INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Anwendungsintegration am Beispiel einer Managementanwendung für ein verteiltes Batch-System

Clemens Durka

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Friedrich Ferstl, Genias Software
	Dr. Bernhard Neumair
	René Wies
Abgabedatum:	15. August 1995

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15. August 1995

Inhaltsverzeichnis

Kapitel 1

Einleitung

Durch die fortschreitende Umstellung der EDV-Landschaft auf Workstationcluster werden die zur Verfügung stehenden Ressourcen weniger ausgelastet, da die Benutzer geneigt sind, ihre Programme lokal auf ihrer Maschine zu rechnen, selbst wenn im selben Workstationverbund einige Rechner gerade wenig ausgelastet sind.

Hier setzen verteilte Batchsysteme an. Sie nehmen Aufträge aus dem Workstationnetz entgegen und rechnen sie auf Rechnern, die gerade wenig Last haben bzw. auf denen keine interaktiven Benutzer die Ressourcen beanspruchen. Durch diese Verteilung und den damit verbundenen Lastausgleich können rechenintensive Aufträge schneller gerechnet werden als auf der belasteten lokalen Maschine. Ebenso ist eine Verteilung der Aufträge auf Zeiten, in denen die Rechner wenig benutzt werden wie nachts oder am Wochenende, möglich.

Die vorliegende Diplomarbeit beschäftigt sich mit dem Management eines solchen verteilten Batchsystems. Bisher werden bei solchen Batchsystemen einige Programme zum Management mitgeliefert. Sinnvoller ist es allerdings, das Management von Batchsystemen in das integrierte Netz-, System- und Anwendungsmanagement von Managementplattformen aufzunehmen, um alle Managementaufgaben von einem Ort aus und unter einer einheitlichen Oberfläche zu steuern und durch Abgleich der Daten Inkonsistenzen zu vermeiden.

Diese Möglichkeit wird in der Diplomarbeit untersucht, wobei Konzepte zur Integration am Beispiel der Managementplattform HP OpenView und des verteilten Batchsystems Codine erarbeitet und eine MIB für Codine vorgestellt werden.

Kapitel 2

Szenarien

Während des Codine-Anwendertreffens bei der GWDG in Göttingen hatte ich die Möglichkeit, die Vorstellung einiger Installationen von Codine in verschiedenen Instituten und Firmen zu verfolgen. In diesem Kapitel werden drei dieser Beispielinstallationen von Codine vorgestellt und die Verwendungsmöglichkeiten klassifiziert. Daraus werden dann Managementanforderungen abgeleitet.

2.1 Installationen von Codine

2.1.1 Codine bei der GWDG

Die GWDG (Gesellschaft für wissenschaftliche Datenverarbeitung) in Göttingen betreibt derzeit die größte Codine-Installation in Deutschland für ca. 5000 Benutzer der Universität Göttingen und der Max-Planck-Gesellschaft. Ziel des Einsatzes von Codine ist eine möglichst hohe Auslastung des Workstationverbundes, der aus ca. 30 CPUs, vor allem DEC- und IBM-Rechnern, besteht. Dieser Workstationverbund wird dediziert für das Batchsystem verwendet, es werden keine oder kaum interaktive Benutzer zugelassen. Die erreichte Auslastung beträgt nach Aussage des Betreibers 74 % für den reinen Batchbetrieb. Dem Benutzer werden je nach Laufzeit und Hauptspeicheranforderung seines Programms mehrere Batch-queues angeboten, wodurch gewährleistet ist, daß seine Programme auf den entsprechenden Maschinen gerechnet werden. Auf Grund der Abwesenheit interaktiver Benutzer stellt sich die Installation nach außen fast wie ein Großrechner dar. [GENI 95a]

2.1.2 Codine bei VW-Gedas

Die Installation bei VW-Gedas, einer Tochtergesellschaft von Volkswagen in Wolfsburg, die die Anwender der VW AG im Bereich Hardware und Software betreut, betreibt eine Codine-Installation auf 22 SGI-Rechnern. Da die verschiedenen Computer hier Arbeitsplatzrechner sind, die in verschiedenen Abteilungen stehen und jeweils einem Benutzer zugeordnet sind, soll das Batchsystem nur dann in Aktion treten, wenn der interaktive Benutzer die Maschine gerade nicht benötigt. Somit ist das Ziel hier, den interaktiven Benutzer möglichst wenig zu beeinflussen und dabei trotzdem die Zeiten zu nutzen, in denen der Rechner nicht beschäftigt ist. Sobald der interaktive Benutzer den Rechner verwendet, werden die Queues suspendiert und aus dem Hauptspeicher entfernt. Um trotzdem keine zu großen Wartezeiten bei der Abarbeitung der Jobs durch die Unterbrechung eines interaktiven Benutzers zu haben, wird Checkpointing unterstützt. Das bedeutet, daß der gesamte Programm- und Datenbereich eines Prozesses auf Hintergrundspeicher gesichert wird (Prozeßimage). Dann kann das Batchsystem dem Prozeß einen neuen Rechner zuordnen, auf dem der Prozeß weitergerechnet wird. Durch diese Prozeßmigration kann auch nach der Verteilung der Jobs noch die Last der Rechner berücksichtigt und dynamisches Loadbalancing realisiert werden [GENI 95a].

2.1.3 Codine beim MPI Mainz

Das Max-Planck-Institut für Polymerforschung in Mainz verwendet Codine auf 4 DEC-Alpha-Rechnern und erlaubt sowohl interaktive Benutzung als auch Batchverarbeitung für größere Berechnungen. Dieser Mischbetrieb von Codine und Interaktion hat die Effektivität des Clusters verbessert. Durch die Gleichberechtigung der beiden Zugriffsarten wird erreicht, daß interaktive Benutzer die Rechner benutzen und trotzdem Batchjobs weiterlaufen können, ohne auf das Freiwerden der Rechenleistung eine unbestimmte Zeit warten zu müssen [GENI 95a].

2.2 Klassifikation der Verwendungsmöglichkeiten von Codine

Wie man anhand der in Abschnitt 2.1 vorgestellten Codine-Installationen feststellt, gibt es verschiedene Zielsetzungen bei einem Einsatz von Codine. Um für jede der Einsatzmöglichkeiten die Managementanforderungen formulieren zu können, ist es sinnvoll, zu klassifizieren. Dabei sollen vor allem die beiden Extreme, der rein dedizierte Einsatz und der Einsatz bei Vorrang interaktiver Benutzung untersucht werden und daraus die Anforderungen für ein Management entwickelt werden. Ein Einsatz im Mischbetrieb wird entsprechend neben den gemeinsamen Anforderungen Aspekte beider Varianten enthalten.

2.2.1 Dedizierter Workstationcluster

Die Installation bei der GWDG entspricht einem Einsatz auf einem dedizierten Workstationcluster. Die Zielsetzung wird bei einem solchen Einsatz eine möglichst gute Auslastung der Systeme sein, unabhängig von der Tageszeit und der Anzahl der momentanen Benutzer. Die Warte- und Antwortzeit spielt hier eine geringere Rolle (bei der GWDG muß man beispielsweise in der Regel doppelt so lange warten, wie der Job rechnet), und im Normalfall werden in den Queues immer Aufträge zur Abarbeitung warten.

2.2.2 Arbeitsplatzrechner mit interaktiven Benutzern

Eine andere Zielsetzung hat die Installation bei VW, die als Beispielinstallation für eine Clusterkonfiguration mit interaktiven Benutzern angesehen werden kann. Da die Besitzer von Workstations, die ihnen alleine zugeordnet sind, die Rechenzeit ihrer Rechner nur abtreten werden, wenn sie dadurch nicht in ihrer Arbeit behindert werden und im Idealfall nichts von der zusätzlichen Last bemerken, liegt der Schwerpunkt hier auf der statischen und dynamischen Lastbalancierung. Unter statischer Lastbalancierung versteht man die Zuteilung der Jobs auf Rechner, die im Moment wenig Last haben. Da sich diese Last ändern kann (zum Beispiel durch Interaktion eines Benutzers), muß die Last auch dynamisch, d. h. auch nach der Zuteilung der Jobs noch überwacht und eventuell neu verteilt werden. Dies wird durch eine Abschaltung der Queues bei interaktiver Last erreicht. Wenn – wie bei VW – auch noch Checkpointing unterstützt werden kann, dann können diese Prozesse auch auf andere Rechner migrieren und müssen nicht auf das Wiederanlaufen der Queue nach Ende der Interaktion warten.

2.3 Managementaufbau

2.3.1 Dedizierter Workstationcluster

Aus der Sicht eines Managers entspricht ein dedizierter Cluster eher einem Großrechner oder einem Parallelrechner. Die Installation wird meist als Ganzes gesehen, d. h. Managementaufgaben beziehen sich häufiger auf den gesamten Cluster als auf einen speziellen Rechner der Installation. Weiterhin werden Zugriffe auf die Installation meist von außen, also von einem Rechner, der nicht am Batchsystem beteiligt ist, stattfinden. Dies spricht für die Auswahl eines zentralen Punktes, an dem das Managementsystem mit dem Batchsystem in Kontakt tritt. Sinnvollerweise wird das der Hauptserver des Batchsystems sein. Dieser übernimmt dann alle Kommunikation mit den Managementanwendungen und gibt Anfragen und Anforderungen, die er nicht selber erfüllen kann, an den entsprechenden Rechner über ein eigenes Protokoll weiter, über das er auch die Antworten zurückgibt.

2.3.2 Arbeitsplatzrechner mit interaktiven Benutzern

Bei einer Installation mit interaktiven Benutzern tritt der Aspekt der Verteiltheit, der Heterogenität der Rechner und der Zuständigkeiten verschiedener Besitzer stärker in den Vordergrund. Dadurch ergibt sich eine andere Sicht des Managements, das mehr auf die beteiligten Hosts bezogen ist. Es wird mehr Anfragen geben, die sich nur auf einzelne Rechner beziehen, wie z. B. eine Anfrage nach der Auslastung aller Rechner in einem

Zuständigkeitsbereich. Diese Anfragen können durchaus auch von Rechnern kommen, die Teil des Batchsystems sind oder die – von der Netztopologie gesehen – näher an den zu managenden Rechnern stehen als der Hauptserver, der ja auch in einem anderen Subnetz liegen kann. Daher ist es sinnvoller, mehrere Kommunikationspunkte zwischen den Managementanwendungen und dem Batchsystem einzurichten. Somit bietet sich hier an, jedem am Batchsystem beteiligten Rechner eine Kommunikationsschnittstelle für das Management mitzugeben. Dadurch werden die Übertragungszeiten des Netzes für den Fall entfernter Server umgangen, und Anfragen können innerhalb einer Hierarchiestruktur lokal bearbeitet werden.

Kapitel 3

Anforderungen

3.1 Integration

Das Management einer verteilten Anwendung soll in das integrierte Netz-, System- und Anwendungsmanagement einer Managementplattform einbezogen werden. Da die Fülle unterschiedlicher Managementaufgaben bisher zu einer Fülle von unterschiedlichen Werkzeugen geführt hat und dies auch eine große Anzahl verschiedener Schnittstellen zum Zugriff auf Managementfunktionen zur Folge hatte, soll durch eine Integration eine Vereinheitlichung des Managements erreicht werden. Hierbei ergeben sich die folgenden Anforderungen an eine Integration:

- Die verschiedenen Managementfunktionen und der Zugriff auf die enthaltenen Objekte sollen über einheitliche Schnittstellen ansprechbar sein.
- Zur Vermeidung redundanter und inkonsistenter Daten sollen Informationen, die von verschiedenen Werkzeugen benötigt werden, nur einmal gespeichert und bei Bedarf von dort angefordert werden.
- Unterschiedliche Sichten und abgestufte Zugriffsrechte sollen entsprechend dem Status des Benutzers unterstützt werden.
- Eine einheitliche Benutzerschnittstelle soll für den Zugriff auf alle Managementfunktionen bestehen.
- Das gesamte Management eines Netzes inklusive der laufenden Anwendungen soll von einer Person von einem Ort aus geschehen.

3.2 Grundanforderungen

Aus Gesprächen mit den Betreibern der Batchsysteme des Leibniz-Rechenzentrums in München und mit verschiedenen Benutzern von Codine während des Codine-Anwendertreffens in Göttingen hat sich folgendes Bild herauskristallisiert:

- Alle Betreiber von Batchsystemen benutzen bisher zum Management die mit dem jeweiligen System mitgelieferten Programme und haben entsprechend ihrer Bedürfnisse noch einige eigene Programme geschrieben, die spezielle Anforderungen erfüllen.
- Nur wenige Installationen benutzen eine Managementplattform zum Netzmanagement.
- Die Personen, die für die Batchsysteme zuständig sind, beschäftigen sich häufig nicht mit dem Netzmanagement.

Daraus folgt, daß eine wichtige Anforderung bei der Integration des Managements in eine Plattform der Erhalt der bisherigen Managementfunktionalität ist. Das bedeutet, daß alle von den bisher mitgelieferten Managementwerkzeugen erfüllten Funktionen auch von einer Plattform aus ausführbar sein müssen. Diese Funktionen sind in Tabelle A.1 auf Seite 68 zusammengestellt. Die Anforderungen der Betreiber der Batchsysteme unterscheiden sich je nach Installation, wobei folgende Funktionen häufig ausgeführt werden.

Diese Funktionen betreffen folgende Bereiche:

- Konfigurationsmanagement
  - Installation der Software
  - Konfiguration der beteiligten Rechner (Hinzufügen, Entfernen, Ändern der Konfiguration)
  - Konfiguration der beteiligten Queues und Queuefamilien
- Fehlermanagement
  - Starten und Stoppen der Queues
  - Starten und Stoppen der beteiligten Server und Clients
  - Reaktion auf bestimmte Ereignisse und Überschreiten bestimmter Grenzwerte, die von der Plattform bereitgestellt werden
- Leistungsmanagement
  - Überwachung der allgemeinen Auslastung, des Arbeitsaufkommens und der Antwortzeiten.
  - Berücksichtigung von äußeren Parametern bei der Verteilung der Jobs (installierte Software, verfügbarer Plattenplatz)

- Abrechnungsmanagement
  - Accounting
  - Benutzermanagement
- Sicherheitsmanagement
  - Zugriffsschutz

3.3 Datenintegration

Eine weitere Anforderungen ist, Daten, die sowohl in der Managementumgebung als auch im Batchsystem zur Verfügung stehen, nur einmal zu speichern, um Inkonsistenzen zu vermeiden. Hierbei kann man folgende Daten unterscheiden:

- Allgemeine Daten des Netzmanagements

  Allgemeine Daten des Netzes, die dem Netzmanagement zur Verfügung stehen, sind in der Regel für mehrere Anwendungen in dem Bereich interessant und sollten daher von der Managementplattform verwaltet und exportiert werden. Hierzu zählen z. B. Rechnernamen, IP-Adressen, Speicherausstattung, installierte Software ...

  Das Batchsystem soll bei der Verwendung dieser Daten auf die Datenbank der Plattform zurückgreifen.

- Daten des Batchsystems

  Die Daten des Batchsystems, die nicht von der Plattform kommen, sollen in einer möglichst universellen Weise anderen Managementanwendungen zur Verfügung stehen. Hierzu ist eine MIB (Management Information Base) zu definieren, in der die Daten gehalten werden, da es bisher noch keine standardisierte MIB dafür gibt. Wenn es möglich ist, sollte diese MIB so allgemein gehalten werden, daß sie einfach auf andere verteilte Batchsysteme übertragen werden kann.

3.4 Weitere Managementszenarien

Neben den in Abschnitt 3.2 auf der vorherigen Seite genannten Anforderungen, die zum größten Teil bereits heute mit den codine-eigenen Werkzeugen, die in Tabelle A.1 auf Seite 68 aufgeführt sind, realisiert werden können, kann man sich auch noch weitere Managementszenarien vorstellen, deren Realisierbarkeit im Rahmen eines integrierten Managements wünschenswert wäre. Dies geht aber über eine Integration bestehender Produkte hinaus, da hierzu zum Teil größere Änderungen in Codine vorgenommen werden müssen. Sie werden aber trotzdem hier aufgeführt, damit bei verschiedenen Managementkonzepten eine mögliche Erweiterbarkeit und Realisierbarkeit dieser Anforderungen untersucht werden kann.

3.4.1 Änderung der Rechnerkonfiguration

Bei einer Änderung der Rechnerkonfiguration, also dem Hinzufügen oder Entfernen eines Rechners aus einem Cluster, wäre es praktisch, wenn man diesen Rechner nicht getrennt in der Konfiguration des Netzmanagements und des Batchsystems behandeln müßte. Durch ein Attribut innerhalb von OpenView, das angibt, ob ein Rechner Codine unterstützt, soll es möglich sein, nach dem Hinzufügen eines Rechners und dem Einrichten in der Plattform die Umkonfigurierung zur Integration des Rechners in das Batchsystem automatisch von der Plattform vorzunehmen. Diese übernimmt dann das Aufspielen der Software und das Hinzufügen in den entsprechenden Konfigurationsdateien.

3.4.2 Lastsenkung bei interaktiver Last

Bei interaktiver Last sollte es eine Vielzahl von Möglichkeiten geben, das Batchsystem über die Plattform zu beeinflussen. So könnte man sich verschiedene Strategien zur Lastsenkung vorstellen, die wählbar sind und die Wahl des jeweiligen Grenzwertes ermöglichen.

- Nach dem Einloggen eines neuen Benutzers wird das Batchsystem für eine bestimmte Zeit suspendiert. Falls dann die Last unter einem bestimmten Lastwert liegt, wird die Arbeit wiederaufgenommen.
- Die Last darf benutzerabhängig einen bestimmten Wert nicht überschreiten, wobei z. B. Studenten weniger Last zugestanden wird als Assistenten.
- Die Lastgrenze soll tages- und tageszeitabhängig sein.

3.4.3 Beschränkungen für bestimmte Benutzergruppen

Neben den oben genannten, nach Benutzergruppen verschiedenen Lastgrenzen kann man sich noch weitere Attribute vorstellen, die von der Art des Benutzers abhängen:

- die maximale CPU-Zeit, die dem Benutzer zur Verfügung steht,
- den maximalen Speicherbedarf eines Jobs,
- die Anzahl der gleichzeitig in der Queue stehenden Jobs,
- eine voreingestellte Priorität, die die Jobs erhalten.

Diese Attribute sollten von der Plattform aus einsehbar und einstellbar sein.

3.4.4 Abhängigkeiten von äußeren Bedingungen

Der Start bestimmter Jobs kann von Bedingungen außerhalb des Batchsystems abhängig sein, z. B. von der Verfügbarkeit bestimmter Software, Hardware oder bestehenden Netzverbindungen. Hier ist eine Zusammenarbeit mit der Plattform besonders sinnvoll, da sie über Netzverbindungen und die zur Verfügung stehenden Ressourcen Bescheid weiß. Diese Information soll der Scheduler vor dem Start eines Jobs berücksichtigen und mit den Anforderungen für den zu startenden Job vergleichen. Stellt er fest, daß nicht alle Bedingungen für einen Jobstart vorhanden sind, wird ein anderer Job zum Start bestimmt.

Als Bedingung sollen alle Informationen der Managementplattform zulässig sein wie der Inhalt von MIB-Variablen oder der Status einer bestimmten Komponente, der von der Managementplattform regelmäßig abgefragt wird. Als Beispiel einer MIB-Variablen ist es vorstellbar, daß Lizenzserver, die eine bestimmte Anzahl Lizenzen eines Produkts zur Verfügung haben, ihre Informationen ebenfalls in einer MIB über SNMP zur Verfügung stellen. So kann die Plattform und das Batchsystem die Anzahl der freien Lizenzen abfragen und entsprechend reagieren.

3.4.5 Unterstützung von Wartungsarbeiten

Bei Wartungsarbeiten, die regelmäßig zu festen Terminen stattfinden, müssen in der Regel die Rechner heruntergefahren und somit alle Anwendungen unterbrochen werden. Zu diesem Zeitpunkt sollten die Queues leergelaufen sein, um zu vermeiden, daß die im Moment laufenden Jobs abgebrochen werden. Falls Checkpointing unterstützt wird, ist es relativ einfach, alle Jobs zu checkpointen und nach den Wartungsarbeiten fortzusetzen. Ansonsten ist für ein Verfahren zu sorgen, daß vor dem entscheidenen Zeitpunkt nur noch Jobs zur Ausführung kommen, die noch rechtzeitig beendet werden können. Die Schwierigkeit hierbei ist es, im voraus festzustellen, welcher Job wieviel Rechenzeit beanspruchen wird.

3.4.6 Zeitmanagement

Ein weiterer Wunsch, der von Betreibern von Batchsystemen geäußert wurde, ist, das Batchsystem in ein Zeitmanagement zu integrieren, so daß in Abhängigkeit der Tageszeit und des Wochentags das Batchsystem in einer anderen Konfiguration läuft. So ist es z. B. sinnvoll, an Werktagen während der Arbeitszeit weniger Queues anzubieten oder dem Batchsystem größere Beschränkungen aufzuerlegen, um die interaktiven Benutzer nicht zu stören. Nachts, an Wochenenden und Feiertagen sollen dann verstärkt Batchsysteme insbesondere für rechenintensive Jobs genutzt werden.

Das einzige Werkzeug, das auf Unixsystemen vorliegt und eine Art Zeitmanagement realisieren kann, ist allerdings das Unix-Werkzeug `cron`, das zu bestimmten Uhrzeiten Programme starten kann. Damit sind die erwähnten Anforderungen nur schwer und umständlich zu realisieren.

Es besteht zwar die Möglichkeit, solch einen Mechanismus in das Batchsystem zu integrieren. Dann ist er aber nur innerhalb des Batchsystems verfügbar. Da auch noch andere Anwendungen wie Backup-Programme ein mächtigeres Zeitmanagement nutzen könnten, wäre es denkbar, dies im Rahmen der Managementplattform anzubieten. Das Batchsystem sollte dann über die nötigen Schnittstellen verfügen, um in ein solches Zeitmanagement einbezogen zu werden.

3.4.7 Zusammenarbeit mit der Managementplattform

Bei einer Integration eines isolierten Werkzeugs in eine Managementplattform ist es natürlich wünschenswert, daß nicht nur eine Vereinheitlichung der Benutzeroberfläche und der Datenbasen erfolgt, sondern daß auch die in der Plattform vorhandene Infrastruktur genutzt werden kann.

So sollen die Basisanwendungen[1], wie die Zustandsüberwachung, die Schwellwertüberwachung, das Ereignismanagement, die Konfigurationsanwendungen und die Leistungsüberwachung, auch für die integrierte Anwendung zur Verfügung stehen. Damit werden folgende Managementaufgaben realisierbar:

- Mit Hilfe der Schwellwertüberwachung können für Daten des Batchsystems Grenzwerte definiert werden, bei deren Überschreitung Aktionen ausgelöst oder Traps gesendet werden.
- Das Ereignismanagement kann vom Batchsystem erzeugte Traps auf dem gleichen Wege wie andere Traps weiterverarbeiten.
- Über die Konfigurationsanwendungen können alle Komponenten des Netzes und des Batchsystems einheitlich konfiguriert werden.
- Accountinginformationen des Batchsystems können einem integrierten Accounting zugeführt werden, so daß die Gesamtheit der benutzten Ressourcen abgerechnet werden kann.
- Die Leistungsüberwachung kann auf das Batchsystem erweitert werden, und Leistungswerte des Batchsystems können mit Werten des restlichen Netzes in Korrelation gebracht werden.

[1] Diese Basisanwendungen werden in Abschnitt 4.1.4 genauer vorgestellt

Kapitel 4

Analyse der beteiligten Komponenten

4.1 HP OpenView

4.1.1 Überblick

Managementplattformen gewinnen mit der fortschreitenden Standardisierung von Managementprotokollen und -informationen immer mehr an Bedeutung. Gegenüber traditionellen isolierten Managementwerkzeugen stellen Managementplattformen eine Infrastruktur bereit, in die verschiedene Managementanwendungen eingebettet werden können. Sie laufen in offenen Systemumgebungen, bieten eine grafische Oberfläche, integrieren mehrere Funktionsbereiche und sind modular aufgebaut. Managementplattformen existieren von verschiedenen Herstellern, die bekanntesten Produkte sind wohl HP OpenView, Cabletron Spectrum, SunNet Manager und IBM NetView.

4.1.2 Nodemanager

Der zentrale Teil der Benutzeroberfläche von OpenView ist der Nodemanager. Der Nodemanager stellt die in der OpenView-Datenbank enthaltenen Objekte, die den Komponenten des Netzes entsprechen, in verschiedenen Submaps dar. Eine Hierarchie von Submaps stellt eine Map dar, in der eine ausgezeichnete Submap die Rootmap ist. Durch Anklicken von Objekten einer Submap können jeweils neue Submaps angezeigt werden. So können, von der Rootmap ausgehend, durch Anwählen der Submaps immer detailliertere Darstellungen des zu managenden Netzes und dessen Komponenten angezeigt werden.

Beim Laden einer Map werden alle in der Map enthaltenen Objekte in der Datenbank angelegt. Darüberhinaus bietet OpenView auch die Möglichkeit, selber eine Map anzulegen, indem es versucht, alle im Netz vorhandenen Ressourcen zu finden und zu identifizieren.

4.1.3 OpenView Datenbank

In der OpenView Datenbank werden alle Objekte der Managementplattform gehalten. Ein Objekt ist dabei eine interne Repräsentation einer physischen oder logischen Einheit oder Ressource, die im Netz existiert, und besteht aus einem eindeutigen Objekt-Identifikator und einigen Feldern (*fields*), die die Charakteristiken des Objekts beschreiben. Ein Teil dieser Objektattribute beschreibt dabei die *Capability* eines Objekt (wie z. B. `isNode`, `isDevice`, `isComputer`, `isSNMPSupported`). Darüber lassen sich Auswahlkriterien für bestimmte Operationen definieren, um Aktionen nur für passende Objekte zu erlauben. So können z. B. SNMP-Managementfunktionen nur auf Objekte angewandt werden, die die SNMP-Capability haben.

Diese *Fields* werden entweder beim Start von OpenView aus *Field Registration*-Dateien eingelesen oder mit API-Funktionen von einer Managementanwendung innerhalb OpenView erzeugt. Das Erzeugen von Objekten geschieht ebenfalls mit API-Funktionen.

4.1.4 Basisanwendungen der Managementplattform

Managementplattformen wie OpenView bieten eine Reihe von Basisanwendungen an, die vor allem dazu dienen, Informationen von Netzkomponenten und Endsystemen zu visualisieren.

Zustandsüberwachung

Zur Überwachung der Ressourcen pollt die Managementstation regelmäßig bestimmte Attribute dieser Ressourcen, um Informationen über den Zustand zu erhalten. Die Realisierung hängt dabei vom angewendeten Managementprotokoll ab. Bei Testprotokollen wie ICMP sendet die Station laufend Testpakete und registriert deren Antworten, woraus auf die Verfügbarkeit des entsprechenden Systems geschlossen werden kann. Ähnlich geschehen Abfragen über SNMP, der Manager sendet einen Request und wartet auf die Antwort. Da SNMP aber auf dem verbindungslosen Protokoll UDP beruht, kann aus einem Nichteintreffen einer Antwort nicht auf einen Ausfall des Systems geschlossen werden. Dies sichert eine Überwachung mit einem verbindungsorientierten Managementprotokoll wie CMIP, was aber auch einen höheren Aufwand verlangt.

Schwellwertüberwachung

Neben der allgemeinen Verfügbarkeit bieten Managementsysteme auch die Überwachung bestimmter Attribute auf Über- oder Unterschreiten von Schwellwerten. Diese Schwellwerte werden in der Managementstation definiert. Die Managementplattform liest dann regelmäßig die entsprechenden Werte ein und vergleicht sie mit den Schwellwerten. Bei deren Überschreitung werden Traps ausgelöst, die weitere Aktionen verursachen können.

Ereignismanagement

Ereignisse sind sowohl interne Traps, die andere Module der Managementplattform wie die Schwellwertüberwachung liefern, als auch Traps, die von externen Komponenten über Managementprotokolle geliefert werden. Das Ereignismanagement übernimmt das Sammeln dieser Ereignisse und sorgt für deren Weiterverarbeitung. Dies kann durch Abspeichern in Logdateien, durch Melden an den Benutzer, durch Weiterleiten an Anwendungen oder durch Ausführen von Aktionen geschehen. Hierzu werden in Konfigurationsdateien die Reaktionen, die für bestimmte Ereignisse oder Ereignisklassen definiert sind, abgelegt.

Konfigurationsanwendungen

Über Konfigurationsanwendungen können Informationen über Objekte angezeigt und modifiziert werden. OpenView stellt Anwendungen zur Verfügung, mit denen SNMP-Tabellen und einfache SNMP-Variablen übersichtlich dargestellt und modifiziert werden können. Mit Hilfe des MIB-Browsers können alle Variablen einer MIB ausgelesen werden.

Leistungsüberwachung

Ähnlich der Schwellwertüberwachung können auch Leistungsmessungen durchgeführt werden. Hierzu werden die zu überwachenden Attribute, die Meßintervalle und die Meßdauer festgelegt. Die Ergebnisse werden in Dateien protokolliert und können zu einem späteren Zeitpunkt ausgewertet werden.

4.1.5 Schnittstelle für Erweiterungen

OpenView bietet für die verschiedenen Bereiche APIs, mit denen eigene Anwendungen unter Zuhilfenahme von Teilen von OpenView realisiert werden können. So existieren APIs für den Zugriff zur Datenbank, zur Kreation und Interaktion mit Submaps und zum Management mit SNMP.

4.2 Codine

4.2.1 Funktionsweise von Codine

Codine ist ein verteiltes Batchsystem, das Rechenaufträge in Form von Unix-Shellskripten annimmt und diese auf einem Workstationcluster bearbeiten läßt. Das Batchsystem berücksichtigt bei der Verteilung der Aufträge die Last der jeweiligen Rechner und versucht, wenig benutzte Maschinen zur Abarbeitung der Aufträge zu verwenden. Hierbei werden die *Workload* (die Last der jeweiligen Maschine) und die Tatsache, daß ein interaktiver Benutzer an dem Rechner ist, bei der Verteilung berücksichtigt. Wenn sich die Lastsituation an einem Rechner ändert, z. B. durch Beginn der Arbeit eines interaktiven Benutzers, stoppt Codine die Abarbeitung seiner Aufträge und veranlaßt bei bestimmten Programmen die Migration der Prozesse auf andere Rechner.

Somit realisiert Codine Lastbalancierung und Prozeßmigration innerhalb eines Workstationverbundes.

4.2.2 Aufbau von Codine

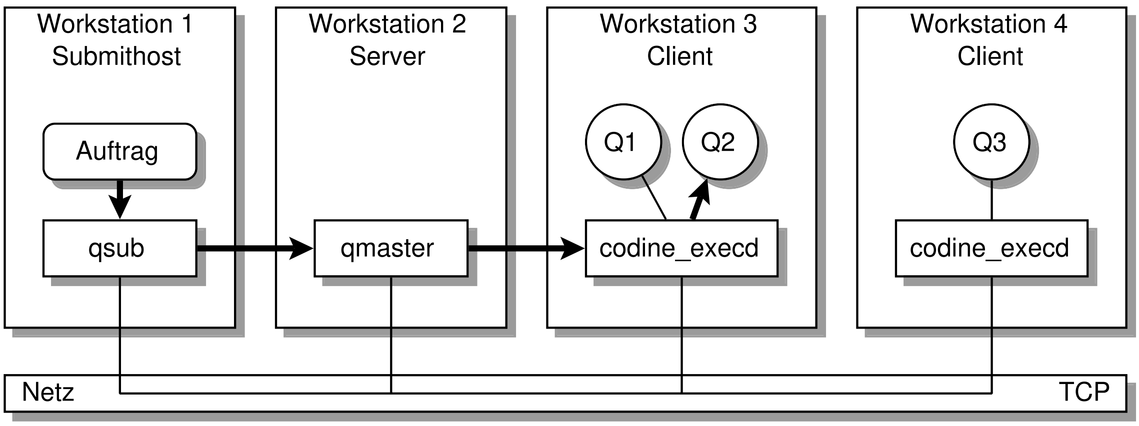

Abbildung 4.1: Aufbau von Codine

Codine ist eine Client-/Serveranwendung. Auf dem Server läuft der Codine Masterdämon `qmaster`, der die Verteilung der Aufträge und administrative Aufgaben übernimmt. Rechner, auf denen die Aufträge gerechnet werden, heißen Executionhosts. Auf ihnen läuft je ein Codine Executiondämon `codine_execd`. Dieser nimmt die Aufträge vom Masterdämon entgegen und bringt sie zur Ausführung. Außerdem liefert er regelmäßig Statusmeldungen an den Server.

Die Abbildung 4.1 zeigt den Aufbau von Codine. Von der Workstation 1 (einem Submithost) wird ein Auftrag abgegeben, der vom Masterdämon auf Workstation 2 in Empfang genommen wird und zur Ausführung an die Queue Q2 der Workstation 3 weitergegeben wird. Die Kommunikation zwischen den beteiligten Rechnern geschieht über TCP/IP.

Die einzelnen Queues können zu Queuefamilien zusammengeschlossen werden, um dann Parameter gemeinsam für alle Queues einer Familie ändern zu können. Aufträge besitzen eine Reihe von Attributen (wie z. B. eine bestimmte Prozessorarchitektur) um die Ausführung auf einer bestimmten Untergruppe der Queues zu verlangen.

4.2.3 Management von Codine (Status Quo)

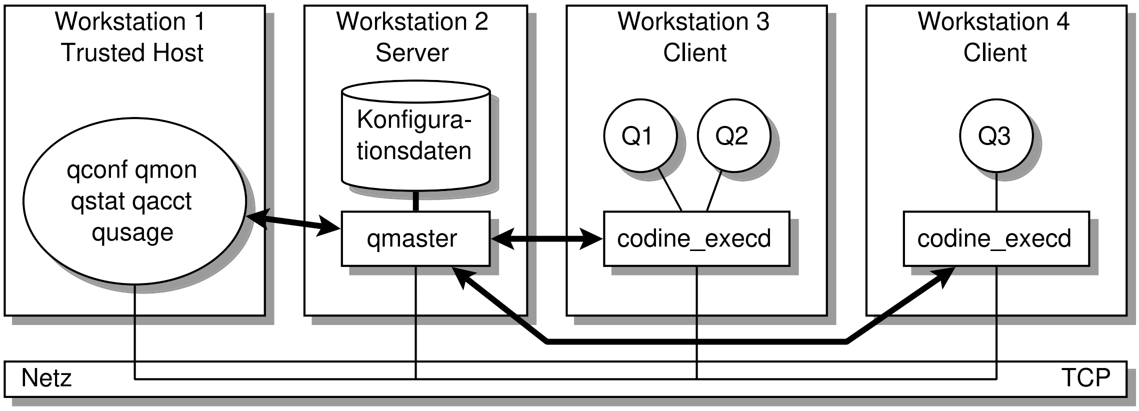

Abbildung 4.2: Derzeitiges Management von Codine

Das Management von Codine erfolgt bisher mit einigen von Codine mitgelieferten Programmen, die keinen Anschluß an eine Managementplattform vorsehen. Die für das Batchsystem relevanten Daten sammelt der Masterdämon.

Wie in Abbildung 4.2 zu sehen ist, werden diese Managementprogramme von einem Rechner, der als sicher gilt (*trusted host*), aufgerufen und kommunizieren mit dem Masterdämon. Dieser steht mit den Executiondämonen in Verbindung und hat Zugriff auf die Daten.

Die Programme werden normalerweise aus der Kommandozeile aufgerufen. Es gibt auch ein Programm mit grafischer Benutzeroberfläche (qmon), das den Aufruf der anderen Programme ermöglicht und den Status der Queues anzeigt. Dies kommt der Oberfläche der Managementprogramme unter HP OpenView wohl am nächsten.

In Tabelle A.1 auf Seite 68 im Anhang sind die Managementfunktionen, die Codine derzeit anbietet, zusammengefaßt.

4.2.4 Codine Version 4.0

Mit der Version 4.0 von Codine, die im Herbst veröffentlicht wird, werden einige Erweiterungen im Aufbau der Anwendung eingeführt. Um eine sichere Prozeß-zu-Prozeß-

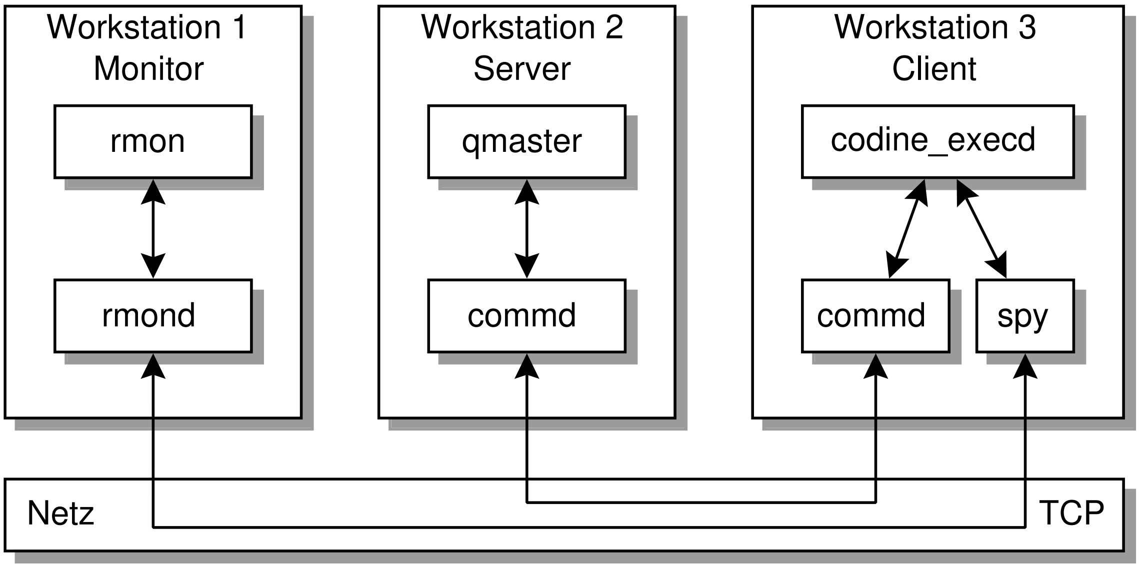

Abbildung 4.3: Aufbau von Codine Version 4.0

Kommunikation zu gewährleisten, wird auf jedem Rechner ein Kommunikationsdämon installiert. Dieser ermöglicht das sichere Versenden von komplexeren Datenstrukturen an die an Codine beteiligten Rechner mit der Möglichkeit der Bestätigung, daß der empfangende Prozeß die Daten erhalten hat.[1] Diesen Kommunikationsdämon können nun Managementanwendungen benutzen und somit relativ einfach auf die codineinternen Datenstrukturen zugreifen und sie verändern.

Weiterhin wird eine Möglichkeit zum *Remote Monitoring* zur Verfügung gestellt, so daß Clients ihre Logmeldungen direkt an eine zentrale Instanz schicken können, ohne dabei über den Masterdämon zu gehen. Diese Meldungen können dann von der zentralen Instanz des *Remote Monitoring*, dem `rmond`, angefordert werden. Hierzu läuft auf jedem Client ein weiterer Dämon `spy`, der mit dem `rmond` und der eigentlichen Anwendung direkt in Verbindung steht. Alle Meldungen vom Anwendungsprogramm werden an den `spy` übergeben und dann vom `rmond` in regelmäßigen Intervallen gepollt. Eine Managementanwendung, die für das Monitoring der verteilten Anwendung verantwortlich ist, kann auf diese Weise alle für sie relevanten Logmeldungen erhalten, filtern und daraus dann Traps erzeugen, die der Managementplattform zugestellt werden. Dieses *Remote Monitoring* stellt also eine zentrale Instanz für Mitteilungen in Form von Logdateien vom Batchsystem an das managende System dar, hat aber nichts mit der RMON-MIB ([RFC 1757]) zu tun, die zur Fernüberwachung von Netzkomponenten und Netzverkehr dient und vor allem statistische Daten liefert.

[1] TCP bestätigt nur die Zustellung am anderen Rechner, nicht aber den Empfang durch den anderen Prozeß.

4.2.5 Schnittstelle für Erweiterungen

Der Zugriff von Erweiterungen zu den internen Datenstrukturen von Codine geschieht über den Kommunikationsdämon, der von Codine 4.0 zur Verfügung gestellt wird. Hierüber werden die internen Daten, die in Listen verwaltet werden, über eine einer Datenbankabfragesprache ähnlichen Schnittstelle angeboten, die in eigene Programme eingebunden werden kann. Somit kann man die Listen vom Masterdämon holen, nach bestimmten Kriterien filtern und die Information weiterverarbeiten. Ebenso ist es möglich, Listen zu verändern und dem Server zu übergeben. Diese Schnittstelle (*CULL - Common Usable List Library*) ist in [GENI 95b] beschrieben.

Diese Library wird im Moment als kompiliertes Objektfile geliefert, so daß der Programmierer der Erweiterung direkt die zur Verfügung stehenden Aufrufe nutzen kann, wenn er das Programm dann mit der Library linkt. Die Kommunikation zwischen dem Master und der Anwendung ist in der Library integriert.

Die zur Verfügung stehenden Daten und Befehle sind in Tabelle A.2 auf Seite 69 im Anhang aufgeführt.

4.3 Managementprotokolle

4.3.1 Überblick

In diesem Abschnitt werden kurz die beiden wichtigen Managementprotokolle, die zum Netz-, System- und Anwendungsmanagement eingesetzt werden, vorgestellt. Das Managementprotokoll des Internetmanagements SNMP (*Simple Network Management Protocol*), das vom IAB (*Internet Activities Board*) standardisiert ist, hat durch seine einfachen Implementierungsmöglichkeiten eine große Verbreitung erfahren, wogegen CMIP (*Common Management Information Protocol*), Teil des von der ISO standardisierten OSI-Modells, zwar sehr mächtig ist und einen durchgehend objektorientierten Ansatz hat, sich aber wegen der aufwendigen Implementierung in der heutigen Rechnerwelt kaum durchsetzen konnte.

Da die hier behandelten Batchsysteme in der Unixwelt eingesetzt werden, wird für die Anbindung an ein Management auch SNMP verwendet.

4.3.2 CMIP - Common Management Information Protocol

Das OSI-Managementmodell wird allgemein als geeigneterer Ansatz für das Management von Rechnernetzen und Systemen gesehen ([HeAb 93]). Aufgrund der hohen Komplexität hat es sich aber nur im Bereich des Managements von Telekommunikationsnetzen etablieren können.

4.3.3 SNMP - Simple Network Management Protocol

SNMPv1

SNMP ist vom IAB (*Internet Activities Board*) in verschiedenen RFCs (*Request For Comments*, den Veröffentlichungen des IAB) standardisiert. [RFC 1157] definiert das Transportprotokoll (SNMP - *Simple Network Management Protocol*), [RFC 1155] die Struktur der Managementinformation (SMI - *Structure of Management Information*). Die eigentliche Information, also die Beschreibung der zu managenden Objekte, stehen in einer MIB (*Management Information Base*). Die Standard-MIB, die die wichtigsten Informationen vor allem zur Netzwerküberwachung liefert, ist die in [RFC 1213] definierte MIB-II. Sie wird von den meisten SNMP-Agenten zur Verfügung gestellt.

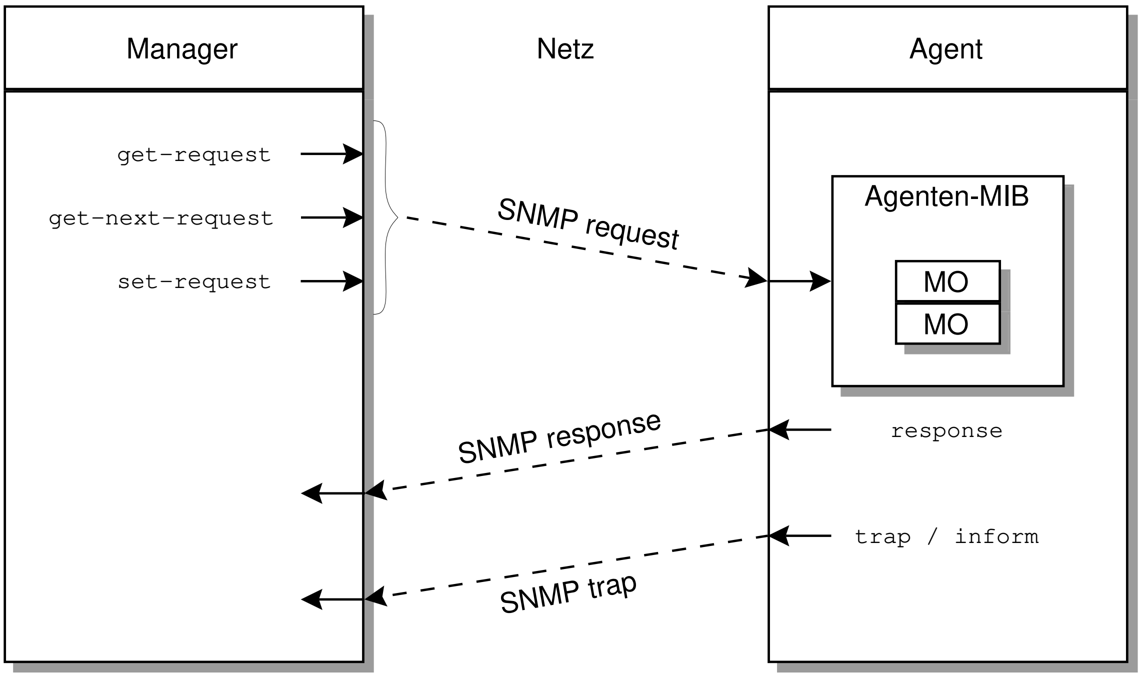

Abbildung 4.4: Kommunikation mit SNMP

Der Aufbau eines Managements mit SNMP ist in Abb. 4.4 zu sehen. Auf der Seite des zu managenden Systems wird ein SNMP-Agent installiert, der über einen bekannten UDP-Port (161) angesprochen werden kann. Der SNMP-Anwendung auf der Managerseite stehen die Befehle `get`, `getnext` und `set` zur Verfügung, die sich auf Objekte in der MIB beziehen. Der Agent kann mit `trap` asynchron Nachrichten an den Manager übermitteln. Die Zustellung dieser Traps ist allerdings nicht gesichert, da sie auf UDP beruhen. Zur Sicherheit dient ein *Communitystring*, der in der PDU (*Protocol Data Unit*) unverschlüsselt transportiert wird. Dieser *Communitystring* ist also mit *sniffer*-Programmen einfach abzuhören. Somit kann man relativ einfach unberechtigt an Informationen gelangen, die über SNMP bereitgestellt werden. Wenn der jeweilige Agent auch noch das Setzen von SNMP-Variablen erlaubt, kann jeder, der den Communitystring abgehört hat, Änderungen am System vornehmen.

SNMPv2

Inzwischen existiert eine neue Version des SNMP-Protokolls: SNMPv2, das in [RFC 1441]-[RFC 1452] definiert ist. Verbessert wurden vor allem Sicherheitsaspekte und die Möglichkeit, eine große Menge von Variablen auf einmal zu lesen. So wird durch die Einführung von Verschlüsselungs- und Authentifizierungsmechanismen ein unerlaubter Lese- und Schreibzugriff unterbunden. Dies ermöglicht dann die sichere Verwendung von `set`-Operationen. Zur schnelleren Übermittlung der Daten dient der neue Befehl `getbulk`, der eine größere Anzahl von `get-getnext` Aufrufen ersetzt. Eine weitere Neuerung ist der `inform`-Befehl. Dieser ermöglicht eine sichere Zusendung asynchroner Meldungen.

Diese Erweiterungen, insbesondere die zusätzlichen Sicherheitsvorkehrungen, die für einen Agenten, der auch das Setzen von Variablen zuläßt, unabdingbar sind, verlangen die Verwendung von SNMPv2.

Leider gibt es im Moment noch keine Managementplattform, die die Möglichkeiten von SNMPv2 vollständig implementiert hat. Auch auf der Agentenseite fehlt noch eine vollständige Implementierung, die alle Sicherheitsaspekte beinhaltet. Bis jetzt gibt es nur Agenten, die sowohl das SNMPv1- als auch das SNMPv2-Protokoll verstehen und bessere Performance haben, aber die neuen Sicherheitsaspekte ausklammern. Daher wird der Agent für das Batchsystem ein solcher zweisprachiger SNMPv1- und SNMPv2-Agent sein. Eine Migration zu einer vollständigen SNMPv2-Implementierung sollte aber geschehen, sobald die entsprechenden Produkte vorhanden sind, um die Sicherheitslücken zu schließen.

4.3.4 Standardisierte MIBs

Bei einer Integration des Managements eines Batchsystems soll es möglichst viele Informationen über das System von der Managementplattform oder von anderen SNMP-Agenten erhalten. Daher werden hier kurz standardisierte MIBs vorgestellt, aus denen diese Informationen kommen könnten.

- MIB-II [RFC 1213]

  Die MIB-II enthält Informationen zum System, zu den Netzwerkinterfaces und Netzprotokollen.

- Host Resources MIB [RFC 1514]

  Die Host Resources MIB enthält Informationen zum System, zum Speicher (Hauptspeicher und Hintergrundspeicher) und zu den laufenden Prozessen.

- Network Services Monitoring MIB [RFC 1565]

  Die Network Services Monitoring MIB ermöglicht das Monitoring von Netzwerkanwendungen, indem er eine Tabelle der verfügbaren Netzwerkanwendungen und deren Verbindungen liefert.

- LRZ-UNIX MIB [KRIE 94]

  Die LRZ-UNIX MIB entstand durch einige Arbeiten am Leibniz-Rechenzentrum München. Sie ist allerdings eine experimentelle MIB und somit nicht standardisiert. Sie implementiert ähnliche Informationen wie die Host Resources MIB, geht dabei allerdings genauer auf Unix ein und stellt mehr Variablen zu den Bereichen System, Speicher, Gerätetreiber, Prozessor, Drucker, Platten, Filesysteme, Prozesse und Benutzer bereit.

4.3.5 Subagenten

Da es unter Unix nicht möglich ist, daß zwei Prozesse am gleichen Port liegen, kann es pro Rechner nur einen SNMP-Agenten geben, der an Port 161 gebunden ist. Dieser eine Agent muß die Daten aller MIBs zur Verfügung stellen, die auf dem System angeboten werden. Um die Erweiterbarkeit dieses Agenten zu ermöglichen, ohne ihn jedesmal stoppen und mit neuen MIBs kompilieren zu müssen, wurden Protokolle entwickelt, mit denen ein Hauptagent Anfragen an Subagenten weitergeben kann, die eigenständige Prozesse sind. Diese können sich zur Laufzeit des Hauptservers für einen Teilbaum einer MIB an- und abmelden und bekommen dann die entsprechenden SNMP-Befehle vom Hauptserver weitergereicht (Abb. 4.5 auf der nächsten Seite). Somit ist es möglich, daß jede Anwendung einen SNMP-Subagenten mitliefert, der beim Starten der Anwendung geladen wird.

Leider gibt es mehrere Protokolle zur Anbindung von Subagenten und keinen einheitlichen Standard, welches Protokoll eingesetzt werden sollte. IBM hat in [RFC 1592] ein solches Protokoll, SNMP DPI (*SNMP Distributed Protocol Interface*), definiert und der Öffentlichkeit zugänglich gemacht. Auf diesem Protokoll aufbauend, existieren an der Universität und am Leibniz-Rechenzentrum München schon einige Implementierungen. Der SNMP-Agent der *Carnegie Mellon University* (CMU-SNMP-Agent) wurde um dieses Protokoll erweitert, so daß ein Hauptagent zur Verfügung steht, der die MIB-II realisiert und den Anschluß von Subagenten ermöglicht ([HAIN 94]). Dieser Agent funktioniert bisher unter SunOS 4. Im Rahmen dieser Arbeit wurde er auch auf Linux portiert, wo er dann auch noch die Host Resources MIB implementiert. Weitere Portierungen, z. B. auf HPUX, sind in Arbeit. Dazu existieren bisher schon einige Subagenten, unter anderem für die LRZ-Unix-MIB ([HaHa 94]) und für eine Drucker-MIB ([HAIN 95]). Daher wird auch der Subagent für Codine mit DPI-Schnittstelle realisiert.

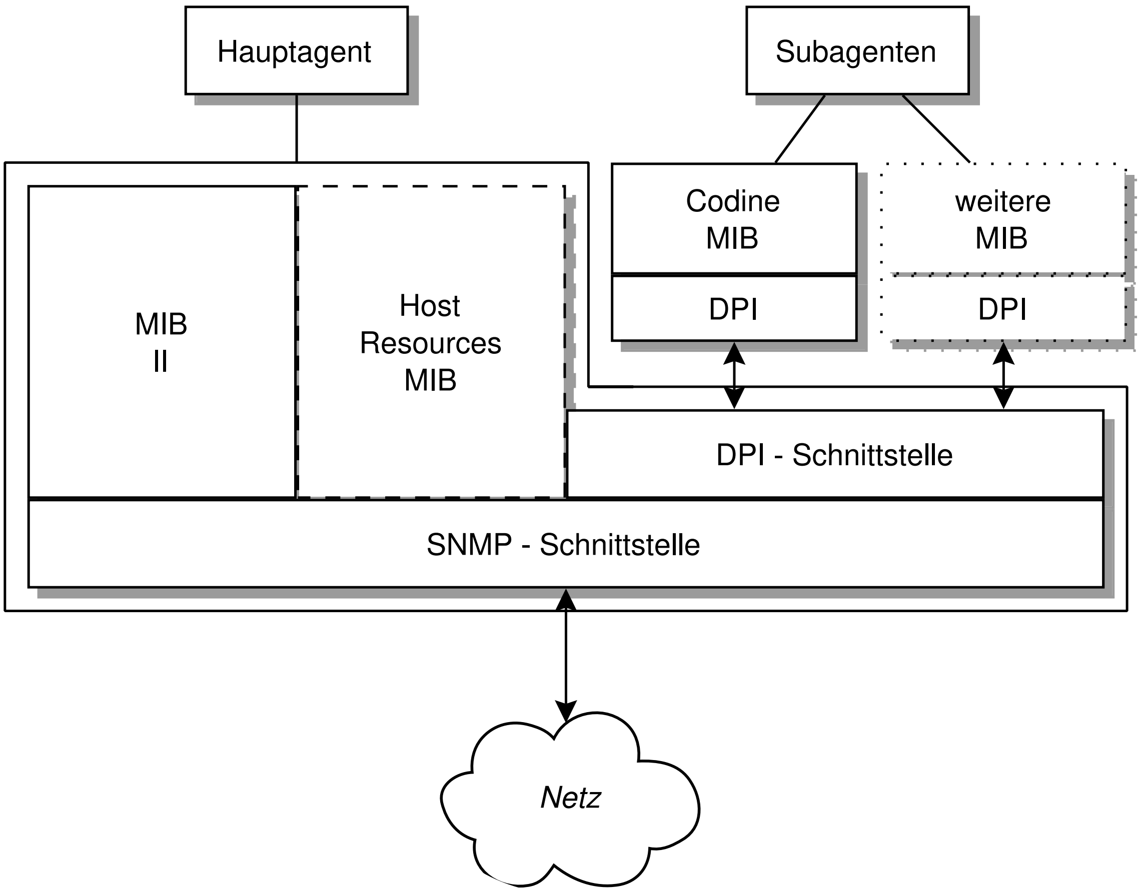

Abbildung 4.5: SNMP-Agent mit Subagenten

Kapitel 5

Anforderungen an die MIB

In diesem Kapitel werden anhand der Anforderungen an das Management, die in Kapitel 3 vorgestellt wurden, Anforderungen an eine MIB für Codine abgeleitet. Neben den Anforderungen, die direkt aus den zu realisierenden Managementaufgaben folgen, gibt es auch solche, die durch das Protokoll SNMP bedingt sind.

5.1 Anforderungen des Managements

In den folgenden Abschnitten werden die für die einzelnen Aspekte des Managements nötigen Daten untersucht, die in die Codine-MIB aufgenommen werden müssen. Objekte, die für mehrere Aspekte relevant sind, werden nur beim ersten Mal ausführlich erläutert.

5.1.1 Konfigurationsmanagement

Installation der Software

Bisher geschieht die Installation der Software über Installationsskripten, die die einzelnen Komponenten von Codine auf den vorgesehenen Rechnern installieren.

Im Rahmen eines integrierten Managements sollte auch die Installation der Software mit Hilfe der Managementplattform möglich sein. Hierzu fehlen allerdings bisher einheitliche Konzepte und die Unterstützung von Seiten der Plattform. Eine Unterstützung dieser Aufgaben kann auch nicht durch die Codine-MIB erfolgen, da diese MIB von dem Codine-SNMP-Agenten zur Verfügung gestellt wird, der erst nach Installation des Codinesystems funktionsfähig ist. Somit besteht die Möglichkeit, hierfür eine eigene Codine-Software-Installations-MIB vorzusehen, was aber einen unverhältnismäßig hohen Aufwand darstellt. Daher ist es sinnvoller, die Codine-Installation mit den bisherigen Skripten fortzuführen bis ein einheitliches Softwareinstallationswerkzeug vorliegt.

Konfiguration der beteiligten Rechner

Die Konfiguration der Rechner geschieht bisher durch ein Programm `qconf`. Dies liefert eine Vielzahl von Optionen zur Modifikation der Parameter und erlaubt zum Teil das Editieren einiger Konfigurationsdateien.

Viele der hier einzustellenden Parameter beziehen sich nur auf das Batchsystem, sind nicht für andere Anwendungen relevant und auch noch nicht als Information in der Managementplattform vorhanden. Daher ist es nicht sinnvoll, alle Konfigurationsparameter in die MIB aufzunehmen, sondern nur diejenigen, die für Managementaufgaben relevant sind. Für die restlichen Parameter steht weiterhin das codineeigene Programm `qconf` bereit.

In der folgenden Aufzählung werden die Parameter der Konfigurierung der Hosts bezüglich ihrer Verwendbarkeit in der Codine-MIB untersucht.

- Typ des Hosts

  Der Typ des Hosts, also die Rolle, die ein Rechner in Codine übernimmt, ist eine Information, die für verschiedene Managementaufgaben benötigt wird. Dies muß daher in die MIB aufgenommen werden. Mögliche Werte dieser Variablen sind: Masterserver, Shadowmaster, Executionhost und Submithost.

- Besitzer des Hosts

  Da der Besitzer des Host das Recht hat, die Queues, die auf seinem Rechner laufen, zu suspendieren, wird die Information hierüber auch in der MIB benötigt.

- Hostname

  Der Name des Rechners wird auch in die MIB übernommen, da Codine bisher den Namen eines Rechners als Identifikator benutzt und nicht dessen Internetadresse. Werden also die bisherigen Programme zum Management von Codine in eine SNMP-Umgebung integriert, so benötigen sie weiterhin den Rechnernamen zur Identifizierung des Rechners.

- Architektur und Betriebssystem

  Die Prozessorarchitektur des Rechners und das installierte Betriebssystem können bisher auch über `qconf` konfiguriert werden. Diese Informationen werden beim Starten von Programmen benötigt, die in der Regel nur auf einer bestimmten Prozessorarchitektur und einem bestimmten Betriebssystem ablaufen können. Es besteht zwar noch die Möglichkeit, diese Informationen aus der Host Resources MIB zu erhalten (siehe Abschnitt 8.3.2 auf Seite 59), aber die Information dort ist nicht normiert. Daher müssen diese Informationen in der Codine-MIB enthalten sein.

- Speicher

  Auch die Größe des verfügbaren Speichers wird bisher von Hand konfiguriert. Diese Information kann aus der Host Resources MIB bezogen werden (siehe Abschnitt 8.3.2 auf Seite 59).

Konfiguration der beteiligten Queues

Bisher geschieht die Konfiguration der Queues ebenfalls über `qconf`. Auch hier werden die für das Management wichtigen Daten in die MIB aufgenommen.

- Namen der Queue und des Rechners

  Die Namen der Queue und des Rechners dienen zur Identifizierung der Queue und der Zuordnung zu einem Rechner und werden deshalb in die MIB aufgenommen.

- Name des Besitzers

  Der Name des Besitzers der Queue entspricht normalerweise dem Namen des Besitzers des Rechners, auf dem die Queue läuft. Er wird ebenfalls in die MIB aufgenommen, da er das Recht hat, bestimmte Operationen, die die Queue betreffen, auszuführen.

- Queuefamilie

  Die Zuordnung zu einer Queuefamilie geschieht bisher ebenfalls über Konfigurationsdateien. Es ist sinnvoller, diese Information in der MIB zu integrieren, um Anfragen, die sich auf alle Queues einer Familie beziehen, bearbeiten zu können.

- Priorität

  Die Information über die voreingestellte Priorität der Jobs, die in einer Queue zur Ausführung gebracht werden, wird ebenfalls in die MIB aufgenommen. Somit kann sie im Rahmen des Managements geändert werden, um z. B. bestimmte Rechner zu entlasten.

Konfiguration der beteiligten Queuefamilien

Folgende Informationen über die Queuefamilien werden in der MIB benötigt:

- Name der Queuefamilie

  Der Name der Queuefamilie dient wieder zur Identifizierung und wird daher in der MIB benötigt.

- Liste der Queues

  Eine Liste der Queues, die in der Queuefamilie enthalten sind, wird ebenfalls für Aufgaben benötigt, die sich auf eine Familie beziehen.

Weitere Informationen über Queuefamilien, die in Codine konfiguriert werden können, sind sehr speziell und werden nur innerhalb von Codine benötigt. Da sie für Managementaufgaben nicht relevant sind und normalerweise nur einmal konfiguiert werden, empfiehlt es sich nicht, diese in der MIB zu integrieren. Somit wird auch eine Überladung der MIB mit unnötigen, selten zu verändernden Daten vermieden. Die Änderung solcher Parameter muß auch weiterhin mit dem codineeigenen Werkzeug `qconf` geschehen, wobei der Aufruf dieses Werkzeugs in die Plattform integriert werden kann (siehe Oberflächenintegration, Abschnitt 8.1).

Konfiguration der Parameter von Jobs

Die Parameter der Jobs werden beim Abschicken der Jobs angegeben. Eine nachträgliche Änderung einiger Parameter ist erst mit der neuen Version Codine 4 möglich. Hierzu steht der Befehl `qalter` zur Verfügung. Damit kann man z. B. durch eine Änderung der Prioritäten der Jobs die Reihenfolge der Abarbeitung verändern.

Die Parameter der Jobs werden im folgenden auf eine Aufnahme in die Codine-MIB hin untersucht.

- Jobnummer und Jobname

  Diese Informationen werden wieder zur Identifizierung in der MIB benötigt.

- Jobfile

  Der Name der Datei, in der die Beschreibung des Jobs enthalten ist, wird in der MIB benötigt. Damit können z. B. genauere Informationen über die im System vorhandenen Jobs ausgegeben werden.

- Execfile

  Dies ist der Name der auszuführenden Datei eines Jobs. Auch diese Information ist in der MIB sinnvoll, da ein Manager nicht nur den Namen des Jobs, sondern auch den Namen des laufenden Prozesses, der mit dem Namen der auszuführenden Datei identisch ist, für Managementaufgaben benötigt. Damit kann man dann in der Prozeßtabelle weitere Informationen erhalten.

- Queue und Host des Jobs

  Um eine Zuordnung von Jobs auf Queues und Hosts, in bzw. auf denen sie abgearbeitet werden, zu ermöglichen, bedarf es dieser Information in der MIB.

- Priorität des Jobs

  Die Priorität des Jobs kann man, wie oben beschrieben, mit `qalter` verändern. Dies und die damit verbundene Änderung der Reihenfolge der Abarbeitung der Jobs ist eine Aufgabe, die ein Manager über SNMP realisieren wird. Daher gehört dieses Attribut in die MIB.

- Maximale Rechenzeit eines Jobs

  Die maximale Rechenzeit eines Jobs ist eine obere Schranke der CPU-Zeit, die der Abarbeitung eines Jobs zugestanden wird. Diese Zeit wurde bisher beim Abschicken eines Jobs bestimmt. Durch Aufnahme in die MIB kann diese Grenze, nach der der

Job vom System beendet werden kann, vom Management berücksichtigt und auch noch nachträglich verändert werden.

5.1.2 Fehlermanagement

Starten und Stoppen der Komponenten

Für den Betrieb von Codine müssen der Server (`qmaster`) und die Clients (`codine_execd`) auf den beteiligten Rechnern gestartet werden. Das Starten dieser Programme geschieht bisher durch das Aufrufen eines Skriptes (`codine_startup`) auf jedem Rechner des Clusters. Dazu existiert auch ein Skript, daß das Einloggen auf allen Rechnern übernimmt und mittels `rsh` die Dämonen startet. Das Stoppen geschieht bisher bei Codine durch das Aufrufen der Kommandos `qconf -kqs; qconf -km`. Dies stoppt zunächst die Clients und dann den Masterdämon.

Die Lösung dieser Aufgabe durch die Managementplattform mit der Codine-MIB über SNMP ist schwierig. Das Management via SNMP kann erst in Aktion treten, wenn die entsprechende MIB verfügbar ist. Der Codine-SNMP-Agent ist aber ein eigenständiger Prozeß, der auch erst gestartet werden muß, und er benötigt den Masterserver und den Kommunikationsdämon, um mit Codine in Kontakt zu treten. Daher müssen auch in einer solchen Managementumgebung die einzelnen Programme von Hand oder mit Skripten gestartet werden.

Nach dem Start des Masterservers, des Kommunikationsdämons und des SNMP-Agenten ist das Management funktionsfähig, und es wäre möglich, die Clients vom Server starten zu lassen. Hierzu werden in der Codine-MIB Einträge benötigt, die das Starten der Clients veranlassen können.

Das Stoppen der Komponenten über SNMP ist allerdings möglich, da die SNMP-Kommunikationsinfrastruktur vor dem Stoppen noch besteht. Weitere Einträge in der MIB müssen das Stoppen realisieren. Wie solche Aktionen mit SNMP realisiert werden können, wird in Abschnitt 5.2.4 auf Seite 35 beschrieben.

Auslösen von Traps bei Überschreiten bestimmter Grenzwerte

Bisher bietet Codine keine Möglichkeit, bei Überschreiten bestimmer Grenzwerte automatisch Aktionen auszulösen.

Ein Managementsystem erlaubt es, für bestimmte Werte, die z. B. aus einer MIB ausgelesen werden, Schranken zu setzen, bei deren Über- oder Unterschreitung Traps gesendet werden. So kann z. B. bei einer zu langen Queue die Annahme neuer Aufträge gesperrt oder bei Überschreiten einer gewissen Laufzeit ein Job suspendiert werden, wenn viele Anfragen warten. Bei häufigem Überschreiten der Last kann z. B. der Systemadministrator benachrichtigt werden, nach den Ursachen zu suchen. Voraussetzung hierfür ist die Verfügbarkeit der zu überwachenden Informationen in MIB-Variablen.

5.1.3 Leistungsmanagement

Überwachung des Zustands der Queues

```
--> qstat

QUEUE     GROUP ARCHITECTURE STATUS    OWNER  REQUEST JOB_ID P
durin.q   sun4  sparc       suspended
balin.q   sun4  sparc       suspended
gloin.q   sun4  sparc       running   ste    parall  44     0
thorin.q  sun4  sparc       suspended
ori.q     sun4  sparc       running   ste    parall  44     0
eomer.q   hpux  parisk      running   ferstl test.sh 46     0
eowyn.q   hpux  parisk      suspended
legolas.q osf   alpha       running   andy   ckpt.sh 45     0
bilbo.q   irix4 sgi         idle
aragorn.q rios  rs6000      idle

======Pending Jobs======
Resource   Requests  Owner  Jobscript  Job ID   P
a=parisk   1         ferstl test.sh    47       5
g=sun4     2         ferstl test.sh    48       0
```

Tabelle 5.1: Ausgabe von `qstat`

Das Programm `qstat` gibt bisher Auskunft über den Zustand der Queues. Es listet alle Queues auf und informiert über den Status und den im Moment laufenden Job. Eine Ausgabe dieses Befehls ist in Tabelle 5.1 zu sehen. Das Programm `qusage` erzeugt eine grafische Anzeige der aktuellen Auslastung der Queues ähnlich dem X Window-Programm `xload`.

Es ist klar, daß auch diese Informationen in der Codine-MIB zu modellieren sind, so daß man von der Managementplattform darauf zugreifen kann. Die Managementanwendung in der Plattform kann dann auf diese Informationen der MIB zugreifen und sie z. B. in einer Tabelle darstellen.

Wie auch die bisherigen Werkzeuge die Informationen in einen Teil, der sich auf Jobs, und einen Teil, der sich auf Queues bezieht, unterteilen, werden in der MIB entsprechende Variablen zu Queues und Jobs benötigt. Im folgenden werden die Informationen des Leistungsmanagements, die sich auf Queues beziehen, aufgeführt.

- Queue-Status

  Zur Überwachung der Queues ist ihr Status eine wichtige Information, die auch in der MIB enthalten sein muß. Mögliche Werte hierzu sind: gestoppt (*disabled*), leer (*idle*) und rechnend (*running*).

- ID des Jobs, der im Moment bearbeitet wird

  Da bei einer Statusabfrage auch der Name oder die ID des Jobs, der gerade abgearbeitet wird, interessant ist, muß eine der beiden Informationen in der MIB enthalten sein. Es genügt, die ID (Jobnummer) zu benutzen, da über sie der Name bestimmt werden kann.

- Anzahl der Jobs in der Queue

  Ebensowichtig für das Leistungsmanagement, insbesondere für die Berechnung von Wartezeiten, ist die Information über die Anzahl der wartenden Jobs in einer Queue, die natürlich auch in der MIB verfügbar sein muß.

Die weiteren Attribute für Queues, die hier noch in der MIB benötigt werden, wurden schon beim Konfigurationsmanagement detaillierter aufgeführt. Es sind: Name der Queue, Architektur des Prozessors und Besitzer der Queue.

Anzeige der Jobs

Die Anzeige der Jobs erfolgt bisher ebenfalls mit `qstat`, wie in der Tabelle 5.1 auf der vorherigen Seite zu ersehen ist. Mit `qdel` können Jobs aus der Queue gelöscht werden.

Im folgenden werden die Informationen aufgeführt, die als Attribute der Jobs in der Codine-MIB enthalten sein müssen, damit der Manager alle nötigen Informationen über die laufenden und wartenden Jobs erhalten kann. Daraus kann er ähnliche Tabellen erstellen wie die Ausgabe von `qstat`.

- Job-Status

  Der Status des Jobs (wartend, rechnend, beendet) wird in der MIB benötigt, um ihn richtig zu klassifizieren. So hat es wenig Sinn, bei Untersuchungen im Rahmen des Leistungsmanagements die Rechenzeiten von Jobs zu berücksichtigen, die noch nicht gerechnet werden.

Weitere in diesem Zusammenhang wichtige Variablen der Codine-MIB wie die Priorität des Jobs wurden schon für das Konfigurationsmanagement gefordert.

5.1.4 Abrechnungsmanagement

Accounting

Bisher gibt es das Programm `qacct`, das Accountinginformationen nach Job, Queue, Benutzer, Architektur filtern kann oder für den ganzen Cluster ausgibt. Hierbei greift es auf eine Accountingdatei zurück, in der die nötigen Informationen über beendete Jobs stehen. Die Ausgabe des Befehls `qacct` ist in Tabelle 5.2 zu sehen.

```
--> qacct -u -A -G -q

ARC     GROUP   QUEUE      USER      REAL   USER   SYSTEM
=========================================================
sparc   sun4    durin.q    andy      101    0      0
sparc   sun4    durin.q    ferstl    51     0      2
sparc   sun4    thorin.q   ferstl    7      0      2
sparc   sun4    thorin.q   ste       304    0      0
sparc   sun4    ori.q      ferstl    26     0      1
sparc   sun4    gloin.q    andy      880    0      0
sparc   sun4    gloin.q    ferstl    25     0      1
sparc   sun4    balin.q    root      101    0      0
sparc   sun4    balin.q    andy      101    0      0
sgi     irix4   bilbo.q    andy      8853   910    147
```

Tabelle 5.2: Ausgabe von `qacct`

Diese Informationen sind für das Accountingmanagement einer Plattform natürlich sehr interessant, da über eine Managementplattform ein integriertes Accounting aller Dienste, die von der Plattform überwacht werden, realisiert werden kann. Dies ermöglicht dann auch eine gemeinsame Abrechnung der in Anspruch genommen Dienste, wenn ein entsprechendes Abrechnungskonzept vorhanden ist. Leider existieren im Moment noch keine einheitlichen Verfahren, Accountinginformationen zu sammeln und gemeinsam auszuwerten.

Eine Voraussetzung hierzu, nämlich der einfache Zugriff auf die benötigten Informationen von Codine, muß allerdings schon mit der Codine-MIB geschaffen werden. Dafür werden folgende Informationen, die sich auf Jobs beziehen, benötigt:

- Submission-Zeit

  Die Zeit, zu der der Job abgeschickt wurde, wird in der MIB benötigt, um festzustellen, wie lange ein Job schon wartet.

- Startzeit

  Ebenso wird der Zeitpunkt benötigt, zu der die Abarbeitung des Jobs auf einem Rechner begonnen hat.

- Beendigungszeit

  Mit Hilfe der Beendigungszeit (dem Zeitpunkt, zu dem die Berechnungen des Jobs beendet waren) und der Startzeit können Messungen über die Dauer der Jobs durchgeführt werden, oder es kann festgestellt werden, in welchem Zeitraum welche Jobs berechnet wurden.

- Rechenzeit des Jobs

  Neben den absoluten Zeitpunkten wird auch die Zeitdauer der Berechnungen der Jobs in der MIB für Performancemessungen gebraucht. Um die Messungen genau durchzuführen, wird für jeden Prozeß jeweils die reale, also die insgesamt verstrichene Rechenzeit, die System- und die Benutzerrechenzeit, z. B. so, wie sie der Unix-Befehl `time` ausgibt, benötigt. Diese Informationen müssen auch noch nach Beendigung des Jobs zur Verfügung stehen, da Accountinginformationen normalerweise für beendete Jobs benötigt werden.

- Besitzer des Jobs

  Um Accountinginformationen für jeden Benutzer getrennt auswerten zu können, wird natürlich auch eine Information über den Besitzer eines Jobs in der MIB benötigt. Sinnvollerweise geschieht dies durch mehrere Attribute des Jobs, die sowohl die Benutzer-ID als auch den Login-Namen des Benutzers beinhalten.

- Status des Jobs

  Auch für das Accounting wird eine Information über den Status des Jobs benötigt, damit man erkennen kann, welche Jobs schon beendet sind. Diese können dann in ein Abrechnungsmanagement einbezogen werden.

Verwalten der Benutzerrechte

Bisher geschieht die Verwaltung der Benutzer mit verschiedenen Optionen des Kommandos `qconf`. Dabei werden den Benutzern bestimmte Rechte in Form von Benutzerstufen (User, Manager, Operator) zugewiesen.

Die Verwaltung der Benutzerrechte für ein Batchsystem entspricht in etwa der Verwaltung derer eines UNIX-Systems. Für diese normale Benutzerverwaltung gibt es im Bereich des Systemmanagements schon Ansätze, diese in das integrierte Management der Plattform einzubeziehen. Hier bietet es sich an, dieselben Konzepte zu übernehmen und weitere Attribute für die Rechte in Codine hinzuzufügen, sofern die Benutzerverwaltung das Hinzufügen neuer Attribute zuläßt.

Steht eine solche Benutzerverwaltung nicht zur Verfügung, so müssen in der Codine MIB noch folgende Einträge für die Benutzer des Batchsystems hinzugefügt werden.

- Name und ID des Benutzers

  Zur Identifikation in der MIB wird der Name und die ID des Benutzers benötigt.

- Typ des Benutzers

  Entsprechend der Benutzerstufen in Codine ist ein Eintrag in der MIB vorzusehen. Mögliche Werte sind: User, Manager und Operator.

- Benutzungseinschränkungen

  Für benutzerbezogene Benutzungseinschränkungen wie der maximalen Anzahl von Jobs, die sich gleichzeitig im Batchsystem befinden dürfen, ist auch in der MIB ein Eintrag vorzusehen.

Für den Fall, daß eine Managementplattform eine Benutzerverwaltung zur Verfügung stellt, das das Hinzufügen von Attributen erlaubt, und eine einheitliche Schnittstelle für den Zugriff anbietet, über die das Codine-System die Informationen erlangen kann, könnten die Informationen über den Typ des Benutzers und die Benutzungseinschränkungen auch dort integriert werden. Da dies aber noch nicht abzusehen ist, wird die Codine-MIB entsprechende MIB-Variablen enthalten.

5.1.5 Sicherheitsmanagement

Die Schnittstelle CULL von Codine, die zur Implementierung des SNMP-Agenten verwendet wird, realisiert keine Sicherheitsaspekte. Diese werden innerhalb von Codine auf der Ebene der Programme, die CULL benutzen, realisiert. Somit muß auch der SNMP-Agent selber für die Sicherheit sorgen. Dies geschieht durch das Einschränken der Rechte für bestimmte MIB-Variablen, die nicht öffentlich les- und schreibbar sind. Davon sind die Variablen betroffen, deren Abfrage oder Veränderung auch bisher einen bestimmten Benutzerlevel erfordern. Für diese einzelnen Benutzerlevel werden SNMP-Parties definiert und die entsprechenden Rechte zugeordnet.

Eine solche Implementation ist aber im Moment aufgrund der Beschränkungen von SNMPv1 und der noch nicht vollen Verfügbarkeit von SNMPv2 nicht abhörsicher zu realisieren, wie in Abschnitt 4.3.3 auf Seite 19 erläutert.

5.2 Anforderungen aus SNMP

Neben den oben aufgeführten Anforderungen an die MIB, die aus den Managementanforderungen folgen, gibt es auch Anforderungen, die sich durch die Verwendung von SNMP als Managementprotokoll ergeben.

5.2.1 Realisierung der Listen

Die vorgestellten Attribute der Queues, Jobs, Hosts und Benutzer werden innerhalb von Codine in Listen verwaltet. In einer MIB müssen diese Listen auf SNMP-Tabellen abgebildet werden. Die Indizierung dieser Tabellen kann nicht wie bei den Codine-Listen über Namen geschehen, da SNMP dies nicht zuläßt. Zur SNMP-konformen Indizierung gibt es mehrere Möglichkeiten, die im folgenden untersucht werden.

5.2.2 Indizierung der SNMP-Tabellen

Indizierung mit Integern

Bei der Realisierung von Tabellen mit SNMP wird die Objekt-ID des MIB-Baumes in mehrere Teile zerlegt: der Anfang entspricht der gesamten Tabelle, die folgende Komponente der ID steht für die Spalte der Tabelle und der Rest, dessen Länge frei wählbar ist, definiert den Index. So ist z. B. bei der MIB-Variablen `interfaces.ifTable.ifEntry.ifDescr.10` der Interfacetabelle `interfaces.ifTable.ifEntry` der Teil, der die gesamte Tabelle definiert, `ifDescr` die Spalte und `10` die Zeile der Tabelle. Zur Indizierung der Tabelle wurden hier also Integerwerte verwendet. Diese Möglichkeit der Listenindizierung besteht auch bei der Codine-MIB.

Der Vorteil dieser Indizierungsart ist die einfachere Implementierung im Agenten. Die Indizes der Listen haben keine semantische Bedeutung und können somit vom Agenten frei gewählt werden, wobei sich natürlich anbietet, die Reihenfolge der Listen innerhalb von Codine zu belassen.

Ein Nachteil dieser Methode ist aber die Suche nach bestimmten Informationen, d. h.: Wenn man z. B. die Anzahl der Jobs in einer Queue bestimmen will, die auf einem bestimmten Host läuft, muß man erst in der Queuelist durch sämtliche Queues gehen und die Variable `coqlHostname` mit dem gesuchten Hostname vergleichen. Dann kann man mit den gefundenen Indizes in der Queuelist die Variable `coqlNrJobs` auslesen. Die erhaltenen Indizes können aber bei einer weiteren Abfrage nicht wiederverwendet werden, weil sich durch Hinzufügen oder Entfernen von Hosts oder Queues die Indizierung ändern kann.

Somit steht hier eine einfache Implementierung auf der Agentenseite einer umständlichen Weise der Informationsbeschaffung auf der Seite des Managers entgegen.

Indizierung mit der Internetadresse

Eine andere Möglichkeit der Indizierung wird in der `ip.ipAddrTable` realisiert. Hier geschieht die Indizierung mit vier Integern, die Internetadressen entsprechen. So kann auch die Hostliste innerhalb der Codine-MIB indiziert werden. Um nun Informationen eines bestimmten Hosts zu erfahren, genügt es, mit `gethostbyname` die Internetadresse zu bestimmen, um dann mit dieser als Index direkt auf die gewünschte Variable innerhalb der Spalte zugreifen zu können.

Analog hierzu kann man die Queues mit fünf Integern indizieren, wobei die ersten vier der Internetadresse des Hosts, auf dem sie laufen und die fünfte der Numerierung der Queues auf einem Host entsprechen. Dies erleichtert ebenfalls die Suche nach Queues eines bestimmten Hosts innerhalb der Queuetabelle.

Der Preis für die einfachere Handhabung ist eine aufwendigere Implementierung im Agenten, der die Indizierung auf die codineeigene Identifizierung von Hosts und Queues durch Namen abbilden muß.

5.2.3 Einfügen neuer Tabelleneinträge

SNMP bietet keinen einheitlichen Weg zum Anlegen neuer Einträge in einer Tabelle von seiten des Managers an, schlägt aber in [RFC 1212] mehrere Möglichkeiten hierzu vor. Eine Möglichkeit ist, das Schreiben auf einen nicht existierenden Tabelleneintrag nicht mit einer Fehlermeldung zu quittieren, sondern das entsprechende Objekt und eventuell auch die zugehörige Tabellenspalte zu kreieren. Hierzu ist es aber nötig, die Objekt-ID des neuen Objektes inklusive des Indizes schon zu kennen, wozu der Agent die ganze Tabelle nach einem freien Index durchsuchen muß. Dabei muß er die Art der Indizierung, die oben erläutert wurde, berücksichtigen, um nicht einen Eintrag mit unerlaubtem Index zu kreieren.

Um dies zu umgehen wird eine andere Möglichkeit der Einfügung neuer Tabelleneinträge für die Codine-MIB verwendet. Durch Schreiben auf eine spezielle Variable der MIB können Aktionen aufgerufen werden. Eine dieser Aktionen ist `add`, die, auf eine Liste angewandt, einen neuen Eintrag kreiert. Da diese Art des Auslösen von Aktionen, die im Abschnitt 5.2.4 auf der nächsten Seite beschrieben wird, sowieso für weitere Aktionen benötigt wird, ist es sinnvoll und praktisch, sie auch zum Einfügen neuer Tabelleneinträge zu verwenden. Das Ergebnis dieser Aktion ist eine neue Zeile in der Tabelle, die leere Einträge enthält. Der Manager sucht daraufhin alle Indizes der Tabelle ab und sucht den freien Eintrag, der für ihn kenntlich sein muß. Sinnvoll ist es, den Namen, der ja die Objekte eindeutig identifiziert, auf `frei` zu setzen. Wenn der Manager den freien Eintrag gefunden hat, kann er durch Setzen der Variablen den neuen Tabelleneintrag anlegen. Diese Suche über bestehende Tabelleneinträge ist einfacher als die für die erste Möglichkeit benötigte Suche nach einem freien Index, da der Manager die Tabelle mit `get-next`-Aufrufen durchgeht und keine genaue Kenntnis der verwendeten Indizierung benötigt.

5.2.4 Auslösen von Aktionen

Da SNMP keine Aktionen auf MIB-Variablen definiert, sondern nur deren Lesen und Setzen, muß ein Verfahren vereinbart werden, durch Setzen bestimmter reservierter Variablen Aktionen auszuführen. Der Codine Agent muß dann bei einem Schreiben der entsprechenden Variablen die entsprechenden Aktionen ausführen.

Objekt	Aktionen					
Host	`add`	`delete`				
Dämon			`start`	`kill`		
Queue	`add`	`delete`	`start`	`stop`	`enable`	`suspend`
Job		`delete`			`enable`	`suspend`
Queuefamilie	`add`	`delete`				
Benutzer	`add`	`delete`				
Index der Aktion	1	2	3	4	5	6

Tabelle 5.3: Aktionen der Codine-MIB

Die möglichen Aktionen mit den zugehörigen Objekten sind in Tabelle 5.3 aufgeführt. Zur Realisierung dieser Aktionen ist es also nötig, in jeder der Listen noch eine Variable `Action` hinzuzufügen und die Aktionen über Integer zu identifizieren. So wird bei der Queueliste noch die Spalte `coqlAction` hinzugefügt, auf die durch Schreiben eines Integers die Aktion mit dem entsprechenden Index ausgeführt wird. Entsprechend kommen bei den anderen Listen die Spalten `cohlAction`, `cojlAction`, `coflAction` und `coulAction` hinzu.

Kapitel 6

Entwurf einer MIB für Codine

In diesem Kapitel wird ein anhand der Anforderungen des vorhergehenden Kapitels entwickelter Entwurf einer MIB für Codine vorgestellt.

6.1 Objekte der Codine-MIB

Die hier vorgestellte Codine-MIB besteht aus den folgenden Teilen:

- Allgemeiner Teil

  In diesem Teil sind allgemeine Informationen des gesamten Codine-Systems untergebracht, die für alle beteiligten Systeme gleich sind. Dieser Teil besteht aus einer Menge einfacher MIB-Variablen.

- Hostliste

  Die Hostliste ist eine SNMP-Tabelle, in der alle am Codine-System beteiligten Rechner enthalten sind. Für jeden Host werden die für Codine relevanten Daten gespeichert.

- Queueliste

  Die Queueliste ist ebenfalls eine Tabelle mit Einträgen für alle Queues. Sie enthält die Attribute jeder Queue.

- Jobliste

  Entsprechend ist auch die Jobliste als Tabelle aufgebaut. Sie enthält für jeden Job die zugehörigen MIB-Variablen. In dieser Tabelle sind alle Arten von Jobs enthalten: diejenigen, die auf Abarbeitung warten, diejenigen, die im Moment gerechnet werden, und schon beendete Jobs, deren Daten noch für das Accounting benötigt werden.

- Queuefamilienliste

  Die Queuefamilienliste enthält für alle Familien einen Eintrag für die zugehörigen Queues und erlaubt Aktionen auf allen zu einer Familie gehörenden Queues.

- Benutzerliste

  Auch die Benutzerliste ist eine SNMP-Tabelle, die Informationen zu allen Benutzern des Codine-Systems anbietet.

Im folgenden werden die einzelnen Teile der Codine-MIB genauer vorgestellt.

6.1.1 Allgemeiner Teil

Dieser Teil besteht aus folgenden einzelnen MIB-Variablen, die für das gesamte Codine-System gültig sind.

- coVersion

  Diese Variable ist ein String, der die Versionsnummer der installierten Codine-Umgebung beinhaltet. Die Information über die Versionsnummer wird benötigt, wenn in einer späteren Version von Codine ein Teil der MIB geändert oder erweitert wird. Dann kann damit überprüft werden, ob die passende Codine-Version installiert ist.

- coMasterServerName

  Diese Variable enthält als String den vollen Namen des Rechners, auf dem der Codine-Masterserver läuft. Damit kann der Manager schnell feststellen, welcher Rechner dies ist.

- coMasterServerIP

  Dies enthält die Internet-Adresse des Codine-Masterservers. Managementprogramme benötigen in der Regel die Internet-Adresse, wogegen der Name für Benutzer praktischer ist.

- coShadowServerName

  Diese Variable enthält als String den vollen Namen des Rechners, auf dem der Codine-Shadowmaster läuft, der bei Ausfall des Masterservers seine Aufgaben übernimmt.

- coShadowServerIP

  Entsprechend beinhaltet diese Variable die Internet-Adresse des Codine-Shadowmasters.

- coAdmin

  Dieser String enthält den Namen des Ansprechpartners für die gesamte Codine-Installation entsprechend der MIB II-Variablen `sysContact`. Somit kann im Fehlerfall schnell die richtige Person kontaktiert werden.

6.1.2 Hostliste

Dieser Teil der MIB von Codine beinhaltet codinespezifische Informationen zu den beteiligten Rechnern. Er hat die Struktur einer SNMP-Tabelle mit einem Tabelleneintrag für jeden Rechner des Codine-Systems und den folgenden Spalten, die sich auf den jeweiligen Rechner beziehen:

- cohlType

  Diese Variable enthält den Typ des Rechners in der Codine-Installation. Einige Managementaufgaben sind nicht auf alle Hosttypen anwendbar. So kann anhand dieses Attributs geprüft werden, ob der richtige Hosttyp vorliegt. Mögliche Werte sind: `Master`, `Shadowmaster`, `Executionhost`, `Submithost` und `Trusted Host`. Bei der Implementierung ist darauf Rücksicht zu nehmen, daß ein Rechner auch mehrere der genannten Typs gleichzeitig repräsentieren kann (Ein Submithost kann z. B. auch Executionhost und Trusted Host sein).

- cohlStatus

  Diese Variable enthält den Status des auf dem Host laufenden Codinedämons. Logischerweise kann sie nicht geschrieben, sondern nur gelesen werden. Managementsysteme, die die zu managenden Objekte als Symbole anzeigen und diese entsprechend ihrem Status färben, können das anhand dieser Information durchführen. Diese Variable enthält nur Informationen über den Zustand des Dämonen und nicht über die laufenden Queues, da dies in der Queueliste modelliert ist.

- cohlOwner

  Hier kann man den Namen des Besitzers des Rechners erfahren. Dies ist nicht notwendigerweise identisch mit der SNMP-Variablen `system.sysContact` der MIB-II, die die Kontaktperson im Fehlerfall bezeichnet. Vielmehr wird diese Information benötigt, da der Besitzer eines Rechners einige Rechte innerhalb Codine bezüglich seines Rechners besitzt. Er kann z. B. die Queues, die auf seinem Rechner laufen, suspendieren. Davor kann mit Hilfe dieser Variable überprüft werden, ob eine berechtigte Person diese Aktion durchführen will. Logischerweise darf diese Variable nur von Personen beschrieben werden, die einen Codine-Managerstatus besitzen.

Folgende Informationen gehören auch zu diesem Bereich und werden zum Management von Codine benötigt, können aber von anderer Stelle bezogen werden. Voraussetzung hierfür ist aber eine SNMP-Library im Codine-Masterserver. Solange dies nicht vorhanden ist, kann über die folgenden Variablen ein Datenabgleich realisiert werden (siehe Datenintegration, Abschnitt 8.3).

- cohlHostname

  Der volle Name des Rechners ist Inhalt dieser Variablen. Diese Information kann auch aus der MIB II (`system.sysName`) anhand der Internet-Adresse erhalten werden. Da viele Anfragen aber über den vollen Hostnamen identifiziert werden, ist es praktisch, diese Information in der Hostliste der Codine-MIB aufzunehmen.

- cohlArch

  Diese Variable enthält die Prozessorarchitektur des Rechners. Diese Information kann zwar auch von der Host Resources MIB (HRM, [RFC 1514]) bezogen werden (`host.hrDevice.hrProcessorTable.hrProcessorEntry.hrProcessorFrwID`). Die HRM ist im Moment allerdings noch nicht auf allen Systemen verfügbar. Und selbst wenn sie vorhanden ist, ist die Inhalt dieser Variablen nicht immer korrekt implementiert[1]. Daher ist es angebracht einen eigene MIB-Variable vorzusehen, solange diese Information nicht überall implementiert ist.

- cohlOs

  Dieses Element der Hostliste bezeichnet das Betriebssystem des Rechners, damit Programme auch auf dem Betriebssystem, das sie angefordert haben, zur Ausführung gebracht werden können. Eine ähnliche Information enthält zwar auch die Variable `system.sysDescr` der MIB II, allerdings als nicht standardisierter String, der ungefähr der Ausgabe des Unix-Befehls `uname -a` entspricht. Es ist einfacher, eine neue MIB-Variable in Codine einzuführen, als nach Möglichkeiten zu suchen, die nicht standardisierte Information der MIB II zu verwerten.

- cohlLoad

  Diese Variable enthält die aktuelle Prozessorlast des Rechners, und zwar als Integer, der das 100-fache der kurzfristigen (einminütigen) *load average* bezeichnet, wie man sie mit den Unix-Befehlen `xload` oder `uptime` erhält. Dies ist für Codine eine wichtige Information, die der Masterdämon regelmäßig von allen Executionhosts abfragt. Sie wird zum Scheduling der Jobs benötigt, um zu gewährleisten, daß Jobs möglichst auf wenig ausgelasteten Maschinen gestartet werden. Wenn diese Information von allen beteiligten Rechnern auch auf andere Weise zur Verfügung gestellt wird, wenn z. B. alle Rechner die HRM implementieren, dann könnte diese Information auch aus der Variablen `host.hrDevice.hrProcessorTable.hrProcessorEntry.hrProcessorLoad` der HRM bezogen werden. Wie oben erwähnt stehen aber noch nicht für alle Rechner eine HRM-Implementierung zur Verfügung und der Codine-Masterdämon verfügt noch über keine SNMP-Library. Somit muß diese Information in der Codine-MIB enthalten sein, bis sich die beschriebene Situation ändert.

- cohlMem

  Auch auf diese Variable, die den installierten Hauptspeicher in Megabyte bezeichnet, kann verzichtet werden, wenn sich die oben beschriebene Situation geändert hat, da die gleiche Information in der HRM als `host.hrStorage.hrMemorySize` steht.

[1] Der Eintrag des Linux-CMU-SNMP-Agenten, dem einzigen mir vorliegenden Agenten, der die HRM schon implementiert, ist leer: `OID: .ccitt .nullOID`.

- cohlFreeMem

  Ebenso wird der im Moment freie Hauptspeicher in einer eigenen MIB-Variablen geführt, solange er nicht aus der HRM aus dem Tabelleneintrag zu Hauptspeicher berechnet werden kann. Er entspricht der Differenz des installierten und des benutzten Speichers: `host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageSize` − `host.hrStorage.hrStorageTable.hrStorageEntry.hrStorageUsed`.

6.1.3 Queueliste

Dieser Teil der Codine-MIB enthält die Attribute aus dem vorhergehenden Kapitel, die sich auf Queues beziehen. Er hat ebenfalls die Form einer SNMP-Tabelle mit einem Eintrag pro Queue und den folgenden Spalten:

- coqlName

  Diese Variable enthält den Namen der Queue als String. Er dient wie auch der Name des Hosts zur Identifizierung für einige Managementanwendungen.

- coqlHostname

  Der volle Name des Rechners, auf dem die Queue läuft, ist Inhalt dieser MIB-Variablen der Queueliste. Mit Hilfe dieser Variablen können Anfragen einfach realisiert werden, die sich auf alle Queues eines bestimmten Rechners beziehen.

- coqlLoad

  Die Last einer Queue ist die Anzahl der wartenden Jobs in dieser Queue. Diese Information ist über diese MIB-Variable zu beziehen, die natürlich nicht schreibbar ist. Mit ihrer Hilfe kann die Auslastung des Systems und die Anzahl aller wartenden Jobs bestimmt werden.

- coqlStatus

  Diese Variable enthält den Status der Queue. Mögliche Werte sind `disabled`, `running` und `idle`. Diese Information kann wieder verwendet werden, um den Queuesymbolen in einer Managementplattform ein entsprechendes Aussehen oder eine andere Farbe zuzuordnen. Diese Variable ist nicht schreibbar, da ein Zustandsübergang zwischen `running` und `idle` selbst geschieht. Um in den Zustand `disabled` zu gelangen, wird die Queue mit Hilfe einer Aktion, also einem Schreibvorgang auf die Aktionsvariable, angehalten.

- coqlAlive

  coqlAlive wird von der Codine-Schnittstelle CULL bereitgestellt und enthält ebenfalls Informationen über den Zustand der Queue, die von einigen Managementanwendungen benötigt wird.

- coqlOwner

  Dies bezeichnet den Name des Besitzers der Queue. Dies ist normalerweise auch der Besitzer des Rechners, wie er in der Hostliste steht. Da dies aber nicht so sein muß, sondern für jede Queue eines Rechners verschieden sein kann, wird diese Variable benötigt.

- coqlPriority

  Diese Variable enthält die voreingestellte (*default*) Priorität eines Jobs dieser Queue, sofern dieser keine bestimmte Priorität beim Start erhalten hat.

- coqlFamily

  Dies bezeichnet die Queuefamilie, die diese Queue enthält. Um eine schnelle Zuordnung von Queues zu Familien und umgekehrt zu gewährleisten, ist sowohl diese als auch der umgekehrte Eintrag in der Queuefamilienliste notwendig.

- coqlActJob

  Wenn die Queue gerade einen Job abarbeitet, so bezeichnet diese Variable den aktuellen Job. Falls die Queue leer ist, enthält diese Variable einen ungültigen Eintrag. Da innerhalb von Codine alle Jobs eine eindeutige Identifikationsnummer besitzen, genügt diese hier als Inhalt der Variablen.

- coqlJobList

  Dies ist eine Liste aller Jobs, die im Moment in der Queue auf Abarbeitung warten. Um die erneute Implementierung einer SNMP-Tabelle zu vermeiden, genügt es hier, eine Sequenz von Integern, die den Identifikationsnummern der wartenden Jobs entsprechen, zu speichern.

- coqlSuspOnCompl

  Suspension on Completion. Ist diese Variable gesetzt, so wird die Queue nach Beenden des aktuellen Jobs beendet. Dies wird benötigt, um Queues anzuhalten, ohne die darauf noch laufenden Prozesse zu stoppen.

- coqlNrJobs

  Diese Variable enthält die Anzahl der wartenden Jobs in der Queue. Neben der Information an sich ist diese Variable in Zusammenhang mit der Variablen `coqlJobList` interessant, da sie der Länge der Sequenz der Job-IDs entspricht.

6.1.4 Jobliste

Auch die Jobliste ist in Form einer SNMP-Tabelle realisiert, wobei jeder Job einem Eintrag entspricht. Diese Liste enthält wartende, rechnende und bereits beendete Jobs, da die Daten letzterer noch für das Accounting benötigt werden.

- cojlJobnummer

  Jedem Job ist zur Identifikation eine eindeutige Nummer zugeordnet, die Inhalt dieser Variablen ist. Über diese Nummer wird der Job für alle managementrelevanten Aufgaben identifiziert.

- cojlJobname

  Diese Variable enthält den vollen Namen des Jobs, da es für den Benutzer angenehmer ist, neben der Nummer auch einen Namen vorzufinden.

- cojlJobFile

  Die Beschreibung der Attribute eines Jobs zusammen mit dem genauen Aufruf des Programms ist in einer Datei gespeichert, dessen Name Inhalt dieser Variablen ist. Bei genaueren Untersuchungen des Jobs mit allen angeforderten Optionen ist es nützlich, auch direkt auf diese Datei zugreifen zu können.

- cojlExecFile

  Diese Variable enthält den Name des aufzuführenden Programms. Diese Information ist wichtig, da der Job, wenn er gestartet ist, mit diesem Namen in der Prozeßtabelle auftauchen wird.

- cojlStatus

  Wie die anderen Statusvariablen, beschreibt sie den Status des Jobs (wartend, rechnend, beendet) und kann vom Managementsystem für eine entsprechende Anzeige eines Symbols des Jobs verwendet werden. Auch für das Accounting ist es wichtig, daß nur Jobs berücksichtigt werden, die schon beendet sind.

- cojlSubmissionTime

  Diese Variable enthält den Zeitpunkt des Abschicken des Jobs in einem standardisieren Zeitformat, das die Berechnung von Zeitvergleichen, Zeitdifferenzen und die Ausgabe der Zeit in einfacher Weise unterstützt. Mit Hilfe dieser Variablen und der aktuellen Zeit kann die Wartezeit berechnet werden.

- cojlStartTime

  Diese Variable enthält im selben Zeitformat den Zeitpunkt des Starts des Jobs. Vor dem Start des Jobs enthält sie einen ungültigen Wert.

- cojlEndTime

  Und diese Variable entsprechend den Zeitpunkt der Beendigung des Jobs.

- cojlOwner

  Der Besitzer des Jobs ist in dieser Variablen mit seinem Login-Namen beschrieben. Er hat das Recht, bestimmte Operationen, wie das Ändern von Parametern und das Löschen seines Jobs zu veranlassen.

- cojlAccount

  Häufig ist der Benutzername nicht identisch mit dem Namen, der für das Abrechenmanagement verwendet wird. Z. B. wenn ein Benutzer in mehreren Projekten arbeitet und die Rechenzeiten dafür auch getrennt in Rechnung gestellt haben will. Deshalb enthält diese Variable einen Namen, der für Abrechnen der Rechenzeit des Jobs verwendet wird.

- cojlUid

  Aus praktischen Gründen ist es sinnvoll, nicht nur den Namen, sondern auch die User-ID eines Jobs in der MIB aufzunehmen. Dies erspart die Umwandlung, da einige Managementaufgaben die User-ID benötigen.

- cojlEuid

  Diese Variable enthält die effektive User-ID des Jobs, also die User-ID, unter der der Job im Moment läuft. Diese Information ist wichtig, da der laufende Prozeß durch `suid`-Aufrufe die User-ID verändern kann.

- cojlQueue

  Diese Variable enthält die Queue, in der der Job läuft, falls er schon einer Queue zugeteilt ist. Dies ist der Fall, wenn er schon rechnet oder wenn nur ein Rechner zur Auswahl steht, auf dem er rechnen kann. Ansonsten wird der Job erst einer Queue zugeteilt, wenn eine Queue leer ist. Solange besitzt diese Variable einen ungültigen Wert.

- cojlHostname

  Entsprechend enthält diese Variable den Namen des Rechners, auf dem der Job läuft, sofern er schon zugeteilt ist. Man könnte zwar auch mit der bekannten Queue in der Queueliste nach dem passenden Hostnamen suchen, aber es ist einfacher und schneller, hier auch noch eine Variable für den Hostname vorzusehen.

- cojlPriority

  Die Priorität des Jobs ist in dieser Variablen enthalten. Ihr Inhalt ist vom Besitzer des Jobs und von Usern, die als Manager oder Operator eingetragen sind, änderbar.

- cojlCpuTimeReal

  Diese Variable enthält die seit dem Start des Jobs reel vergangene Zeit in einer Einheit, die das Abrechnungsmanagement unterstützen kann. Die kleinste Einheit hierfür wäre die interne Frequenz des Rechners, die auch vom Unix-Befehl `time` verwendet wird. Da diese Einheit aber nicht standardisiert ist und auf verschiedenen Prozessoren unterschiedlich ist (ca. 100 Hz bei i[345]86, ca. 1000 Hz bei DEC Alpha), ist es sinnvoller, diese Information als Integer in vollen Sekunden anzugeben.

- cojlCpuTimeUser

Entsprechend enthält diese Variable die seit dem Start des Jobs vergangene User-Rechenzeit, ebenfalls in Sekunden. Dies ist die CPU-Zeit die zur Berechnung des Prozesses in Benutzerumgebung (*Userspace*) benötigt wurde.

- cojlCpuTimeSys

  Diese Variable enthält auf die gleiche Weise die bisher beanspruchte System-Rechenzeit. Dies ist die CPU-Zeit, die zur Berechnung eines Prozesses im Kernel (z. B. für Swappen) benötigt wurde.

- cojlMaxCpuTime

  Diese Variable beschreibt die obere Schranke der Rechenzeit eines Jobs. Nach dieser maximal zur Verfügung stehender CPU-Zeit hat der Job kein Recht mehr, weiterzurechnen und kann abgebrochen werden.

6.1.5 Queuefamilienliste

Wie die anderen Listen ist auch diese eine SNMP-Tabelle mit einem Eintrage für jede Queuefamilie.

- coflName

  Diese Variable bezeichnet den Name der Queuefamilie und dient als Identifikator für die Managementprogramme.

- coflQueueList

  Diese Variable enthält eine Liste der in der Familie enthaltenen Queues. Zusammen mit der Variable `coqlFamily` ist eine Abbildung Queue nach Familie und Familie nach Queue einfach möglich.

6.1.6 Benutzerliste

Auch die Benutzerliste ist eine SNMP-Tabelle mit einem Eintrag für jeden Benutzer.

- coulName

  Diese Variable enthält den Name des Benutzers und dient gleichzeitig als Identifikator. Der Name entspricht dabei dem Login-Namen auf dem System, so daß eine Eindeutigkeit gewährleistet ist.

- coulType

  Diese Variable bezeichnet den Typ des Benutzers, also seine Rolle in Codine. Mögliche Werte sind Operator, Manager oder User. Eine Zugehörigkeit zu einer dieser Gruppen ist für viele managementrelevanten Vorgänge von Bedeutung. So kann z. B. nur der Operator und Manager neue Benutzer eintragen, deren Attribute ändern, Parameter der Queues ändern und Jobs, die ihm nicht gehören, löschen.

- coulMaxUJobs

  Diese MIB-Variable ist ein Integer, der die maximale Anzahl abgesetzter Jobs pro Benutzer bezeichnet. Im Moment kann diese Schranke nur global eingestellt werden, d. h. der Wert wird für alle Benutzer der selbe sein. Da es aber sinnvoll ist, diesen Wert für jeden Benutzer zu definieren, was wohl in Zukunft auch von Codine unterstützt werden wird, ist er Teil der Benutzerliste.

6.2 Traps

Die Traps stellen eine Möglichkeit der asynchronen Kommunikation dar, die es dem zu managenden System ermöglicht, den Manager zu kontaktieren. Die Schnittstelle zu Codine geschieht nicht über CULL, sondern über einen SNMP-Agent, der mit dem codineeigenen Monitoringdämon `rmond` zusammenarbeitet. Er setzt die vom `rmond` kommenden wichtigen Logmeldungen in Traps um und sendet diese an die Managementplattform. SNMP sieht nur 6 verschiedene Arten von Traps vor, wovon einer *enterprise specific* ist. Dieser wird hier verwendet. Weitere Werte spezifizieren den genauen *enterprise specific*-Trap und das Trap-auslösende Objekt.

Wie schon erwähnt, werden bei SNMPv1 Traps mit UDP versendet, was sie Zustellung nicht garantiert. Sobald die Voraussetzungen für einen Umstieg auf SNMPv2 vorhanden sind, sollte umgestiegen werden und SNMPv2-`inform`-PDUs verwendet werden, die eine sichere Zustellung garantieren.

Für folgende Ereignisse sollen von Codine-`rmond` die nötigen Informationen erhalten werden und SNMP-Traps kreiert werden:

- Daemon_down

  Für den Fall, daß ein Dämon gestoppt wird oder abstürzt, wird ein Trap geschickt. Somit wird der Manager alarmiert und über die Trap-Infrastruktur der Managementplattform können Aktionen ausgelöst werden, um den Fehler zu beheben.

- Daemon_up

  Beim Start eines Dämons wird dieser Trap ausgelöst. Generell verschicken viele Instanzen sowohl einen Trap beim Stop als auch beim Start.

- Queue_down

  Falls nicht der Dämon, sondern nur die auf einem Rechner laufende Queue gestoppt wird, wird dafür ein anderer Trap ausgelöst. Dies wird auch hier realisiert.

- Queue_up

  Entsprechend wird auch ein Trap generiert, wenn die Queue wieder startet.

Für die beiden folgenden Situationen ist es auch vorstellbar, Traps zu kreieren:

- Job_scheduled

  Dieser Trap könnte geschickt werden, wenn ein Job vom Batchsystem auf einem Rechner gestartet wurde.

- Job_finished

  Entsprechend würde dieser Trap abgeschickt, wenn ein Job beendet ist.

Allerdings wird eine Implementierung der beiden letzten Traps eine große Menge von Traps erzeugen, nämlich für jeden Job zwei Stück, was das Managementsystem stark belasten kann. Wenn man den damit verbundenen Netzverkehr und Managementaufwand berücksichtigt, ist es sinnvoller, diese beiden Traps nicht zu implementieren. Diese Information ist auch weniger für den Manager als vielmehr für den Benutzer des Batchsystems relevant. Dieser kann auch die Dienste des Codine-`rmond` direkt in Anspruch nehmen, ohne über die Managementplattform zu gehen.

6.3 Zentrale oder Verteilte MIB

Bei der Realisierung der vorgestellten MIB stellt sich noch die Frage, ob sämtliche MIB-Variablen von einem SNMP-Agent an einer Stelle zentral zur Verfügung gestellt werden sollen, oder ob jeder am Codine-System teilnehmende Rechner einen SNMP-Agent haben soll, der den Teil der MIB implementiert, der für den jeweiligen Rechner relevant ist.

6.3.1 Zentrale Codine-MIB

Bei einer zentralen MIB für Codine, die als Schnittstelle zum Server dient, müssen alle Informationen an dieser Stelle zur Verfügung stehen. Diese MIB beinhaltet also alle Listen, die gerade vorgestellt wurden, eine Liste aller Hosts mit ihren jeweiligen Attributen, eine Liste aller Queues, aller Jobs, aller Queuefamilien und aller Benutzer und den allgemeinen Teil. Somit enthält diese MIB alle Objekte des Codine-Systems, unabhängig von ihrem Ort, an dem sie sich befinden.

6.3.2 Verteilte Codine-MIB

Bei einer verteilten Codine-MIB gibt es mehrere Möglichkeiten zur Verteilung der Daten. Zum einen können alle Informationen sowohl über die Server-MIB – wie im zentralen Fall – zugänglich sein. Die Client-MIBs bieten darüberhinaus die zu ihnen gehörenden Daten, also Informationen über den Rechner selber und über Queues und Jobs, die auf dem Rechner laufen, an. Diese Möglichkeit bietet sich vor allem für eine Migration von einer zentralen MIB auf eine verteilte an und zur Evaluierung, welche der beiden letztgenannten Möglichkeiten für eine bestimmte Installation am günstigsten ist.

Eine doppelten Datenhaltung in Client- und Server-MIB kann auch Inkonsistenzprobleme z. B. beim Setzen einer Variable auf einem Client und der Abfrage der gleichen Variable am Server hervorrufen. Um dies zu vermeiden, kann man die Daten auch so auf die Client- und Server-MIB aufteilen, daß über die Server-MIB nur Daten bereitgestellt werden, die keinem Client zugeordnet werden können und alle anderen Informationen in der passenden Client-MIB vorhanden sind. Ein Nachteil der verteilten Datenhaltung ist aber die umständliche Suche nach Informationen, die nicht einem Client zugeordnet werden können, z. B. die Daten eines Jobs, der schon gestartet wurde und von dem man wissen will, auf welchem Rechner er sich befindet. Hierzu müßten alle Client-MIBs durchsucht werden.

Um diese aufwendige Suche zu vermeiden, sollte selbst im verteilten Fall eine lückenhafte Server-MIB existieren, die nur die Elemente enthält, die benötigt werden, die einzelnen Objekte den verteilten Client-MIBs zuzuordnen. Dies wird im Normalfall der Identifikator (Name oder ID) und der zugehörige Rechner sein. Durch eine schnelle Abfrage dieser Server-MIB kann der Ort der Information der Client-MIB bestimmt und dann von dort weitere Informationen bezogen werden. Man erspart sich auch einen Teil der redundanten Datenhaltung, da die Server-MIB nur noch wenige Attribute zu den Objekten enthält. Allerdings ist der Implementierungsaufwand deutlich höher, wie im zentralen Fall.

6.3.3 Wahl der Realisierung

Die Wahl der Realisierung der MIB hängt von der Häufigkeit und Art ihrer Benutzung ab. Die Vorteile der verteilten MIB zeigen sich bei Abfragen von Informationen, denen man einen Rechner und somit eine Teil-MIB zuordnen kann. Diese müssen sich allerdings durch eine umständliche Art der Suche nach Informationen, von denen man den Ort der Verfügbarkeit innerhalb der verteilten MIB nicht kennt, erkauft werden. Um genau feststellen zu können, welcher der Ansätze besser ist, ist es nötig, die genauen Häufigkeiten der Vorkommnisse jeder Operation und den Zeitgewinn durch direkten Zugriff auf eine Client-MIB zu untersuchen. Das würde aber über den Rahmen dieser Arbeit hinausgehen.

Da aber eine zentrale Realisierung alle Möglichkeiten bietet, die genannten Anforderungen zu erfüllen, und die Unterstützung von Codine durch die Schnittstelle CULL auch zentral ausgelegt ist, wird hier eine zentrale MIB realisiert.

Kapitel 7

Integrationstechniken

7.1 Einleitung

Mit integrierten Managementlösungen wird das Ziel verfolgt, eine heterogene Netz- und Systemumgebung gemäß einem einheitlichen, durch die integrierte Managementarchitektur vorgegebenen Vorgehen zu überwachen und zu steuern ([ABEC 95]). Zur Integration isolierter Werkzeuge gibt es mehrere Möglichkeiten, die [ABEC 95], [HeAb 93] und [OVPG 93] definieren.

Im folgenden werden die Integrationstechniken kurz vorgestellt. Konzepte, die diesen Integrationstechniken entsprechen, werden dann unter Einbeziehung der vorgestellten MIB im Kapitel 8 diskutiert.

7.2 Oberflächenintegration

Oberflächenintegration ist die einfachste Form der Integration. Man versteht darunter die Einbindung von Programmteilen in die grafische Benutzeroberfläche der Managementplattform. Die in der Plattform gehaltene Informationsbasis bleibt hierbei von der Integration unberührt.

7.2.1 Lose Oberflächenintegration

Falls die zu integrierende Anwendung schon eine grafische Benutzeroberfläche besitzt, die parallel zur Oberfläche der Managementplattform läuft (z. B. zwei auf X11 beruhende Oberflächen wie Motif und Openlook), so kann sie in die Menüstruktur von OpenView eingebunden werden und ist dann von der Managementanwendung aus aufrufbar. Hierbei bleibt die Anwendung an sich unverändert. [OVPG 93] nennt dies *Drop-In Application.*

Eine kleine Verbesserung der Integration kann man noch durch die Übergabe bestimmter

Umgebungsvariablen, wie die Namen der selektierten Objekte, erreichen. Falls die Anwendung eine solche Eingabe vorsieht, wird sie somit abhängig von der Selektion anders reagieren.

7.2.2 Oberflächenintegration durch Hilfsprogramme von OpenView

Falls die zu integrierende Anwendung keine grafische Oberfläche besitzt, gibt es noch die Möglichkeit, sie über ein von OpenView mitgeliefertes Programm aufzurufen, das die grafische Darstellung übernimmt.

7.2.3 Vollständige Oberflächenintegration

Bei der losen Integration eines Programms stimmt das Erscheinungsbild der Anwendung nicht mit dem der Oberfläche überein. Will man für den Benutzer eine völlig gleich aussehende grafische Schnittstelle schaffen, also gleiche Farben, gleicher Stil der Menüs und Buttons und gleiche Icons (gleiches *Look and Feel*), so kann man mit Hilfe der API von OpenView eine Managementanwendung schreiben, die die gleichen Oberflächenelemente wie OpenView erzeugt und mit der zu managenden Einheit kommuniziert. Diese Kommunikation kann hierbei durch Aufrufen von Programmen, die die Managementaufgaben ausführen, geschehen. Diese Managementanwendung kann aber auch die Kommunikation verwenden, die die bisherigen Managementprogramme verwendet haben. So könnte man z. B. für Codine eine solche Managementanwendung schreiben, die über TCP/IP mit dem Masterdämon kommuniziert, so wie es die mitgelieferten Programme getan haben. [OVPG 93] unterscheidet hier noch *Tool Applications*, die zwar OV APIs verwenden, aber eine eigene Benutzerschnittstelle haben, und *Map Applications*, die selber Maps verändern können.

7.3 Integration über einen Proxy-Agenten

Um eine Koordination der Managementvorgänge in der Plattform und dem zu integrierenden Werkzeug zu erreichen, ist es notwendig, daß der Managementplattform Informationen über die Funktionen des Werkzeugs vorliegen. Durch die Einführung eines Proxy-Agenten kann erreicht werden, daß die Plattform-Informationsbasis um Objekte des zu managenden Werkzeugs erweitert wird. Der Proxy-Agent ermöglicht hierbei den Zugang zu Informationen des Werkzeugs, indem er eine objektorientierte Sicht auf das Werkzeug erzeugt. Somit bildet der Proxy-Agent die Informationsbasis des Werkzeugs ab und ermöglicht der Plattform, durch Zugriff auf die Objekte Daten an das Werkzeug weiterzugeben und Aktionen auszulösen.

Der Proxy-Agent stellt dabei diese Objekte durch eine einheitliche Schnittstelle zur Verfügung, so daß auch andere Anwendungen darauf zugreifen können. Dies wird in einer

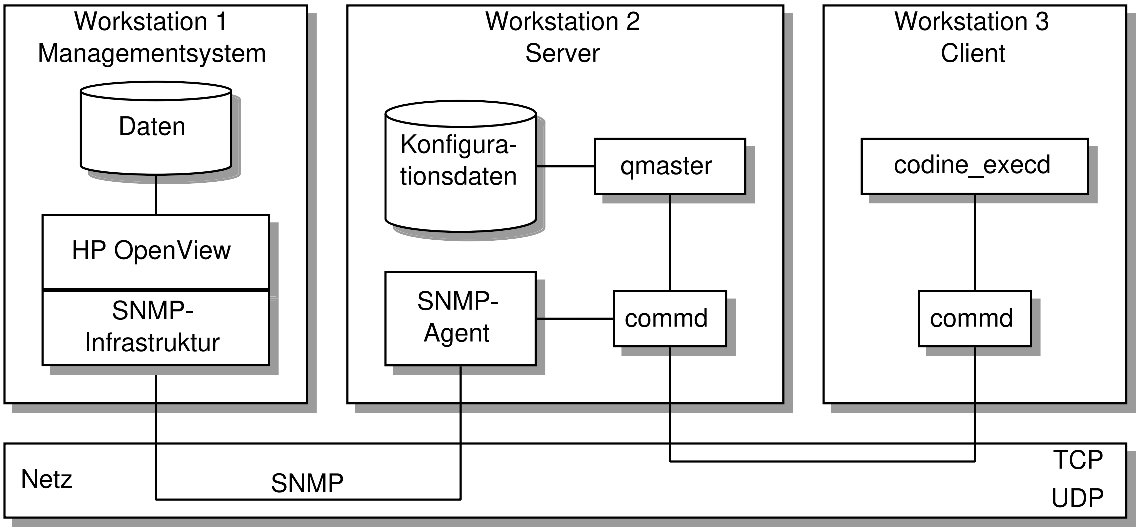

Abbildung 7.1: Integration über einen Proxy-Agenten

MIB (Management Information Base) beschrieben. Der Zugriff hierauf erfolgt über ein standardisiertes Managementprotokoll wie SNMP (Simple Network Management Protocol).

7.4 Datenintegration

Die nächsthöhere Form der Integration ist die Datenintegration, bei der die zu integrierende Anwendung und die Managementplattform auf den gleichen Datenbestand zurückgreifen. Das Zusammenführen der Plattform- und Werkzeug-Informationsbasis wird dabei durch einen Update-Mechanismus gelöst.

Hierbei ist darauf zu achten, daß bei Informationen, die beide Systeme benötigen, keine Inkonsistenzen auftreten. Dies kann vermieden werden, wenn das Werkzeug versucht, möglichst alle doppelten Daten, die auch der Netzmanagementplattform bekannt sind, über diese zu beziehen. Somit ist zu untersuchen, welche für das Werkzeug relevanten Daten der Plattform zur Verfügung stehen.

7.5 Kommunikationsintegration

Bei der Kommunikationsintegration wird auch innerhalb der verteilten Anwendung zur Kommunikation der beteiligten Komponenten ein standardisiertes Managementprotokoll wie SNMP (*Simple Network Management Protocol*) oder CMIP (*Common Management Information Protocol*) verwendet.

Hierzu benötigt jede Komponente einen aktiven und passiven Zugriff auf das Managementprotokoll. Das bedeutet, sie muß über einen SNMP-Agenten, der das Abfragen und Setzen von Variablen erlaubt, und über eine SNMP-Library verfügen, um andere Komponenten über SNMP ansprechen zu können. Außerdem wird eine Abbildung der internen Datenstruktur der verteilten Anwendung auf SNMP benötigt.

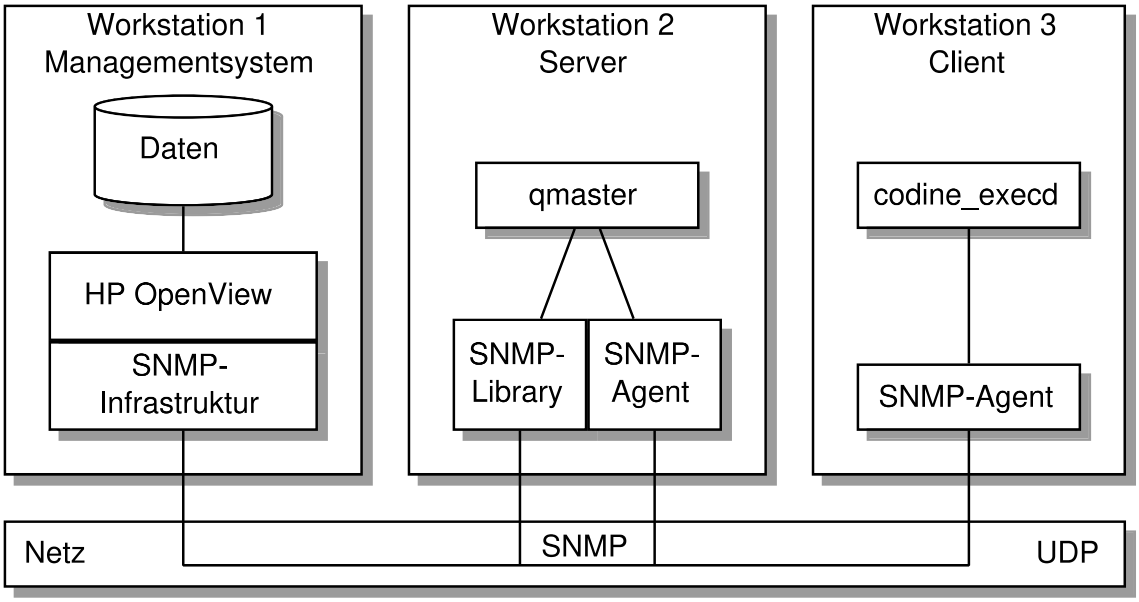

Abbildung 7.2: Kommunikationsintegration

Kapitel 8

Konzepte zur Integration

In diesem Kapitel werden Konzepte vorgestellt, mit denen man die in Kapitel 7 vorgestellten Integrationstechniken unter Einbeziehung der in Kapitel 6 vorgestellten MIB realisieren kann.

8.1 Oberflächenintegration

8.1.1 Lose Oberflächenintegration

Die codineeigenen Managementanwendungen `qmon` und `qusage` laufen wie auch HP OpenView unter X11 und Motif. Dies erfüllt die in 7.2.1 genannten Voraussetzungen und ermöglicht eine einfache Oberflächenintegration durch das Einbinden neuer Menüpunkte in HP OpenView. Bei deren Aktivierung werden dann die zu Codine gehörenden Managementprogramme (`qmon`, `qusage`) aufgerufen. Hierbei wird die aktuelle Selektion der Objekte in OpenView jedoch nicht berücksichtigt. In jedem Fall erscheint das Programm in derselben Form, wie wenn es direkt aufgerufen wird.

In der nächsten Version von Codine soll aber auch der Aufruf der codineeigenen Programme mit Kommandozeilenargumenten möglich sein, die die Auswahl auf bestimmte Hosts einschränken. Damit können dann beim Start der Programme die Namen der selektierten Hosts übergeben werden, so daß die Funktionalität der Programme auf diese Hosts beschränkt ist. Die Funktionsweise der Menüpunkte von Codine entspricht dann der der anderen Menüpunkte, die auch auf die Selektion reagieren.

Damit OpenView bei einer Selektion von Menüpunkten von Codine korrekt reagieren kann, muß es über das Vorhandensein von Codine auf dem entsprechenden Host informiert sein. Hierzu muß zu den Hostattributen die Field Registration noch ein Attribut `isCodine` hinzugefügt werden, das beim Aufruf eines Menüpunktes ausgewertet wird.

8.1.2 Oberflächenintegration durch Hilfsprogramme von OpenView

Falls das zu integrierende Programm noch keine grafische Oberfläche besitzt, kann es evtl. über von OpenView mitgelieferte Programme aufgerufen werden, die die grafische Darstellung übernehmen. Hierbei sind bisher folgende Programme vorgesehen:

- `xnmappmon`: Die textuelle Ausgabe eines Programms wird in einem Fenster dargestellt.
- `xnmgraph`: Die Werte einer MIB werden ausgelesen und als Graph dargestellt.

Voraussetzung für den Einsatz dieser Programme ist die Unterstützung von SNMP auf Seite der zu managenden Anwendung, da SNMP-Variablen als Eingabewerte verwendet werden. Damit können, wenn Codine einen SNMP-Agenten mitliefert, die darin enthaltenen SNMP-Variablen auf einfache Weise angezeigt und verändert werden.

8.1.3 Vollständige Oberflächenintegration

Um das gleiche *Look and Feel* für das Management des Batchsystems zu erhalten wie für die anderen Managementaufgaben innerhalb von HP OpenView, sollte mit Hilfe der von OpenView bereitgestellten API eine Managementanwendung geschrieben werden, die die zum Batchsystem gehörenden Objekte in einer einheitlichen Weise anzeigt.

Diese Managementanwendung wird nach Anwahl von Codinemanagement ein neues Fenster aufmachen, das ein Icon für jede Queue zeigt. Hierbei kann noch die Anzeige der Queues auf diejenigen eingeschränkt werden, die auf den vorher selektierten Hosts laufen. Die Menüleiste stellt dann zu verschiedenen Aspekten des Managements Befehle zur Verfügung. So können z. B. über ein Menü „Queue" Aktionen für die angewählten Queues aufgerufen werden (Starten, Stoppen, Umkonfigurieren). Des weiteren sollen Menüpunkte vorhanden sein, um alle in Kapitel 3.2 genannten Anforderungen zu realisieren. So sollte z. B. die Möglichkeit der Lastüberwachung des Systems durch eine dem `xload` vom X Window System entsprechende grafische Anzeige für jede Queue gegeben sein.

Diese Managementanwendung kommuniziert mit dem Teil des Batchsystems, das die nötigen Informationen bereitstellt, also entweder über ein proprietäres Protokoll direkt mit dem Hauptserver oder mit einem Agenten, der die Informationen von Codine über SNMP anbietet und auch das Ausführen von Aktionen über SNMP erlaubt.

Die Verwendung von SNMP zur Kommunikation ist hier sinnvoller als die Kommunikation über ein proprietäres Protokoll, da Managementplattformen schon einige Funktionalität zum SNMP-Management anbieten, wie z. B. die in Abschnitt 8.1.2 erwähnten Programme, so daß der Implementierungsaufwand auf der Plattformseite geringer ausfällt. Dies erleichtert auch das Portieren auf andere Managementplattformen. Eine vollständige Oberflächenintegration ist also nur in Zusammenhang mit einer Integration über einen Proxy-Agenten sinnvoll.

8.2 Integration über einen Proxy-Agenten

Ein Proxy-Agent muß die Informationen des Batchsystems, die für die Plattform relevant sind, durch eine einheitlichen Schnittstelle zur Verfügung stellen. Hierzu werden die Daten in einer MIB (Management Information Base) modelliert und über das Managementprotokoll SNMP zur Verfügung gestellt. Somit wirkt der Proxy-Agent als *Gateway* zwischen dem Managementprotokoll SNMP und dem internen Kommunikationssystem von Codine und bildet die codineinternen Datenstrukturen auf eine MIB ab.

8.2.1 Aufbau des Proxy-Agenten

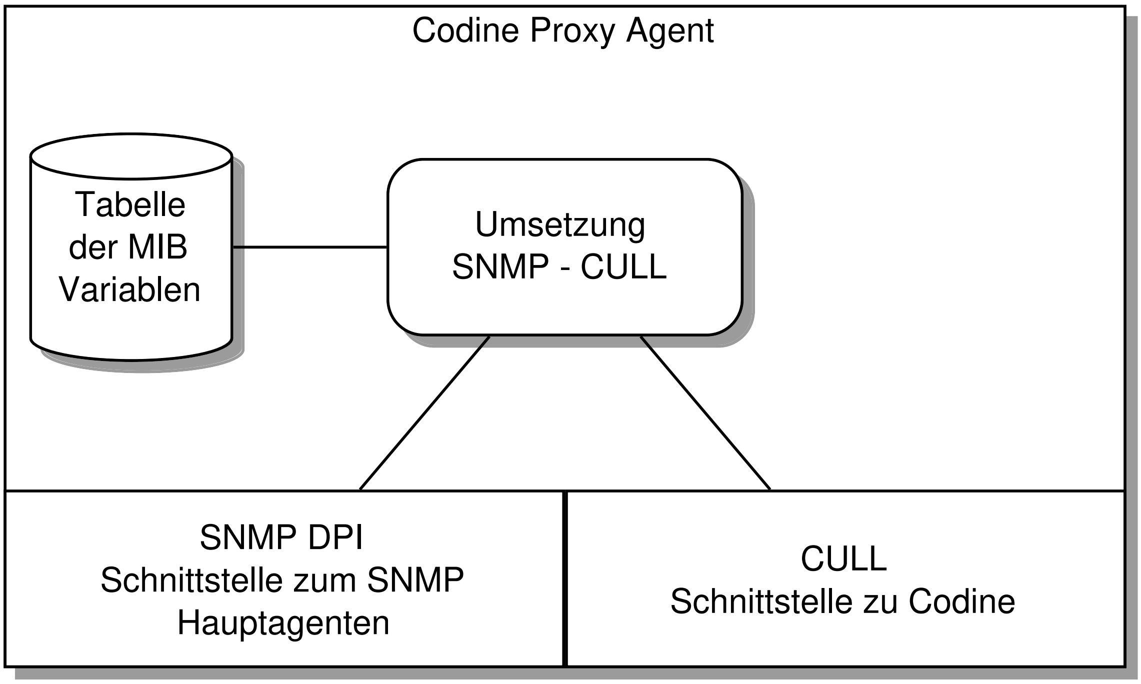

Abbildung 8.1: Aufbau des Proxy-Agenten

Der Proxy-Agent enthält zwei Schnittstellen nach außen, die DPI-Schnittstelle zur Kommunikation mit dem SNMP-Hauptagenten des Systems und die CULL-Schnittstelle von Codine, um auf die Datenstrukturen des Codine-Masterdämons zuzugreifen. Zwischen diesen beiden Schnittstellen setzt er die Anfragen mit Hilfe einer Tabelle, die die Relation von MIB-Variablen zu den codineeigenen Listenstrukturen enthält, um. Dies ist in Abbildung 8.1 veranschaulicht.

8.2.2 Funktionsweise des Proxy-Agenten

Start

Nach dem Start meldet sich der Proxy-Agent mit Hilfe von Aufrufen der DPI-Schnittstelle beim SNMP-Hauptagenten an und reserviert seinen Teilbaum der MIB. Anfragen auf diesen Teilbaum werden dann an den Subagenten weitergeleitet. Eine genauere Beschreibung der Funktionsweise der DPI-Schnittstelle befindet sich in [HaHa 94].

get

Bei einem `get`-Aufruf wird dem Proxy-Agenten die Objekt-ID der Anfrage vom Hauptagenten übergeben. Durch eine Suche in einer Tabelle im Proxy-Agenten kann er diese OID in zwei Teile splitten, dem Teil, der die Variable oder Tabelle innerhalb der MIB spezifiziert, und dem Teil, der als Argument (also z. B. als Index) dient. Zu jeder MIB-Variablen oder -Tabelle existiert eine Funktion, die den zweiten Teil der OID als Argument übergeben bekommt und den gesuchten Wert als Ergebnis zurückliefert. Für Werte, die in den codineeigenen Listen enthalten und die über die CULL-Schnittstelle zu erhalten sind, rufen diese Funktionen eine weitere Funktion auf, die den gesuchten Wert über die CULL-Schnittstelle bezieht. Die Schwierigkeit hierbei ist die Identifizierung der Tabellen bzw. Listeneinträge, da in der MIB über die OID und somit über einen oder mehrere Integer Einträge identifiziert werden, wogegen innerhalb von Codine Namen zur Identifikation dienen.

Von den in Abschnitt 5.2.2 auf Seite 33 vorgestellten Indizierungsmöglichkeiten wird hier die Indizierung mit Internetadressen hergenommen, um einen größeren Teil der Arbeit im Agenten zu erledigen und Abfragen einfacher zu gestalten.

Am Beispiel einer Nachfrage nach dem Status eines Hosts kann man die Funktionsweise des `get`-Befehles zeigen. Angenommen, die Hostliste der Codine-MIB ist unter der MIB-Variablen .1.3.6.1.3.100.6.2.1.1 zu erreichen und die Spaltennummer für die Spalte `cohlStatus` ist 4. Vor der Anfrage wandelt der Manager den Hostnamen `hpsp1` mit `gethostbyname` in die Internetadresse um (129.187.214.32). Die gesuchte Information kann nun mit der OID .1.3.6.1.3.100.6.2.1.1.4.129.187.214.32 abgefragt werden. Der Subagent splittet diese OID in die OID der Hostliste inklusive Spalte (.1.3.6.1.3.100.6.2.1.1.4) und ruft eine Funktion `get_hoststatus` mit der Internetadresse des Rechners als Argument auf. In der Funktion `get_hoststatus` wird die IP-Adresse wieder in einen Rechnernamen zurückübersetzt und dann eine Abfrage der CULL formuliert, die folgendermaßen aussieht:

```
where = lWhere(HostListT, "%s = %s", HO_hostname, "hpsp1");
element = lFindFirst(hostlist, where);
if (element) ergebnis = lGetInt(element, HO_status);
```

In der ersten Zeile wird das Selektierkriterium, die Übereinstimmung der Namen, festgelegt. Dann wird das erste passende Listenelement gesucht (da die Namen eindeutig sind, darf es nur ein entsprechendes Element geben) und der gesuchte Wert ausgelesen. Dieser Wert wird dann als Ergebnis der Funktion zurückgeliefert und dem Hauptagenten weitergegeben, der ihn dann wiederum an den aufrufenden Manager ausliefert.

set

Das Setzen von Variablen geschieht auf analoge Weise wie das Abfragen. Der Manager setzt die OID aus der MIB-Variablen der Tabelle und einem Index zusammen und setzt mit dem SNMP-Befehl `set` den Wert des entsprechenden Objekts. Der Agent splittet wieder die OID und setzt mit `lSetInt` oder `lSetString` das entsprechende Listenelement. Anschließend muß der Agent veranlassen, daß die Änderung an Codine weitergegeben wird.

Damit dies funktioniert, muß die Schnittstelle zu Codine-CULL das Schreiben aller Variablen erlauben, die in der MIB als schreibbar angegeben werden, und der Codinemasterdämon muß die von CULL geänderten Listen berücksichtigen. Die Schnittstelle CULL befindet sich im Moment noch in Entwicklung, aber es ist abzusehen, daß diese Operationen mit CULL und dem Befehl `cod_api` (siehe Anhang A.2 auf Seite 70) möglich sein werden.

Aktionen

Wie in Abschnitt 5.2.4 auf Seite 35 beschrieben wurde, enthält die MIB für Codine Variablen, die die Ausführung von Aktionen ermöglichen. Dabei identifiziert die Zahl, die an diese Stelle geschrieben wird, die Aktion, die ausgeführt wird. Die Schnittstelle CULL und der Befehl `cod_api` ermöglichen neben dem Erhalten und Setzen von Werten auch, neue Einträge zu kreieren bzw. Listenelemente zu löschen.

Für Aktionen, die über SNMP realisiert werden, ist der SNMP-Agent die ausführende Instanz. Er muß diese Aktionen entweder selber ausführen oder deren Ausführung veranlassen. Dies kann über die Schnittstelle CULL geschehen, wenn sie die Operationen ermöglicht, die in der Tabelle 5.3 auf Seite 35 aufgeführt sind. Wenn dies nicht der Fall ist, muß der Agent die codineeigenen Werkzeuge[1], mit denen bisher solche Operationen ausgeführt werden, aufrufen.

Dies sei an einem Beispiel verdeutlicht: Der Manager will die erste Queue auf dem Rechner `hpsp1` (129.187.214.32) suspendieren. Sei 3.1.1 die MIB-Variable für die Queueliste innerhalb der Codine-MIB[2], 2 die Spalte für `action` und 6 der Index für die Aktion `suspend`, so wird die Aktion durch Schreiben des Wertes 6 in die OID 3.1.1.2.129.187.214.32.1 ausgelöst. Der Agent führt daraufhin den Befehl `qmod -s hpsp1.q1` aus oder eine entsprechende Aktion über CULL, sofern das unterstützt wird.[3]

[1] Eine Liste dieser Werkzeuge ist in Tabelle A.1 auf Seite 68 zu finden.

[2] Die Codine-MIB ist bisher als experimentelle MIB unter der OID .1.3.6.1.3.100.6 vorgesehen

[3] Nebenbemerkung: Diese Überlegungen wären sicherlich einfacher gewesen, wenn der Sourcecode von Codine und somit der von `qmod` zur Verfügung gestanden hätte. Dann könnte man wahrscheinlich den Code von `qmod` relativ einfach in den Agenten integrieren.

8.2.3 Einflußnahme auf das Batchsystem

Bei bestimmten Aktionen auf Objekten muß der Proxy-Agent auch Informationen an das Batchsystem weitergeben, so z. B. wenn ein Attribut eines Objektes geändert wird. Ebenso muß er Aktionen auf Objekte in die entsprechenden Befehle für den Server umwandeln, um z. B. das Stoppen einer Queue zu realisieren. Wenn diese Voraussetzungen erfüllt sind, können auch Events, die in der Plattform auftreten, so konfiguriert werden, daß dadurch Aktionen des Batchsystems ausgelöst werden. Damit ist es möglich, die Verteilung der Jobs zu stoppen, wenn beispielsweise nicht mehr genug Plattenplatz zur Verfügung steht.

8.3 Datenintegration

Im Gegensatz zur Integration über einen Proxy-Agenten, die nur die Art des Zugriffs definiert, führt die Datenintegration zu einer Vermeidung redundanter Daten, indem Daten, die beide Partner benötigen, nur noch einmal abgelegt werden und beide darauf zugreifen. Solche Daten stehen entweder in der Codine-MIB zur Verfügung und werden vom Managementsystem oder anderen Anwendungen über SNMP bei Bedarf abgefragt, oder sie sind in einer anderen MIB oder in der Managementplattform enthalten und werden vom Batchsystem abgefragt. Dadurch werden auch Inkonsistenzen vermieden.

8.3.1 Update-Mechanismus

Um von allen beteiligten Systemen immer auf die gleichen Daten zugreifen zu können, wird ein Update-Mechanismus benötigt, der das Abgleichen der Datenbestände übernimmt und Änderungen auf jeder Seite an die andere Seite weitergibt. Hierzu gibt es die Möglichkeit des direkten Zugriffs auf die Information und den Umweg über eine Anforderung über einen SNMP-Trap und den Erhalt der Information durch Schreiben der MIB von Codine.

Direkter Zugriff

Die einfachste Möglichkeit hierzu ist der direkte Zugriff bei jeder Verwendung einer Information auf die Stelle, an der die Information gehalten wird. Dies ist aber nur bei Informationen möglich, die in einer MIB modelliert sind. Zum Zugriff auf fremde MIBs benötigen die Codine-Komponenten, vor allem der Masterdämon, eine SNMP-Library. Da ein solcher Zugriff mit erheblichem Kommunikationsaufwand verbunden ist, ist es sinnvoll, bestimmte Daten, die häufig benötigt werden, wie die Namen und Internetadressen der beteiligten Rechner zu cachen.

Schreiben der MIB von Codine

Einige Daten der OpenView-Datenbank sind aber nicht in einer MIB zur Verfügung gestellt, z. B. die *Capabilities*, die einem Objekt zugeordnet werden. Diese werden aber benötigt, wenn beim Start von Codine auf diese Weise die Rechner bestimmt werden sollen, auf denen das System läuft. Zum Zugriff auf solche Informationen gibt es folgende Möglichkeiten eines Update-Mechanismus unter Verwendung der in Kapitel 6 vorgestellten Codine-MIB.

Da SNMP kein vollkommen symmetrisches Protokoll ist, werden die meisten Aktionen vom Manager durch `get` und `set` initiiert. Der zu managende Client hat keine direkte Möglichkeit, Daten des Managers zu erhalten, da ihm diese Befehle nicht zur Verfügung stehen. Dies kann umgangen werden, wenn beim Start von Codine ein Trap an die Managementplattform gesendet wird, der einer Abfrage der aktuellen Konfiguration entspricht. Die Plattform antwortet durch ein direktes Schreiben der Codine-MIB mit den entsprechenden Daten. So kann die Hostliste der Codine-MIB vom Manager beschrieben und die in der Managementplattform enthaltene Konfiguration von Codine übernommen werden.

Wahl des Verfahrens

Welche Möglichkeit verwendet wird, hängt von der Art der Daten und den Veränderungen in Codine ab. Daten aus einer fremden MIB sollen auch direkt aus ihr gelesen werden. Beim Start von Codine werden allerdings Daten der Managementplattform, die Capabilities der Rechner, benötigt, die in keiner MIB zur Verfügung stehen und nur dem Managementsystem bekannt sind. Bisher liest Codine die aktuelle Konfiguration aus Dateien ein und initiiert die entsprechenden Queues auf den Rechnern, auf denen die Executiondämonen gestartet wurden. Danach stehen diese Daten über die Schnittstelle CULL und somit über die Codine-MIB zur Verfügung. Soll Codine schon beim Start Daten von der Managementplattform erhalten, so muß diese vor dem Einlesen der Konfigurationsdateien mit dem oben beschriebenen Verfahren (Senden eines Traps und Beschreiben der MIB) die aktuelle Konfiguration übermitteln. Dies verlangsamt aber den Start von Codine und lohnt sich nur dann, wenn die Konfiguration sich häufig ändert. Werden immer dieselben Rechner für Codine verwendet, so ist es effizienter und einfacher, deren Konfiguration aus Dateien zu lesen.

Wenn es für Anwendungen wie Codine eine direktere Möglichkeit gäbe, die Informationen aus der Managementstation zu erfahren, dann wäre es sinnvoll, die Konfigurationsdateien aller Anwendungen durch Informationen in der Managementplattform zu ersetzen und diese bei Bedarf auszulesen. Das asymmetrische Protokoll SNMP, das auch die Übertragung strukturierter Daten wie Tabellen erschwert, ist hierzu nicht gut geeignet.

8.3.2 Redundante Daten

In diesem Abschnitt wird untersucht, welche Daten sowohl von Codine als auch von der Managementplattform benötigt werden und bisher redundant in beiden Systemen vorhanden sind.

Last des Rechners

Die Last des Rechners wird von Codine bei der Verteilung der Jobs auf die beteiligten Rechner benötigt. Bisher geben die Codine-Executiondämonen diese Information an den Masterserver weiter. In der MIB-Tabelle `host.hrDevice.hrProcessorTable` der *Host Resources MIB (HRM)* gibt es aber auch einen Eintrag für die Last jedes Prozessors eines Rechners. So ist der Tabelleneintrag für den ersten (und meist einzigen) Prozessor `hrProcessorEntry.hrProcessorLoad.1`. Die hier angegebene Zahl entspricht der durchschnittlichen Last des Prozessors in der letzten Minute in Prozent.

Architektur des Rechners

Die Prozessorarchitektur des Rechners wird für die Zuteilung der Programme benötigt, die in der Regel für nur eine Architektur kompiliert sind. Anstatt diese bei der Konfigurierung eines neuen Rechners anzugeben und in Konfigurationsdateien abzuspeichern, kann der Masterdämon diese auch aus der HRM in der oben genannten Prozessortabelle unter `hrProcessorEntry.hrProcessorFrwID`) erhalten.

Betriebssystem

Die Ausführung eines Programms bedingt in der Regel auch ein bestimmtes Betriebssystem. Die Information kann man möglicherweise aus der MIB II unter `system.sysDescr`) entnehmen. Allerdings ist der Inhalt dieser MIB-Variable ein nicht standardisiertet String, der der Ausgabe des Unix-Befehls `uname -a` bzw. bei Vorhandensein eines `proc`-Dateisystems der Ausgabe von `/proc/version` entspricht. Das erste Wort ist in beiden Fällen der Name des Betriebssystems.

Verfügbarer Speicher

Beim Abschicken von Jobs kann eine untere Schranke an benötigtem Hauptspeicher, Swapspeicher oder Hintergrundspeicher angegeben werden, um sicherzustellen, daß Programme, die viel Speicher zum Rechnen benötigen, auf adäquaten Rechnern gestartet werden. Diese Informationen, die bisher in einer Konfigurationsdatei stehen, können auch aus der MIB-Tabelle `host.hrStorage.hrMemorySize` der HRM bezogen werden. Diese Tabelle enthält unter anderem den Typ jedes Speichers (`hrStorageType` mit den Werten: RAM, virtual

Memory, fixed Disk), die Größe (`hrStorageSize`) und den im Moment belegten Platz (`hrStorageUsed`), aus dem sich der freie Speicher ausrechnen läßt.

Diese redundanten Daten können, wie in Abschnitt 8.3.1 auf Seite 57 beschrieben, entweder direkt von Codine aus den entsprechenden MIBs ausgelesen werden oder nach Aufforderung von Codine vom Manager in die Codine-MIB geschrieben werden. Letztere Möglichkeit empfiehlt sich nur in einer Übergangszeit, solange der Codine-Masterdämon noch nicht über eine SNMP-Library verfügt. Dann müssen die entsprechenden Variablen auch in der Codine-MIB vorhanden sein und vom Managementsystem beschrieben werden.

8.4 Protokoll- und Kommunikationsintegration

Eine Protokoll- und Kommunikationsintegration von Codine in eine Managementplattform würde bedeuten, daß sämtliche Kommunikation auch innerhalb von Codine über ein standardisiertes Managementprotokoll abgewickelt wird. Aufgrund der Einschränkungen von SNMP und des hohen Kommunikationsbedarfs zwischen den Komponenten von Codine, der vor allem aus der Übertragung von Listenstrukturen besteht, ist es mit einem großen Aufwand verbunden, dies alles umzustellen. Daraus würden sich zwar als Vorteile eine geringere Anzahl von Agenten und ein einheitlicher Zugriff auch für andere Anwendungen ergeben. Allerdings existiert bisher keine solche Anwendung, die die internen Informationen von Codine benötigt und nicht über die Codine-MIB, die ja die Schnittstelle nach außen darstellen soll, zugreifen könnte. Auch andere Anwendungen, die eine MIB und einen SNMP-Agenten für ihre nach außen exportieren Daten zur Verfügung stellen, behalten ihre eigene interne Kommunikation bei.[4]

Zudem wurde in der neuen Version 4 von Codine die Kommunikation erst umgestellt und in eigene Prozesse (`commd`) ausgelagert. Diese garantieren eine sichere Prozeß-zu-Prozeß-Kommunikation und sind der verbindungslosen Kommunikation von SNMP weit überlegen. Außerdem stellt diese Schnittstelle relativ mächtige Funktionen für die Übertragung von Listen zur Verfügung.

Daher bringt im Moment eine Umstellung der Kommunikation auf ein Managementprotokoll, selbst wenn SNMPv2 vollständig verfügbar wäre, mehr Probleme und kaum einen Gewinn, der mehr bietet als die vorgeschlagene MIB. Falls es in Zukunft allerdings Managementprotokolle geben wird, die die bemängelten Einschränkungen nicht haben, könnte man die Fragestellung neu erörtern.

8.5 Codine Monitoring

Die in Abschnitt 4.2.4 auf Seite 17 vorgestellte Möglichkeit zum Monitoring von Codine kann auch als eine Schnittstelle zum Management ausgebaut werden. So kann an den

[4] z. B. Druckerspooler, die eine MIB unterstützen; Datenbanksysteme; SAP R/3

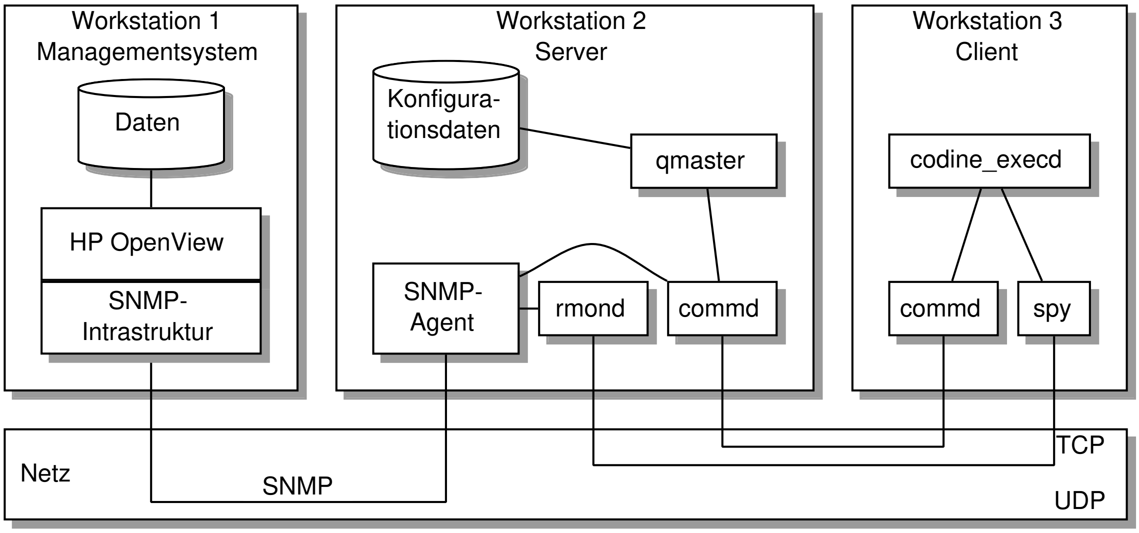

Abbildung 8.2: Aufbau unter Einbeziehung des Codine Monitoring

`rmond` ein SNMP-Agent angehängt werden, der die Meldungen, die der `rmond` erhält, nach bestimmten konfigurierbaren Kriterien auswählt und in einer Codine-Monitor-MIB zur Verfügung stellt. Diese Informationen können dann vom Managementsystem regelmäßig gepollt werden. Dieser Agent kann für wichtige Meldungen auch Traps an den Manager schicken, der daraufhin die in der Plattform konfigurierten Aktionen auslösen wird. Dieser Agent kann in den bisherigen Agenten integriert, so wie das Abbildung 8.2 zeigt, oder in einem weiteren unabhängigen Agenten realisiert werden.

An dieser Stelle können auch noch weitere Informationen über Codine, die nicht über CULL und somit über den Codine-SNMP-Agenten abfragbar sind, verfügbar sein. Über die asynchrone Kommunikation mit Traps werden Meldungen über Ausfälle der Komponenten an die Managementstation gesendet.

8.6 Managementanwendung

8.6.1 Zweck einer Managementanwendung

Als Gegenstück zum SNMP-Agenten für Codine, der die Codine-MIB realisiert, wird auch in der Managementplattform eine Managementanwendung benötigt, die unter Verwendung der Codine-MIB das Management des verteilten Batchsystems und seiner Komponenten ermöglicht.

Standardmäßig unterstützt eine Managementplattform das Management mit SNMP und erlaubt das Editieren und Überwachen von MIB-Variablen (siehe Basisanwendungen, Abschnitt 4.1.4 auf Seite 13). Dies entspricht aber noch nicht den Möglichkeiten der von Codine mitgelieferten Programme, da sich der Benutzer selber informieren muß, welche MIB-Variable welche Information beinhaltet. Um die Möglichkeit zu nutzen, Managementobjekte als Symbole darzustellen und entsprechende Operationen darauf zu erlauben, bedarf es einer Managementanwendung, die auf der Infrastruktur der Managementplattform aufsetzt.

Diese Managementanwendung wird über ein Menü Operationen bereitstellen, die sich auf die Codine-Objekte Host, Queue, Job, Familie und Benutzer und das gesamte System beziehen. Im folgenden werden die einzelnen Teile genauer spezifiziert.

8.6.2 Managementanwendung für Hosts

Hosts sind in der Managementplattform schon als Objekte vorhanden und in den Maps enthalten. Durch das schon früher geforderte Hinzufügen einer *Capability* ist es möglich festzustellen, welche Hosts zum Codinesystem gehören. Daher genügt es, dem Menüpunkt für Codinemanagement ein Untermenü hinzuzufügen, das die im folgenden beschriebenen Operationen enthält:

- Host an- und abmelden

  Dies meldet einen Host durch Verändern des *Capability*-Flags für Codine an oder ab. Es ist für alle Rechner möglich, diese Operation auszuführen. Durch das Einschalten dieses Flags wird es möglich, die anderen Managementoperationen, die Codine betreffen, auszuführen.

- Dämon starten und stoppen

  Durch diesen Unterpunkt können die Dämonen, die auf den selektierten Rechnern laufen, gestartet und gestoppt werden. Das Stoppen geschieht dabei über die MIB. Die Managementanwendung muß die zu den selektierten Rechnern passenden MIB-Action-Variablen mit dem Wert 4, der für Stoppen steht, beschreiben. Das Starten muß, wie schon beschrieben, über `rsh`-Aufrufe geschehen.

- Modifizieren der Parameter der Hosts

  Dieser Unterpunkt listet eine Tabelle aller modifizierbarer Parameter der Codine-MIB wie der Typ und der Besitzer des Rechners für die selektierten Rechner auf und gibt die Möglichkeit, diese zu verändern. Dazu öffnet die Managementanwendung ein Fenster mit der Tabelle, liest die MIB und schreibt am Ende die veränderten Werte in die MIB zurück.

Somit wird der Benutzer des Managementsystems von direkten MIB-Zugriffen verschont. Will er z. B. eine Queue eines bestimmten Rechners stoppen, so genügt es, das Symbol dieses Rechners im Managementsystem anzuwählen und über die Menüleiste die Operation „Stoppen“ aufzurufen. Aufgabe der Managementanwendung ist es, diese Transparenz zu ermöglichen.

8.6.3 Managementanwendung für Queues

Da Queues im Gegensatz zu Hosts im Managementsystem noch nicht als Objekte zur Verfügung stehen, ist es bisher auch noch nicht möglich, eine Map zu öffnen, die Symbole für Queues enthält. Da dies aber wünschenswert ist, um auf die selbe transparente Weise auch das Management von Queues zu realisieren, ist es Aufgabe der Managementanwendung, dies zu verwirklichen. So soll über einen Menüpunkt das Öffnen einer Map möglich sein, die für jede Queue der selektierten Rechner ein Symbol darstellt. In der Menüleiste dieser Map sind dann Menüpunkte vorzusehen, die folgende Operationen realisieren:

- Queue starten, stoppen, suspendieren und wieder freigeben

  Entsprechend der Aktionen die mit der MIB realisiert werden können, sind Menüpunkte vorzusehen, die diese Operationen auf die selektierten Queues anwenden. Die Managementanwendung hat wieder für den Zugriff auf die passenden MIB-Variablen zu sorgen, so daß der Benutzer kein Wissen über die genaue Implementierung der MIB besitzen muß.

- Modifizieren der Parameter für Queues

  Wie auch für Hosts soll die Managementanwendung eine benutzerfreundliche Möglichkeit bieten, die Queueparameter der MIB wie den Besitzer, die Priorität oder die *Suspension on Completion* zu editieren. Weiterhin sollte hier noch das Editieren der Queueparameter von Codine möglich sein, die nicht Teil der MIB sind, sondern in Konfigurationsdateien stehen, da sie sehr speziell sind und nicht für andere Managementaufgaben benötigt werden.

- Last und Status der Queue

  Um dem Benutzer der Managementplattform einen schnellen Überblick über den Zustand des Batchsystems zu ermöglichen, ist es praktisch, den Symbolen entsprechend der Statusvariablen der Codine-MIB ein anderes Aussehen oder eine andere Farbe geben. Dies sollte mit den Variablen `coqlStatus` und `coqlLoad` möglich sein, um sowohl den Zustand (idle, running, disabled) als auch die aktuelle Last in einer übersichtlichen Form darzustellen.

- Anzeige der Jobs der Queue

  Durch Auslesen der MIB-Variablen `coqlJobList` kann die Managementanwendung einen Menüpunkt realisieren, der die in der Queue enthaltenen Jobs anzeigt. Dies kann entweder in Textform geschehen, ähnlich wie `qstat` (siehe Abbildung 5.1 auf Seite 28), oder – mit höherem Programmieraufwand – in Form einer neuen Map mit Symbolen für die Jobs.

8.6.4 Weitere Teile der Managementanwendung

Analog zu der für Queues beschriebenen Vorgehensweise hat die Managementanwendung auch für Jobs diese Funktionalität zu realisieren, um Operationen darauf ähnlich benutzerfreundlich zu gestalten. Erschwerend kommt hier allerdings hinzu, daß Jobs wesentlich dynamischer sind. Es kommen laufend neue zum System hinzu, die schnell Status und Ort ändern.

Entsprechend sind von der Managementanwendung Operationen bereitzustellen, die sich auf Queuefamilien und Benutzer beziehen.

8.7 Änderungen in Codinekomponenten

8.7.1 Masterdämon

Um den in Abschnitt 8.3.1 auf Seite 57 vorgestellten Update-Mechanismus auf die vorgeschlagene Weise zu realisieren, benötigt der Codine-Masterdämon ein SNMP-Library, um direkt über SNMP Informationen von anderen MIBs zu erhalten.

8.7.2 Scheduler

Der Scheduler in Codine ist für die Verteilung der Jobs auf die Queues und den Start der Jobs zuständig. In der neuen Version von Codine ist es möglich, diesen Algorithmus an eigene Bedürfnisse anzupassen, wobei allerdings noch kein Anschluß an ein integriertes Management vorgesehen ist. Durch den Einbau einer SNMP-Library könnte der Scheduler auch beliebige SNMP-Variablen abfragen und den Start von Jobs von deren Wert abhängig machen. Auf diese Weise lassen sich Starts von Jobs unterbinden, wenn die dafür benötigten Ressourcen nicht zur Verfügung stehen und dies über SNMP zu testen ist, wie z. B. Netzverbindungen, freier Speicher oder freier Plattenplatz. In Zukunft werden immer mehr Anwendungen ihre Informationen über SNMP zur Verfügung stellen, was einer solchen Schnittstelle viele Erweiterungsmöglichkeiten bietet. So ist es im Moment nicht einfach, einen Job in Abhängigkeit von der Verfügbarkeit einer Softwarelizenz zu starten, da Lizenzserver keine einheitliche Möglichkeit bieten, die Anzahl der freien Lizenzen abzufragen. Falls diese Lizenzen auch außerhalb von Codine verwendet werden, nützt auch keine Beschränkung der Queues, die diese Software anbieten. Würde dies in einer Lizenz-MIB stehen, so können darauf alle Programme zurückgreifen.

8.8 Weitere Managementszenarien

Im folgenden wird untersucht, inwieweit die in Abschnitt 3.4 vorgestellten Managementszenarien mit dem vorgestellten Ansatz realisiert werden können. Das Ziel dabei ist nicht, eine konkrete Realisierungsmöglichkeit vorzustellen, es soll vielmehr gezeigt werden, daß es mit den vorgestellten Konzepten grundsätzlich möglich ist, den Anforderungen zu entsprechen, wobei aber meistens die anderen am Management beteiligten Systeme noch nicht die benötigte Funktionalität haben.

8.8.1 Lastsenkung bei interaktiver Last

Grundsätzlich ist es möglich, die Last, wie in Abschnitt 3.4.2 auf Seite 9 beschrieben, von Parametern in der Plattform abhängig zu machen, da die MIB diese Operationen erlaubt und sie von der Managementplattform aus gestartet werden können. Bisher fehlen allerdings noch Möglichkeiten, auf der Seite der Plattform die nötigen Eingabeparameter als Entscheidungskriterien für eine Lastsenkung zur Verfügung zu stellen. So kann die Managementstation über SNMP zwar die Last eines Prozessors abfragen, aber dabei nicht unterscheiden, ob die Last von interaktiven Benutzern, vom Batchsystem oder von anderen Komponenten verursacht wird. Auch eine Lastsenkung beim Einloggen eines neuen Benutzers ist über eine Plattform nur schwer realisierbar, da diese nicht sofort und direkt davon erfährt. Da eine solche Maßnahme aber nur sinnvoll ist, wenn eine sofortige Suspendierung der Queues möglich ist, da in den ersten Sekunden nach dem Einloggen der Rechenbedarf hoch ist, sollten solche Vorkehrungen direkt beim Client ohne Einwirken des Managementsystems geschehen. Über die Plattform sind dann nur noch die Parameter über entsprechende MIB-Variablen einzustellen.

8.8.2 Beschränkungen für bestimmte Benutzergruppen

Um für bestimmte Benutzergruppen Beschränkungen, wie eine voreingestellte Priorität, die maximale Anzahl der Jobs oder die maximale CPU-Zeit, festzulegen, muß Codine diese Limits für jeden User unterstützen. Bisher gibt es aber nur Beschränkungen, die für eine Queue oder für die gesamte Installation gelten und die über die entsprechenden MIB-Variablen modifiziert werden. Wenn dies pro Benutzer geregelt werden soll, müssen diese MIB-Variablen vom queuespezifischen und globalen Teil der MIB in den benutzerbezogenen Teil versetzt werden.

8.8.3 Abhängigkeit von äußeren Bedingungen

Durch das Hinzufügen einer SNMP-Bibliothek für den Codine-Scheduler (8.7.2) kann der Start von Jobs auch von Variablen in verschiedenen MIBs abhängig gemacht werden.

Dies bietet sich insbesondere in Zusammenarbeit mit Lizenzservern an, die mit der Managementplattform kooperieren. Allerdings gibt es auch hier noch keine Implementierungen und auch keine standardisierte Lizensserver-MIB.

8.8.4 Zeitmanagement

Mit den bisher zur Verfügung stehenden Mitteln kann man nur ein einfaches Zeitmanagement realisieren. Hierzu kann man mit dem Unixtool `cron` zu bestimmten Zeiten Skripten starten, die die Umkonfiguration des Systems übernehmen. So läßt sich der Betrieb tageszeitabhängig steuern.

Diese Methode ist aber relativ umständlich und nicht sehr flexibel, und besondere Tage wie Feiertage und Ferien werden nicht erkannt. Praktischer wäre es, wenn die Managementplattform ein Zeitmanagement zur Verfügung stellen würde, das die Definition mehrerer Zeitniveaus erlaubt. Zu jedem dieser Niveaus und zu einem Niveauwechsel können dann Aktionen ausgeführt werden, die bestimmte Anwendungen starten oder sie umkonfigurieren. Da dies innerhalb der Managementplattform geschieht, kann auf die Plattforminfrastruktur zurückgegriffen werden, um die Aktionen über SNMP auszuführen oder um sie von weiteren SNMP-Variablen abhängig zu machen. Genauso kann jedes Zeitniveau bzw. der Wechsel von einem Kalender und äußeren Bedingungen abhängen.

Ein vorstellbares Szenario in Zusammenhang mit einem Batchsystem wäre die Definition dreier Niveaus: Nacht, Tag und Hochlast. Der Wechsel von Nacht- zu Tagbetrieb erfolgt zu bestimmten Uhrzeiten zu Arbeitsbeginn, Hochlast wird definiert, wenn bestimmte, in der Managementplattform überwachte Schwellwerte, wie z. B. die Anzahl der Benutzer oder die Netzauslastung, überschritten werden, und die Umstellung von Tag- auf Nachtbetrieb geschieht, wenn eine festgelegte Anzahl Benutzer unterschritten wird. Das Batchsystem läuft entsprechend bei Nachtbetrieb auf allen Maschinen, bei Tagbetrieb nur auf einigen Maschinen mit Vorrang für den interaktiven Benutzer, und bei Hochlast werden alle Queues suspendiert, um das gesamte Netz zu entlasten.

Durch die Möglichkeit, mit der vorgestellten MIB auf das Batchsystem einzuwirken, es umzukonfigurieren und Queues zu suspendieren, kann ein solches Zeitmanagement realisiert werden, sobald die Plattform die entsprechenden Dienste zur Verfügung stellt.

8.8.5 Unterstützung von Wartungsarbeiten

Auch die Unterstützung von Wartungsarbeiten kann mit einem solchen Zeitmanagement implementiert werden. Hierzu werden rechtzeitig vor dem Wartungsdatum die Rechner in einen entsprechenden Zustand gesetzt, der dann z. B. nur noch kleine Jobs oder Jobs, die checkpointen können, startet.

8.8.6 Weitere Zusammenarbeit mit der Managementplattform

Nach Implementierung des SNMP-Agenten für Codine, das die Codine-MIB zur Verfügung stellt, und der in Abschnitt 8.7 auf Seite 64 vorgestellten Änderungen durch Hinzufügen von SNMP-Bibliotheken ist Codine auch für eine weitere Zusammenarbeit mit dem Managementsystem vorbereitet, da an allen wichtigen Punkten eine Kommunikation mit dem Managementsystem besteht. Codine kann selber auf alle MIB-Variablen zugreifen und über Traps die Managementstation kontaktieren, und die Managementstation kann sich über den Zustand von Codine durch Lesen der MIB informieren und ihn durch Schreiben verändern.

Kapitel 9

Zusammenfassung

In der vorliegenden Diplomarbeit wurde untersucht, wie eine bestehende verteilte Anwendung in das integrierte Management einer Plattform einbezogen werden kann. Dabei wurden ausgehend von den Anforderungen, die an ein solches Management gestellt werden, Konzepte für die verschiedenen Integrationsstufen entwickelt und eine MIB vorgestellt.

Um nun diese Anbindung von Codine an ein Managementsystem zu realisieren, ist als erster Schritt die Implementierung des SNMP-Agenten nötig, der die vorgestellte Codine-MIB implementiert. Dann kann in der Managementplattform eine Managementanwendung geschrieben werden, die die Codine-MIB verwendet und mit ihrer Hilfe das Management von Codine von der Plattform aus möglich macht. Wenn die Integration noch weitergeführt werden soll, so sind die beschriebenen Änderungen an den Codine-Komponenten – insbesondere das Hinzufügen von SNMP-Libraries – nötig, damit Codine auch seinerseits Informationen über SNMP beziehen kann. Eine Protokoll- und Kommunikationsintegration kann jedoch im Moment aufgrund der beschriebenen Einschränkungen des Managementprotokolls und des erhöhten Aufwandes nicht empfohlen werden.

Anhang A

Codine

A.1 Managementfunktionen von Codine

Objekt	Aktion	Codine-Befehl
Master	status	qstat
	kill	qconf -km
Execdämon	kill	qconf -kqs
Cluster	modify	qconf -mc
	show	qconf -sc
Queue	add	qconf -aq
	delete	qconf -dq
	modify	qconf -mq
	show	qconf -sq
	stop	qconf -dq
	suspend	qmod -s
	enable	qmod -e
Queuefamily	add	qconf -af
	delete	qconf -df
	modify	qconf -mf
	show	qconf -sf
Job	submit	qsub
	delete	qdel
User	add	qconf -au (am, ao)
Manager	delete	qconf -du (dm, do)
Operator	show	qconf -su (sm, so)

Tabelle A.1: Managementfunktionen von Codine

In Tabelle A.1 sind die Managementfunktionen, die Codine derzeit anbietet, zusammengefaßt. Dies ist ausführlich in [GENI 94] beschrieben.

A.2 Common Usable List Library

CULL Eintrag	Beschreibung	entspricht in MIB
JB_job_number	Jobnummer	cojlJobnumer
JB_job_name	Jobname	cojlJobName
JB_job_file	Jobfile	cojlJobFile
JB_exec_file	Ausführbares Programm	cojlExecFile
JB_status	Status	cojlStatus
JB_submission_time	Zeit des Abschickens des Jobs	cojlSubmissionTime
JB_start_time	Startzeit des Jobs	cojlStartTime
JB_end_time	Endzeitpunkt des Jobs	cojlEndTime
JB_owner	Jobbesitzer	cojlOwner
JB_uid	User-ID des Jobs	cojlUid
JB_euid	Effektive User-ID	cojlEuid
JB_priority	Priorität	cojlPriority
QU_priority	Default Priorität der Queue	coqlPriority
QU_status	Status der Queue	coqlStatus
QU_load_avg	Last der Queue	coqlLoad
QU_alive	Aktive Queue	coqlAlive
QU_job_list	Jobliste der Queue	coqlJobList

Tabelle A.2: Datenstrukturen von Codine

Die Tabelle A.2 zeigt einen Teil der Daten, die über die *Common Usable List Library* von Codine zu erhalten sind. Die Spezifikation der Daten, die über CULL zu erhalten sind, ist noch nicht abgeschlossen. Daher werden in der Tabelle beispielshaft nur einige Variablen aufgeführt, über die schon Informationen erhältlich waren, und die für die beschriebenen Managementanwendungen und die MIB relevant sind. Informationen für die restlichen vorgeschlagenen MIB-Variablen sollen ebenfalls über die CULL-Schnittstelle zur Verfügung stehen.

Die folgende Aufzählung zeigt eine Auswahl der Funktionen von CULL zur Bearbeitung von Listen, die für eine Realisierung des SNMP-Agenten relevant sind. Genauere Informationen zur Schnittstelle CULL werden in [GENI 95b] beschrieben.

- `lWhere(Beschreibung der Bedingung)`

  Formuliert eine Bedingung für einen `lFind`-Aufruf.

- `lFindFirst(List, Bedingung)`

Sucht in einer Liste nach dem ersten Element, daß die Bedingung erfüllt.

- `lFindNext(Listenelement, Bedingung)`

  Sucht das nächste Element, daß die Bedingung erfüllt.

- `lGetInt(Listenelement)`

  Extrahiert einen Integerwert aus dem Listenelement, das Daten in einem internen Format abspeichert.

- `lGetString, lGetChar, lGetFloat, ...`

  Entsprechende Funktionen zur Extraktion von Strings, Zeichen, Fließkommazahlen usw.

- `lSetInt, lSetString, lSetChar, lSetFloat, ...`

  Analoge Funktionen zum Setzen von Werten in Listenelementen.

Die Funktion `cod_api(Kommando, Liste, ...)` stellt die Verbindung zum Masterdämon dar. Damit sind folgende Kommandos vorgesehen:

- `COD_API_GET`: Holt vom Masterserver die mit `Liste` angegebene Liste.
- `COD_API_ADD`: Fügt der Liste ein Element hinzu.
- `COD_API_DEL`: Löscht ein Element der Liste.
- `COD_API_MOD`: Modifiziert die Liste.

A.3 Codine Remote Monitoring

rmond	Remote Monitor Dämon
rmon	Remote Monitor Applikation
spy	Client, der die Daten vom Anwendungsprogramm liefert
mconf	Konfigurationstool zur Einstellung des Systemdefaults
mlevel	Einstellen des Monitorlevels
mjob	Ermöglicht das Tracing von Jobs
mstat	Zeigt Informationen über das Remote Monitoring System
msysstat	Zeigt Informationen über das Remote Monitoring System
mdel	Löscht einen spy
mquit	Beendet das Monitoring System

Tabelle A.3: Remote Monitoring-Funktionen von Codine

Tabelle A.3 enthält die vom Codine-*Remote Monitoring*-System zur Verfügung gestellten Befehle.

Anhang B

Codine-MIB

B.1 Codine-MIB als Tabelle

Codine-MIB			
MIB-Variable			Beschreibung
coSystem	coVersion		Versionsnummer
	coMasterServerName		Name des Masterservers
	coMasterServerIP		IP-Adr. des Masterservers
	coShadowServerName		Name des Shadowservers
	coShadowServerIP		IP-Adr. des Shadowservers
	coAdmin		Codine-Administrator
coHostlist	cohlEntry	cohlIndex	Index für die Hostliste
		cohlAction	Auslösen von Aktionen
		cohlType	Typ des Rechners
		cohlStatus	Status des Dämons
		cohlOwner	Besitzer des Rechners
		cohlHostname	Name des Hosts
		cohlArch	Prozessorarchitektur
		cohlOS	Betriebssystem
		cohlLoad	Last
		cohlMem	Speicher
		cohlFreeMem	Freier Speicher
coQueuelist	coqlEntry	coqlIndex	Index für die Queueliste
		coqlAction	Auslösen von Aktionen
		coqlHostnameIP	Host, auf dem die Queue läuft
		coqlHostname	Host, auf dem die Queue läuft
		coqlName	Name der Queue
		coqlLoad	Last der Queue(Anz. wartender Jobs)
Fortsetzung auf nächster Seite			

Fortsetzung von vorhergehender Seite			
MIB-Variable			Beschreibung
		coqlStatus	Status der Queue
		coqlAlive	Aktive Queue
		coqlOwner	Besitzer der Queue
		coqlPriority	Default Priorität der Jobs dieser Queue
		coqlFamily	zugehörige Queuefamilie
		coqlActJob	Aktuell berechneter Job
		coqlJobList	wartende Jobs
		coqlSuspOnCompl	Suspension on Completion
		coqlNrJobs	Anzahl der wartenden Jobs
coJoblist	cojlEntry	cojlIndex	Index für Jobliste = Jobnummer
		cojlAction	Auslösen von Aktionen
		cojlJobname	Name des Jobs
		cojlJobFile	Datei der Jobbeschreibung
		cojlExecFile	Auszuführendes Programm
		cojlStatus	Status des Jobs
		cojlSubmissionTime	Submissionzeit
		cojlStartTime	Startzeitpunkt
		cojlEndTime	Endzeitpunkt
		cojlOwner	Jobbesitzer
		cojlAccount	Account zum Abrechnen
		cojlUid	User-ID des Jobs
		cojlEuid	Effektive User-ID des Jobs
		cojlQueue	Queue, in der der Job gerechnet wird
		cojlHostname	Host, auf dem der Job gerechnet wird
		cojlPriority	Priorität
		cojlCpuTimeReal	Reale Rechenzeit
		cojlCpuTimeUser	User CPU-Zeit
		cojlCpuTimeSys	System CPU-Zeit
		cojlMaxCpuTime	Maximale CPU-Zeit
coQFamlist	coflEntry	coflIndex	Index für Queuefamily-Liste
		coflAction	Auslösen von Aktionen
		coflName	Name der Familie
		coflQueueList	Enthaltene Queues
coUserlist	coulEntry	coulIndex	Index für Benutzer
		coulAction	Auslösen von Aktionen
		coulName	Name des Benutzers
		coulType	Typ des Benutzers
		coulMaxUJobs	Max. Anzahl Jobs pro User

Tabelle B.1: Codine-MIB

B.2 Codine-MIB in ASN.1

Im folgenden wird die Codine-MIB in ASN.1-Notation aufgeführt.

```
-- CODINE-MIB
-- Diplomarbeit an der TU-Muenchen
-- Aufgabensteller: Prof. Dr. H.-G. Hegering
-- Autor: Clemens Durka
-- e-mail: durka@informatik.tu-muenchen.de


SNMPv2-CODINE DEFINITIONS ::= BEGIN;

    IMPORTS
            mgmt, NetworkAddress, IpAddress, Counter, Gauge,
                    TimeTicks
                FROM RFC1155-SMI
            OBJECT-TYPE
                FROM RFC-1212;


    nullOID       OBJECT IDENTIFIER ::= { ccitt 0 }
    internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
    directory     OBJECT IDENTIFIER ::= { internet 1 }
    mgmt          OBJECT IDENTIFIER ::= { internet 2 }
    experimental  OBJECT IDENTIFIER ::= { internet 3 }
    private       OBJECT IDENTIFIER ::= { internet 4 }
    enterprises   OBJECT IDENTIFIER ::= { private 1 }



   -- currently the Codine MIB is defined under the LRZ experimental
   -- MIB, which is fixed under experimental.100 !
   -- with registration the tree probably has to be moved.

   LRZ OBJECT IDENTIFIER ::= { experimental 100 }

   -- FoPra Bastian Pusch (puschb@informatik.tu-muenchen.de)
   -- LRZ-LOG OBJECT IDENTIFIER ::= {LRZ 1}

   -- Diplomarbeit Uwe Krieger (krieger@informatik.tu-muenchen.de)
   -- LRZ-UNIX OBJECT IDENTIFIER ::= {LRZ 2}

   -- Diplomarbeit Erwin Hainzinger (hainzing@informatik.tu-muenchen.de)
```

```
-- LRZ-PRINTER OBJECT IDENTIFIER ::= {LRZ 5}

-- Diplomarbeit Clemens Durka (durka@informatik.tu-muenchen.de)
CODINE OBJECT IDENTIFIER ::= {LRZ 6}

-- textual conventions
Boolean ::= INTEGER{true(1),false(2)}
KBytes ::= INTEGER(0..2147483647)
Bytes ::= INTEGER(0..2147483647)
DisplayString ::= OCTET STRING (SIZE (0..255))


-- Groups in LRZ-UNIX-MIB

coSystem       OBJECT IDENTIFIER ::= {CODINE 1}
coHost         OBJECT IDENTIFIER ::= {CODINE 2}
coQueue        OBJECT IDENTIFIER ::= {CODINE 3}
coJob          OBJECT IDENTIFIER ::= {CODINE 4}
coQFam         OBJECT IDENTIFIER ::= {CODINE 5}
coUser         OBJECT IDENTIFIER ::= {CODINE 6}

-- Codine System Group

coVersion OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The Version Number of the Codine Implementation."
   ::={ coSystem 1 }

coMasterServerName OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "The full qualified hostname of the Codine Master Server."
   ::={ coSystem 2 }

coMasterServerIP OBJECT-TYPE
   SYNTAX     IpAddress
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
```

```
            "The IP address of the Codine Master Server."
      ::={ coSystem 3 }

   coShadowServerName OBJECT-TYPE
      SYNTAX     DisplayString
      MAX-ACCESS read-write
      STATUS     current
      DESCRIPTION
            "The full qualified hostname of the Codine Shadow Server."
      ::={ coSystem 4 }

   coShadowServerIP OBJECT-TYPE
      SYNTAX     IpAddress
      MAX-ACCESS read-write
      STATUS     current
      DESCRIPTION
            "The IP address of the Codine Shadow Server."
      ::={ coSystem 5 }

   coAdmin OBJECT-TYPE
      SYNTAX     DisplayString
      MAX-ACCESS read-write
      STATUS     current
      DESCRIPTION
            "The name of the person administrating Codine."
      ::={ coSystem 6 }

   -- Codine Hostlist
   -- This part of the Codine MIB holds information on the hosts
   -- pariticipating in Codine.

   coHostlist OBJECT-TYPE
      SYNTAX SEQUENCE OF cohlEntry
      MAX-ACCESS not-accessible
      STATUS     current
      DESCRIPTION
            "The table with the host information."
      ::={ coHost 1 }

   cohlEntry OBJECT-TYPE
      SYNTAX     CohlEntry
      MAX-ACCESS not-accessible
      STATUS     current
      DESCRIPTION
```

```
            "One entry for each host."
    INDEX { cohlIndex }
    ::={ coHostlist 1 }

  CohlEntry ::= SEQUENCE {
        cohlIndex         IpAddress,
        cohlAction        INTEGER,
        cohlType          INTEGER,
        cohlStatus        DisplayString,
        cohlOwner         DisplayString,
        cohlHostname      DisplayString,
        cohlArch          DisplayString,
        cohlOS            DisplayString,
        cohlLoad          INTEGER,
        cohlMem           KBytes,
        cohlFreeMem       KBytes
  }

 cohlIndex OBJECT-TYPE
    SYNTAX     IpAddress
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The IP Address of the host which serves as well as index."
    ::= { cohlEntry 1 }

 cohlAction OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "This is the Action field. Writing a number in this field
             performs an action to the corresponding object. Possible
             values are 1 = add, 2 = delete the host to the Codine
             system and 3 = start, 4 = stop the daemon which is
             running on the host."
    ::= { cohlEntry 2 }

 cohlType OBJECT-TYPE
    SYNTAX     INTEGER {
                 trusted(1),
                 submit(2),
                 submit_trusted(3),
                 execution(4),
```

```
                  execution_trusted(5),
                  execution_submit(6),
                  execution_submit_trusted(7),
                  shadow(8),
                  shadow_trusted(9),
                  shadow_submit(10),
                  shadow_submit_trusted(11),
                  shadow_execution(12),
                  shadow_execution_trusted(13),
                  shadow_execution_submit(14),
                  shadow_execution_submit_trusted(15),
                  master(16),
                  master_trusted(17),
                  master_submit(18),
                  master_submit_trusted(19),
                  master_execution(20),
                  master_execution_trusted(21),
                  master_execution_submit(22),
                  master_execution_submit_trusted(23),
                  none(32)
                }
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "The type of the host in Codine. There are the five
            following possibilities: Masterserver, Shadowserver,
            Executionhost, Submithost, Trustedhost. As almost all
            combinations exists, the information is stored in a
            5 bit field where each bit is one kind of host.
            So you get the number of the type by adding the
            following types: master(16), shadow(8),
            executionhost(4), submithost(2), trusted(1)."
   ::= { cohlEntry 3 }

cohlStatus OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The status of this host."
   ::= { cohlEntry 4 }

cohlOwner OBJECT-TYPE
   SYNTAX     DisplayString
```

```
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The owner of this host."
    ::= { cohlEntry 5 }

cohlHostname OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The fullqualified hostname."
    ::= { cohlEntry 6 }

cohlArch OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     optional
    DESCRIPTION
            "The architecture of this host."
    ::= { cohlEntry 7 }

cohlOS OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     optional
    DESCRIPTION
            "The Operation system of this host."
    ::= { cohlEntry 8 }

cohlLoad OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     optional
    DESCRIPTION
            "The load of this host."
    ::= { cohlEntry 9 }

cohlMem OBJECT-TYPE
    SYNTAX     KBytes
    MAX-ACCESS read-write
    STATUS     optional
    DESCRIPTION
            "The total amount of memory."
```

```
   ::= { cohlEntry 10 }

cohlFreeMem OBJECT-TYPE
   SYNTAX     KBytes
   MAX-ACCESS read-write
   STATUS     optional
   DESCRIPTION
           "The free memory."
   ::= { cohlEntry 11 }


-- Codine Queuelist
-- This part of the Codine MIB holds information on the queues
-- pariticipating in Codine.

coQueuelist OBJECT-TYPE
   SYNTAX SEQUENCE OF coqlEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "The table with the queue information."
   ::={ coQueue 1 }

coqlEntry OBJECT-TYPE
   SYNTAX     CoqlEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "One entry for each queue. The index is composed of the
            Internetaddress of the host, where the queue is running
            and an index for the queue on this host.
            for example: 129.187.214.32.5 is the fifth queue on the
            host 129.187.214.32."
   INDEX { coqlHostnameIP,
           coqlIndex       }
   ::={ coQueuelist 1 }

 CoqlEntry ::= SEQUENCE {
       coqlIndex         INTEGER,
       coqlAction        INTEGER,
       coqlHostnameIP    IpAddress,
       coqlHostname      DisplayString,
       coqlName          DisplayString,
       coqlLoad          INTEGER,
```

```
        coqlStatus       INTEGER,
        coqlAlive        DisplayString,
        coqlOwner        DisplayString,
        coqlPriority     INTEGER,
        coqlFamily       DisplayString,
        coqlActJob       INTEGER,
        coqlJobList      DisplayString,
        coqlSuspOnCompl  Boolean,
        coqlNrJobs       INTEGER
 }

coqlIndex OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The number of the queue of a specific host.
            This is part of the index for this SNMP table."
   ::= { coqlEntry 1 }

coqlAction OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "This is the Action field. Writing a number in this field
            performs an action to the corresponding object. Possible
            values are 1 = add, 2 = delete the queue,
            3 = start, 4 = stop the queue and 5 = enable, 6 = suspend
            the queue."
   ::= { coqlEntry 2 }

coqlHostnameIP OBJECT-TYPE
   SYNTAX     IpAddress
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The internet address of the host, the queue is running on.
            This is part of the index for this SNMP table."
   ::= { coqlEntry 3 }

coqlHostname OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-only
```

```
   STATUS     current
   DESCRIPTION
           "The name of the host, the queue is running on."
   ::= { coqlEntry 4 }

coqlName OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "The name of the queue."
   ::= { coqlEntry 5 }

coqlLoad OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The load of the queue."
   ::= { coqlEntry 6 }

coqlStatus OBJECT-TYPE
   SYNTAX     INTEGER {
                idle(1),
                running(2),
                disabled(3)
              }
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The status of the queue."
   ::= { coqlEntry 7 }

coqlAlive OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "Another status variable of the queue."
   ::= { coqlEntry 8 }

coqlOwner OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-write
```

```
    STATUS     current
    DESCRIPTION
            "The name of the owner of the queue."
    ::= { coqlEntry 9 }

coqlPriority OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The default priority for jobs running on this queue."
    ::= { coqlEntry 10 }

coqlFamily OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The name of the family, the queue belongs to."
    ::= { coqlEntry 11 }

coqlActJob OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The number of the actual job running on the queue."
    ::= { coqlEntry 12 }

coqlJobList OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "A sequence of integers with the numbers of all the
             waiting jobs for this queue written in a string, as
             it is too complicated and not urgent to implement a
             real SNMP table here."
    ::= { coqlEntry 13 }

coqlSuspOnCompl OBJECT-TYPE
    SYNTAX     Boolean
    MAX-ACCESS read-write
    STATUS     current
```

```
   DESCRIPTION
           "If this flag is set, the actually running job will be
            completed, but no other will be scheduled. So the
            queue becomes suspended after completion of this job."
   ::= { coqlEntry 14 }

coqlNrJobs OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The number of jobs waiting in this queue. This is equal
            to the length of the coqlJobList."
   ::= { coqlEntry 15 }


-- Codine Joblist
-- This part of the Codine MIB holds information on the jobs
-- in Codine. It hold information on all kind of jobs: jobs
-- which are waiting, running and which are finished.

coJoblist OBJECT-TYPE
   SYNTAX SEQUENCE OF cojlEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "The table with the job information."
   ::={ coJob 1 }

cojlEntry OBJECT-TYPE
   SYNTAX     CojlEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "One entry for each job."
   INDEX { cojlIndex }
   ::={ coJoblist 1 }

 CojlEntry ::= SEQUENCE {
        cojlIndex        INTEGER,
        cojlAction       INTEGER,
        cojlJobname      DisplayString,
        cojlExecFile     DisplayString,
        cojlStatus       INTEGER,
```

```
        cojlSubmissionTime INTEGER (0..2147483647),
        cojlStartTime     INTEGER (0..2147483647),
        cojlEndTime       INTEGER (0..2147483647),
        cojlOwner         DisplayString,
        cojlAccount       DisplayString,
        cojlUid           INTEGER,
        cojlEuid          INTEGER,
        cojlQueue         DisplayString,
        cojlHostname      DisplayString,
        cojlPriority      INTEGER,
        cojlCpuTimeReal   INTEGER,
        cojlCpuTimeUser   INTEGER,
        cojlCpuTimeSys    INTEGER,
        cojlMaxCpuTime    INTEGER
 }

cojlIndex OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The number of the job serves as Index."
   ::= { cojlEntry 1 }

cojlAction OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "This is the Action field. Writing a number in this field
            performs an action to the corresponding object. Possible
            values are 2 = delete a job from the Codine
            system and 5 = enable, 6 = suspend a job."
   ::= { cojlEntry 2 }

cojlJobname OBJECT-TYPE
   SYNTAX     DisplayString
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The name of the job."
   ::= { cojlEntry 3 }

cojlExecFile OBJECT-TYPE
```

```
   SYNTAX     DisplayString
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The name of the file which is executed, when the job is
            running."
   ::= { cojlEntry 4 }

cojlStatus OBJECT-TYPE
   SYNTAX     INTEGER {
                waiting(1),
                running(2),
                finished(3),
                disabled(4)
              }
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The status of the job."
   ::= { cojlEntry 5 }

cojlSubmissionTime OBJECT-TYPE
   SYNTAX     INTEGER (0..2147483647)
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The time, the job was submitted to the Codine system.
            The time is measured in seconds since 00:00:00 GMT,
            January 1, 1970 as usual in Unix environment. This makes
            it easy to calculate differences if submission, start and
            end time."
   ::= { cojlEntry 6 }

cojlStartTime OBJECT-TYPE
   SYNTAX     INTEGER (0..2147483647)
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The time, the job got started. As well in seconds since
            01.01.1970."
   ::= { cojlEntry 7 }

cojlEndTime OBJECT-TYPE
   SYNTAX     INTEGER (0..2147483647)
```

```
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The time, the job was finished in seconds since 01.01.1970."
    ::= { cojlEntry 8 }

cojlOwner OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The name of the owner of the job."
    ::= { cojlEntry 9 }

cojlAccount OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The name of the account, the cpu time of the job added."
    ::= { cojlEntry 10 }

cojlUid OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The user ID under which the job is running."
    ::= { cojlEntry 11 }

cojlEuid OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The effective user ID under which the job is running."
    ::= { cojlEntry 12 }

cojlQueue OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The name of the queue in which the job becomes executed
```

```
             or empty, if the job is not scheduled yet."
    ::= { cojlEntry 13 }

cojlHostname OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The name of the host, the job is running on of empty,
             if the job is not scheduled yet."
    ::= { cojlEntry 14 }

cojlPriority OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The priority of the job."
    ::= { cojlEntry 15 }

cojlCpuTimeReal OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The real cpu time in seconds elapsed since the job
             was started. If the job is finished, the time is
             stopped, so any accounting software can use this
             value."
    ::= { cojlEntry 16 }

cojlCpuTimeUser OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The user cpu time in seconds elapsed since the job
             was started."
    ::= { cojlEntry 17 }

cojlCpuTimeSys OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-only
    STATUS     current
```

```
   DESCRIPTION
           "The system cpu time in seconds elapsed since the job
            was started."
   ::= { cojlEntry 18 }

cojlMaxCpuTime OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "The maximum of cpu time, a job is allowed to run."
   ::= { cojlEntry 19 }


-- Codine Queuefamilylist
-- This part of the Codine MIB holds information on the queuefamilies.

coQFamlist OBJECT-TYPE
   SYNTAX SEQUENCE OF coflEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "The table with the queue family information."
   ::={ coQFam 1 }

coflEntry OBJECT-TYPE
   SYNTAX     CoflEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "One entry for each queue family."
   INDEX { coflIndex }
   ::={ coQFamlist 1 }

 CoflEntry ::= SEQUENCE {
        coflIndex         INTEGER,
        coflAction        INTEGER,
        coflName          DisplayString,
        coflQueueList     DisplayString
 }

coflIndex OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-only
```

```
    STATUS     current
    DESCRIPTION
            "The index for the queue families."
    ::= { coflEntry 1 }

coflAction OBJECT-TYPE
    SYNTAX     INTEGER
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "This is the Action field. Writing a number in this field
             performs an action to the corresponding object. Possible
             values are 1 = add, 2 = delete a queue family to or
             from the Codine system."
    ::= { coflEntry 2 }

coflName OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
            "The name of the queue family."
    ::= { coflEntry 3 }

coflQueueList OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
            "The list of all queues which belong to this family
             written in a string, because it is too complicated and
             not usefull to implement a real SNMP table. To modify
             you have to modify the queue object and the results will
             show up here."
    ::= { coflEntry 4 }


-- Codine Userlist
-- This part of the Codine MIB holds information on the users
-- of the Codine system, which are not available with other
-- user management tools. If these attributes will be integrated
-- in another user management, this part of the MIB becomes obsolete.

coUserlist OBJECT-TYPE
```

```
   SYNTAX SEQUENCE OF coulEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "The table with the user information."
   ::={ coUser 1 }

coulEntry OBJECT-TYPE
   SYNTAX     CoulEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
           "One entry for each user."
   INDEX { coulIndex }
   ::={ coUserlist 1 }

 CoulEntry ::= SEQUENCE {
        coulIndex         INTEGER,
        coulAction        INTEGER,
        coulName          DisplayString,
        coulType          INTEGER,
        coulMaxUJobs      INTEGER
 }

coulIndex OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
           "The index for the user which is equal to the userID."
   ::= { coulEntry 1 }

coulAction OBJECT-TYPE
   SYNTAX     INTEGER
   MAX-ACCESS read-write
   STATUS     current
   DESCRIPTION
           "This is the Action field. Writing a number in this field
            performs an action to the corresponding object. Possible
            values are 1 = add, 2 = delete a user to or from the Codine
            system."
   ::= { coulEntry 2 }

coulName OBJECT-TYPE
```

```
      SYNTAX     DisplayString
      MAX-ACCESS read-write
      STATUS     optional
      DESCRIPTION
              "The name of the user. This is not already part of an
               other MIB, but the same information is written in the
               passwd file. If it is easier to get the information there
               this variable can be dismissed."
      ::= { coulEntry 3 }

   coulType OBJECT-TYPE
      SYNTAX     INTEGER {
                   none(1),
                   manager(2),
                   operator(3),
                   user(4)
                 }
      MAX-ACCESS read-write
      STATUS     current
      DESCRIPTION
              "The type of the user in the Codine system. None is
               included for users which have no access to the Codine
               system, so that this could be integrated in another
               user management."
      ::= { coulEntry 4 }

   coulMaxUJobs OBJECT-TYPE
      SYNTAX     INTEGER
      MAX-ACCESS read-write
      STATUS     current
      DESCRIPTION
              "The maximal number of jobs, a user can have waiting
               and running in the Codine system."
      ::= { coulEntry 5 }
END
```

Anhang C

Realisierung

In diesem Anhang wird anhand einiger Beispiele kurz beschrieben, wie einzelne Aspekte der Realisierung durchgeführt werden können.

C.1 Einfügen eines neuen Menüpunktes in HP OpenView

Zur losen Oberflächenintegration wird wie in Abschnitt 8.1.1 vorgestellt, ein neuer Menüpunkt eingefügt, der den Aufruf der verschiedenen Programme von Codine mit der aktuellen Selektion als Argument ermöglicht.

Beim Starten von `ovw` werden die Definitionen für die Menüs, die unter der Verzeichnisstruktur `$OV/registration` als Dateien abgelegt sind, eingelesen und nach diesen Vorgaben die Menüs erstellt. In einer solchen Datei wird spezifiziert, an welches bestehende Menü ein neuer Punkt für Codine angehängt werden soll. Wie eine solche Einfügungen beschrieben wird ist in der folgenden Beschreibung in Tabelle C.1 auf der nächsten Seite ersichtlich.

Mit `MenuBar` und `Menu` wird der Eintrag in die Menustruktur festgelegt, `Action` beschreibt die auszuführende Handlung. Hierbei kann die Ausführung noch von einigen Parametern abhängig gemacht werden, z.B. von der minimalen und maximalen Anzahl der selektierten Objekte (`MinSelected`, `MaxSelected`) oder von den Eigenschaften, die die selektierten Objekte haben müssen (`SelectionRule`), um beispielsweise nur Hosts einer bestimmten Firma zu erlauben. Nach Command folgt dann der Name des aufzurufenden Programms einschließlich der Parameter, wobei mit `$OVwSelections` die Namen der selektierten Objekte übergeben werden können.

Für die vollständige Oberflächenintegration muß die Managementanwendung unter HP OpenView geschrieben werden, die die API zum Anzeigen von Oberflächenelementen in OpenView nutzt.

```
Application "Codinemanagement"
{
    Description { "Menuentries for Calling Codine Management Programms", }
    MenuBar "Administer"
    {
              "Codine Management"       _C  f.menu "CodMgmt";
    }

    Menu "CodMgmt"
    {
              "qmon..."                 _q  f.action "qmon";
              "qusage..."               _q  f.action "qusage";
              "qstat..."                _q  f.action "qstat";
    }

    Action "qmon"
    {
        SelectionRule (isNode && isCodine);
        MinSelected 1;
          Command "/proj/batchsys/CODINE/bin/hp/qmon \\
    -hosts $OVwSelections ";
        NameField "IP Hostname", "Selection Name";
    }

    Action "qusage"
    {
        SelectionRule (isNode && isCodine);
        MinSelected 1;
          Command "/proj/batchsys/CODINE/bin/hp/qusage \\
    -hosts $OVwSelections ";
        NameField "IP Hostname", "Selection Name";
    }

    Action "qstat"
    {
        SelectionRule (isNode && isCodine);
        MinSelected 1;
          Command "/proj/batchsys/CODINE/bin/hp/qstat \\
    -hosts $OVwSelections ";
        NameField "IP Hostname", "Selection Name";
    }
}
```

Tabelle C.1: Beschreibung eines Menüeintrages für Codine

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CODINE	Computing in Distributed Networked Environments
CMIP	Common Management Information Protocol (ISO)
CMU	Carnegie Mellon University
GENIAS	Gesellschaft für numerisch intensive Anwendungen und Supercomputing, Neutraubling
GWDG	Gesellschaft für wissenschaftliche Datenverarbeitung mbH, Göttingen
HP	Hewlett Packard
HPUX	Betriebssystem von HP
IAB	Internet Activities Board
IBM	International Business Machines Corp.
ID	Identification (Number)
IP	Internet Protocol
LAN	Local Area Network
MIB	Management Information Base
MO	Managed Object
NFS	Network File System
OSI	Open Systems Interconnection (ISO)
PDU	Protocol Data Unit
RFC	Request for Comments
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SNMPv2	Simple Network Management Protocol Version 2
SNMP DPI	Simple Network Management Protocol Distributed Protocol Interface
SunOS	Betriebssystem von Sun
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VW	Volkswagen AG, Wolfsburg
X11	X Window System Version 11

Literaturverzeichnis

[ABEC 95] SEBASTIAN ABECK. *Integrationstechniken im Netzmanagement.* Informatik aktuell: Kommunikation in Verteilten Systemen. Springer, Februar 1995, pp. 133–126.

[BLAE 94] JAKOB BLÄTTE. *Konzeption des Managements von Lastbalancierung und Datenmigration in Workstation-Farms.* Diplomarbeit, TU München, Institut für Informatik, August 1994.

[EBER 92] MARTIN EBERT. *Analyse vorhandener Werkzeuge des Performance Management auf ihre Eignung für eine Einbindung in HP OpenView.* Diplomarbeit, TU München, Institut für Informatik, Juni 1992.

[GENI 94] GENIAS SOFTWARE GMBH. *Codine User's Guide.* Erzgebirgstr. 2, 93073 Neutraubling, 1994.

[GENI 95a] GENIAS SOFTWARE GMBH. *Protokoll des Codine-Anwendertreffens.* Erzgebirgstr. 2, 93073 Neutraubling, 24. April 1995.

[GENI 95b] ANDRE ALEFELD; ANDREAS HAAS; FRITZ FERSTL. *Common Usable List Library (CULL).* Genias Software GmbH, Erzgebirgstr. 2, 93073 Neutraubling, 6. Juli 1995.

[HaHa 94] NEDO HAUBELT; RAINER HAUCK. *Implementierung einer MIB für Systemmanagementaufgaben.* Fortgeschrittenenpraktikum, TU München, Institut für Informatik, November 1994.

[HAIN 94] ERWIN HAINZINGER. *Erweiterung eines SNMPv2-Agenten um eine Schnittstelle zur Interprozeßkommunikation.* Fortgeschrittenenpraktikum, TU München, Institut für Informatik, September 1994.

[HAIN 95] ERWIN HAINZINGER. *Objektorientierte Analyse und Implementierung von Managementinformation und -funktionalität ausgewählter systemnaher Basisdienste.* Diplomarbeit, TU München, Institut für Informatik, Mai 1995.

[HeAb 93] HEINZ-GERD HEGERING; SEBASTIAN ABECK. *Integriertes Netz- und Systemmanagement.* Addison-Wesley, 1993.

[KaNe 94] J. A. Kaplan; M. L. Nelson. A comparison of queueing, cluster and distributed computing systems. NASA Technical Memorandum 109025, NASA Langley Research Center, 1994.

[KRIE 94] Uwe Krieger. *Konzeption einer Managementinformationsbasis für das Management von UNIX-Endsystemen.* Diplomarbeit, TU München, Institut für Informatik, Mai 1994.

[NeWi 95] Bernhard Neumair; René Wies. Using management policies: The case of managing distributed queueing systems. Tech. rep., LMU München, Institut für Informatik, Apr. 1995.

[OVAD 92] Hewlett Packard. *HP OpenView Windows Application Design and Style Guide*, June 1992. HP Manual Part Number: J2310-90002.

[OVAR 93] Hewlett Packard. *HP OpenView SNMP Management Platform Administrator's Reference*, Dec. 1993. HP Manual Part Number: J2312-90002.

[OVPG 93] Hewlett Packard. *HP OpenView Windows Programmer's Guide*, Dec. 1993. HP Manual Part Number: J2319-90002.

[OVPR 93] Hewlett Packard. *HP OpenView Windows Programmer's Reference*, Dec. 1993. HP Manual Part Number: J2319-90009.

[OVUG 93] Hewlett Packard. *HP OpenView Windows User's Guide*, Dec. 1993. HP Manual Part Number: J2311-90002.

[RFC 1155] K. McCloghrie; M. Rose. *Structure and Identification of Management Information for TCP/IP-based Internets.* RFC 1155, IAB, Oct. 1990.

[RFC 1157] M. Schoffstall; M. Fedor; J. Davin; J. Case. *A Simple Network Management Protocol (SNMP).* RFC 1157, IAB, Oct. 1990.

[RFC 1212] M. Rose K. McCloghrie. *Concise MIB Definitions.* RFC , IAB, Mar. 1991.

[RFC 1213] K. McCloghrie; M. Rose. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II.* RFC 1213, IAB, Mar. 1991.

[RFC 1441] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Introduction to version 2 of the Internet-standard Network Management Framework.* RFC 1441, IAB, Mar. 1993.

[RFC 1442] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1442, IAB, Mar. 1993.

[RFC 1443] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1443, IAB, Mar. 1993.

[RFC 1444] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1444, IAB, Mar. 1993.

[RFC 1445] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1445, IAB, Mar. 1993.

[RFC 1446] J. Galvin; K. McCloghrie. *Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1446, IAB, Mar. 1993.

[RFC 1447] K. McCloghrie; J. Galvin. *Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1447, IAB, Mar. 1993.

[RFC 1448] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1448, IAB, Mar. 1993.

[RFC 1449] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1449, IAB, Mar. 1993.

[RFC 1450] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2).* RFC 1450, IAB, Mar. 1993.

[RFC 1451] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Manager to Manager Management Information Base.* RFC 1451, IAB, Mar. 1993.

[RFC 1452] J. Case; K. McCloghrie; M. Rose; S. Waldbusser. *Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework.* RFC 1452, IAB, Mar. 1993.

[RFC 1514] P. Grillo; S. Waldbusser. *Host Resources MIB.* RFC 1514, IAB, Sept. 1993.

[RFC 1565] N. Freed; S. Kille. *Network Services Monitoring MIB.* RFC 1565, IAB, Nov. 1994.

[RFC 1592] G. Carpenter; B. Wijnen. *SNMP-DPI - Simple Network Management Protocol Distributed Program Interface.* RFC 1592, IAB, Mar. 1994.

[RFC 1757] S. Waldbusser. *Remote Network Monitoring Management Information Base.* RFC 1757, IAB, Feb. 1995.

[ROSE 94] Marshall T. Rose. *The Simple Book – An Introduction to Internet Management*, second ed. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1994.

[SLOM 94] Morris Sloman, Ed. *Network and Distributed Systems Management.* Addison-Wesley, 1994.

[STAL 93] William Stallings. *SNMP, SNMPv2 and CMIP - The Practical Guide to Network-Management Standards.* Addison-Wesley, 1993.