INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

SEP

Ein CIM-basiertes Modell der Rechnernetzepraktikum-Infrastruktur

Bearbeiter: Emanuel Heidinger
Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Dipl.-Inform. Michael Brenner
Dipl.-Inform. Vitalian Danciu

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

SEP

Ein CIM-basiertes Modell der Rechnernetzepraktikum-Infrastruktur

Bearbeiter: Emanuel Heidinger

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Dipl.-Inform. Michael Brenner
Dipl.-Inform. Vitalian Danciu

Abgabetermin: 31. Januar 2003

Zusammenfassung

Web-Based Enterprise Management (WBEM) ist ein offener Standard, der Grundlagen für ein plattformunabhängiges Netz- und Systemmanagement definiert. Den Kern dieser Architektur bildet das *Common Information Model (CIM)*, ein objektorientiertes Informationsmodell für die einheitliche Beschreibung von Managementinformationen.

Das Rechnernetzepraktikum deckt seit einigen Jahren einen wichtigen Teil des technischen Hauptstudiums der Informatik ab. Neuerdings bietet der MNM-Lehrstuhl auch ein technisches Praktikum zu IPSicherheit an, welches mit derselben Hardwarekonfiguration auskommen muss. Die neue Situation zwei Praktika parallel zu veranstalten, erfordert ein völlig neues Konzept, die vorhandene Hard- und Software zu managen. Diese neue Problemlage ist so noch nicht gelöst.

Diese Arbeit versucht jetzt, die Brauchbarkeit der Verwendung des Common Information Models in Bezug zum Management des Rechnernetzepraktikums sowie des IPSec Praktikums zu setzen.

Inhaltsverzeichnis

1 Thematische Einordnung **5**

2 Grundlagen **7**
2.1 Web-Based Enterprise Management 7
2.1.1 Motivation und Funktionsprinzip 7
2.1.2 Common Information Model 8
2.1.3 Interoperabilität 12

3 CIM Software **17**
3.1 CIM Object Manager 17
3.1.1 Funktionalität eines CIM Object Manager 17
3.1.2 Begriffsklärung Provider 18
3.2 CIMOM Implementierungen 19
3.2.1 Pegasus 19
3.2.2 SNIA 21
3.2.3 WBEMServices 22
3.3 CIM Tools 24
3.3.1 WBEMServices CIM Workshop 24
3.3.2 SNIA Browser 26
3.3.3 CIM Navigator 27
3.4 Resümee 28

4 Repräsentation der Managed Objects mittels CIM **30**
4.1 Übersicht der Vererbungshierarchie 30
4.2 Repräsentation der physischen Komponenten 31
4.3 Repräsentation der logischen Komponenten 33

5 Beispielmodellierungen **37**
5.1 Beispielmodellierung Rechner mit Ethernet-Switch 37
5.2 Beispielmodellierung 2 41

6 Fazit **48**

Quellenverzeichnis **51**

Abbildungsverzeichnis

2.1 CIM Meta Schema 9

3.1 WBEM-Architektur 18
3.2 Verzeichnisstruktur des Pegasus CIM Object Manager 21
3.3 CIM Workshop 25
3.4 Screenshot des SNIA CIM Browser 26
3.5 CIM Navigator 27

4.1 Übersicht der Vererbungshierarchie 31
4.2 Managed Object PhysicalElement 32
4.3 Managed Object ComputerSystem 33
4.4 Managed Object LogicalDevice 34
4.5 Managed Object Service 35
4.6 Managed Object ServiceAccessPoints 36

5.1 Beispielmodellierung Rechner mit Ethernet-Switch 38
5.2 Beispielmodellierung pcrnp10 39
5.3 Beispielmodellierung swnm1 40
5.4 Beispielmodellierung des Netzmanagementversuchs 41
5.5 Beispielmodellierung swnm1 42
5.6 Beispielmodellierung pcnm1ov 43
5.7 Beispielmodellierung pcnm1prot 44
5.8 Beispielmodellierung Hub 2 45
5.9 Beispielmodellierung VLANSWITCH 2 46

Kapitel 1

Thematische Einordnung

Der Betrieb eines Rechnernetzes umfasst eine Vielzahl von Aufgaben, die unter dem Begriff *Netzmanagement* zusammengefasst werden und sich nach OSI (siehe auch [1]) grob in die Funktionsbereiche

- Fehlermanagement,
- Konfigurationsmanagement,
- Leistungsmanagement,
- Abrechnungsmanagement und
- Sicherheitsmanagement

gliedern lassen. Die vorliegende Arbeit befasst sich vor allem mit den Teilgebieten des Konfigurationsmanagements und Fehlermanagements. Mit zunehmender Größe von Rechnernetzen schnellt deren Komplexität und der Zeitbedarf, der zur Wartung des Rechnernetzes nötig wird, in die Höhe.

Gerade bei Firmen mit einer großen Netzwerk-Infrastruktur stellen Rechnernetze die Administration vor große Aufgaben. Auf der einen Seite soll das Funktionieren des Netzes rasch und kostengünstig gewährleistet sein, auf der Anderen will sie den etwaigen Problemfall schnell erledigt wissen. Die Konfiguration von Rechnernetzen ist im allgemeinen sehr kurzlebig, da laufend neue Rechner hinzukommen, ausfallen, Mitarbeiter neue Software benötigen, etc. Durch diese ständigen Arbeiten sollte das Rechnernetz keine Stabilität einbüßen, der Großteil der Beschäftigen sollte von der Wartung ganz unberührt bleiben, der Betrieb läuft reibungslos weiter.

Hier setzen eine Vielzahl von Managementtools an, die gegebenenfalls eigene Protokolle verwenden, um dem neuen Informationsfluss Herr zu werden.

Diese Werkzeuge müssen die schwierige Aufgabe bewältigen, eine Balance zwischen vernünftigem Management und unnötigem Datenfluss zu finden, also ein vertretbares Mit-

telmaß. Freilich befindet sich schon eine Reihe solcher Managementtools auf dem Markt, teils kommerziell, teils Opensource, welche die Aufgaben mehr oder weniger gut erledigen. Für einen solch speziellen Anwendungszweck gibt es natürlich noch keine vorgefertigten Lösungen, im Besonderen wird zu für diesen Anwendungsfall eine eigene Entwicklung diskutiert.

An diesem Punkt setzt die vorliegende Arbeit an. Sie untersucht die Verwendbarkeit des von der Distributed Management Task Force (DMTF, siehe [2]) verabschiedeten Standards, Common Information Model (CIM, siehe [3]), auf das Konfigurations- und Fehlermanagement der Praktika Rechnernetze (RNP) und IT-Sicherheit (IPSec).

Kapitel 2

Grundlagen

2.1 Web-Based Enterprise Management

2.1.1 Motivation und Funktionsprinzip

1996 riefen die Firmen BMC Software, Cisco Systems, Compaq Computer, Intel und Microsoft die *Web-Based Enterprise Management (WBEM) Initiative* ins Leben, um eine offene, herstellerneutrale Architektur für ein WWW-basiertes Management der gesamten DV-Infrastruktur von Unternehmen zu entwickeln. Im Laufe der Jahre schlossen sich eine ganze Reihe weiterer Firmen dem Konsortium an, u. a. Computer Associates, IBM/Tivoli und Hewlett-Packard.

Systemadministratoren stehen heutzutage oft vor dem Problem, auf verschiedene Plattformen verteilte Managementdaten zu verwalten. Dabei müssen sie häufig auf herstellerabhängige APIs zurückgreifen oder für jede Managementanwendung eine separate Konsole benutzen. WBEM kann jedoch *eine gemeinsame* Schnittstelle für unterschiedliche Plattformen zur Verfügung stellen, weil es unabhängig von Programmiersprachen, Betriebssystemen und Nutzerschnittstellen ist. WBEM definiert einen allgemeinen Mechanismus für den Austausch von Managementinformationen, schreibt den Herstellern aber nicht vor, wie sie ihre Managementlösungen zu implementieren haben. Für WBEM ist keine Laufzeitumgebung oder besondere Programmiersprache erforderlich, es wird keine Nutzung spezieller Managementanwendungen, Betriebssysteme oder GUIs verlangt. WBEM stellt eine konsistente Sicht auf die zu verwaltenden Ressourcen bereit, ohne seine Nutzer auf bestimmte Systeme, Protokolle oder Plattformen festzulegen.

Zwei Hauptziele motivierten die WBEM-Gründer zur Schaffung einer plattformübergreifenden Management-Technologie. Erstens musste die Darstellung von Managementdaten standardisiert werden, da nur mit einem einheitlichen Informationsmodell ein uneingeschränkter, verlustfreier Austausch von Information gewährleistet ist. Zu diesem Zweck wurde der DMTF (*Desktop Management Task Force*, später umbenannt in *Distributed*

Management Task Force) im Jahre 1997 die (Weiter-)Entwicklung eines standardisierten Datenmodells namens *Common Information Model (CIM)* übertragen. CIM ist ein objektorientiertes Schema zur Beschreibung der Managementobjekte eines Systems. Es stellt einen einheitlichen, erweiterbaren Mechanismus zur Datenbeschreibung dar, der prinzipiell auf alle Netze und Netzkomponenten, Managementwerkzeuge, Peripheriegeräte und Datenbanken anwendbar ist. CIM unterstützt objektorientierte Konzepte wie Vererbung und Assoziationen und ist programmiersprachenunabhängig. So können mittels CIM definierte Objekte z. B. in Java, DCOM (*Distributed Component Object Model*), oder CORBA (*Common Object Request Broker Architecture*) implementiert werden.

Zweitens brauchten die WBEM-Gründer eine standardisierte Methode für den Zugriff auf Managementinformationen. Bisher mussten Administratoren für jede Managementumgebung, von der sie Daten abrufen wollten, auf herstellereigene API-Aufrufe und speziell ausgelegte Software zurückgreifen. WBEM stellt eine einheitliche Methode für den Zugriff auf Managementdaten aus verschiedenartigen Quellen zur Verfügung.

2.1.2 Common Information Model

Da es in heterogenen Umgebungen von Haus aus keine Vereinbarungen über die Information gibt, die zu Managementzwecken ausgetauscht werden muss, spielt das Informationsmodell eine zentrale Rolle innerhalb einer Managementarchitektur. Es spezifiert die Methoden zur Modellierung und Beschreibung von Managementobjekten und legt damit fest, *was* überhaupt verwaltet werden soll, welche Ressourcenmerkmale und Vorgänge relevant sind.

Das *Common Information Model (CIM)* bildet deshalb den wichtigsten Bestandteil der WBEM-Architektur. Es verfolgt bei der Strukturierung der Managementinformation durchweg einen objektorientierten Ansatz. Bei seiner Entwicklung wurde Wert darauf gelegt, dass sich vorhandene Informationsmodelle (wie z. B. SNMP-SMI) möglichst verlustfrei auf CIM abbilden lassen.

Die CIM-Spezifikation ([3]) – die von der DMTF verwaltet wird und auf der Website der Organisation (siehe [2]) eingesehen werden kann – unterteilt CIM in drei Schichten:

- Das *Core Model* ist ein „Kern“ von einigen wenigen Klassen, Assoziationen und Eigenschaften, die eine Basis für Verfeinerungen und ein grundlegendes Vokabular für die Analyse und Beschreibung zu verwaltender Systeme bilden.

  So steht die allgemeinste Klasse `CIM_ManagedElement` für jegliche Art von (evtl. verschachtelten) Managementobjekten; davon abgeleitet ist die Klasse `CIM_ManagedSystemElement`, welche allgemein gültige Merkmale wie den Namen oder das Installationsdatum eines Elements besitzt. Wiederum hiervon abgeleitet sind die Klassen `CIM_PhysicalElement` und `CIM_LogicalElement`, welche physische Elemente (also Hardware) bzw. logische Elemente wie Prozesse, Dateien oder

Aggregationen physischer Elemente beschreiben.

- Das *Common Model* besteht aus einer Menge von Klassen die konkreter gefasst sind als im Core Model und bereits als Basis für eine Reihe von Managementanwendungen dienen können. Sie sind aber immer noch unabhängig von bestimmten Technologien und Implementierungen. Abgedeckt werden hierbei Bereiche wie Systeme, Applikationen, Netze, Endgeräte und Benutzerverwaltung, wobei mit fortschreitender Entwicklung des Modells neue Anwendungsfelder hinzukommen können.
- *Extension Schemas* stellen technologiespezifische Erweiterungen des Common Model dar und sind auf spezielle Umgebungen (insbesondere Betriebssysteme) zugeschnitten. Als Beispiel sei *Win32 Schema* genannt, das bei Microsofts WBEM-Implementierung zum Einsatz kommt.

Core und *Common Model* gemeinsam nennt man *CIM Standard Schema* oder einfach *CIM Schema* (nicht zu verwechseln mit dem *CIM Meta Schema* im folgenden Abschnitt). Es wird von der DMTF ständig erweitert und in seinen verschiedenen Entwicklungsstufen (derzeit die Versionen 2.0 bis 2.7) auf [2] veröffentlicht.

Meta Schema

Das *Meta Schema* ist eine formale Definition des Modells. Es legt die im CIM verwendeten Begriffe sowie deren korrekten Gebrauch und die Semantik fest. Das Meta Schema selbst wird mit Hilfe der *Unified Modeling Language (UML)* definiert. Seine Struktur zeigt Abbildung 2.1.

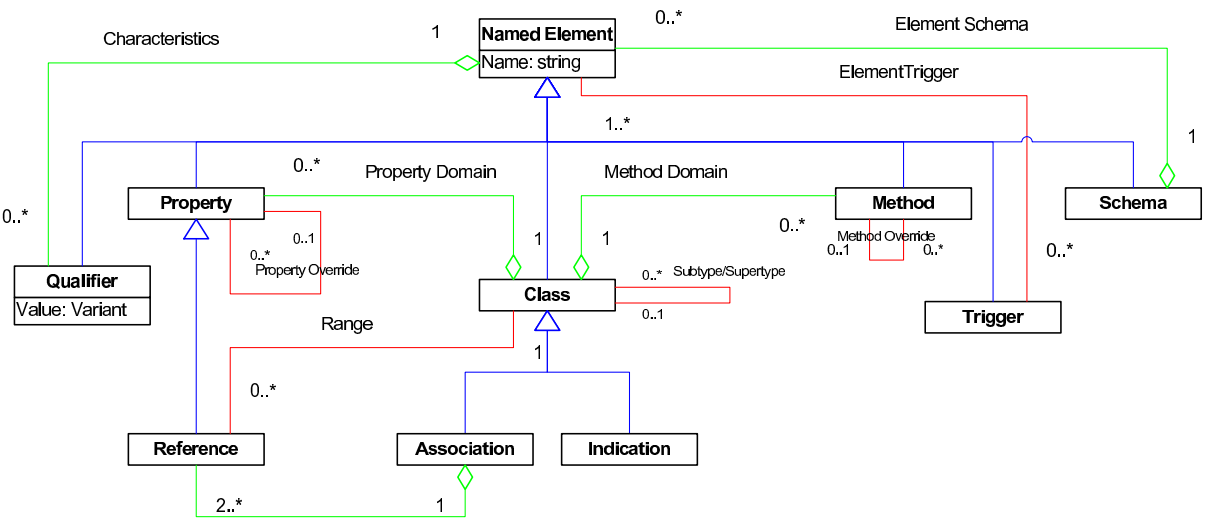

Abbildung 2.1: CIM Meta Schema

Die Elemente des Modells sind Schemata, Klassen (mit Ereignisklassen und Assoziationen als Untertypen), Eigenschaften (mit Referenzen als Untertyp) und Methoden.

Ein Schema ist eine Gruppe von Klassen, die aus Verwaltungsgründen zusammengefasst wurden. Schemata definieren Namensräume; ein voll qualifizierter Klassenname hat die Form „`<Schema>_<Klasse>`“. Innerhalb eines Schemas müssen die Klassennamen eindeutig sein.

Eine Klasse („Class“) ist eine Menge von Objekten desselben Typs. Von einer Klasse abgeleitete Unterklassen erben deren Eigenschaften und Methoden, können diese jedoch überschreiben; Mehrfachvererbung ist nicht möglich. Ein konkretes Objekt als Ausprägung einer Klasse nennt man Instanz.

Eine Eigenschaft („Property“) ist ein Wert, mit dem Instanzen einer Klasse charakterisiert werden. Man kann sie sich als ein Paar von *get*- und *set*-Funktionen vorstellen die, angewandt auf ein Objekt, den Zustand zurückliefern bzw. verändern.

Eine Methode („Method“) definiert die Signatur (Methodenname, Rückgabetyp und Parameter) einer Operation, die auf Instanzen der Klasse anwendbar ist. Methoden repräsentieren das „Verhalten“ von Klassen.

Ein Trigger liegt vor, wenn eine Zustandsänderung einer Instanz erkannt wird. Dabei kann es sich um Zugriffe auf Eigenschaften handeln sowie um Erzeugung, Löschung oder Update der Instanz selbst.

Eine Ereignisklasse („Indication“) repräsentiert einen Typ von Ereignismeldungen. Eine Instanz davon stellt eine konkrete Ereignismeldung dieses Typs dar. Sie wird als Reaktion auf einen Trigger erzeugt.

Eine Assoziation („Association“) ist eine Klasse, die mindestens zwei Referenzen enthält. Sie repräsentiert eine Beziehung zwischen Objekten. Die referenzierten Klassen werden dabei nicht beeinflusst; d. h. insbesondere, dass ihre Schnittstellen nicht modifiziert werden müssen, wenn sich an der Assoziation etwas ändert.

Eine Referenz („Reference“) ist eine spezielle Eigenschaft, die einen Verweis auf eine Klasse darstellt. Nur Assoziationen können über Referenzen verfügen.

Ein Qualifikator („Qualifier“) ist ein Mittel, mit dem man eine Klasse, Eigenschaft oder Methode näher charakterisieren kann. Beispielsweise läßt sich eine Eigenschaft mit den Qualifikatoren `Read` und `Write` als les- und schreibbar auszeichnen. Neben einer Reihe von Standard-Qualifikatoren sind auch benutzerdefinierte Qualifikatoren erlaubt. Auf diese Weise lässt sich das Meta Schema in beschränktem Umfang erweitern.

Managed Object Format (MOF)

Klassendiagramme sind ein probates Mittel, um Managementinformation für Menschen übersichtlich darzustellen. Das *Managed Object Format (MOF)* dient dagegen primär dem Zweck, die Informationen maschinenlesbar zu formatieren. Es handelt sich dabei um eine

Sprache zur textuellen Beschreibung CIM-konformer Information. Die Hauptkomponenten einer MOF-Spezifikation sind Klartextbeschreibungen von Klassen, Assoziationen, Eigenschaften, Referenzen, Methoden, Instanzdeklarationen sowie die zugehörigen Qualifikatoren. Eine MOF-Textdatei kann mittels eines Compilers verarbeitet werden. Auf diese Weise lassen sich statische Daten wie das CIM Schema in ein Repository einlesen.

Die vollständige MOF-Syntax ist in der CIM-Spezifikation ([3]) beschrieben. Zur Veranschaulichung folgen Auszüge aus der MOF-Beschreibung der Klasse `CIM_PhysicalElement`:

```
   [Abstract, ... ][1]

class CIM_PhysicalElement : CIM_ManagedSystemElement[2] {

      [Key[3] , MaxLen (256)[4] , Description[5]  (

        "An arbitrary string that uniquely identifies the Physical"

        "Element and serves as the Element's key. ...") ]

   string Tag;

...

      [MaxLen (256), Description (

        "The name of the organization responsible for producing the "

        "PhysicalElement. ...") ]

   string Manufacturer;

...
```

[1] Qualifikator `Abstract` besagt, dass die Klasse nicht instantiiert werden kann
[2] `CIM_PhysicalElement` ist Unterklasse von `CIM_ManagedSystemElement`
[3] Qualifikator `Key` zeichnet Eigenschaft `Tag` als eindeutiges Unterscheidungsmerkmal der Klasse aus
[4] Qualifikator `MaxLen` legt Maximallänge der Zeichenkette `Tag` fest
[5] Qualifikator `Description` liefert textuelle Beschreibung der Eigenschaft

```
        [MaxLen (64), Description (
          "A manufacturer-allocated number used to identify the Physi-
cal"
          "Element.") ]
    string SerialNumber;
        [MaxLen (64), Description (
          "A string indicating the version of the PhysicalElement.") ]
    string Version;
...
        [Description (
          "Boolean indicating that the PhysicalElement is powered on "
          "(TRUE), or is currently off (FALSE).") ]
    boolean PoweredOn;
        [Description (
          "Date that this PhysicalElement was manufactured.") ]
    datetime ManufactureDate;
};
```

2.1.3 Interoperabilität

In den ursprünglichen WBEM-Spezifikationen war keine Transportschicht vorgesehen. Aus diesem Grund ist derzeitige WBEM-Software trotz Standardkonformität nicht zwangsläufig interoperabel. Erst 1999 wurden Mappings für die Darstellung von CIM-Informationen

im XML-Format und der Transfer über das *Hypertext Transfer Protocol (HTTP)* definiert.[6]

CIM-Darstellung in XML

XML (Extensible Markup Language) ist eine universelle Auszeichnungssprache und bildet als solche ein mächtiges Werkzeug zur Datenmodellierung. Als plattformunabhängiger und vergleichsweise einfach zu implementierender Standard, insbesondere aber im Hinblick auf die Verwendung von HTTP als Transportprotokoll, bietet sich das XML-Format für die Einhüllung von CIM-Daten an.

Die exakten Abbildungsvorschriften dieser Einhüllung sind in [4] spezifiziert. Als Beispiel wird wieder die Core-Klasse `CIM_PhysicalElement` herangezogen (siehe 2.1.2), diesmal in XML-Darstellung:

```
<CLASS NAME="CIM_PhysicalElement" SUPERCLASS="CIM_ManagedSystemElement">
  <QUALIFIER NAME="Abstract" TYPE="boolean" OVERRIDABLE="false" TOSUBCLASS="false">
    <VALUE>
      TRUE
    </VALUE>
  </QUALIFIER>
...
  <PROPERTY NAME="Tag" CLASSORIGIN="CIM_PhysicalElement" TYPE="string">
    <QUALIFIER NAME="Key" TYPE="boolean" OVERRIDABLE="false">
      <VALUE>
        TRUE
      </VALUE>
    </QUALIFIER>
    <QUALIFIER NAME="MaxLen" TYPE="uint32">
      <VALUE>
        256
```

[6] Was immer noch fehlt, ist eine standardisierte Schnittstelle zwischen CIMOM und Provider. Providermodule müssen derzeit noch auf den verwendeten CIMOM zugeschnitten werden.

```
      </VALUE>
    </QUALIFIER>
    <QUALIFIER NAME="Description" TYPE="string" TRANSLATABLE="true">
      <VALUE>
        An arbitrary string that uniquely identifies the PhysicalElement
        and serves as the Element&amp;apos;s key. ...
      </VALUE>
    </QUALIFIER>
  </PROPERTY>
...
  <PROPERTY NAME="Manufacturer" CLASSORIGIN="CIM_PhysicalElement" TYPE="string">
    <QUALIFIER NAME="MaxLen" TYPE="uint32">
      <VALUE>
        256
      </VALUE>
    </QUALIFIER>
    <QUALIFIER NAME="Description" TYPE="string" TRANSLATABLE="true">
      <VALUE>
        The name of the organization responsible for producing the
        PhysicalElement. ...
      </VALUE>
    </QUALIFIER>
  </PROPERTY>
...
</CLASS>
```

CIM-Operationen via HTTP

Die in [5] spezifizierte Abbildung ordnet WBEM-Operationen den Requests von HTTP zu und schafft so Interoperabilität zwischen den Transportprotokollen. Per *POST* oder *M-POST* werden XML-Seiten verschickt, die Methodenaufrufe und Datenaustausch beinhalten. Neben den Vorteilen eines gut verstandenen Protokolls, für das performante Implementierungen verfügbar sind, bringt HTTP quasi automatisch Mechanismen zur verschlüsselten Übertragung mit sich. Bei erhöhten Anforderungen an die Sicherheit wird einfach auf eine SSL-Verbindung zurückgegriffen.

Das folgende Beispiel zeigt, wie so ein Request aussehen kann. Mit einer Anfrage dieser Art würde ein Client die Klassendefinition von `CIM_PhysicalElement` beim CIMOM anfordern:

```
M-POST /cimom HTTP/1.1

HOST: http://www.myhost.com/

Content-Type: application/xml; charset="utf-8"

Content-Length: xxxx

Man: http://www.dmtf.org/cim/mapping/http/v1.0 ; ns=73

73-CIMOperation: MethodCall

73-CIMMethod: GetClass

73-CIMObject: root/cimv2

<?xml version="1.0" encoding="utf-8" ?>

<CIM CIMVERSION="2.0" DTDVERSION="2.0">

 <MESSAGE ID="4711" PROTOCOLVERSION="1.0">

 <SIMPLEREQ>

  <IMETHODCALL NAME="GetClass">

   <LOCALNAMESPACEPATH>

    <NAMESPACE NAME="root"/>

    <NAMESPACE NAME="cimv2"/>
```

```
    </LOCALNAMESPACEPATH>
    <IPARAMVALUE NAME="ClassName"><CLASSNAME NAME="CIM_PhysicalElement"/>
       </IPARAMVALUE>
    <IPARAMVALUE NAME="LocalOnly"><VALUE>FALSE</VALUE></IPARAMVALUE>
   </IMETHODCALL>
  </SIMPLEREQ>
  </MESSAGE>
 </CIM>
```

Die Antwort des CIMOM enthielte im Erfolgsfall unter anderem die vollständige Klassendefinition im XML-Format (siehe 2.1.3).

Kapitel 3

CIM Software

3.1 CIM Object Manager

3.1.1 Funktionalität eines CIM Object Manager

Abbildung 3.1 zeigt einen Überblick der WBEM-Architektur. Am unteren Bildrand sind einige mit WBEM nutzbare Teilsysteme und Komponenten dargestellt. Die darüber befindliche *CIM Object Provider*-Ebene fungiert als Übersetzungsschicht für den *CIM Object Manager (CIMOM)*. Der CIMOM ist die zentrale Komponente eines WBEM-Systems. Er regelt die Interaktionen zwischen dem *CIM Repository*, den Managementanwendungen und dem CIM Object Provider. Zusätzlich übernimmt der CIMOM Aufgaben wie Authentifizierung und Event-Handling. Das *CIM Repository* ist eine Datenbank, die statische Managementdaten in Form von CIM-Objekten enthält. Die „Managementanwendung“ am oberen Bildrand repräsentiert alle Applikationen, die Managementdaten weiterverwerten. Ein Beispiel hierfür wäre ein WWW-Server, der die Daten (mittels CGI-Modul, Java-Servlet etc.) als HTML-Seiten aufbereitet und für den Zugriff durch WWW-Browser verfügbar macht.

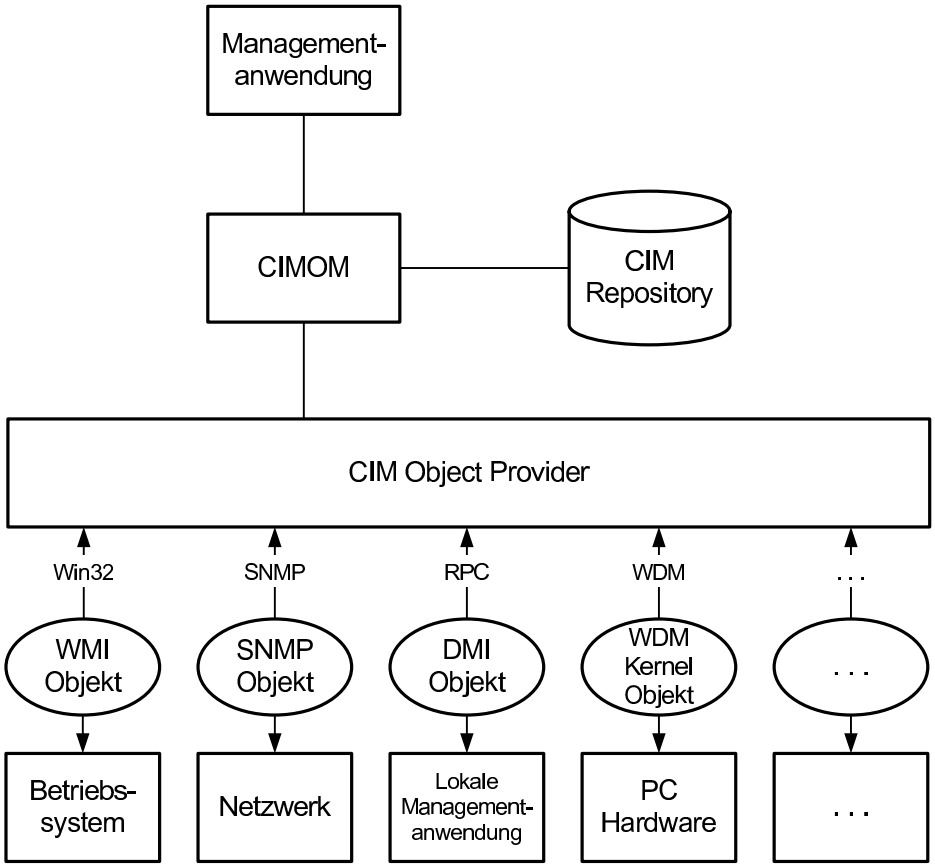

Abbildung 3.1: WBEM-Architektur

3.1.2 Begriffsklärung Provider

Die CIM Object Manager bieten die Möglichkeit, Erweiterungen zur Laufzeit an diesem vorzunehmen, vergleichbar einem Internetbrowser, welcher durch Plugins neue Fähigkeiten erhält. Die Hersteller der CIMOMs stellen dafür geeignete Schnittstellen zur Verfügung. Die Erweiterungen werden in diesem Zusammenhang als Provider bezeichnet. Hierbei ist zu beachten, dass jeder CIM Object Manager, seine eigenen Methoden, Templates bereitstellt, Provider einzubinden und zu implementieren.

Ein Provider ist also ein Stück Code, das zur Laufzeit in den CIMOM eingebunden wird. Mittels geeigneter Methoden, registriert sich dieser Provider beim CIM Object Manager für bestimmte Klassen und Instanzen. Somit ist dieser Provider nach der Registrierung für die Verwaltung von Instanzen der entsprechenden registrierten Klasse verantwortlich.

Genaugenommen handelt es sich hierbei um einen Instanzprovider, welcher die Kontrolle über bestimme Klassen von Instanzen übernimmt. Der CIM Object Manager gibt hier also die Kontrolle vollständig aus der Hand, der Provider wird selbst für die Datenhaltung und Integrität verantwortlich.

Es gibt noch andere Arten solcher Provider, wie etwa Methodprovider. Die Instanzprovider stellen allerdings die wichtigste Gattung dar, daher wird hier auf solche Exoten nicht weiter eingegangen.

Die Verwendung von Providern ist eine flexibel Art, CIM Object Managager zu erweitern. Es gilt jedoch einige Fallstricke zu umgehen.

Um den Bezug zur eigentlichen Intension dieser Arbeit herzustellen, sollte der Leser sich die Möglichkeit vor Augen führen, dass der CIMOM zentral das Sammeln der benötigten Daten auslöst. Dies könnte praktisch mit solchen Providern realisiert werden. Da dem Provider aber Fähigkeiten zur Fernwartung gegeben werden müssten, ein nicht ganz einfacher Lösungsweg; WBEM verliert hier de facto ein Teil seiner Plattformunabhängigkeit.

3.2 CIMOM Implementierungen

3.2.1 Pegasus

The Open Group ist ein internationales firmen- und technologieneutrales Konsortium, das sich als Vermittler zwischen Nutzern und Herstellern von Informationstechnologie versteht und dessen erklärtes Ziel die Förderung von Interoperabilität in allen Bereichen der IT ist. Sponsoren der Open Group sind die Firmen Fujitsu, Hewlett-Packard, Hitachi, IBM und Sun Microsystems.

Mit *Pegasus* hat die Open Group ein WBEM-Projekt gestartet, das vollständig in C++ realisiert ist. Es läuft unter den meisten UNIX-Arten, unter Linux und Microsoft Windows. Pegasus wird unter der *MIT open source license* vertrieben, die aktuelle Releasefassung hat die Versionsnummer 2.1. Das Projekt-Portal befindet sich auf [9].

Eigenschaften von Pegasus

Das Pegasus Paket enthält über den eigentlichen CIMOM hinaus eine Reihe nützlicher Tools, die dem Entwickler bei seiner Arbeit behilflich sind. Das Kernstück stellt allerdings der *cimserver* dar, der standardmäßig als Daemon startet und im Hintergrund auf TCP Port 5988 (bzw. 5989) lauscht und CIM-over-HTTP (bzw. CIM-over-HTTPS) Anfragen entgegennimmt. Die Datenhaltung erfolgt in einem sogenannten Repository, das auf einem sekundären Speichermedium in XML Darstellung abgelegt wird. Das Repository enthält auf unterster Ebene mehrere *Namespaces*, zur parallelen Verwaltung zahlreicher Schemata. Standardmäßig wird unter Pegasus das aktuelle CIM-Schema in dem *Namespace* root/cimv2 verwaltet. Frühere CIM Versionen verwendeten dabei den *Namespace* root, seit Einführung der zweiten Version hat sich dies jedoch geändert. Jeder *Namespace* enthält zur Datenhaltung drei Verzeichnisse, classes, instances und qualifiers.

Der Mof-Compiler *cimmof* bzw. *cimmofl* ist ein weiteres nützliches Hilfsmittel und sollte hier keinesfalls unerwähnt bleiben. Er formt MOF-Files in XML um und lädt sie ins Repository, im Normalfall also in das Verzeichniss classes; hier wird zentral die Klassenhierarchie verwaltet.

Installation des Pegasus CIMOM

Benötigt werden die Quelldateien des Pegasus Projekts. Zwei Möglichkeiten stehen hier zur Wahl, zum einen kann ein fertiger Release heruntergeladen werden, zum anderen ein aktueller Snapshot der CVS Entwicklungsumgebung. Aufgrund der recht aktiven Entwicklergemeinde, welche dieses Opensource Projekt besitzt, wurde auf die zweite Möglichkeit zurückgegriffen.

Die Projektseite der Pegasus Opengroup ist im Internet unter [9] zu finden.

Der Pegasus-Opengroup CIMOM wurde mit C++ entwickelt, in dem betrachteten Testlauf kam die GNU Compiler Collection ([8]) in der Version 3.3 zur Verwendung.

Im Besitz der Quelldateien lässt sich mit der eigentlichen Erstellung des Kompilats fortfahren. Hierzu muss entschieden werden, auf welcher Plattform Pegasus verwendet wird. Mit diesem Wissen lassen sich die Umgebungsvariablen PEGASUS_HOME und PEGASUS_PLATFORM auf dem Testrechner mit korrekten Werten belegen.

Pegasus unterstützt dabei folgende Plattformen:

- WIN32_IX86_MSVC
- LINUX_IX86_GNU
- AIX_RS_IBMCXX
- HPUX_PARISC_ACC
- ZOS_ZSERIES_IBM

In dem vorliegenden Beispiel sind demnach folgende Anweisungen nötig:

- export PEGASUS_HOME=/mnt/disk/_cim/pegasus
- export PEGASUS_PLATFORM=LINUX_IX86_GNU

Des Weiteren sollte das Binary-Verzeichnis (bin, befindlich im Pegasus-Wurzelverzeichnis) und das Library-Verzeichnis (lib, ebenfalls im Pegasus-Wurzelverzeichnis) den entsprechenden Pfaden angefügt werden (PATH und LD_LIBRARY_PATH).

Mittels *make* im Pegasus-Wurzelverzeichnis wird der Pegasus CIMOM erstellt.

Nach Erstellung des Pegasus ist des weiteren ein anfänglich erstelltes Repository nötig, welche das gültige CIM-Schema enthält und in welchem später alle Instanzen als XML gespeichert werden. Der zugehörige Befehl lautet *make repository*.

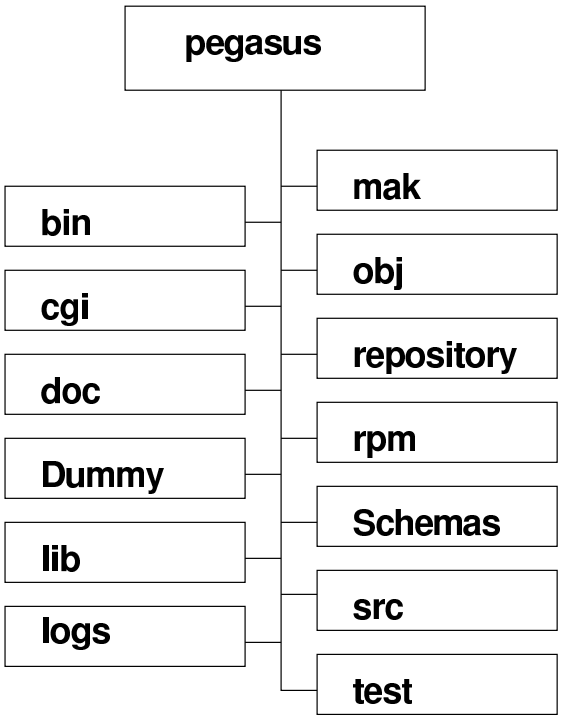

Abbildung 3.2: Verzeichnisstruktur des Pegasus CIM Object Manager

3.2.2 SNIA

Die Organisation die hinter dem Opensource-CIMOM steht, ist die Storage Networking Industry Association (SNIA). Sie hat sich der Förderung der Speichernetzwerk-Technologie verschrieben. Unter anderem sind ihr AT&T, IBM, Nortel Networks, Hewlett-Packard, Intel und Cisco Systems angehörig.

Eigenschaften des SNIA CIMOM

Die Client-API des Opensource Projektes SNIA CIMOM ist CIM-over-HTTP konform (auch HTTPS/SSL), also auch mit Pegasus weitestgehend kompatibel. Des Weiteren kann der Server über RMI bzw. über lokale Routinen angesprochen werden. Als Authentifizierungsdienst kommt standardmäßig die HTTP Basic Authentication zum Einsatz, weitere Methoden können als Plugins eingebunden werden. Ein MOF-Compiler ist für das Einlesen statischer Daten in das Repository verantwortlich.

Ein generischer CIM Browser ist ebenfalls mit von der Partie, er zeigt die Baumstruktur des Repository an, lädt verschiedene MOF-Dateien, generiert Instanzen, löscht oder modifiziert diese. Der CIM Browser sollte bei Verwendung mit anderen CIMOM keine größere Probleme bereiten, da sie ja auf CIM-over-HTTP basieren; zumindest mit der Pegasus Suite arbeitete der CIM Browser in der Testumgebung problemlos zusammen.

Des Weiteren sind Beispielprovider zu Demonstrationszwecken vorhanden.

Das Repository findet sich ausgehend vom SNIA-Wurzelverzeichnis im Verzeichnis persistence. Hier unterscheidet SNIA die instances, classes und qualifiers. Die darauffolgende Stufe gliedert die Namespaces auf, standardmäßig gibt es den Namespace root. Die Datenspeicherung erfolgt in diesem persistenten Repository aber in eigenem Format, es wird kein Klartext oder XML verwendet.

Installation des SNIA CIMOM

Der SNIA CIMOM benötigt über die Quellen hinausgehend Java 1.3.1, Java Swing und den XML Parser Xerces (in diesem Beispiel der Version 2.5.0).

Die Projektseite dieses CIMOM findet sich unter [7].

Zur Installation wird der Einfachheit halber das vorgesehene Skript *make.sh* verwendet, die Umgebungsvariable CLASSPATH muss hier angepasst werden, damit die zusätzlichen Packages Xerces und Swing erreicht werden.

Auf dem Testrechner *pcheger3* waren folgende Anweisungen nötig:

export PATH=/usr/lib/SunJava2-1.3.1/bin:$PATH

export JAVA_HOME=/usr/lib/SunJava2-1.3.1/

export CLASSPATH=$CLASSPATH:
/home/heidinge/xerces-2_5_0/build/xercesImpl.jar:
/home/heidinge/xerces/2_5_0/build/xercesSamples.jar:
/home/heidinge/xerces-2_5_0/build/xml-apis.jar

3.2.3 WBEMServices

Bereits 1999 stellte die Firma Sun Microsystems die WBEM-Implementierung für ihr Betriebssystem Solaris vor: *Solaris WBEM Services*. Von da an standen die Systeminformationen von Solaris-Maschinen im CIM-Format zur Verfügung. Für Anwendungsentwickler wurde ein spezielles SDK (Software Development Kit) mitgeliefert.

Zwei Jahre später veröffentlichte Sun den Quellcode der WBEM Services. Formell wurde die Codebasis als „Gründungskapital“ an *WBEMsource* übergeben, eine Open Source Initiative deren Ziel es ist, die Verbreitung der WBEM-Technologie zu fördern.[1]

So kommt es, dass heute neben der kommerziellen Variante für Suns Betriebssystem (derzeit Version 2.5) eine betriebssystemunabhängige Open-Source-Fassung der WBEM Services mit der Versionsnummer 0.95c existiert. Sie wird unter der *Sun Industry Standards*

[1] Heute dient WBEMsource im Wesentlichen als Koordinierungsforum für diverse WBEM-Projekte, um die Interoperabilität der unterschiedlichen Implementierungen zu verbessern.

Source License (SISSL) vertrieben und unter Suns Federführung weiterentwickelt. Das Web-Portal zum Projekt findet sich unter [6].

Eigenschaften der WBEMServices

Als Programmiersprache kommt Java zum Einsatz, die Services können also prinzipiell auf jedem System genutzt werden, für das eine aktuelle Java Virtual Machine (JVM) zur Verfügung steht. Des Weiteren ist das XML Parse Package Jakarta nötig.

Client-seitig wird neben CIM-XML/HTTP auch RMI[2] unterstützt, verschlüsselte Kommunikation über HTTPS[3]/SSL[4] ist möglich. Mit einem voll funktionsfähigen CIMOM und einem Repository für statische CIM-Daten ist die Grundausstattung für ein WBEM-System vorhanden. Das Repository wird in eigenem Format (wiederum kein XML oder Klartext) auf Platte gespeichert.

Weitere Werkzeuge umfassen

- einen MOF-Compiler zum Einlesen von MOF-Definitionen in das Repository,
- einen MOF-to-JavaBeans-Generator, mit dem sich aus CIM-Klassen-Beschreibungen direkt JavaBeans (wiederverwendbare Softwarekomponenten) erzeugen lassen sowie
- den *CIM Workshop*, eine graphische Client-Applikation, mit der man durch das CIM-Schema „browsen“ und CIM-Operationen auf Klassen und Instanzen ausführen kann.

Wird die Kompatibilität zu anderen Produkte betrachtet, so fällt ins Auge, dass die WBEMServices sich inkompatibel mit dem SNIA Package zeigen; so führt der Zugriff auf den CIMOM mittels SNIA Browser zu dessen Programmabsturz. Des Weiteren benötigt dieser CIMOM standardmäßig Root Rechte, was so kaum nachvollziehbar scheint. Wird die Prüfroutine des Startup Skripts nach der Nutzerkennung entsprechend modifiziert (in diesem Fall auskommentiert), startet der CIMOM problemlos mit jeder Kennung.

Die Anfragen die an diesen CIMOM gestellt werden, scheinen sehr langsam erledigt zu werden. Dies liegt wohl an der verwendeten Programmiersprache Java, mit der Geschwindigkeit von einem C++ CIMOM wie etwa Pegasus kann dieses Produkt jedenfalls nicht konkurrieren.

Installation der WBEMServices

Die Quellen dieses Opensource CIMOM sind unter [6] erhältlich.

[2] Remote Method Invocation (Java-Standard für Methodenaufrufe über Rechnergrenzen hinweg)
[3] HTTP Secure
[4] Secure Sockets Layer

Nach Anpassung des Skripts start_cimom.sh, lassen sich die Wbemservices auch ohne Root-Rechte ohne Weiteres starten. Jedoch funktioniert das Shutdown Skript stop_cimom.sh mit dieser Modifikation nicht, es ist daher zum Beenden des CIMOM eine kill-Operation vonnöten.

Im Speziellen waren auf *pcheger3* die folgenden Operationen nötig.

export PATH=/soft/IFI/lang/j2sdk-1.4.0-sun/iX86-unknown-linux/bin:$PATH

export JAVA_HOME=/soft/IFI/lang/j2sdk-1.4.0-sun/iX86-unknown-linux

3.3 CIM Tools

Das folgende Kapitel befasst sich mit einer Reihe von CIM Tools, die nicht dem obigen Kapiteln zuordbar sind, aber dennoch wichtige Aufgaben der CIM Welt erledigen. Teils verstehen sie sich als einzelne Projekte, teils sind sie Teil der obig besprochenen CIMOM Projekte, wie etwa WBEMServices CIM Workshop oder SNIAs CIM Browser.

3.3.1 WBEMServices CIM Workshop

Eigenschaften

Wie die Namensgebung schon vermuten lässt, ist dieser generischer CIM Browser Teil des Cimom Paketes WBEMServices. Wie schon WBEMServices selbst, ist dieses Teilprojekt ebenfalls in der plattformübergreifenden Sprache Java realisiert. Dieses Client-Werkzeug bietet lediglich rudimentäre Funktionen für den Zugriff auf das CIM Repository. So können mittels diesem Tool Instanzen angelegt, modifiziert, gelöscht oder etwa abgefragt werden. Eine eigene Funktionalität zur Verwaltung von Referenzen bietet diese Softwarelösung hingegen nicht. Dieser Aspekt schränkt die Funktionalität zwar nicht ein, diese Vorgehensweise erweist sich aber als sehr umständlich. Darüber hinaus erlaubt CIM Workshop die Kommunikation mit dem CIM Object Manager sowohl mittels CIM-over-HTTP als auch über CIM-over-RMI. Im Gegensatz zu später besprochenen Varianten, verlangt CIM Workshop eine sich lokal befindliche Mof-Datei. Die Klassenhierarchie welche dieser CIM Browser verwendet, stammt also nicht aus dem Repository direkt, sondern aus einer eben lokal geöffneten Mof-Datei. Dies könnte schlimmstenfalls zu Versionskonflikten führen.

Der Quellcode ist, wie bei dem Mutterprojekt ebenfalls, OpenSource.

Im Projektverzeichnisbaum wird CIM Workshop im Verzeichnis *bin* mittels *start_workshop.sh* aufgerufen. Die Umgebungsvariablen sind vom WBEMServices CIMOM zu übernehmen.

Erwähnenswert wäre an dieser Stelle überdies, dass dieses Werkzeug im Normalfall mit den anderen CIM Object Manager Implementierungen problemlos zusammenarbeitet (soweit dies bei schnelllebigen Softwareprojekten zu behaupten ist).

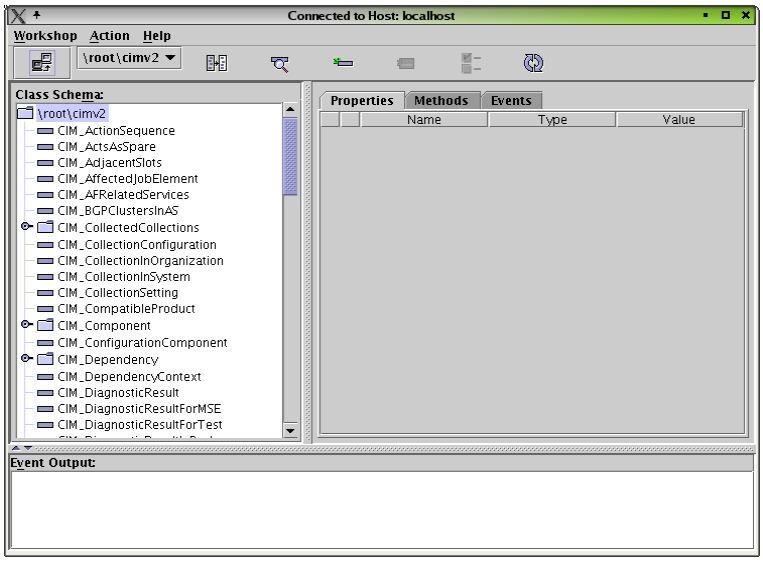

Abbildung 3.3: CIM Workshop

Installation

Nur der Vollständigkeit halber sei dieser Punkt vermerkt, da die Installation CIMOM WBEMServices und CIM Workshop Hand in Hand gehen.

Im Speziellen sind auf *pcheger3* die folgenden Voreinstellungen nötig:

export PATH=/soft/IFI/lang/j2sdk-1.4.0-sun/iX86-unknown-linux/bin:$PATH

export JAVA_HOME=/soft/IFI/lang/j2sdk-1.4.0-sun/iX86-unknown-linux

Mittels *bin/start_workshop.sh* wird CIM Workshop gestartet.

3.3.2 SNIA Browser

Eigenschaften

Der SNIA eigene CIM Browser mutet schon auf den ersten Eindruck etwas antiquiert an. Nach dem Start lädt der SNIA Browser zu allererst die gesamte im CIM Repository befindliche Klassenhierarchie in den flüchtigen Speicher. Aufgrund diesen Umstands benötigt der SNIA Browser von der Java Virtual Machine zusätzlich zur Verfügung gestellten

Speicher und eine gewisse Zeit zum Starten. Auch dieser Browser stellt rudimentäre Funktionen zur Verfügung, geeignete Funktionen zur Verwaltung von Referenzen fehlen auch hier.

Eine Parallele zum CIM Workshop ist die eng verwurzelte Entwicklung mit dem Mutterprojekt. Es ist daher an dieser Stelle hervorzuheben, dass die Verwendung mit anderen CIM Object Managern, zu keinen Problemen führt.

Auch diese Software kann sowohl mit CIM-over-HTTP als auch CIM-over-RMI kommunizieren.

Des Weiteren kann dieses Softwareprodukt lediglich Instanzprovider ansteuern, andere Provider, wie etwa Methodprovider bleiben hier außen vor.

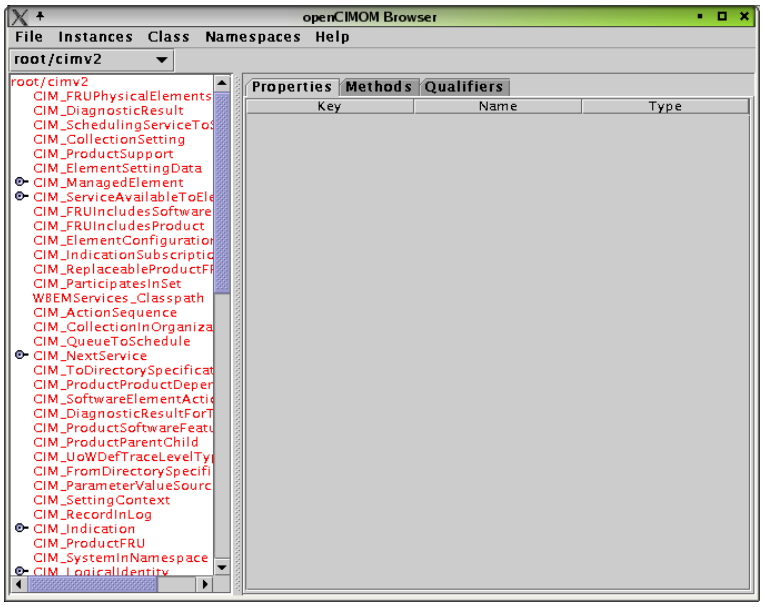

Abbildung 3.4: Screenshot des SNIA CIM Browser

Installation

Die Installation des SNIA Browsers geht mit der Installation des SNIA CIMOM einher.

Über die Grundeinstellungen hinausgehend, wie schon vom SNIA CIMOM bekannt, sind keine weitere Anpassungen nötig.

Der Aufruf erfolgt über *cimbrowser.sh*.

3.3.3 CIM Navigator

Eigenschaften des CIM Navigator

Der CIM Navigator entspricht im groben einem generischen CIM Browser wie etwa dem CIMWorkshop oder dem SNIA CIMBrowser. Der CIM Navigator bringt darüber hinaus eigene Optionen zum Verwalten von Referenzen mit sich, erleichtert demnach deren Umgang ungemein. Ein besonderer Bonus stellt das grafische Fenster dar, in dem sich ein Geflecht aus CIM Objekten beliebig umfangreich grafisch darstellen lässt. Die Homepage dieses Projektes findet sich im Netz unter [10]. Dieses Projekt gehört nicht der Opensource Gemeinde an, gleichwohl ist der CIM Navigator frei zum Download verfügbar. Der Quellcode ist leider nicht verfügbar. Die aktuelle Version 0.85 deutet aber schon darauf hin, dass dieses Werkzeug noch nicht ganz den Kinderschuhen entwachsen ist.

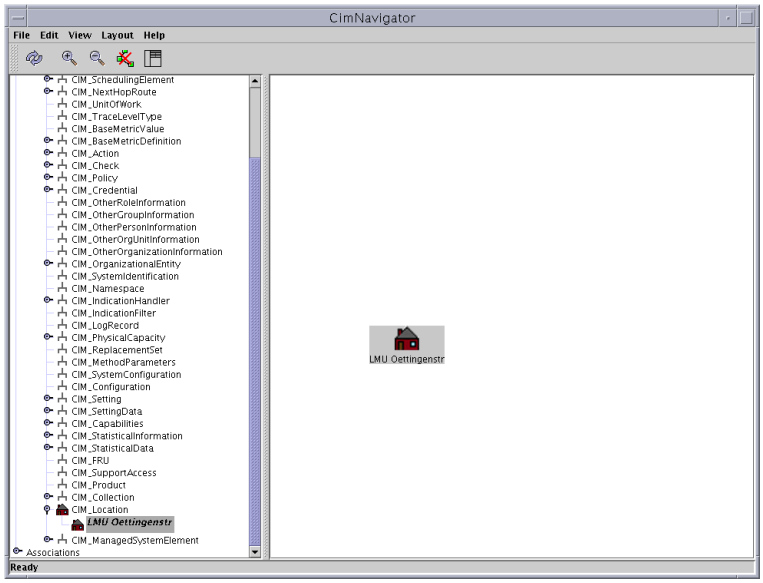

Abbildung 3.5: CIM Navigator

Installation des CIM Navigator

Der CIM Navigator wird in einer fertigen jar-Datei ausgeliefert, benötigt wird zum korrekten Ausführung Java der Version 1.4.0.

Der CIM Navigator wird mittels des Skripts cimnav.sh gestartet, das jedoch noch entsprechend den Rahmenbedingungen angepasst werden muss.

In dem betrachteten Testlauf waren auf dem Testrechner *pcheger3* folgende Anweisungen nötig:

export PATH=/soft/IFI/lang/j2sdk-1.4.0-sun/iX86-unknown-linux/bin:$PATH

export JAVA_HOME=/soft/IFI/lang/j2sdk-1.4.0-sun/iX86-unknown-linux

3.4 Resümee

Zusammenfassend wird die Nützlichkeit der obig besprochenen Produkte in Bezug auf die Möglichkeiten im betrachteten Anwendungsfall untersucht. Für die Verwendung sind unter anderem folgende Punkte wichtig:

- Einhaltung vorgegebener Standards,
- durchgängige XML-Fähigkeiten,
- ressourcensparende Verwendung,
- aktiver Support und künftige Weiterentwicklung.

Im Weiteren wird nun Punkt für Punkt abgehandelt.

Alle obig genannten Produkte unterstützen das auf XML basierende CIM-over-HTTP Protokoll, scheinen demnach in diesem Punkt gleichwertig. Wünschenswert wäre darüberhinaus aber ein durchgängiger XML-Support, beispielsweise sollte die Speicherung auf einem sekundären Speichermedium ebenfalls in XML erfolgen. Dies ist jedoch lediglich bei dem Pegasus CIMOM der Fall, andere CIM Object Manager nutzen hingegen eigene Datenformate, teils erfolgt die Speicherung als Klartext, teils in einem eigenen nicht spezifizierten Format. Die Möglichkeit der bloßen Ausgabe des gesamten Datenbestands in eine XML-Datei stellt keinen der betrachteten CIM Object Manager vor allzu große Probleme, da es sich hierbei ja lediglich um eine EnumerateInstance-Operation auf einen spezifizierten Namespace handelt.

Des Weiteren werden die nötigen Ressourcen betrachtet. Hierbei fällt ins Auge, dass sowohl der SNIA CIMOM als auch die Wbemservices eine Java Runtime Environment benötigen, also schon die Basis auf welche der CIM Object Manager aufbaut eine hohe Prozessorlast erzeugt. Pegasus hingegen ist komplett unter C++ entwickelt und gilt daher als recht ressourcensparend. Eine gewisse Plattformunabhängigkeit wird hier durch die zahlreichen Portierungen erreicht. Das Verhalten der Java- bzw. C++ CIM Object Manager in Bezug auf den Ressourcenverbrauch zeigt sich auch wie erwartet in den praktisch ausgeführten Tests.

Ein Support wird wie in solchen Opensource Projekten üblich, mit Mailinglisten geleistet. Sie sind frei zügänglich, es steht jedem frei, hier aktiv zu werden. Unter diesem Gesichtspunkt zeugten alle Projekte von einer recht aktive Entwicklergemeinde; hinter jedem dieser Projekte stehen zahlreiche, renommierte Firmen, daher sollte zukünfitg keine Problematik mit weiterem Support und weiterer Fortentwicklung entstehen.

Bleibt nun noch die Schlußfolgerung zu ziehen. Da für dieses Projekt vor allem ein durchgängiger XML-Support erwünscht ist und der CIMOM darüber hinaus die gegebenen

Ressourcen am besten nutzt, wird für die Aufgabe der Erfassung einer Netzwerktopologie der Opengroup Pegasus CIMOM [9] empfohlen.

Auf der Seite der Managementanwendungen stehen hingegen die noch recht unausgereift wirkenden CIM Browser. Sie bieten zwar rudimentäre Funktionen zur CIM Darstellung, jedoch muss davon ausgegangen werden, dass ein potentieller Nutzer der Managementanwendung nicht unbedingt mit dem CIM Modell vertraut sein muss. Soll bei der Topologieerfassung eine intuitiv benutzbare Managementanwendung zum Einsatz kommen, so ist eine Eigenentwicklung also (noch) kaum umgänglich. Hier müessten die Hersteller solcher Softwareprodukte etwas nachziehen.

Kapitel 4

Repräsentation der Managed Objects mittels CIM

Das nachfolgende Kapitel soll dem Anspruch gerecht werden, erstmals einen detailreicheren Einblick in die Möglichkeiten von CIM geben. So unterteilt der CIM Standard beispielsweise in physische sowie logische Komponenten. Die genauen Möglichkeiten, Assoziationen zu knüpfen bleiben hier der Übersichtlichkeit halber weitestgehend außen vor. Sie werden bei Bedarf verwendet. Interessierte seien hier an [2] verwiesen.

Zum besseren Verständnis werden gegen Ende des folgenden Kapitels zwei einfache Beispieltoplogien mit CIM modelliert. Vorweg sollte angemerkt werden, dass im folgenden Attribute von Klassen nicht zwingend erst an jenen Stellen eingeführt werden, an denen sie vermerkt sind; vielmehr werden diese Klassen später instantiiert. Diese Vorgehensweise soll also lediglich der besseren Übersichtlichkeit dienlich sein.

4.1 Übersicht der Vererbungshierarchie

Ausgehend von der obersten Klasse, dem *ManagedElement*, wird von nun an jedes weitere Element abgeleitet. Abbildung 4.1 vermittelt einen ersten Eindruck über die wichtigsten Elemente der vorliegenden Vererbungshierarchie. Zu diesem Zeitpunkt findet schon die Aufsplittung in physisches und logisches Element statt. Alle weiteren Elemente besitzen vererbt vom *ManagedSystemElement* unter anderem einen Namen und ein Einrichtungsdatum. Auffällig ist darüber hinaus, dass dem *PhysicalElement* ein eindeutiges Schlüsselattribut zugewiesen ist, der Tag. Ein Schlüssel für *LogicalElement* wird hingegen erst späteren Generationen (im allgemeinen *Name*) verliehen. Der Schlüssel Tag soll in den Ausprägungen das Format Name-der-Klasse_Hostname_ID besitzen. Hierbei ist die ID eine Nummer, welche fortlaufend vergeben wird. Alle IDs sind eindeutig zu wählen, also auch in Bezug auf anderen Schlüsselattributen, welche eventuell erst später eingeführt werden.

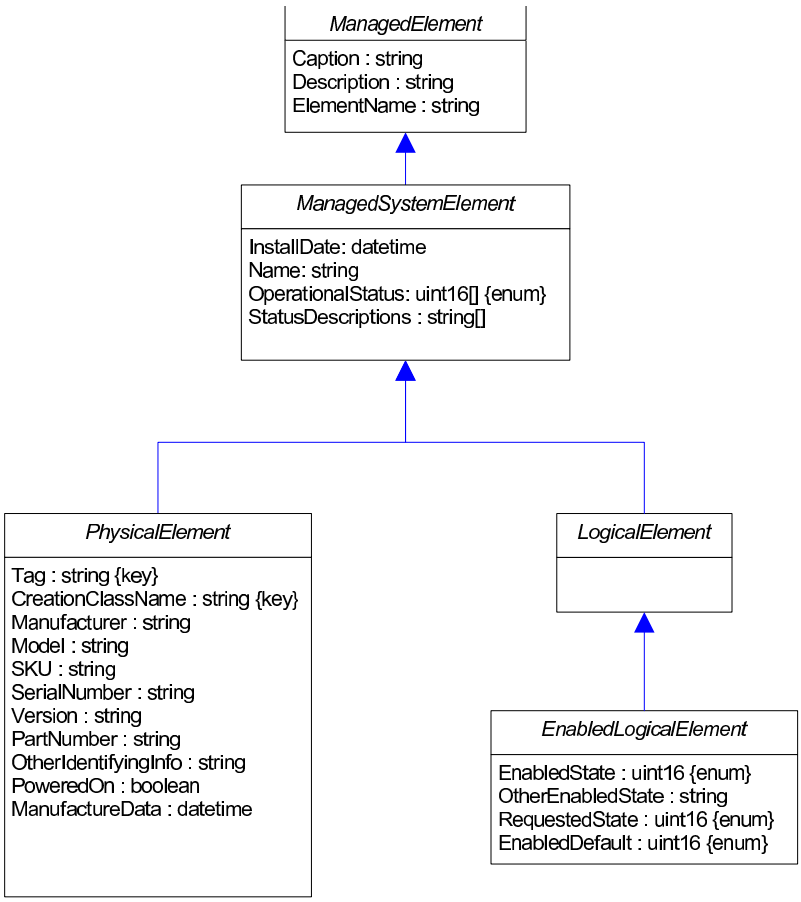

Abbildung 4.1: Übersicht der Vererbungshierarchie

4.2 Repräsentation der physischen Komponenten

Des Weiteren wird die weitere Vererbung der physischen Komponenten und die mögliche Repräsentierung durch Ausprägungen dieser Klassen betrachtet. Über den Weg Physical-Package und PhysicalFrame gelangt man zu Klassen, die später als Instanzen physische Teile repräsentieren, hier also *Chassis* und *Rack*. *Rack* besitzt unter anderem eine Höhe, Tiefe, Breite und ein Gewicht. Darüber hinaus besitzt ein *Rack* einen Herstellernamen und eine Modellbezeichnung; diese Eigenschaft hat *Rack* mit *Chassis* gemein.

Ein anderer Zweig führt zur Klasse *SystemBusCard*, eine PCI Ethernetkarte wäre beispielsweise eine solche *SystemBusCard*. Diese Instanz bietet Platz für Informationen zu Hersteller, Modell, Namen, *BusType* (PCI) und *BusWidth* (64 Bit).

Eine *SystemBusCard* steckt in einem *Slot*, mit einem bestimmten *ConnectorType* (z.B. PCI), einem *MaxDataWidth* (beispielsweise 64 Bit) und einer vom Motherboard festgelegt und eindeutigen Nummer.

Mittels eines *PhysicalLink*s erhält die *SystemBusCard* eine Verbindung zu anderen physischen Komponenten.

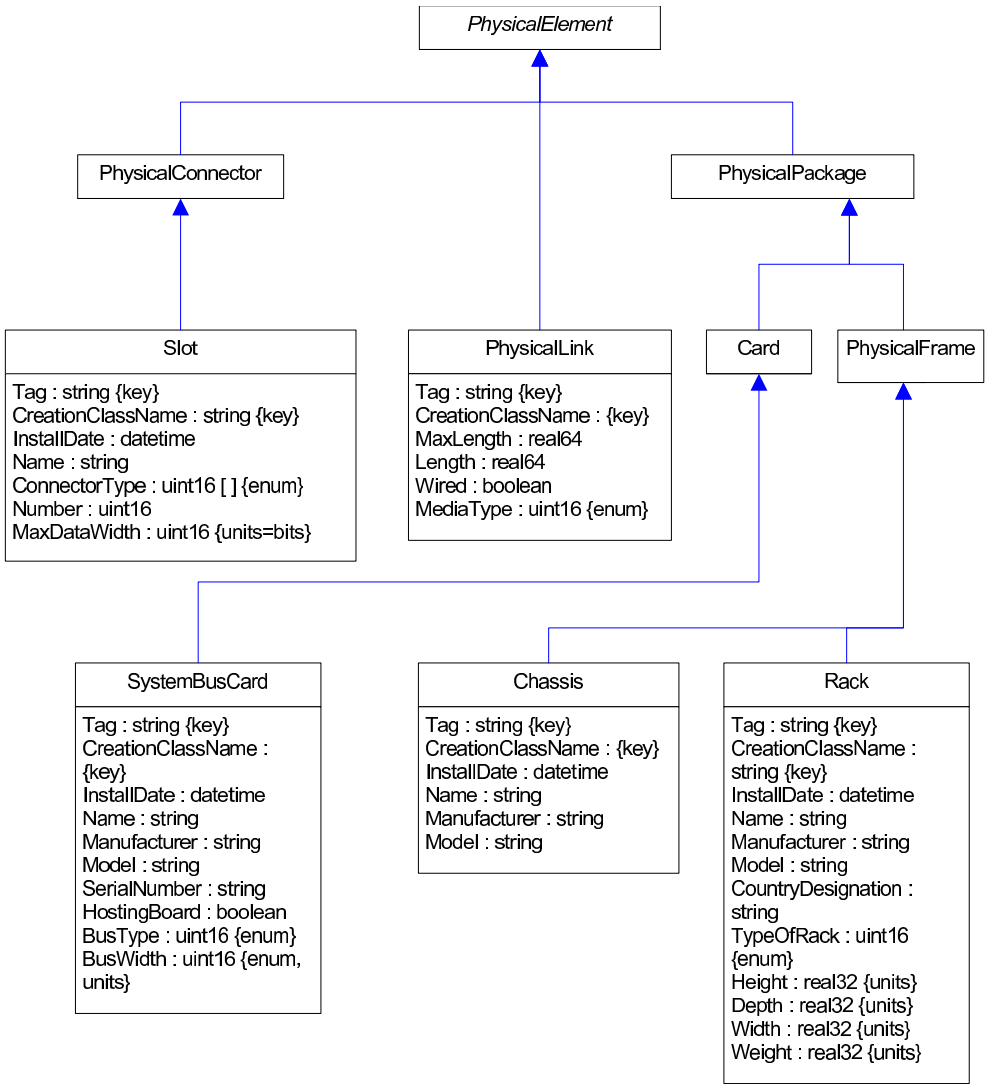

Abbildung 4.2: Managed Object PhysicalElement

4.3 Repräsentation der logischen Komponenten

Bei Betrachtung der folgenden logischen Komponenten fällt ins Auge, dass dem Attribut *Name* im Normalfall eine Schlüsselrolle zugewiesen wird; lediglich das *LogicalDevice* erhält einen eigenen Schlüssel, *DeviceID*.

Die Schlüssel *Name* bzw. *DeviceID* sollen, vergleichbar mit dem Schlüssel *Tag* der physischen Elemente, der Namensgebung Name-der-Klasse_Hostname_ID folgen, es sei denn, das Format wird eigens durch ein Attribut *NameFormat* spezifiziert. Die IDs werden auch hier eindeutig sein.

Darüber hinaus besitzen alle logischen Komponenten Attribute zur Erfassung des *PrimaryOwner* und des *PrimaryOwnerContact*.

Abbildung 4.3 gibt weiteren Aufschluss über die Klasse *ComputerSystem*, welche ein Computersystem im Allgemeinen repräsentiert, d.h. ein Layer-3-Switch wird durch einer Instanz der Klasse *ComputerSystem* repräsentiert, ebenso wie ein Router, eine Firewall oder ein Fileserver.

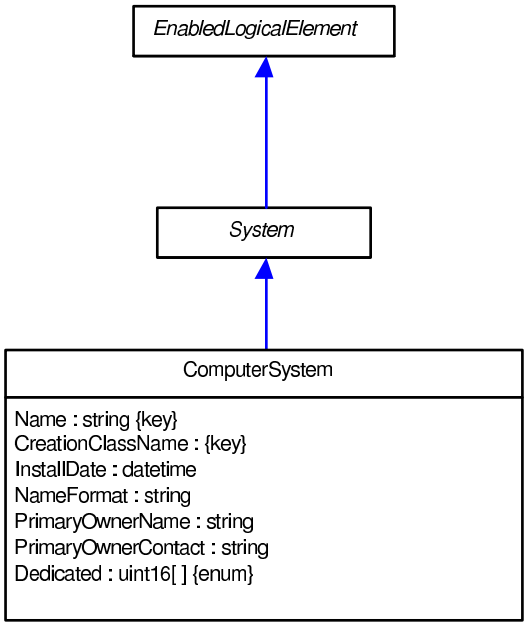

Abbildung 4.3: Managed Object ComputerSystem

Abbildung 4.4 zeigt ein *LogicalDevice*. Hierbei läßt eine Ausprägung dieser Klasse mit einem Systemdevice vergleichen. So wird beispielsweise das Linux Networkdevice eth0 mit einer Instanz dieser Klasse repräsentiert. Daher ist ein *LogicalDevice* nicht zwingend notwendig, beispielsweise besitzt ein Layer 3 Switch keine Instanz eines *LogicalDevice*. Wie bereits angedeutet erhält ein *LogicalDevice* neben *Name* das Schlüsselattribut *DeviceID*.

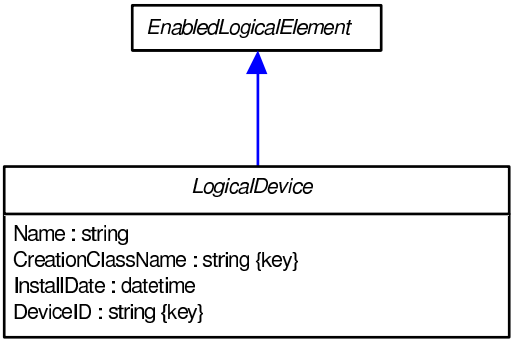

Abbildung 4.4: Managed Object LogicalDevice

Die nachfolgende Zeichnung 4.5 stellt einen *Service* dar. Damit wird jener Teil Software bezeichnet, welcher für die Kommunikation mit der Hardware zuständig ist. *Service* meint also beispielsweise einen Ethernet-Service oder dergleichen.

Darüber hinaus leitet sich von dieser Klasse *VLANService* und *SwitchService* ab. Ein *VLANService* entscheidet, welcher Port welchem VLAN angehört, ein *SwitchService* ist für das Durchleiten der Pakete erforderlich.

Wohl an dieser Stelle sollte auf eine Problematik hingewiesen werden. In unserem Modell wird ein Hub (also ein Multiport-Repeater) als *SwitchService* dargestellt. Dies mag zwar nicht recht der Semantik entsprechen, ist aber unabdingbar, will man mit CIM einen Hub modellieren, da es auf anderem Weg (noch) nicht möglich ist.

Ein weiteres Problem erwies sich bei Assoziationen der Instanz vom Typ *PhysicalLink* mit einer Instanz vom Typ *SwitchPort*. Für dieses Problem konnte keine Ableitung von *CIM_Dependency* gefunden werden. Soll auf eigene Anpassungen verzichtet werden, muss die Klasse *CIM_Dependency* direkt verwendet werden, wodurch aber die semantische Bedeutung auf der Strecke bleibt. Dieser Vorgehensweise wird sich auch bei den später besprochenen Beispielmodellierungen bedient.

In solchen Spezialfällen besitzt das Common Information Model also demnach noch Ungereimtheiten.

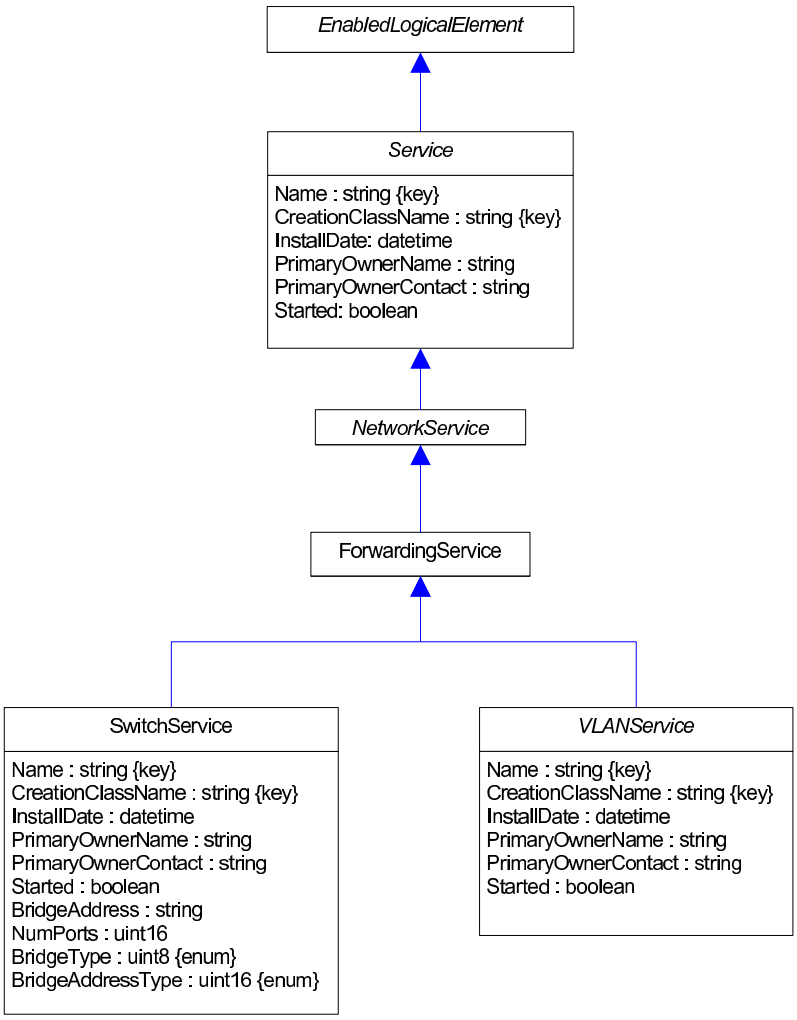

Abbildung 4.5: Managed Object Service

ServiceAccessPoints (4.6) öffnen, wie der Name schon vermuten läßt, den Zugang zu jeweiligen Services. Neben einer von *ServiceAccessPoint* direkt abgeleiteten Klasse *VLAN*, welche das jeweilige VLAN mit einer eindeutigen Nummer identifiziert, gibt es sogenannte Protokollendpunkte. Je nach betrachtetem Protokoll ist eine Klasse dafür zuständig. Die gängigsten sind dabei *LANEndpoint*, *IPProtocolEndpoint* und *SwitchPort*.

Der *LANEndpoint* erhält Information über die MAC Adresse, der *IPProtocolEndpoint* über die IP Adresse. Eine Instanz der Klasse *SwitchPort* repräsentiert den Port eines Switches.

Je nach Bedeutung können bestimmte *ProtocolEndpoint*s aneinandergehängt werden, so hängt ein *IPProtocolEndpoint* im Normalfall an einem *LANEndpoint*.

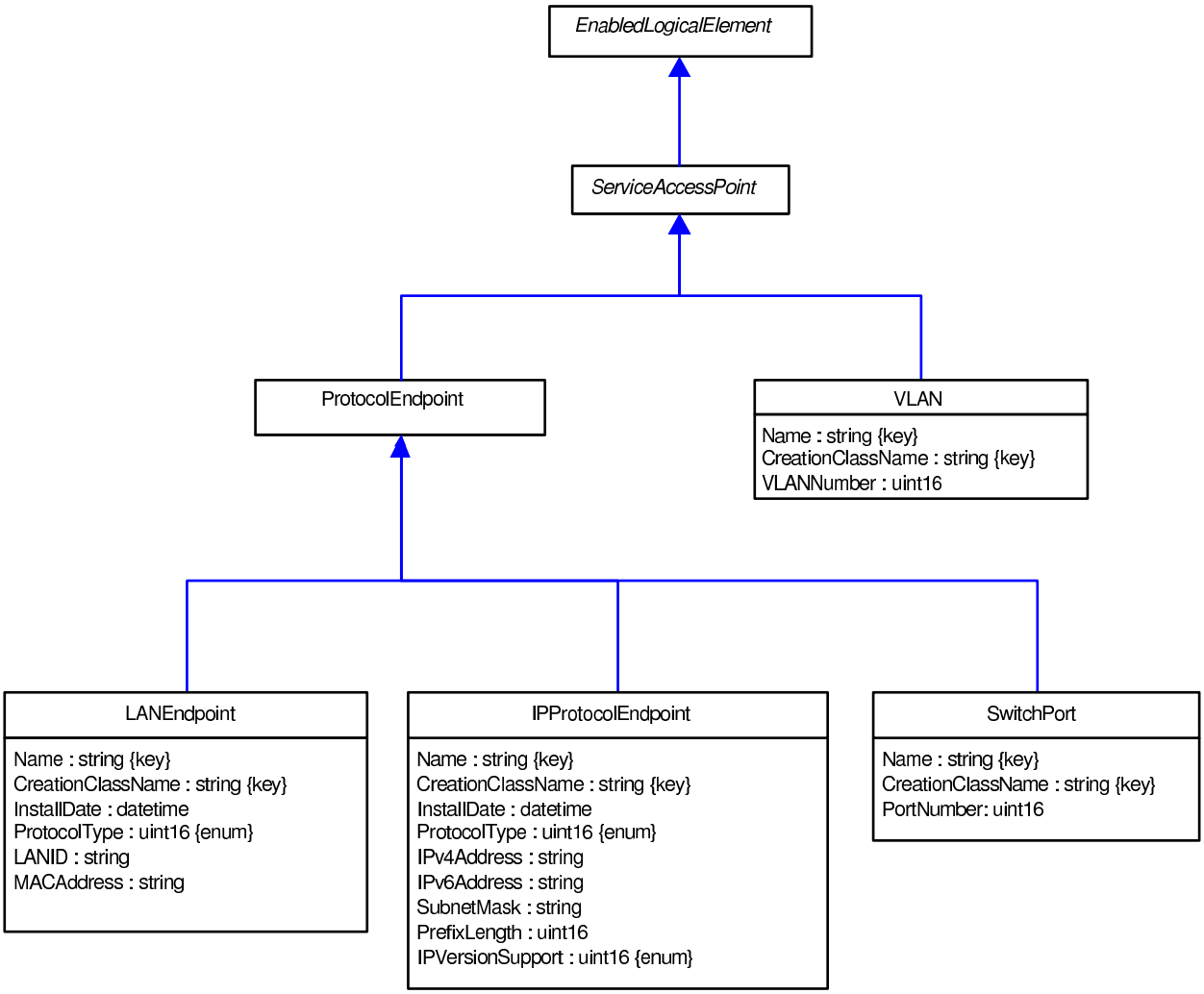

Abbildung 4.6: Managed Object ServiceAccessPoints

Kapitel 5

Beispielmodellierungen

Die obigen Ausführungen und Beschreibungen des CIM Standards, lassen schon vermuten, dass sich die Umsetzung relativ schwierig gestaltet.

Aus diesem Grund wurde auch die Ausführung über mögliche Assoziationen und Aggregationen zunächst ausgespart, da die genaue Verwendung nicht vorgegeben ist und sich keinesfalls einheitlich gestalten läßt (aus Sicht unterschiedlicher Anwendungsfälle).

Daher folgen Beispielmodellierungen, deren Ausführung ungefähren Aufschluß über die Verwendung von Instanzen, Assoziationen und Aggregationen in der Praxis gibt.

5.1 Beispielmodellierung Rechner mit Ethernet-Switch

Die nachfolgende Abbildung 5.1 stellt eine einfache Netzwerktopologie dar, deren Modellierung im weiteren Verlauf abgehandelt wird.

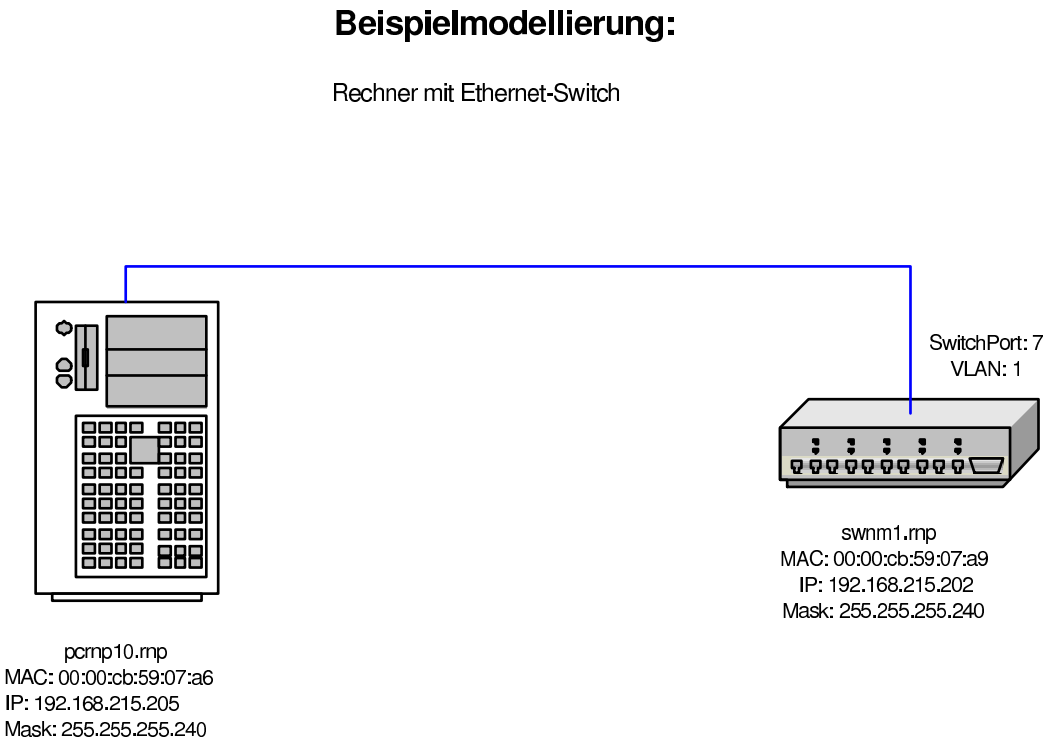

Abbildung 5.1: Beispielmodellierung Rechner mit Ethernet-Switch

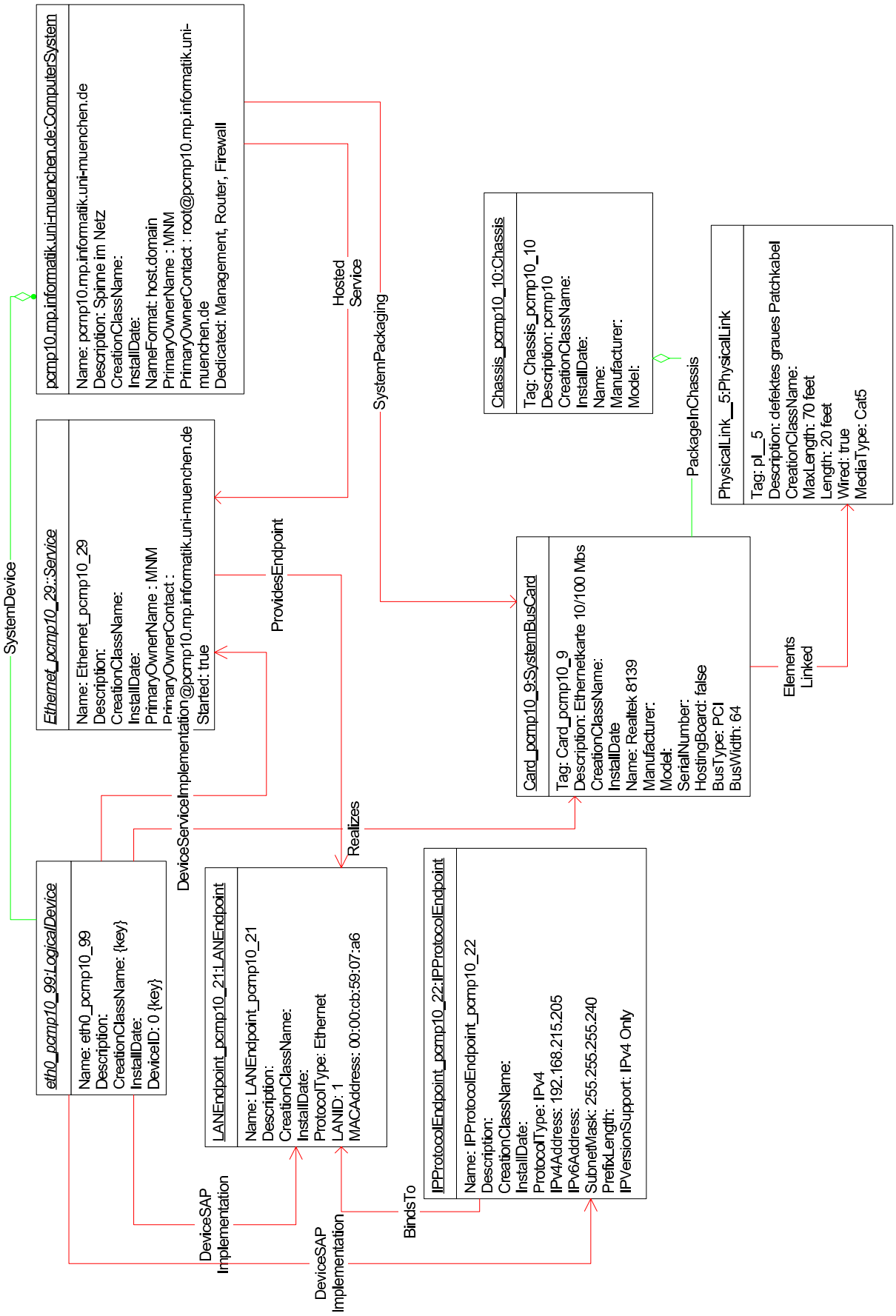

Abbildung 5.2: Beispielmodellierung pcmp10

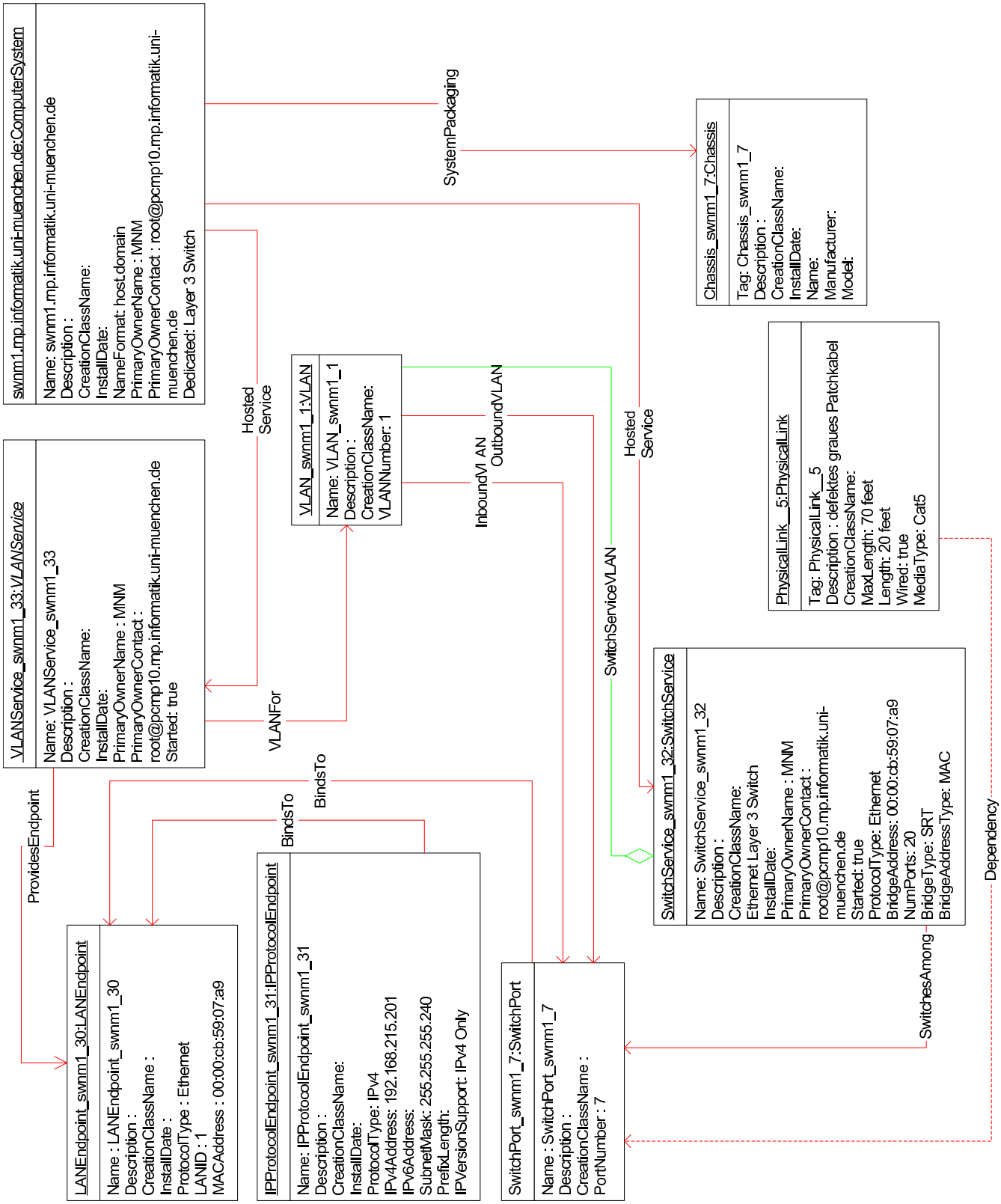

Abbildung 5.3: Beispielmodellierung swnm1

5.2 Beispielmodellierung 2

Die zweite Beispielmodellierung greift einen realen Versuch des Rechnernetzepraktikums auf und versucht eine anschauliche Modellierung. Vorab sei ein grober Überblick des Versuchaufbaus gegeben, es wird die erste Gruppe modelliert (5.4). Jedoch wird im folgenden lediglich der VLANSWITCH2 modelliert, da der VLANSWITCH1 lediglich für das IT-Sec Praktikum von Bedeutung ist.

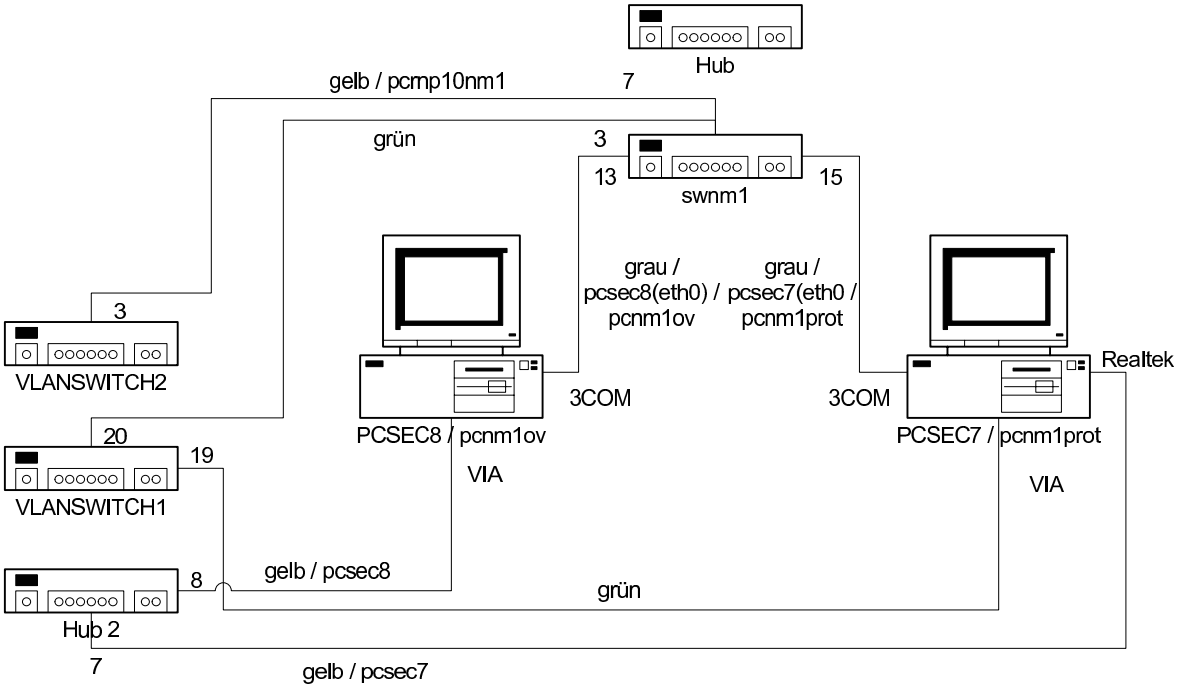

Abbildung 5.4: Beispielmodellierung des Netzmanagementversuchs

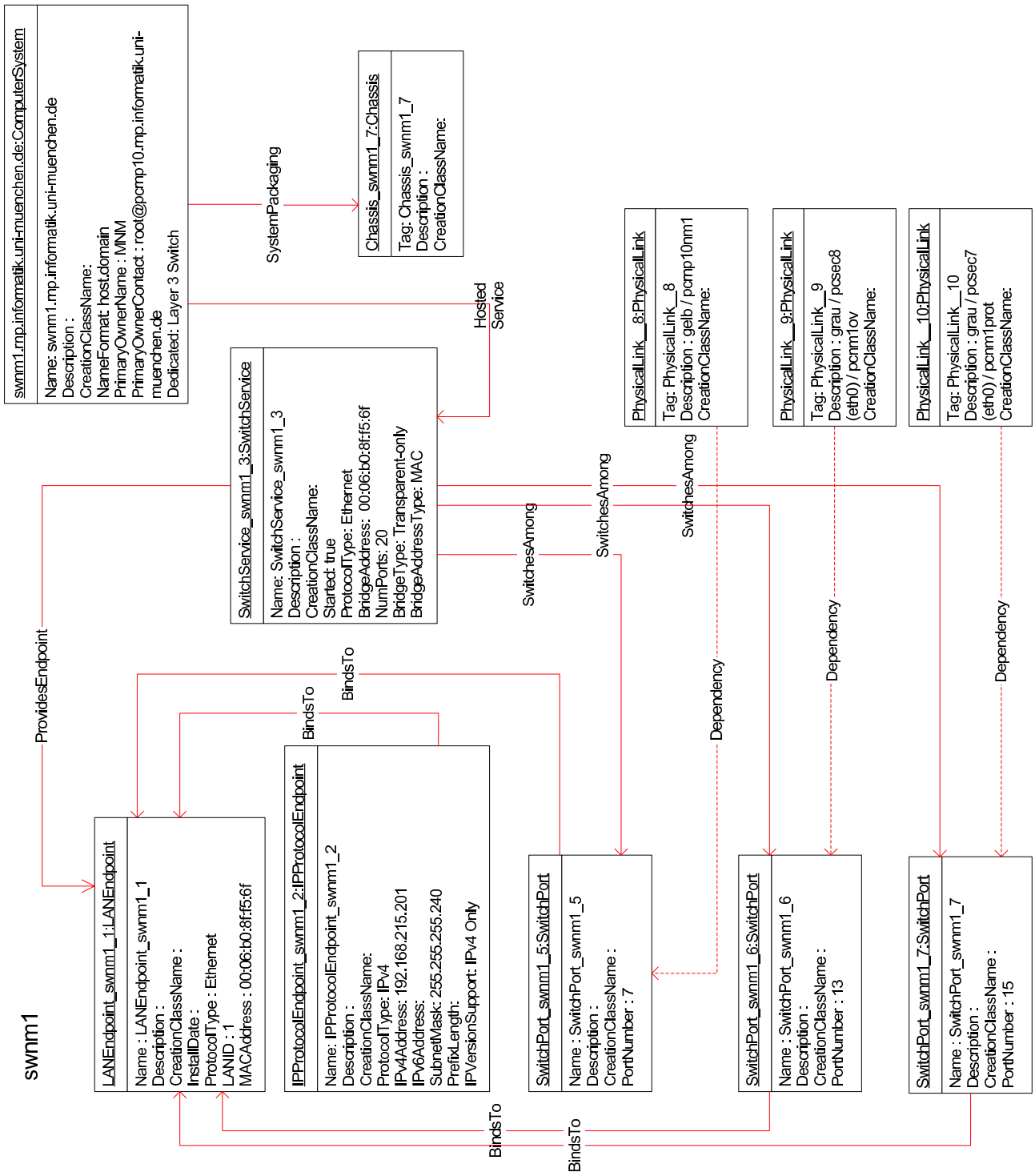

Abbildung 5.5: Beispielmodellierung swnm1

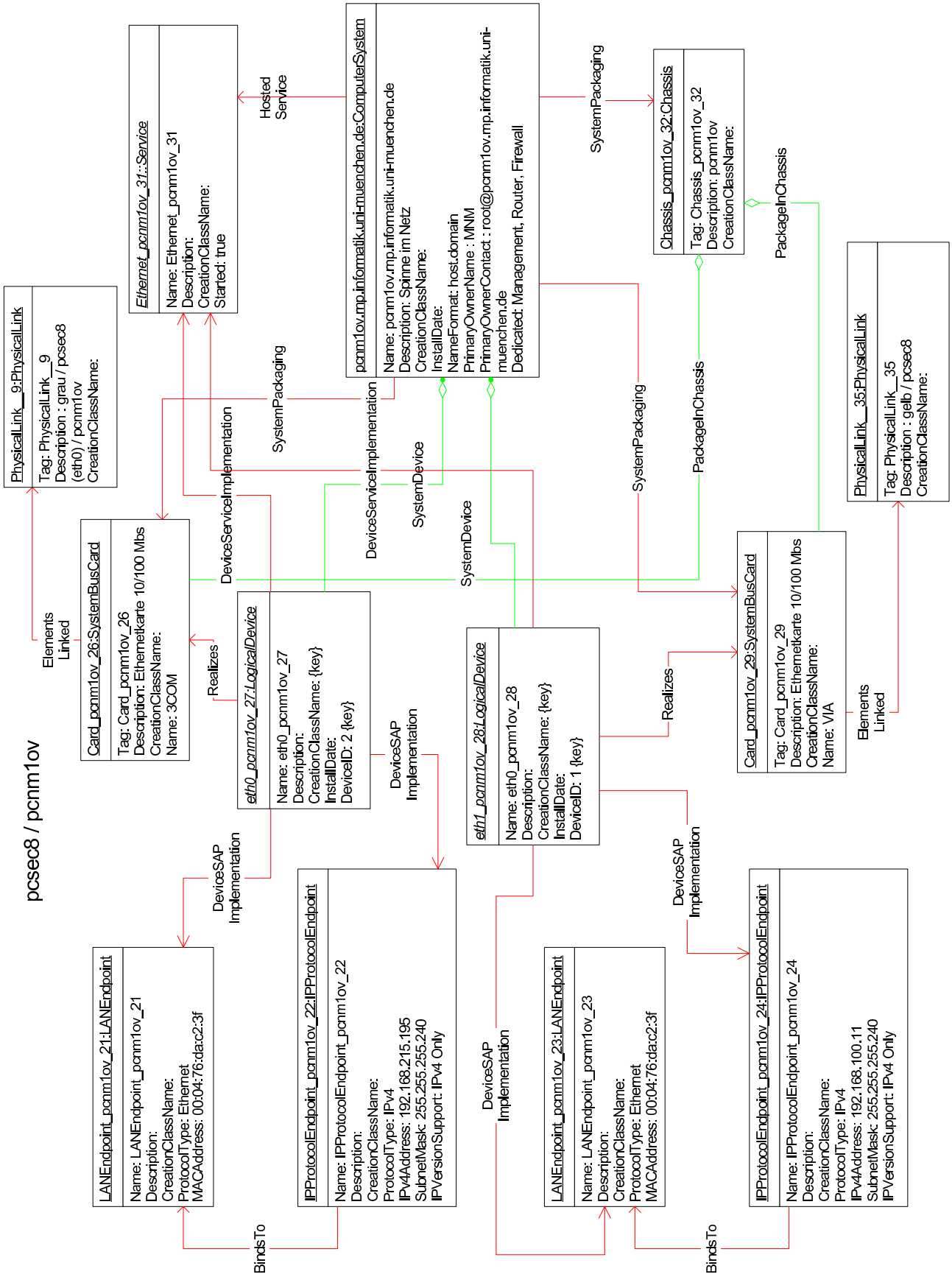

Abbildung 5.6: Beispielmodellierung pcnm1ov

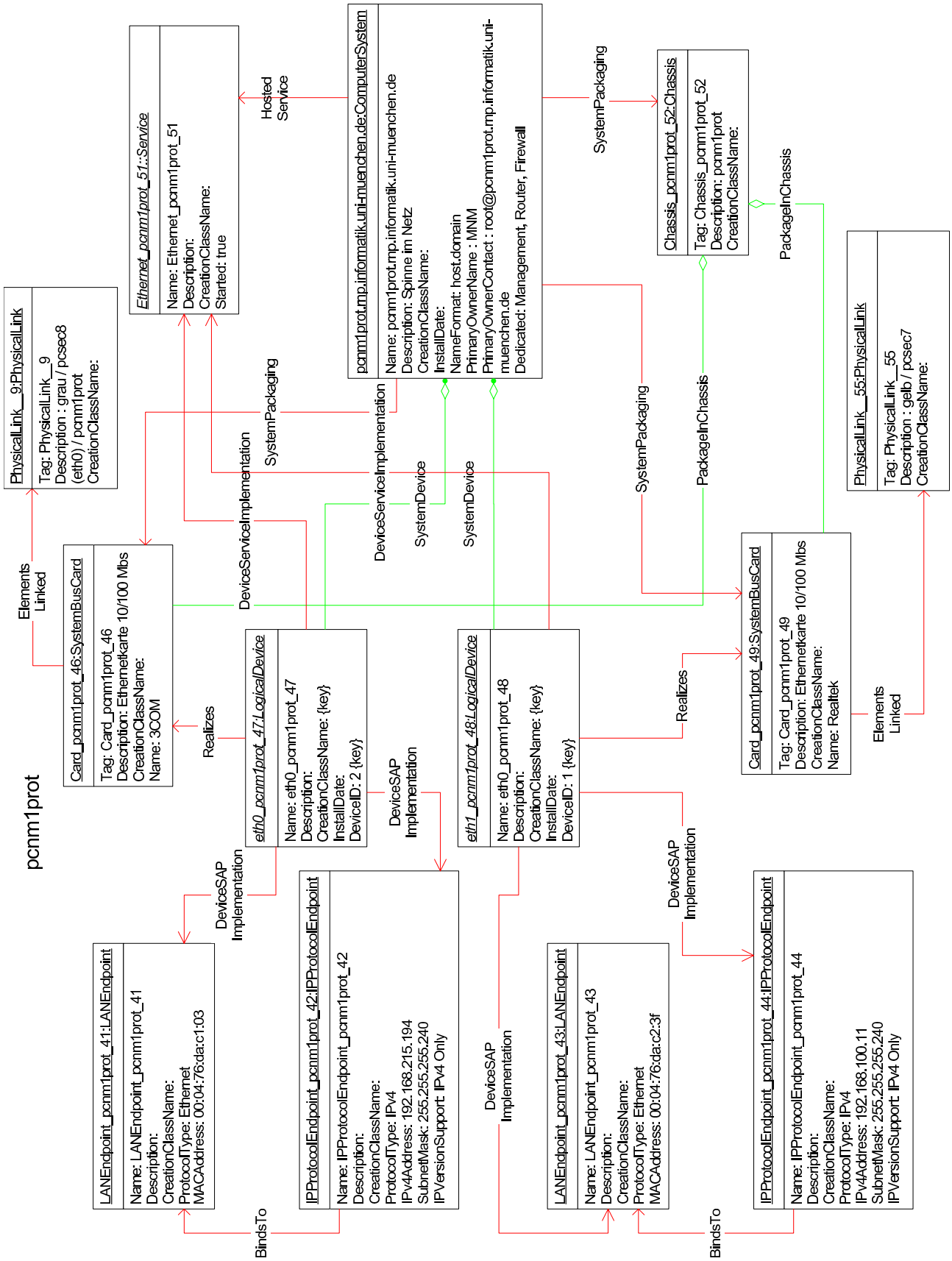

Abbildung 5.7: Beispielmodellierung pcnm1prot

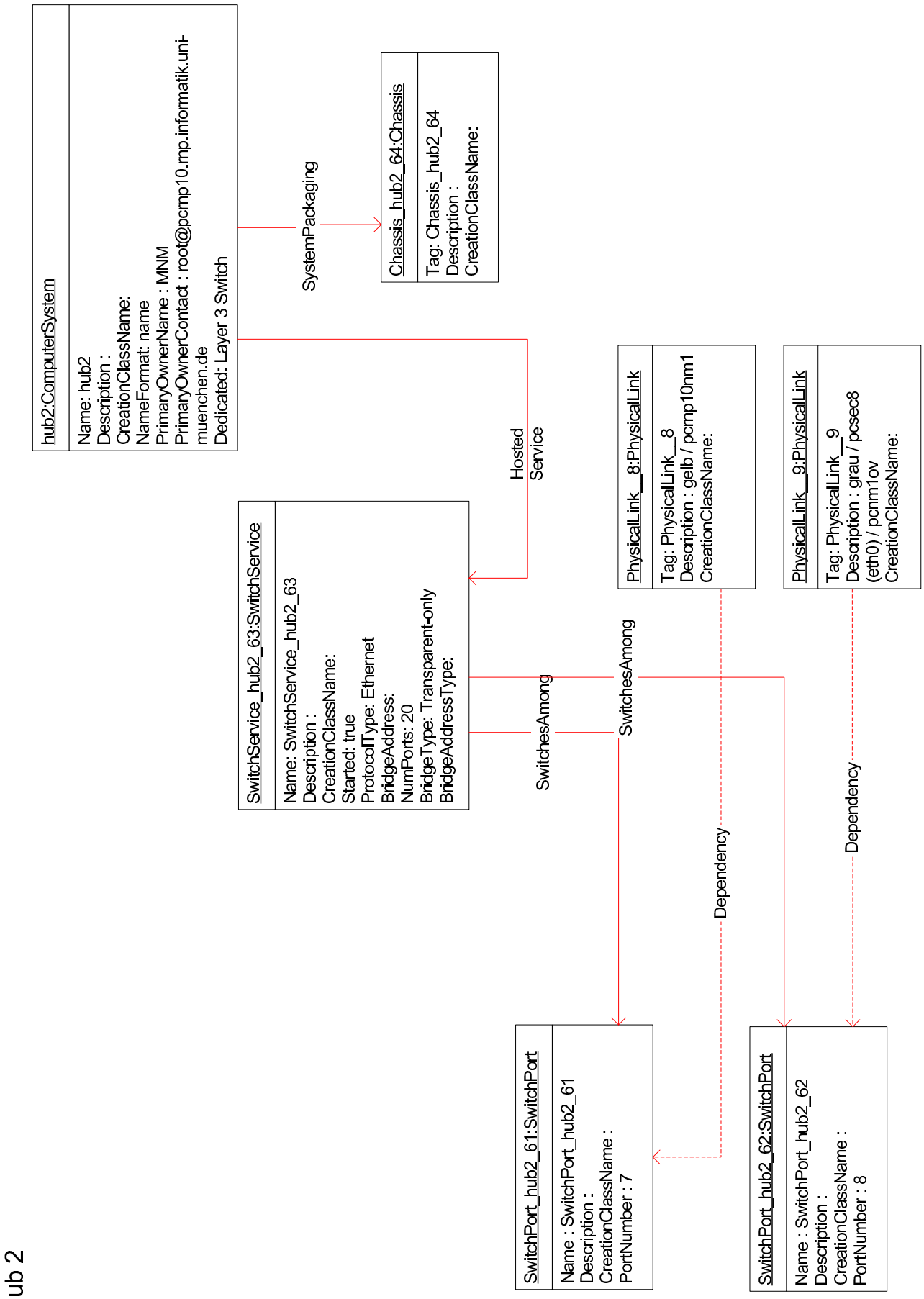

Abbildung 5.8: Beispielmodellierung Hub 2

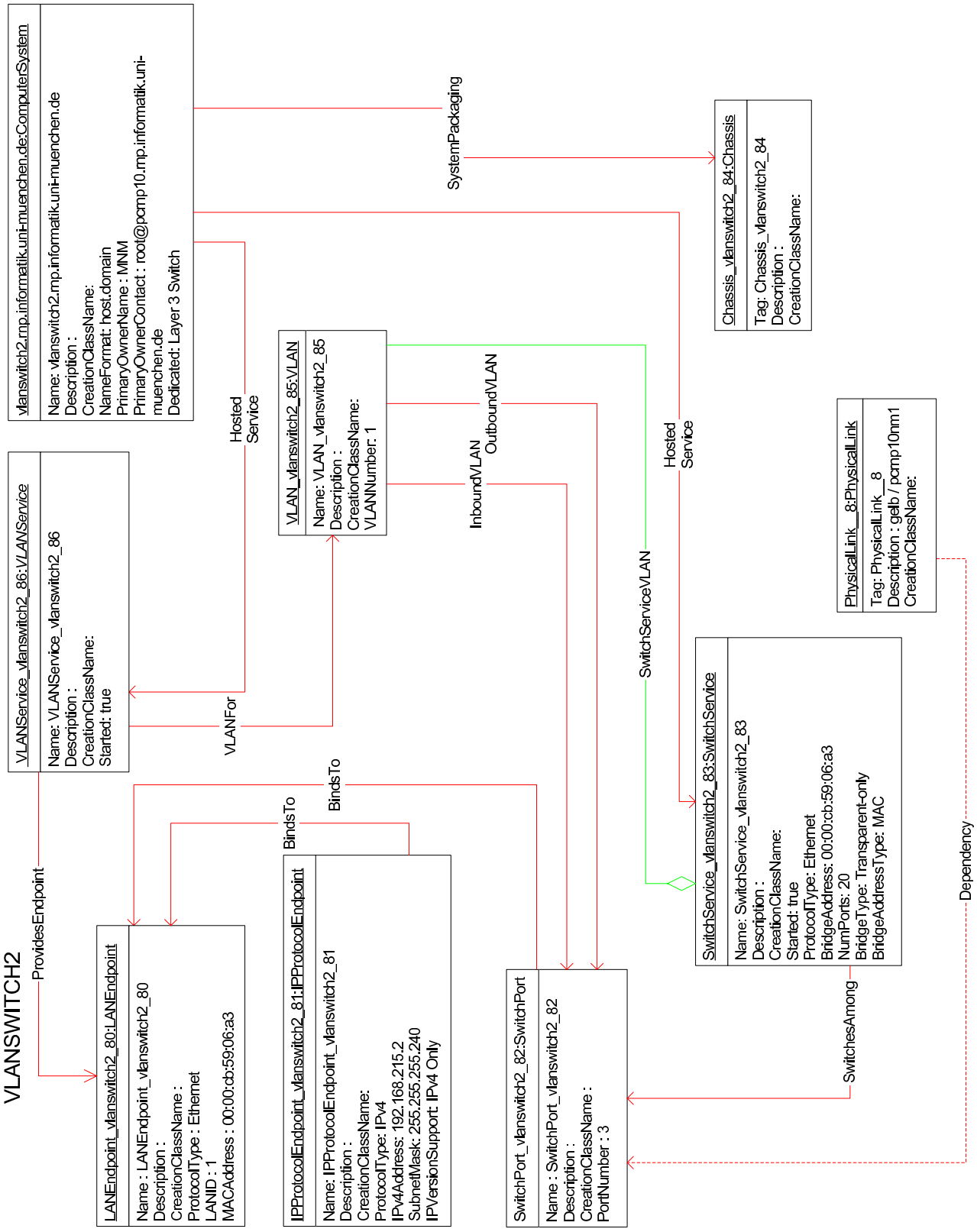

Abbildung 5.9: Beispielmodellierung VLANSWITCH 2

Es leuchtet ein, dass eine solche Verwaltung der Netzwerktopologie einigen Aufwand mit sich bringt. So bedeutete die Modellierung zweier Hosts verbunden durch einige Switches/Hubs schon einen recht hohen Aufwand. Die Modellierung des Versuchaufbaus erstreckt sich dabei über fünf Seiten, eine einzige Ansammlung von Instanzdiagrammen, eine Praxisrelevanz zeigt ein solches Modell in dieser Form also nicht.

Kapitel 6

Fazit

Bleibt nun noch ein abschließendes Resümee der vorliegenden Arbeit zu ziehen, welche sich mit den Fähigkeiten des Common Information Models beschäftigt und den Bezug zur „Rechnernetze Realität“ darstellt.

CIM erhebt den Anspruch, eine hohe Plattformunabhängigkeit zu realisieren. Wohl auch daher sind die meisten bisherig verfügbaren Produkte in Java implementiert, was tatsächlich eine gewisse Plattformunabhängigkeit garantiert. Brauchbare Produkte sind vor allem auf der Serverseite zu finden; jede betrachtete Implementierung ist für sich durchaus als ausgereift zu bezeichnen, obwohl noch einige Kompatibilitätsprobleme vorhanden sind. Hier sind zwei Punkte relevant, zum einen Unterschiede bei der CIM-over-HTTP Implementierung, zum anderen bei der Providerimplentierung. Die erfolgten CIM-over-HTTP Implementierungen halten sich im Normalfall an die vorgegebene API; in Einzelfällen unterscheidet sich jedoch die unterstützte Schnittstelle, wenn auch nur um Nuancen. Hingegen verhält sich die Sache bei der Providerimplementierung dramatischer. Bei einer Umsetzung eines Projektes ist durchaus damit zu rechnen, dass Providerimplementierungen stark von der verwendeten Plattform abhängen. Auch müssten vorhanden Provider im Falle einer Migration auf einen anderen CIM Object Manager entsprechend angepasst werden, da Providervorlagen prinzipiell je nach Hersteller variieren.

Im Normalfall wird diese Problematik indes so gelöst, dass auf eine Datenbank festgelegt und das entstehende Produkt daraufhin ausgerichtet wird, also eine vergleichbare Vorgehensweise zu herkömmlichen Datenbanken.

Ein ganz entscheidender Punkt des Common Information Modell ist die herstellerneutrale Verwaltung heterogener Netzwerke. Diese Eigenschaft ist jedoch nicht ganz umsonst, jedem zu verwaltenden Objekt müssen die CIM Fähigkeiten nachträglich gegeben werden. Dies bedeutet, dass für jedes Betriebssystem und jedes Objekt eigene Routinen zur Verfügung gestellt werden müssen, welche sich um die Umsetzung in CIM kümmern. So wäre es beispielsweise wünschenswert, wenn das betrachtete Objekt gleich selbst solche Routinen mitbringt, um sich beim CIM Object Manager ordnungsgemäß anzumelden, was so aber

noch nicht der Fall ist.

Auf der Clientseite sieht es dagegen nicht derart rosig aus. Derzeit sind noch keine brauchbaren, auf CIM ausgerichteten Managementwerkzeuge auf dem Markt. In Bezug auf das Rechnernetzepraktikum wäre es sinnvoll, zwei Arten von Clients zu betrachten. Zum einen würde ein Client benötigt, welcher dem Systemadministrator zur Hand geht, Objekte zu managen und zu verwalten. Diese Managementsoftware sollte eine intuitiv bedienbare, graphische Oberfläche bieten. Die derzeitigen Standardlösungen die CIM zur Datenhaltung verwenden, setzen jedoch noch große Kenntnisse der CIM Modellierung voraus. Die verfügbaren, generischen CIM Browser bieten zwar Grundoperationen, die durchaus sinnvoll implementiert sind, jedoch fehlen Funktionen zum bequemen Managen der Objekte. So wäre beispielsweise eine übersichtliche Ansicht der Topologie wünschenswert. Dies ist aber derzeit noch Zukunftsmusik. Auch fehlt die weitereführende Unterstützung aller CIM Object Manager Eigenschaften. So gibt es keinerlei Funktionalität, Methodprovider aufzurufen, die Funktion kann bislang nur über „pures“ XML erfolgen. Die Eigenschaften der CIM Object Manager werden derzeit also noch nicht völlig ausgeschöpft.

Zum anderen müssten Clients automatisch auf Veränderung im Topologiesystem reagieren. Es ist sinnvoll, diese beiden Gattungen zu unterscheiden, da solche „Agenten“ auf mehreren Systemen verteilt zum Einsatz kommen und so die Integrität des Datenbestands garantieren. Die Datenintegrität ließe sich ebenfalls durch Verwendung von Providern herstellen. Hier würde die Datensammlung zentral vom CIM Object Manager gesteuert. Den Providern müssten Fähigkeiten zum entfernten Managen und Abfragen von Komponenten gegeben werden. Im einen Fall würde also eine PULL-Operation, im anderen eine PUSH-Operation erfolgen.

Ein Pluspunkt sammelt CIM bei der gelungenen XML Unterstützung. Der Pegasus CIMOM speichert sein CIM Repository bereits in reinem XML, bei WBEMServices und SNIA Cimom ist dies leider nicht der Fall.

Das Common Information Model wurde daraufhin ausgelegt, für möglichst alle gängigen Plattformen verwendbar zu sein. Daher erklären sich auch die umfangreichen Spezifikationen. Der Preis für diese Plattformunabhängigkeit ist aber eine hohe Komplexität bei der Anwendung. Die zweite Beispielmodellierung gibt hier einen leisen Einblick. Es wurde lediglich versucht, einen Netzmanagement Versuch des Rechnernetzepraktikums zu modellieren; dabei wurde bewußt auf das ITSec Praktikum verzichtet, obwohl hier dieselben Rechner zum Einsatz kommen. Das Resultat sind zahlreiche Instanzen, welche sich in dieser Fassung auf fünf Seiten erstrecken. Es ist leicht einsichtig, dass dieses Modell als kompliziert zu erachten ist, zumal die Instanzen darüber hinaus beliebig verknüpft werden können. CIM gibt hier keine Vorgaben, in welcher Weise die Assoziations- und Aggregationsklassen zu verwenden sind. Die Gefahr, dass sich hier ein Fehler einschleichen könnte ist immens.

Betrachtet man hingegen den Aufbau des Rechnernetzepraktikum und ITSicherheitspraktikum, so scheint dieser Aufwand kaum gerechtfertigt. Viele Attribute der verwendeten

Instanzen würden unbesetzt bleiben, das Potenzial welches CIM bietet, würde hier also keinenfalls ausgeschöpft.

Für eine Netzwerktopologie dieser Größe ist das CIM Modell demnach nicht praktikabel. Darüberhinaus müssten zahlreiche Eigenentwicklungen angefertigt werden, welche dieses Modell umsetzen. Diese beiden Sachverhalte würden demnach erhöhten Aufwand bedeuten.

Wird eine solche Lösung trotzdem angestrebt, so sind im folgenden die weiteren Schritte nötig.

- Entwicklung von Werkzeugen zum Einfangen der Managementinformationen
- Entwicklung einer intuitiv benutzbaren Managementanwendung

Wäre das Einfangen von Managementinformationen noch zu bewältigen, wenn auch mit relativ großem Aufwand, würde auf der anderen Seite eine Managementsoftware nötig, welche die erhaltenen Daten auswertet und geeignet darstellt. Demnach würde der zweite Punkt hingegen eine hohe Entwicklungsarbeit fordern. Die Forderungen nach Komfort und Intelligenz der Managementsoftware, die selbständig Fehlerfälle findet und eventuell auch auflöst, sind hoch gegriffen.

Die letztliche Folgerung ist, dass die nächsten Schritte der Hersteller abzuwarten sind, inwieweit ihre Produkte in der nächsten Zeit CIM Unterstützung erhalten. Dies wäre die Voraussetzung dafür, dass CIM für den Enduser attraktiv wird.

Quellenverzeichnis

[1] H.-G. Hegering, S. Abeck, B. Neumair. *Integriertes Management vernetzter Systeme. Konzepte, Architekturen und deren betrieblicher Einsatz.* dpunkt-Verlag, Heidelberg, 1999

[2] `http://www.dmtf.org/` (Website der *DMTF*)

[3] *Common Information Model (CIM) Specification, Version 2.2.* DMTF, 1999 (`http://www.dmtf.org/standards/standard_cim.php`)

[4] *Specification for the Representation of CIM in XML, Version 2.1.* DMTF, 2002 (`http://www.dmtf.org/standards/standard_wbem.php`)

[5] *Specification for CIM Operations over HTTP, Version 1.1.* DMTF, 2002 (`http://www.dmtf.org/standards/standard_wbem.php`)

[6] `http://wbemservices.sourceforge.net/` (Website des *WBEM Services* Projekts)

[7] `http://www.opengroup.org/snia-cimom/` (Website des *SNIA Open Source CIMOM* Projekts)

[8] `http://gcc.gnu.org` (Website der *GNU Compiler Collection*)

[9] `http://www.openpegasus.org/` (Website des *OpenPegasus* Projekts)

[10] `http://www.cimnavigator.com/` (Website des *CIM Navigator* Projekts)