INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenen-Praktikum

Erweiterung der MASA-Umgebung um Accounting-Funktionalität

Bernhard Lorenz

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Christian Ensel
Igor Radisic

Abgabetermin: 14. August 2001

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenen-Praktikum

Erweiterung der MASA-Umgebung um Accounting-Funktionalität

Bernhard Lorenz

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Christian Ensel Igor Radisic
Abgabetermin:	14. August 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Die Qualität des Netzmanagements, respektive des Managements im allgemeinen, ist in hohem Maße abhängig von den Informationen über die zu verwaltenden Einheiten, die dem Manager zur Verfügung stehen.

Um effizient Entscheidungen struktureller oder konzeptioneller Art treffen zu können, oder auf bestimmte Ereignisse adäquat reagieren zu können, bedarf es verschiedener Informationen. Deren Zugänglichkeit bzw. Verfügbarkeit - besonders im Hinblick auf die zeitliche Komponente - ist hierbei entscheidend.

Hinsichtlich der heterogenen Strukturen im Netzmanagement ist die Schaffung einer einheitlichen Schnittstelle für die unterschiedlichsten Quellen managementrelevanter Informationen wünschenswert.

1.2 Das MASA-System

MASA ist eine plattformunabhängige Laufzeit- und Kommunikationsarchitektur für mobile Agenten. Genauere Ausführungen über die im Folgenden lediglich kurz beschriebenen Komponenten sind in [GHR 99] zu finden. Weiterführend bzw. vertiefend zu diesem Thema ist [Kemp 98] anzuführen, zum Thema Java bzw. Corba [Flan 97] und [OrHa 95].

Auf einem zu managenden Endsystem (managed system) wird ein Agentensystem (agent system) ausgeführt, das die Laufzeitumgebung für alle Agenten (mobile agents) auf dem Endsystem darstellt.

Die Kommunikation z.B. der Agenten untereinander geschieht über die Common Object Request Broker Architecture (CORBA). Zudem ist ein Webserver Teil eines jeden Agentensystems welcher für die Kommunikation zur Steuerung sowohl der Agenten, als auch des Agentensystems selbst eigesetzt wird.

Als Benutzeroberfläche von Agentensystemen und Agenten dienen Applets, welche Benutzern den Zugriff über einen Browser ermöglichen.

Innerhalb dieser Umgebung sollen die notwendigen Strukturen für Accounting geschaffen werden.

1.3 Aufgabenstellung

Unter dem Stichwort „Accounting-Funktionalität“ soll für das MASA-System eine Schnittstelle geschaffen werden, welche durch ihren generischen Charakter auf ein breites Spektrum der anfallenden Informationen, d.h. Daten und Datentypen anwendbar sein soll. Diese zunächst relativ restriktiv erscheinende Forderung ermöglicht eine übersichtliche und nicht zu komplexe Struktur der Komponenten.

Die Bereitstellung dieser Schnittstelle geschieht durch einen Agenten der Klasse `DataProvider`, dessen Funktionalität durch eine Erweiterung der Klasse in einem Subagenten (z.B. `CpuDataProvider`) um die aus der spezifischen Aufgabe resultierenden speziellen Anforderungen ergänzt bzw. erweitert wird.

Anhand zweier unterschiedlicher Anwendungen soll die Funktion der Schnittstelle und deren Möglichkeiten dargestellt werden. Zum einen sollen Informationen über die CPU-Nutzung von Prozessen gesammelt werden (vgl. `CpuDataProvider`), zum anderen soll die Anbindung von Komponenten und die Datenabfrage über das SNMP-Protokoll realisiert werden (vgl. `SnmpDataProvider`).

1.4 Struktur der Ausarbeitung

In Kapitel 2 wird die grundlegende Struktur der Erfassung, Aufbereitung und Bereitstellung des Datenmaterials beschrieben, sowie auf Designentscheidungen und Anforderungskriterien eingegangen.

Die darauffolgenden Kapitel behandeln Konzeption, Methoden, GUI und sonstige Eigenschaften des `DataProvider` (Kap. 3), `CpuDataProvider` (Kap. 4) und `SnmpDataProvider` (Kap. 5).

2 Accounting-Modell

Im weiteren Sinne des Begriffs „Accounting“ sollen Informationen erfaßt, aufbereitet und ausgewertet werden. Informationen treten sowohl zu bestimmten Zeitpunkten auf, als auch über kürzere oder längere Perioden bzw. Zeiträume hinweg. Damit werden sowohl einmalige Ereignisse (Login eines Benutzers, Druck eines Dokuments), als auch Zeitreihen bzw. Wertefolgen (Auslastungsinformation über einen Server, Durchsatz einer Netzkomponente) erfaßt. Zumeist treten diese Daten in Verbindung mit anderen (ebenso als spezifisch erachteten) (Zusatz-)Informationen auf, d.h. beispielsweise ein Datum zusammen mit einer Herkunftsangabe (Quelle) und einem Bezugszeitpunkt.

Ein Anforderungskatalog für ein Accountingsystem müßte also zumindest die folgenden Kriterien beinhalten:

- Verschattung der Heterogenität der Informationen durch eine einheitliche Schnittstelle.
- Möglichkeiten zur Filterung und/oder Verfeinerung
- Flexible Handhabung eines Datums, d.h. verschiedene Datentypen zur Erfassung möglich
- Erweiterungsmöglichkeiten für besondere Einsatzgebiete, d.h. Möglichkeiten zur Erweiterung der abstrakten (generischen) Klasse um spezifische Funktionalität

An eine Erweiterung des MASA-Systems um Accounting-Funktionalität werden zusätzlich zu den eben angesprochenen Merkmalen flexibler Datenerfassung weitere Anforderungen gestellt. Das Einsatzgebiet des MASA-Systems ist beispielsweise von einem hohen Grad an Heterogenität geprägt, was im besonderen auch etwaige Accounting-Funktionen in deren Konzeption beeinflußt. Desweiteren stellen sich auch hier die für alle Bestandteile von MASA geltenden Fragen nach Sicherheit, Performance und Stabilität.

Alle diese Anforderungen zusammengenommen, wurde ein Accountingmodell entwickelt, welches im Folgenden genauer dargestellt werden soll.

2.1 Komponenten

Ausgehend von der eigentlichen Datenquelle bzw. der datenerzeugenden Komponente sind der Reihe nach folgende Einheiten[1] an der Datenerfassung beteiligt (vgl. Abbildung 1):

- **Erzeuger**: Je nach Beschaffenheit und Komplexität der gewünschten Daten kommen an dieser Stelle, angefangen von Standardprotokollen und -produkten (Bspw. SNMP) bis hin zu proprietären Programmen, verschiedene Lösungen zum Einsatz. Dabei kann die Zwischenhaltung der Daten sowohl vom Erzeuger, als auch vom Subagenten übernommen werden.

  Ein Beispiel für die erzeugerseitige Pufferung der Daten ist der `CpuDataProvider`, welcher die gesammelten Auslastungsinformationen in einer temporären csv-Datei ablegt. Diese wird dann bei späteren Anfragen entsprechend der Suchkriterien durchlaufen und liefert die Rohdaten für den/die Rückgabewert(e).

  Die Pufferung auf Seiten des Subagenten liegt beispielsweise beim `SnmpDataProvider` vor, dessen Datenquellen ausschließlich vorhandene SNMP-fähige Komponenten sind. Diese liefern auf SNMP-Requests die gewünschten Daten, welche in einer dynamischen Datenstruktur im Subagenten abgelegt werden.

- **`DataProvider`-Agent**: Eine einheitliche Schnittstelle nach außen und Schaffung von Transparenz für die anfragenden Prozesse ist die wesentliche Aufgabe des `DataProvider`-Agenten. Hierdurch erhält der Agent den vermittelnden Charakter eines Informationsbrokers oder Proxies.

[1] Gemäß der Vorgaben betreffend Flexibilität sind die angegebenen Komponenten als Rollen zu verstehen. D.h. im weiteren Sinne kann ein Abnehmer (welcher seine Daten von einem Erzeuger bezieht), wiederum in der nächsten Verarbeitungsinstanz als Erzeuger auftreten. Die Schnittstelle des `DataProvider` (also diejenige oberhalb proprietärer Standards) erlaubt das konkatenieren mehrerer Komponenten.

Auf der einen Seite werden Anfragen von außen entgegen genommen, welche dann auf die spezifischen Anforderungen des Erzeugers übersetzt und umgesetzt werden. Im Falle einer lokalen Pufferung werden die relevanten Daten aus der Struktur herausgefiltert, oder - falls die Pufferung erzeugerseitig geschieht - die Anfrage entsprechend an einen geeigneten Erzeuger weitergegeben.

Desweiteren findet sich hier eine grafische Benutzeroberfläche, welche Zugriff auf die darunterliegenden Methoden bietet und gegebenenfalls um spezifische Funktionen erweitert werden kann.

- **Abnehmer**: Die weitere Verarbeitung geschieht nach Anfrage bestimmter Daten durch einen Abnehmer. Wahlweise kann hier zu Managementzwecken über die Benutzeroberfläche kontollierend und regelnd eingegriffen werden.

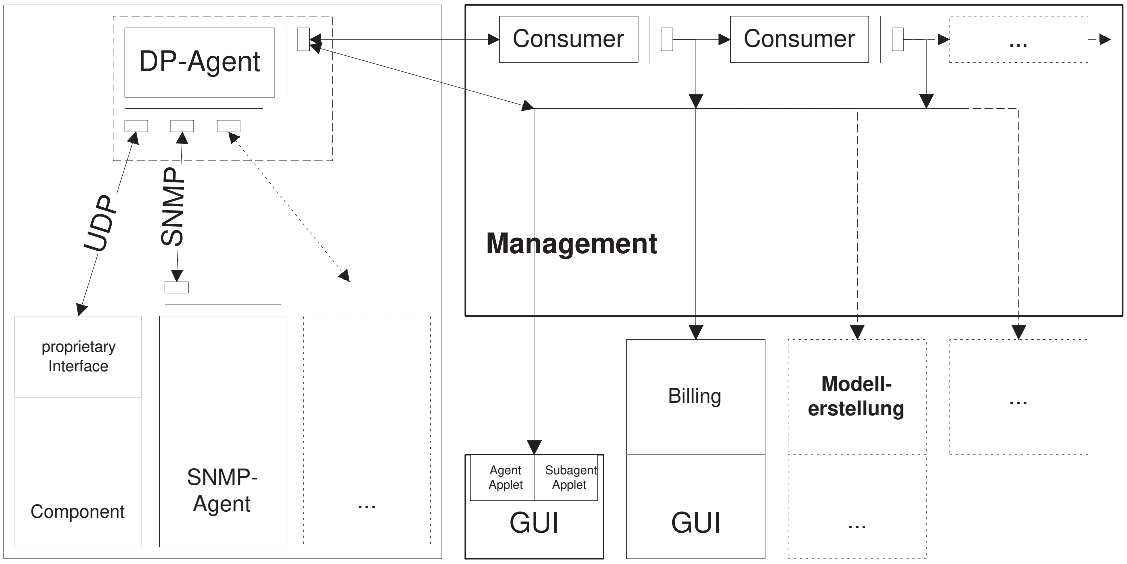

Abbildung 1: Modell der Komponenten

2.2 Strukturierung der Daten

Für die angestrebte Funktionalität wird eine Datenstruktur[2] benötigt, welche unterschiedliche Daten aufnehmen und zugleich einen gewissen generischen Charakter bewahren kann. Es müssen also zusammengesetzte Datentypen genutzt werden, um verschiedenartige Informationen aufnehmen zu können. Hinzu kommt die Notwendigkeit, zu jedem erfaßten Datum einen Zeitstempel, der den Zeitpunkt bzw. -raum der Erfassung kennzeichnet, mit aufzunehmen (vgl. Abbildung 2).

2.2.1 `Source_Id`

Zunächst muß jede Datenquelle eindeutig identifiziert werden können, was mit Hilfe einer `Source_Id` geschieht.

2.2.2 `TimeStamp`

Ein Zeitstempel dient zur Zuordnung eines Datums zu einem genauen Zeitpunkt. Im zugrundegelegten Modell wird davon ausgegangen, daß die Auswertung eines einzelnen Datums grundsätzlich

[2](vgl. hierzu auch 3.1.1)

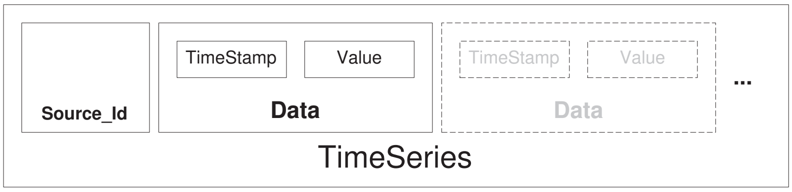

Abbildung 2: Strukturierung der Daten

einer Zeitangabe bedarf, d.h. Datum und Uhrzeit der Aufzeichnung bzw. des Auftretetens. Hierbei ist eine sekundengenaue Ermittlung des Zeitpunktes ausreichend.

2.2.3 `Data`

`Data` ist der eigentliche Träger der Information. Es setzt sich aus einem `TimeStamp` und dem eigentlichen Datum (üblicherweise ein Fließkommawert) zusammen.

2.2.4 `TimeSeries`

Eine `TimeSeries` wiederum ist eine `Source_Id`-markierte Sammlung eines oder mehrerer `Data`-Werte im Sinne einer Zeitreihe.

2.3 Eine Beispielanfrage

Geht man davon aus, daß der Manager eine zu überwachende Komponente gefunden und die notwendigen Maßnahmen getroffen hat, den Erzeuger inklusive dazu passenden Agenten zu initialisieren, so würde sich eine Anfrage wie folgt darstellen.

Der Manager identifiziert die Komponente anhand einer `Source_Id` und spezifiziert durch einen Start- sowie Endzeitpunkt den Zeitraum, innerhalb dessen die gewünschten Daten aufgetreten sind. Er setzt also eine Query mit diesen Anfrageinformationen ab.

Je nach Implementierung wertet nun der zuständige Agent diese Query entweder selbst aus, oder aber er gibt sie an eine weitere Einheit weiter, die die Auswertung übernimmt. Auswertung heißt, daß geprüft wird, ob Daten innerhalb des angefragten Zeitraumes aufgetreten sind und aufgezeichnet wurden. Falls dies zutrifft, werden diese Daten dann als `TimeSeries` zurückgegeben und können vom Manager (oder einer anderen anfragenden Einheit) weiter ausgewertet werden.

2.4 Beispiele für Informationen

Das Angebot von Beispielen für angefragte Informationen ist ebenso breit und vielfältig wie die heterogenen Strukturen moderner Netze und Systeme es erahnen lassen. Dennoch soll exemplarisch am Beispiel von CPU-Auslastungsinformation die hohe Komplexität schon eines einzelnen Teilbereichs aufgezeigt werden.

CPU-Auslastungsinformation ist zunächst die Zeit, die die CPU zur Bearbeitung eines Prozesses aufgewendet hat, im Vergleich zur insgesamt abgelaufenen Zeit. Aufgrund der Komplexität moderner Betriebssysteme, die Multitasking und Multiuser unterstützen, ist es jedoch nicht trivial, festzustellen, wieviel CPU-Zeit einem Prozess zur Verfügung stand. Genauer gesagt liegt die Schwierigkeit darin, folgende Faktoren in die Berechnung mit einzubeziehen:

- Wieviel Zeit wurde die CPU genutzt?
- Wieviel Zeit war das System im Leerlauf? Bzw.
- Wieviel Zeit war das System zu 100% ausgelastet? (Mit anderen Worten, hat der Prozess die benötigte Rechenzeit auch in der Zeit bekommen, in der sie benötigt wurde?)

- Haben andere Faktoren die Ausführung des Prozesses beeinflußt? (Beispielsweise Überlastung, Defekt oder Totalausfall des Systems, was zum Abbruch und Neustart des Prozesses geführt haben könnte?)

Es wird also klar, daß bei der Erzeugung der aufgezeichneten Daten, die relevanten Informationen speziell ausgewertet werden müssen, um in diesem Falle so etwas wie ein „Aktivitätsmaß“zu erhalten.

Abgesehen davon gibt es natürlich weit trivialere Beispiele: einfache Zähler (gelesene Bytes), einfache textuelle Informationen (die Emailadresse eines Administrators) oder diskrete Zustände (Drucker Online/Offline), um nur ein paar zu nennen.

3 Dataprovider

Die Hauptanforderung an die Schicht der Datensammlung aus der Sicht der darüberliegenden Modellerstellung ist es, die heterogene Natur der verschiedenen Datenquellen zu verschatten und somit eine einzige (homogene) Schnittstelle zur Abfrage dieser Daten zu ermöglichen.

Dies wird dadurch realisiert, daß allen Agenten, die potentiell Kontakt mit der Modellerstellungsschicht haben, die einheitliche Schnittstelle `DataProvider` implementieren. Deren wesentlicher Bestandteil ist die Festlegung der Datenabfrage und des Formates der zurückgelieferten Daten (in diesem Falle Zeitreihen) bzw. die Ausnahmeregelungen für Fehlerfälle.

Hinsichtlich der Implementierung liegt es nahe, die Bestandteile eines Agenten in solche aufzuteilen, die generischen Charakter haben und solche die speziell für ein bestimmtes Einsatzgebiet angepaßt sein müssen und deren Spezifität höher ist.

Folgende Bestandteile sind im `DataProvider` implementiert:

- GUI (zumindest der generische Teil hiervon)
- Schnittstelle zur Konfiguration
- Gerüste für Schnittstelle zur Steuerung

Folgende Bestandteile müssen in einem Subagenten des Typs `DataProvider` implementiert sein:

- Schnittstelle zur Steuerung:

  Wie eingehend in 3.1.2 beschrieben, müssen entsprechende Methoden implementiert werden, die die Steuerung von Datenquellen ermöglichen. So z.B. Methoden zur Initialisierung, Konfiguration, Abrage von Daten oder Beenden der Datensammlung, um nur einige zu nennen.

- evtl. Strukturen zur Datensammlung:[3]

  Aufgrund der Heterogenität der Datenquellen und der Beschaffenheit von MASA sind erst auf dieser Ebene solche Strukturen möglich bzw. nötig. Hierbei handelt es sich um einfache Datenhaltung, je nach zu erwartender Datenmengen in Form von Arrays/Vektoren oder beispielsweise auch sequenzieller Dateien.

Im `DataProvider` ist also der generische Teil der Funktionalität zu finden, der für alle Agenten gleichermaßen notwendig und wiederverwendbar ist, d.h. Gerüste für Schnittstellen, die Benutzeroberfläche, entsprechende Datentypen, -strukturen und Methoden. Im besonderen hervorzuheben ist die Tatsache, daß es sich hierbei um eine abstrakte Klasse handelt, da es nicht möglich oder sinnvoll wäre, einen `DataProvider` alleine zu instantiieren. Diesem würden in jedem Falle die eigentlichen Bestandteile zur Datensammlung bzw. -erfassung fehlen.

Demgegenüber muß die Schnittstelle im Subagenten entsprechend der spezifischen Anforderungen implementiert werden, d.h. das vorhandene Gerüst des `DataProvider` muß mit entsprechenden eigenen Methoden überschrieben werden.

An dieser Stelle sei auf eine Änderung im Makefile des Systems verwiesen. In Ergänzung zu `make newagent` wurde die Prozedur, das Gerüst für einen neuen Subagenten zu erstellen, in `make newsubagent` weitgehend automatisiert. Um das Gerüst für einen neuen Subagenten zu Erstellen, genügt ein Aufruf von `make newsubagent`, die Angabe des Superagenten (z.B. `DataProvider`) und des gewünschten Namens für den neuen Subagenten. Daraufhin werden die notwendigen Dateistrukturen, Links und Namenszuweisungen erstellt.

3.1 Schnittstellen des DataProvider

Die Schnittstellen des `DataProvider` umfassen ein Paket generischer Bestandteile, welche - unabhängig von der darunterliegenden Implementierung der Methoden - einheitlich für die verschiedenen Datenquellen ist, um deren Heterogenität abzuschirmen.

[3] Falls diese nicht im Subagenten implementiert sind, so müssen sie entsprechend tiefer in der Hierarchie implementiert werden.

3.1.1 Datentypen

Folgende Datentypen werden bereitgestellt (vgl. Abb. 2):

- Datentyp `Source_ID`: Hierbei handelt es sich um eine **eindeutige** ID zur Identifizierung einer Datenquelle. Prinzipiell könnte man hier beliebige eindeutige Schlüssel verwenden, also im einfachsten Fall einen `Integer`, jedoch werden hier im Sinne einer gesteigerten Lesbarkeit beispielsweise folgende IDs verwendet werden, welche intern durch einen `String` repräsentiert sind: „`de.muc.uni.nm.httpd1`" oder „`de.muc.uni.nm.webdb2`".

- Datentyp `TimeStamp`: Die Repräsentation eines Zeitpunktes erfolgt wie üblich durch einen Zahlenwert, welcher die verstrichene Zeit seit „Epoch", d.h. seit dem 01.01.1970, 00:00 Uhr, enthält. Auch hierbei wird eine „lesbare" Darstellung der Form `DD.MM.YYYY hh:mm` sowohl bei Ein- als auch Ausgaben favorisiert. Interne Darstellung erfolgt üblicherweise in Form von `integer` bzw. `long` Werten.

- Datentyp `Data`: Die Heterogenität der zu erfassenden Daten gebietet eine entsprechende Flexibilität an der Basis der Erfassung, d.h. bei der internen Darstellung und dem „Format" eines Datums.

  Verwendbar ist in einem solchen Kontext lediglich ein zusammengesetzter oder auch strukturierter Datentyp, welcher aus zwei Bestandteilen besteht: Zum einen ist ein `TimeStamp` enthalten, als eindeutiger Identifikator eines Zeitpunktes für zeitpunktbezogene Daten, zum anderen ein weiteres „Datum". Dieses „Datum" ist nicht weiter restriktiv gehandhabt oder vorgegeben, um an dieser Stelle der Heterogenität Rechnung zu tragen. Das Datum kann hierbei (fast) beliebigen Typs sein (i.w.S. also `java.lang.Object`, d.h. `Integer`, `Float`, `String` o.ä.).

- Datentyp `TimeSeries`: Im Hinblick auf die Verarbeitung nicht nur zeitpunktbezogener, sondern auch zeitreihenbezogener Daten, wird ein weiterer zusammengesetzter Datentyp verwendet, eine sogenannte `TimeSeries`.

  Diese beinhaltet eine `Source_ID` und eine aus einem oder mehreren `Data` bestehende Datenreihe.

3.1.2 Methoden

Die schon angesprochenen Methodengerüste sind die Folgenden:

- Methode `init`: Initialisierung einer Datenquelle. Je nach der zugrundeliegenden Struktur und Implementierung werden hierbei die Parameter auf Richtigkeit überprüft (indem eine für die weitere Funktionalität repräsentative Anfrage durchgeführt wird (vgl. `SnmpDataProvider`)) und/oder die für die Speicherung notwendigen Datenstrukturen werden erstellt.

  – Parameter: `s` (`Source_ID`): Eindeutige ID der Datenquelle.

  – Rückgabewert: `TimeStamp`

- Methode `query`: Diese Methode fordert ein Datum oder eine Datenreihe von der jeweiligen Datenquelle an.

  Der Benutzer spezifiziert eine `Source_ID` um den Bezug zu einer Datenquelle herzustellen. Des weiteren muß durch einen Start- und Endezeitpunkt ein Zeitfenster angegeben werden, welches die für die Abfrage relevanten Daten beinhaltet (vgl. Abb. 3).

  – Parameter:
  `s` (`Source_ID`): Eindeutige ID der Datenquelle .
  `t_start` (`TimeStamp`): Zeitstempel des frühesten gewünschten Datums.
  `t_end` (`TimeStamp`): Zeitstempel des spätesten gewünschten Datums.

Abbildung 3: Query Dialog eines (Snmp)DataProvider

  - Rückgabewert:
    `TimeSeries`

- Methode `start`: Diese dient dem Start einer Datenquelle. D.h. ein solcher Aufruf veranlasst die Datenquelle mit der Aufzeichnung von Informationen zu beginnen und im Erfolgfall einen `TimeStamp` zurückzuliefern, mit dem Zeitpunkt der Aufzeichnung des ersten Datums.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.
  - Rückgabewert: `TimeStamp`
    Zeitstempel für den Beginn der Datenaufzeichnung.

- Methode `stop`: Dient zum Stoppen einer Datenquelle.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.

- Methode `quit`: Entspricht einem umgekehrten Initialisierungsvorgang. Es werden bisher gesammelte Daten verworfen und die entsprechenden Strukturen freigegeben. Der Zustand wird wieder auf „configured“ gesetzt, was die Bearbeitung der Konfiguration ermöglicht.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.

Im Fehlerfall stehen folgende Exceptions zur Disposition:

- `AskAnotherException`: Die gefragte Datenquelle hat das gewünschte Datum entweder nicht, oder aber es ist - zum Beispiel im Sinne einer dynamischen Lastverteilung - ratsam oder notwendig, eine andere Quelle gleichen Typs zu befragen.

  Hiermit verbunden ist die Rückgabe einer alternativen ID, zur weiteren Befragung. Hierbei handelt es sich jedoch nicht um die `Source_ID` einer anderen Datenquelle, sondern diejenige eines anderen Agenten.

  Das Modell einer `Source_Id` erlaubt hierbei die Beziehungen eines Agenten und einer Datenquelle strukturiert abzulegen. Dabei wird die ID einer Datenquelle an die ID eines Agenten angehängt: Lautet beispielsweise die ID eines Agenten `de.uni.muc.mnm.cluster01`, so könnten dessen Datenquellen IDs nach folgendem Schema haben: `de.uni.muc.mnm.cluster01.websever07` oder `de.uni.muc.mnm.cluster01.dbms03`.

- `IDNotSupportedException`: Die angegebene `Source_Id` ist unkorrekt oder wird von der Datenquelle nicht unterstützt.

  Im Zuge der normalen Fehlerbehandlung gibt es hierfür keinen speziellen Rückgabewert.

- `IDConfigException`: Die angegebene Konfiguration ist unkorrekt. Dies kann auftreten beim Aufruf einer Konfiguration, welche nicht vorhanden ist, oder beim Einfügen einer neuen Konfiguration welche eine schon vorhandene ID verwendet.

  Im Zuge der normalen Fehlerbehandlung gibt es hierfür keinen speziellen Rückgabewert.

Die vom `DataProvider` implementierten Methoden für das Handling von Kofigurationen sind diese (vgl. Abb. 7):

- Methode `addConfig`: Mit dieser Methode kann eine Konfiguration hinzugefügt werden. Hierbei wird ein neues Element an den Vektor der Konfigurationen angehängt.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.
  - Parameter: `config` (`String[]`)
    String-array der Parameter.

- Methode `editConfig`: Verändern einer Konfiguration. Es wird die der angegebenen `Source_ID` entsprechende Konfiguration mit einer neuen, den Parametern entsprechenden, überschrieben.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.
  - Parameter: `config` (`String[]`)
    String-array der Parameter.

- Methode `clearConfig`: Setzt die Parameter einer Konfiguration auf die vom Subagenten spezifizierten Default-Werte zurück.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.

- Methode `deleteConfig`: Diese Methode löscht nicht mehr benötigte Konfigurationseinträge aus der Tabelle. Das entsprechende Element wird aus dem Vector der Konfigurationen gelöscht.
  - Parameter: `s` (`Source_ID`)
    Eindeutige `Source_ID` der Datenquelle.

3.2 Benutzerschnittstelle des DataProvider

Benutzer eines Agenten des Typs `DataProvider` werden zunächst auf dessen generischen, für alle Typen gleichen Teil der grafischen Oberfläche treffen. Diese ist allerdings als Bestandteil eines instanziierten Agenten zugänglich, beispielsweise in Verbindung mit dem `CpuDataProvider` (vgl. Abbildung 7)

Im Tabfolder „Generic actions“ sind die allgemeinen Bedien- und Anzeigeelemente enthalten. Es handelt sich zum einen um eine tabellenartige Darstellung verschiedener Datenquellen und deren Konfigurationen, und zum anderen, in Form von Buttons, um die entsprechenden Methoden der Schnittstelle.

3.2.1 Datenquellen und Konfigurationen

Entsprechend der Verschiedenartigkeit der Datenquellen liegt eine flexible, tabellenartige Struktur der Darstellung zugrunde. In der Liste sind grundsätzlich eine eindeutige `Source_ID` und ein Statusfeld des Typs `int` enthalten, unabhängig vom Typ der Datenquelle. Je nach repräsentiertem Typ folgen nun weitere Felder unterschiedlicher Datentypen. Ein solcher Datensatz stellt eine „Konfiguration" dar, die alle für den Betrieb einer Datenquelle relevanten Informationen enthält. Die interne Darstellung einer Konfiguration ist ein 3-tupel des Typs `DataProviderConfig`. Diese enthält neben den zwei vorgenannten ein drittes Datum, und zwar die spezifischen Parameter des Agenten in Form eines `String[]` (Stringarray).

Diese Konfigurationen können angelegt bzw. verändert oder gelöscht werden.

3.2.2 Aktionen

Folgende Aktionen stehen in Form einer Buttonleiste auf der rechten Seite zur Verfügung (vgl. hierzu Abschnitt 3.1.2):

- **init**: (Nur aktiv, wenn eine Datenquelle selektiert ist.)

  Ist die Initialisierung erfolgreich, so wird ein Dialog geöffnet, welcher den genauen Zeitpunkt der Initialisierung zurückmeldet. Anderenfalls wird eine Fehlermeldung signalisiert.

- **query**: (Nur aktiv, wenn eine Datenquelle selektiert ist und zuvor erfolgreich initialisiert wurde.)

  Ein Dialog zur Eingabe der entsprechenden Parameter wird geöffnet, anschließend kann die Abfrage abgesendet werden (vgl. Abb. 3).

- **start**: (Nur aktiv, wenn eine Datenquelle selektiert ist und zuvor erfolgreich initialisiert wurde.)

- **stop**: (Nur aktiv, wenn eine Datenquelle selektiert ist und sich im Zustand „running" befindet.)

- **quit**: (Nur aktiv, wenn eine Konfiguration selektiert ist und sich im Zustand „initialized" befindet.)

- **add config**: (Nur aktiv, wenn eine Konfiguration selektiert ist und sich im Zustand „configured" befindet.)

  Mit dieser Funktion können Konfigurationen hinzugefügt werden (vgl. Abb. 4). In Abhängigkeit der Voreinstellungen (vgl. Angang B) werden hier Default-Werte vorgegeben, welche nach Bedarf geändert werden können.

- **edit config**: (Nur aktiv, wenn eine Konfiguration selektiert ist und sich im Zustand „configured" befindet.)

  Öffnet einen Dialog zur Modifikation einer Konfiguration (vgl. Abb. 5).

- **delete config**: (Nur aktiv, wenn eine Konfiguration selektiert ist und sich im Zustand „configured" befindet.)

  Löschen der selektierten Konfiguration.

Unterhalb der Tabelle befindet sich ein weiteres Bedienelement:

- **refresh**: Aktualisieren der Konfigurationsliste. Diese Funktion ist notwendig, da über die Schnittstelle des Agenten auch von außen alle Aktionen zur Verfügung stehen. Desweiteren dient sie zur Aktualisierung des Tabelleninhaltes, für den Fall, daß von der Datenquelle regelmäßig veränderte Werte in die Datenstruktur eingebracht werden (vgl. `SnmpDataProvider`).

Abbildung 4: Add config Dialog eines (Cpu)DataProvider

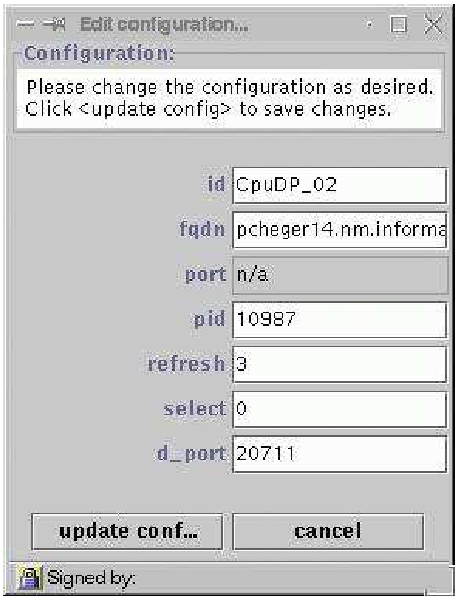

Abbildung 5: Edit config Dialog eines (Cpu)DataProvider

4 CpuDataProvider und CpuMeter

4.1 CpuDataProvider

Der `CpuDataProvider` ist ein Agent des Typs `DataProvider`, und stellt Daten über CPU-Auslastung eines Prozesses zu Verfügung. Dabei ist er auf einen auf der zu überwachenden Maschine laufenden Prozess angewiesen (vgl. Abschnitt 4.2). Dieser wiederum sammelt und puffert die eigentliche Information, wobei die Steuerung dieses Überwachungsprozesses einzig und alleine dem dazugehörenden `CpuDataProvider` obliegt. Auf dessen Anfrage werden die gewünschten Informationen per Datagrammkommunikation übermittelt (vgl. hierzu Abschnitt. 4.3).

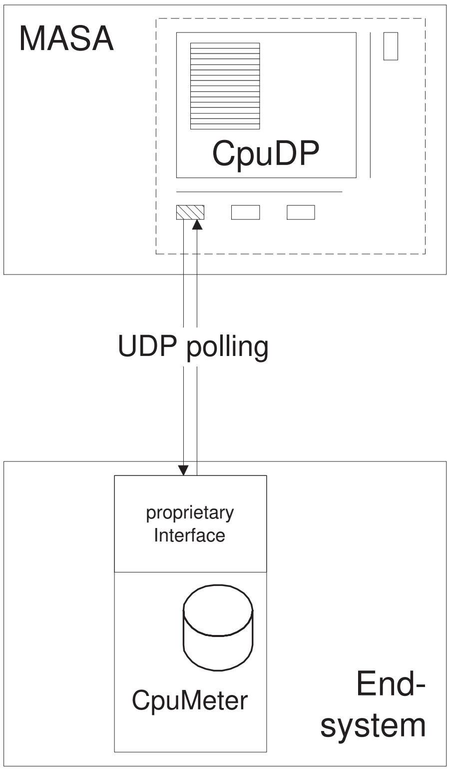

Abbildung 6: Modell des CpuDataProvider

Im Folgenden soll die Funktionsweise des `CpuDataProvider` und dessen Zusammenwirken mit dem CpuMeter beleuchtet werden.

Wie alle Agenten des Typs `DataProvider` differenziert sich dieser Subagent von anderen Subagenten dieses Typs primär durch seine proprietären Konfigurationsparameter. Die folgenden Parameter bilden zusammen mit den generischen Bestandteilen `Status` und `Source_Id` eine vollständige `CpuDataProvider`-Konfiguration:

- **fully qualified domain name** (fqdn):

  Der fqdn identifiziert ein zu überwachendes Gerät anhand des Domänennamens, der intern auf die entsprechende IP-Adresse übersetzt wird. Beides ist (bzw. sollte zumindest) weltweit eindeutig.

- **port**:

  Enthält die Portnummer für den Zugriff auf den jeweiligen `CpuMeter`. Diese ist nicht frei wählbar, sondern wird bei der Initialisierung eines neuen Meters automatisch generiert, d.h. es wird im Bereich ab 25000 ein freier Port gesucht.

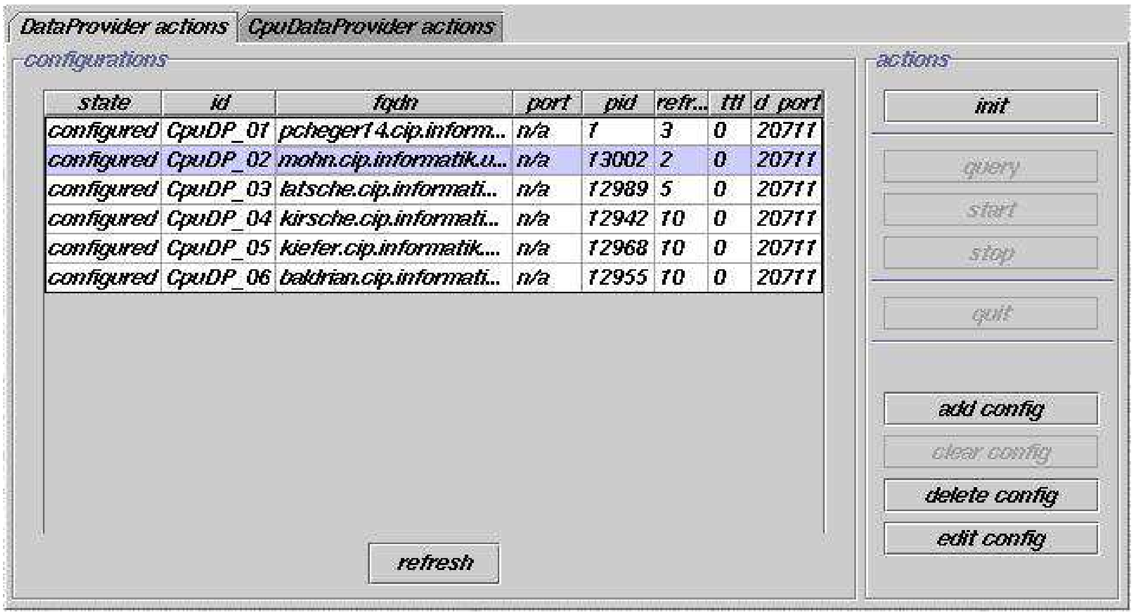

Abbildung 7: Frontpanel des CpuDataProvider

- **process id** (pid):

  In diesem Wert ist die Prozeßnummer des zu überwachenden Prozesses abgelegt.

- **refresh**:

  In diesem Intervall (Angabe in Sekunden) sammelt ein initialisierter und laufeneder `CpuMeter` Daten über den durch `pid` identifizierten Prozess.

- **select**:

  Mittels des `select`-Parameters wird das für das Aktivitätsmaß relevante Datum selektiert, wobei 0 die idle-Zeit selektiert, 1 die nice-Zeit usw. (vgl. Abschnitt 4.2.1).

- **d_port**:

  Diese Portnummer dient der Kommunikation zwischen dem `CpuMeter` (Mutter-)Prozess, d.h. zur initialisierung eines neuen `CpuMeter`. Diese ist prinzipiell systemabhängig frei wählbar, Defaulteinstellung ist 20711.

Anhand dieser Parameter können `CpuDataProvider` und `CpuMeter` eindeutig per UDP kommunizieren und Daten austauschen. Eine genauere Erklärung über die Implementation der Methoden findet sich in Abschnitt 4.2, hier soll lediglich auf die besonderen Merkmale im Vergleich zur jeweiligen generisch angedachten (`DataProvider`-)Methode eingegangen werden. Informationen über das UDP-Protokoll finden sich in [Post 80], die verwendete Methode der UDP-Kommunikation über Sockets entspricht konzeptionell der in [HaGr 98] und [HHSB 98] beschriebenen.

- `init`

  Die Initialisierung eines `CpuMeter` erwartet als Parameter `pid`,`refresh`,`select`.

  Ein zuvor auf der zu überwachenden Maschine gestarteter `CpuMeter` übernimmt die Funktion eines Spawning Process, d.h. er generiert auf einen `init` hin einen Kindprozess, welcher die eigentliche Funktion der Datensammlung erfüllt (vgl. Abschnitt 4.2).

- `query` (Keine abweichende Funktionalität)

- `start` (Keine abweichende Funktionalität)
- `stop`

  Stoppen eines bereits laufenden Meters. Dies betrifft lediglich die Datensammlung. Konfiguration und Meterprozess bleiben für eine spätere Wiederaufnahme der Aufzeichnung erhalten.
- `quit`

  Beenden eines Meterprozesses, **nachdem** die gewünschten Daten bereits abgerufen wurden. Diese Aktion beendet den Prozeß auf dem jeweiligen Gerät und löscht auch die Aufzeichnungsdatei, d.h. alle gesammelten Informationen.

Die untere Ebene der eigentlichen Datensammlung erfolgt durch den `CpuMeter` oder auch `meterd` (Kommandozeilenaufruf).

4.2 CpuMeter

Um auf unterster Ebene CPU-Auslastungsinformationen auf einem UNIX-System effizient sammeln zu können, erfolgt die Implementierung des `CpuMeter` in C (Als eines der Standardwerke zum Thema C-Programmierung ist [KeRi 88] zu empfehlen). Ohne selbst nennenswert in die Auslastungsstatistik einzugehen, überwacht der `CpuMeter` die Bearbeitungszeiten der CPU für einen bestimmten Prozeß und protokolliert dies in regelmäßigen Intervallen in einer temporär angelegten Datei.

Die Installation eines `CpuMeter` erfolgt in zwei Schritten. Zunächst muß manuell auf dem zu überwachenden Gerät ein `meterd` gestartet werden, welcher seine Kommunikation defaultmäßig auf dem Port 20711 betreibt. Dieser `meterd` ist sozusagen Mutterprozess für alle eigentlichen `CpuMeter`, die auf diesem Gerät laufen sollen. Er selbst übernimmt keine Sammelfunktion, sondern sorgt für die Generierung von Kindprozessen, die diese Aufgabe übernehmen. Mit einem entsprechenden Aufruf seitens des Agenten kann ein solcher `meterd` einen Prozess starten, welcher die Datensammlung übernimmt.

4.2.1 Befehle und Format

Der `CpuMeter` erwartet Befehle über seinen Kommunikationsport in Form eines Strings im CSV-Format. Hierbei lautet die Syntax `<befehl>, <parameter>, <parameter>, <parameter>, <parameter>, ....`

Die unterstützten Befehle decken sich mit den Methoden eines `DataProvider`, wobei die Parameter entsprechend angepasst sind:

- `init,<pid>,<refresh>,<start>,<select>`

  Wobei `start` ein flag darstellt, ob der `CpuMeter` sofort gestartet werden soll (0=nein, 1=ja), und `select` auf das gesuchte Datum zeigt.
- `query,<starttime>,<endtime>`

  Zeitangaben müssen in Form von Sekunden seit Epoch angegeben werden.
- `start` (keine weiteren Parameter)
- `stop` (keine weiteren Parameter)
- `quit` (keine weiteren Parameter)

Soll ein neuer Kindprozess gestartet werden, so wird zunächst überprüft, ob die angegebene Prozess-ID existiert. Ist dies der Fall, so wird ein `fork` ausgeführt und der Mutterprozess geht wieder in Grundstellung, d.h. wartet auf weitere Befehle.

Der Kindprozess sucht für die weitere Kommunikation einen freien Port ab der Nummer 25000 und erzeugt eine temporäre Datei für die Überwachungsdaten. Diese Datei wird im `/tmp`-Verzeichnis abgelegt und wird nach folgendem Muster benannt: `acclog.XXXXXX`. Die letzten sechs Zeichen werden durch den Systemaufruf `mkstemp` auf eine bestimmte Art und Weise festgelegt, sodaß die Eindeutigkeit des Dateinamens garantiert ist.

In dieser Datei werden nun die folgenden Daten im ASCII-Format, zeilenweise und durch Komma getrennt, abgelegt:

- `TimeStamp`
- Idle-Zeit (1/100 Sek.)
- Nice-Zeit (1/100 Sek.)
- User-Zeit (1/100 Sek.)
- System-Zeit (1/100 Sek.)
- User-Zeit der Kindprozesse (1/100 Sek.)
- System-Zeit der Kindprozesse (1/100 Sek.)

4.2.2 Solaris Version

Aufgrund von Unterschieden zwischen Linux und Solaris ergeben sich einige Einschränkungen bei der Solaris-Variante des `CpuMeter`.

Während die `/proc`-Dateistruktur unter Linux frei zugänglich ist (Lesezugriffe) und eine weitgehende Datensammlung ermöglicht, ist die Handhabung dieser Daten unter Solaris deutlich restriktiver. So ist z.B. der Zugriff auf verschiedene Prozessdaten nur mit root-Privilegien möglich, was bedeutet, daß ein uneingeschränkt funktionstüchtiger `CpuMeter` mit `setuid root` installiert und gestartet werden muss.

Eine weitere Umstrukturierung der Datensammlung betrifft die Verwaltung von `idle`- bzw. `nice`-Zeitinformation. Diese ist unter Solaris nicht trivial auszulesen. Auch Monitorprogramme, wie z.B. `top`, können hierbei lediglich verschiedene CPU-Statusinformationen verarbeiten. Diese werden dann für eine relativ ungenau hochgerechnete Angabe von bestimmten `idle`-Werten herangezogen, wobei die Genauigkeit nicht einmal ansatzweise für eine Auswertung im Rahmen der zugrundeliegenden Datensammlung ausreicht. Zeitinformation, welche den `nice` Status betrifft, ist gar nicht verfügbar und könnte ebenso nur sehr ungenau und aufwendig aus anderen Werten hochgerechnet werden.

Die Verwertung von Informationen reduziert sich unter Solaris dementsprechend auf die Prozessorzeiten des Prozesses und die seiner Kindprozesse, im Zusammenhang mit einem `TimeStamp`.

4.3 Kommunikations bzw. Ablaufmodell

Eine Datenerfassung beginnt grundsätzlich mit der Initialisierung der Beteiligten Komponenten, d.h.:

- Starten des `CpuMeter` auf der zu überwachenden Maschine, gegebenenfalls die Anpassung bezüglich des Ports für die Kommunikation.
- Erstellen einer Konfiguration im `CpuDataProvider`, d.h. Angabe von Adresse und Portnummer entsprechend des Systems, auf dem der `CpuMeter` gestartet wurde. Zudem die Vergabe von `pid`, `refresh` und `select`.
- Dann folgt das Initialisieren dieser Konfiguration. Sind alle Parameter korrekt und tritt kein weiterer Fehler auf, so befindet sich der `CpuMeter` im Status `initialized`[4].

[4] Falls vom start-Flag Gebrauch gemacht wurde, entfällt der nächste Schritt, da dieser automatisch ausgeführt wird, der Zustand ist dann `running`

- Anschließend muß der `CpuMeter` mittels `start` gestartet werden, um die Aufzeichnung von Auslastungsinformation zu beginnen. Während der Aufzeichnung befindet sich der `CpuMeter` im Zustand `running`.

Wird für eine Auswertung nun eine Reihe von CPU-Auslastungsinformationen benötigt, so wird dies mittels eines `queryData`-Aufrufs an der `CpuDataProvider`-Schnittstelle signalisiert. Der Aufruf enthält die entsprechende `Source_ID` und den gewünschten Zeitabschnitt in Form eines `TimeStamp` für Start- bzw. Endezeit.

Aus seiner internen Datenstruktur ermittelt der `CpuDataProvider` nun die zur ID passende Adresse und Portnummer des Zielrechners, auf dem der überwachte Prozeß läuft. Daraufhin wird eine Datagramm-Kommunikation initiiert und dem zuständigen `CpuMeter` ein Query-String entsprechend dem ursprünglichen Aufruf (siehe vorheriger Absatz) übermittelt. Dieser wiederum liest die gesuchten Informationen aus der temporären Datei und sendet sie zum `CpuDataProvider`.

5 SNMPDataprovider

Um die Flexibilität des Accountingmodells hervorzuheben, liegt die Betrachtung einer zum `CpuDataProvider` konzeptionell relativ unterschiedlichen Ausprägung eines `DataProvider` nahe. In diesem Abschnitt soll der `SnmpDataProvider` beschrieben werden, der die Beschaffung von Informationen über das SNMP-Protokoll realisiert.

5.1 SNMP

Der Begriff „Management“ im Zusammenhang mit Netzen bzw. verteilten Systemen wird seit geraumer Zeit meist mit dem Simple Network Management Protocol (kurz SNMP) in Verbindung gebracht. 1990 als SNMPv1 veröffentlicht, hat sich SNMP im Laufe der Zeit zum Standard für das Netzmanagement entwickelt.

SNMP bietet einerseits Funktionen zur Verwaltung und Überwachung eines Rechnernetzes, andererseits definiert es Strukturen, die eine systematische Ablage der relevanten Informationen ermöglicht.

In Anbetracht der Zielsetzung des Accountingmodells liegt es nahe, die große installierte Basis an SNMP-konformen Komponenten als Datenquellen zu nutzen, und die passende Schnittstelle zwischen SNMP und dem MASA-Accounting zu schaffen.

5.2 SnmpDataProvider

Der `SnmpDataProvider` implementiert also die Schnittstelle `DataProvider` und sorgt für eine Umsetzung der über SNMP bezogenen Daten. Es können prinzipiell alle per SNMP zugänglichen Daten bezogen werden, wobei das Hauptaugenmerk hierbei auf numerischen Daten liegt, welche isbesondere für das Accounting einen höheren Stellenwert besitzen, als beispielsweise Informationen mit einem vergleichsweise administrativen Charakter.

Als Beispiele für besonders prädestinierte Daten wären anzuführen:

- Durchsatz einer Komponente in Bytes (kumulativ, pro Zeiteinheit)
- Belegungszeiten verschiedener Komponeten
- Anzahl belegten Speichers (Diskquota)
- Anzahl gedruckter Seiten (Druckerquota)

Diese Aufzählung bietet natürlich nur einen geringen Einblick in die Vielfalt der Anwendungsmöglichkeiten.

Um an die gewünschten Informationen zu gelangen benötigt man entsprechende Angaben, die das gewünschte Datum eindeutig identifizieren (Object Identification), den rechtmäßigen Zugriff dokumentieren (Community String) und weitere Parameter. Eine eingehende Beschreibung von SNMP findet sich in [HeAb 99]. Alle für diesen Agenten relevanten Parameter werden im Folgenden genauer dargestellt.

Die Konfiguration einer Datenquelle für den `SnmpDataProvider` enthält zusätzlich zu `Status` und `Sourc_Id` die folgenden proprietären Parameter:

- **fully qualified domain name** (fqdn):

  Der fqdn der zu überwachenden Komponente.

- **community string**:

  Der passende Community String für die angegebene Komponente.

- **object id**:

  Der sogenannte Object Identifier der gewünschten Information.

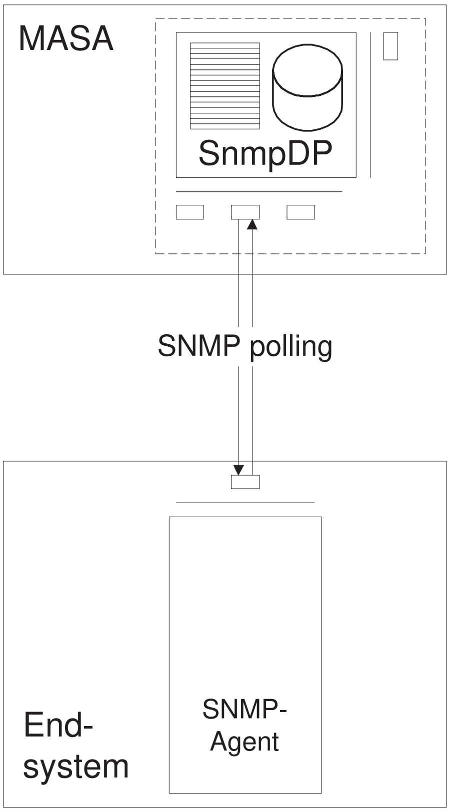

Abbildung 8: Modell des SnmpDataProvider

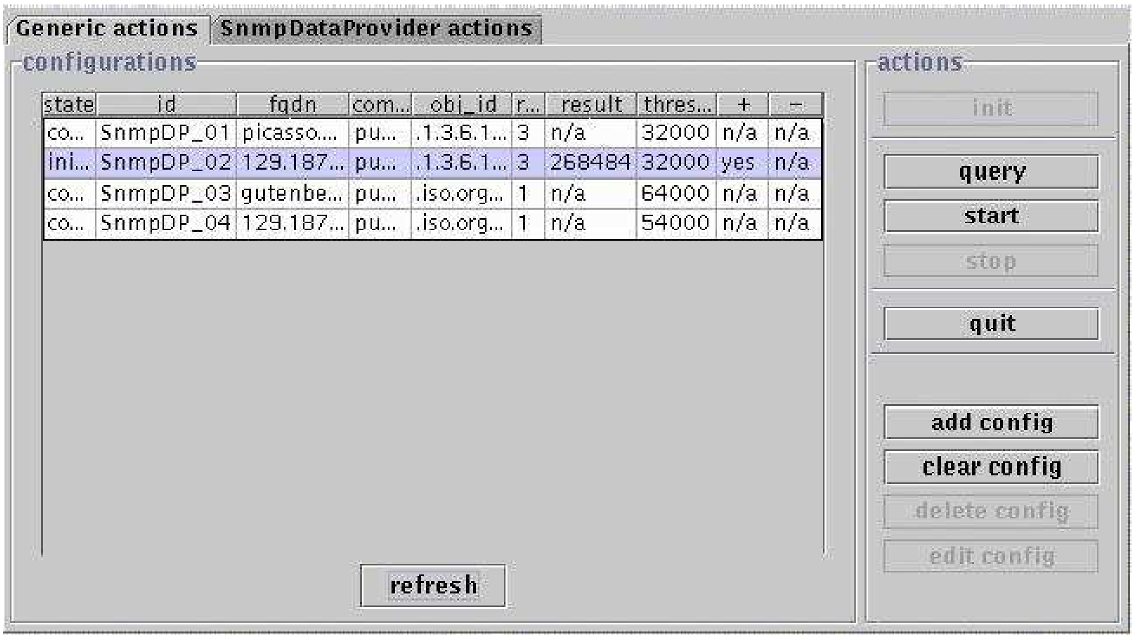

Abbildung 9: Frontpanel des SnmpDataProvider

- **refresh**:

  Zeitintervall in Sekunden für die regelmäßige Anfrage der gewünschten Information; Polling-Intervall.

- **result** (read-only):

  Dieser Wert ist nicht Bestandteil der eigentlichen Konfiguration, denn hier wird lediglich der zuletzt abgefragte Wert abgelegt.

- **threshold**:

  Insofern ein numerischer Wert durch die Object ID bestimmt wird, kann hier ein Schwellenwert angegeben werden. In Abhängigkeit des Vergleichs zwischen `threshold` und `result` werden die zwei folgenden Flags gesetzt.

- **+** (read-only):

  Schwellenwert Flag. Wird auf true/yes gesetzt, falls `result` größer als `threshold` ist bzw. war. Dieses Flag kann nur durch einen erneuten `init` wieder gelöscht werden.

- **-** (read-only):

  Schwellenwert Flag. Wird auf true/yes gesetzt, falls `result` kleiner als `threshold` ist bzw. war. Dieses Flag kann nur durch einen erneuten `init` wieder gelöscht werden.

5.3 Datenhaltung

Insofern es die verschiedenen Konfigurationen betrifft, so besteht im Hinblick auf die Datenhaltung kein Unterschied zwischen `SnmpDataProvider` und `CpuDataProvider`. Sie nützen die vom `DataProvider` bereitgestellten, generischen Strukturen dafür.

Es ergibt sich jedoch die Frage, wie die Datenhaltung der aufzuzeichnenden Informationen im `SnmpDataProvider` geschieht, welcher auf keinerlei sekundäre Strukturen (vgl. `CpuMeter` in Abschnitt 4.2) zurückgreifen kann. Ein im Vergleich zum `CpuDataProvider` grundlegend unterschiedlicher Ansatz findet sich also hier: die Datenhaltung innerhalb des Agenten selbst.

In Anbetracht der restriktiven Vorgaben von MASA bezüglich Sicherheit (bspw. Rechte für File-Access) als auch weiterer Gründe (möglichst universeller Einsatz auf verschiedenen Endsystemen) entschied man sich für eine dynamische Datenhaltung im Hauptspeicher.

Dynamische Datenhaltung heißt in diesem Falle, daß nicht im Voraus bekannt sein muß, wieviele Daten aufgezeichnet werden (sollen). Es wird also eine flexible Darstellung der Daten im Speicher gewählt (Vector), der ohne vorherige, feste Allokation von Speicher auskommt. Ein weiteres Merkmal der Datenhaltung ist die Realisierung durch Threads. Für jede initialisierte Konfiguration wird ein Thread gestartet, welcher den Vector bereithält und die Abfrage der Information in regelmäßigen Abständen ausführt. Der Lebenszyklus eines solchen Threads ist also mit der einer Konfiguration verbunden, d.h. sobald eine Konfiguration initialisiert wird, konstruiert der Agent einen neuen Thread, wird diese Konfiguration mittels `quit`-Befehls wieder in den Zustand configured gesetzt, so wird der dazugehörende Thread - und damit alle gesammelten Daten - terminiert.

5.4 Kommunikations- bzw. Ablaufmodell

Auch hier beginnt die Datenerfassung mit der Initialisierung der beteiligten Komponenten:

- Auswahl einer SNMP-fähigen Komponente und Festlegung der gewünschten Object Identification.

- Erstellung einer dazu passenden Konfiguration im `SnmpDataProvider` und Vergabe der weiteren Parameter (vgl. Abschnitt 5.2).

- Die Initialisierung der Konfiguration sorgt einerseits für eine Überprüfung der Erreichbarkeit und SNMP-Fähigkeit der bezeichneten Komponente (Abfrage von .iso.org.dod.internet.mgmt.mib-2.snmp.snmpInGetRequests) . Andererseits werden, wie zuvor schon erläutert, die notwendigen Strukturen geschaffen.
- Die Aufzeichnung (Regelmäßiges Polling von Informationen und gegebenenfalls Setzen der Threshold-Flags) wird durch Aufruf von `start` begonnen.

Eine `query` auf die gesammelten Informationen erfolgt analog der generischen Vorgaben, d.h. von Außen transparent und auf dieselbe Art und Weise wie beispielsweise beim `CpuDataProvider`. D.h. der Aufruf enthält wiederum eine `Source_Id` und einen durch Start- und Endzeitpunkt definierten Zeitraum. Der Agent sucht daraufhin die passende Threadstruktur, welche wiederum die eigene Datenstruktur auf das Vorhandensein passender Daten durchsucht. Im Erfolgfall wird eine entsprechende `Timeseries` aufgebaut und zurückgeliefert.

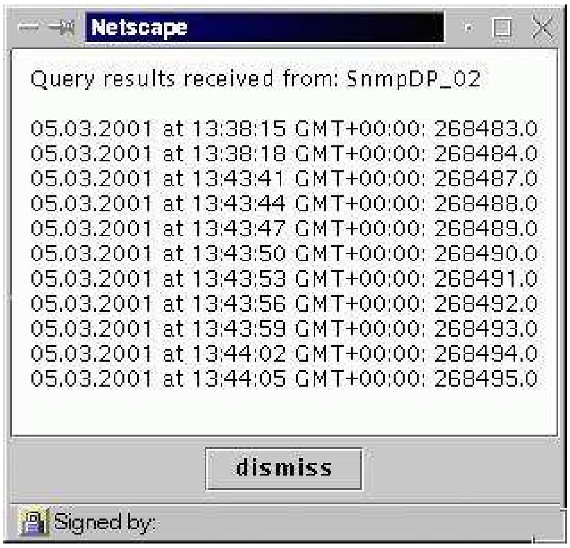

Abbildung 10: Ergebins einer Anfrage

6 Zusammenfassung

Im Rahmen der vorliegenden Ausarbeitung wurde ein Abrechnungsmodell anhand zweier exemplarischer Beispiele dargestellt. Der `CpuDataProvider` und der `SnmpDataProvider` zeigen auf, welche Möglichkeiten zur flexiblen Datenerfassung und -aufzeichnung das Modell bereitstellt. Die unterschiedlichen Ansätze zur Datensammlung bzw. -speicherung lassen sich somit auf vielerlei Art und Weise an gegebene Problemstellungen und Anforderungen anpassen, und sie ermöglichen, zusammen mit dem vom `DataProvider` zur Verfügung gestellten Gerüst an Methoden, eine rasche Implementierung weiterer Agenten.

A Erster Anhang: Properties und PolicyWishlist CpuDataProvider

```
##############################################################################
#
# DO NOT EDIT. Generated by ${0}.
#
##############################################################################

##############################################################################
#
#  Properties needed by the CpuDataProvider agent
#
#  dpType         This is the DP's identifier string
#  columnCount    Number of DP's parameters including State and ID
#
# (The following are comma separated strings with the same number of values
# as indicated by columnCount) (Yeah, well, its _that_ simple...)
#
#  columnNames    The column's names (State, ID, etc... )
#  defaultValues  (achtung einer weniger) If applicable, provide defaults
#   here
#  valueEditable  (achtung einer weniger) set to 'false' if field is not to
#   be changed in EditConfig dialog
#  toolTipText    Some useful tips for the plagued user go here (No tip for
#   the first column, because it is never user-set)
#
${MASA_PACKAGE_thisagent}.dpType=CpuDataProvider
${MASA_PACKAGE_thisagent}.columnCount=8
${MASA_PACKAGE_thisagent}.columnNames=state,id,fqdn,port,pid,refresh,select,/
d_port
${MASA_PACKAGE_thisagent}.defaultValues=none,n/a,n/a,n/a,10,0,20711
${MASA_PACKAGE_thisagent}.valueEditable=true,true,false,true,true,true,true
${MASA_PACKAGE_thisagent}.toolTipText=The DPs ID. Has to be set to a        /
distinct value. (Obligatory),The fully qualified domain name of the /
host to be queried. (Obligatory),The port number of the      /
corresponding meter. (set automatically), The numerical process_id  /
of the process to be monitored. (Obligatory),Logging interval time  /
in seconds. (Obligatory),Selects activity value (0) or raw data     /
(1-4),The meterd's port. (Default: 20711)

#  To start the agent with a number of default configurations, you can do so
#  by defining a single String, containing the necessary paramters, here.
#  The syntax is as follows:
#
#  defaultConfig = <configuration>[;<configuration>]
#
#  configuration = <sourceID>,<parameter>[,<parameter>]
#
${MASA_PACKAGE_thisagent}.defaultConfig=CpuDP_01,pcheger3.nm.informatik.uni-/
muenchen.de,n/a,3045,3,0,20711;CpuDP_02,pcheger14.nm.informatik.uni-/
muenchen.de,n/a,10987,3,0,20711;CpuDP_03,sunheger1.nm.informatik.uni/
-muenchen.de,n/a,10987,3,0,20711
```

```
//############################################################################
//
// DO NOT EDIT. Generated from ${0}.
//
//############################################################################

//############################################################################
//
// Special permissions needed by the CpuDataProvider agent
//

permission ${MASA_PACKAGE_agentSystem}.ClassAllowUsePermission /
\"${MASA_PACKAGE_superagent}.-\";
permission ${MASA_PACKAGE_agentSystem}.ClassAllowDefinePermission /
\"${MASA_PACKAGE_superagent}.-\";

permission java.util.PropertyPermission /
\"${MASA_PACKAGE}.namingwebserver.port\", \"read\";
permission java.util.PropertyPermission /
\"${MASA_PACKAGE}.namingwebserver.host\", \"read\";
```

B Zweiter Anhang: Properties und PolicyWishlist SnmpDataProvider

```
############################################################################
#
# DO NOT EDIT. Generated by ${0}.
#
############################################################################

############################################################################
#
#  Properties needed by the SnmpDataProvider agent
#
#  dpType         This is the DP's identifier string
#  columnCount    Number of DP's parameters _plus_2_ !!! (State and ID are  /
#   obligatory)
#
# (The following are comma separated strings with the same number of values
# as indicated by columnCount) (Yeah, well, its _that_ simple...)
#
#  columnNames    The column's names (State, ID, etc... )
#  defaultValues  If applicable, provide defaults here
#  valueEditable  set to 'false' if field is not to be changed in EditConfig/
#    dialog
#  toolTipText    Some useful tips for the plagued user go here (No tip for /
#   the first
#                 column, because it is never user-set)
#
${MASA_PACKAGE_thisagent}.dpType=SnmpDataProvider
${MASA_PACKAGE_thisagent}.columnCount=10
```

```
${MASA_PACKAGE_thisagent}.columnNames=state,id,fqdn,community,obj_id,refresh/
,result,threshold,+,-
${MASA_PACKAGE_thisagent}.defaultValues=none,n/a,public,n/a,10,n/a,32000,n/a/
,n/a
${MASA_PACKAGE_thisagent}.valueEditable=true,true,true,true,true,false,true,/
false,false
${MASA_PACKAGE_thisagent}.toolTipText=The DPs ID. Has to be set to a        /
distinct value. (Obligatory),The fully qualified domain name of the /
host to be queried. (Obligatory),The community string. (Optional),  /
The object_id of the object to be queried. (Obligatory),Logging     /
interval time in seconds. (Obligatory),The last result received     /
(only available when DP is running).,Threshold value. (Optional),   /
Threshold (at least once) exceeded since DP started logging.,       /
Queried values (were at least once) short of threshold value since  /
DP started logging.

#  To start the agent with a number of default configurations, you can do so
#  by defining a single String, containing the necessary paramters, here.
#  The syntax is as follows:
#
#  defaultConfig = <configuration>[;<configuration>]
#
#  configuration = <sourceID>,<parameter>[,<parameter>]
#
${MASA_PACKAGE_thisagent}.defaultConfig=SnmpDP_01,picasso.nm.informatik.uni-/
muenchen.de,public,.1.3.6.1.2.1.11.15.0,3,n/a,32000,n/a,n/a;SnmpDP_0/
2,129.187.214.72,public,.1.3.6.1.2.1.11.15.0,3,n/a,32000,n/a,n/a;Snm/
pDP_03,gutenberg.nm.informatik.uni-muenchen.de,public,.iso.org.dod.i/
nternet.mgmt.mib-2.snmp.snmpInGetRequests.0,1,n/a,64000,n/a,n/a;Snmp/
DP_04,129.187.214.70,public,.iso.org.dod.internet.mgmt.mib-2.snmp.sn/
mpInGetRequests.0,1,n/a,54000,n/a,n/a

//############################################################################
//
// DO NOT EDIT. Generated from ${0}.
//
//############################################################################

//############################################################################
//
// Special permissions needed by the SnmpDataProvider agent
//

permission ${MASA_PACKAGE_agentSystem}.ClassAllowUsePermission \"${MASA_PACKAG/
E_superagent}.-\";
permission ${MASA_PACKAGE_agentSystem}.ClassAllowDefinePermission \"${MASA_PAC/
KAGE_superagent}.-\";

permission java.net.SocketPermission \"*:0-\", \"accept,connect,listen,resolve/
\";

permission ${MASA_PACKAGE_agentSystem}.ClassAllowUsePermission \"com.adventnet/
.snmp.snmp2.*\";
permission ${MASA_PACKAGE_agentSystem}.ClassAllowDefinePermission \"com.advent/
```

```
net.snmp.snmp2.*\";

permission java.util.PropertyPermission \"${MASA_PACKAGE}.namingwebserver.port/
\", \"read\";
permission java.util.PropertyPermission \"${MASA_PACKAGE}.namingwebserver.host/
\", \"read\";
```

Literaturverzeichnis

[Flan 97] Flanagan, D.: *Java in a Nutshell.* O'Reilly, Zweite Auflage, Mai 1997, `http://www.oreilly.com/catalog/javanut2/`.

[GHR 99] Gruschke, B., S. Heilbronner und H. Reiser: *Mobile Agent System Architecture — Eine Plattform für flexibles IT–Management.* Technischer Bericht 9902, Ludwig-Maximilians-Universität München, Institut für Informatik, München, August 1999, `http://wwwmnmteam.informatik.uni-muenchen.de/common/Literatur/MNMPub/Publikationen/ghr99/ghr99.shtml`.

[HaGr 98] Keith Haviland, Marcus Gray und Ben Salama: *UNIX System Programming.* Addison Wesley, 1998.

[HeAb 99] Heinz-Gerd Hegering, Sebastian Abeck, Bernhard Neumair: *Integriertes Management vernetzter Systeme.* dpunkt-Verlag, Heidelberg, 1999.

[HHSB 98] Merlin Hughes, et al.: *Java Network Programming [2nd Edition].* Manning, 1998.

[Kemp 98] Kempter, B.: *Entwurf eines Java/CORBA-basierten Mobilen Agenten.* Diplomarbeit, Technische Universität München, August 1998.

[KeRi 88] Brian W. Kernighan, Dennis M. Ritchie: *The C Programming Language.* Prentice Hall, 1988.

[OrHa 95] Robert Orfali, Daniel Harkey: *The Essential Distributed Objects Survival Guide.* John Wiley and Sons, 1995.

[Post 80] Postel, J.: *RFC 768: User Datagram Protocol.* RFC, IETF, August 1980, `ftp://ftp.isi.edu/in-notes/rfc768.txt`.