = {
    "GA_Entity",
    "GA_Attribute",
    "GA_AttributeValue",
    "GA_InstanceOfEntity",
    "GA_Relation",
    "GA_Dependency",
    "GA_Attribute_Entity",
    "GA_AttributeValue_Attribute",
    "GA_AttributeValue_IoE",
    "GA_BaseMetricValue_AttributeValue",
    "GA_Entity_IoE",
    "GA_LogicalElement_Entity",
    "GA_MetricDefinition_Attribute"
  };

  public MObjectsEnum ( String hostns ) throws CIMException {
    CIMInstance inst = null;
    Enumeration<CIMInstance> e = null;
    this.cns = new CIMNameSpace( hostns );

    this.cimClient = new CIMClient( this.cns,
                      new UserPrincipal("bla"), new PasswordCredential("blub") );

    for( int i=0; i<MOEnumTargets.length; i++) {
      e=cimClient.enumerateInstances(new CIMObjectPath(MOEnumTargets[i]), true );
      while(e.hasMoreElements()) {
```

```
          inst = e.nextElement();
          instanceVector.add(inst);
        }
      }
      instanceEnumeration = instanceVector.elements();
    }

    public boolean hasMoreElements() {
      return instanceEnumeration.hasMoreElements();
    }

    public CIMInstance nextElement() {
      return instanceEnumeration.nextElement();
    }
}
```