



**Fachhochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Abschlussarbeit

im Bachelor Studiengang

Darstellung von Leistungsdaten in einem Browser-basierten Cluster-Monitoring-System

von
Karsten Reineck

Erstbetreuer: Prof. Dr. Rudolf Berrendorf
Zweitbetreuer: Prof. Dr. Norbert Jung

Eingereicht am: 7. August 2006

Kurzdarstellung

Die Fachhochschule Bonn-Rhein-Sieg betreibt im Rahmen einer interdisziplinären Plattform zur Förderung computerwissenschaftlicher Anwendungen und Forschung einen Parallelcluster [FAC-O], übersetzt aus dem Englischen, A. d. V.].

Für diesen Cluster soll ein Monitor-System entwickelt werden, welches zeitabhängige Leistungsdaten der einzelnen Knoten bzw. des Gesamtsystems anzeigen soll. Die Monitor-Applikation besteht dabei aus drei großen Komponenten: Die Cluster-Komponente umfasst die Erfassung von Leistungsdaten auf den Knoten inklusive ihrer Bereitstellung auf dem Cluster-Server (dem Datensammler). Die zweite Komponente besteht aus einem Browser-basierten Framework, über das die Cluster-Komponente und die Web-Applikation gesteuert werden. Die Aufgabe der Darstellungskomponente besteht in der Anzeige der gemessenen Daten innerhalb des Frameworks auf der Webseite.

Diese Bachelor-Arbeit beschäftigt sich mit der Planung, dem Entwurf und der Implementierung der Darstellungskomponente. Die Arbeit lässt sich inhaltlich in drei Teile gliedern:

Im ersten Teil werden die Anforderungen an den Cluster-Monitor und insbesondere an die Darstellungskomponente ermittelt. Dabei werden zum Einen geeignete Diagrammtypen anhand der vorliegenden Leistungsdaten ermittelt, hierzu zählen z. B. Liniendiagramme, die textuelle Darstellung oder ein Bargraph. Zum Anderen wird ein Protokoll zur Kommunikation zwischen der Darstellungs- und der Cluster-Komponente entwickelt. Dieses berücksichtigt Anforderungen wie Latenz, Bandbreite und Flexibilität.

Im zweiten Teil werden verfügbare Monitor-Produkte und Möglichkeiten der grafischen Darstellung von Animationen auf Webseiten evaluiert. Hierzu zählen z. B. Java-Applets, Flash und Scalable Vector Graphics, die unter anderem auf Browser-Kompatibilität, Flexibilität, Dynamisierbarkeit und Schnittstellen geprüft werden. Dabei etablieren sich Java-Applets, auf deren Technologie die Darstellungskomponente aufsetzen wird.

Im dritten Teil wird noch einmal die Architektur des Cluster-Monitors aufgegriffen, anhand derer die Schnittstellen zwischen den einzelnen Komponenten entwickelt werden. Auch findet hier der Entwurf der Diagrammkomponente statt, die schließlich zusammen mit den Schnittstellen implementiert wird.

Zum Schluss werden die drei einzelnen Komponenten zur Monitor-Applikation zusammengeführt.

Inhaltsverzeichnis

1	Einleitung	4
1.1	Das Cluster-System der Fachhochschule Bonn-Rhein-Sieg.....	4
1.2	Beschreibung des Projekts „Cluster-Monitor“	4
1.2.1	Zielsetzung und Motivation.....	4
1.2.2	Komponenten des Cluster-Monitors	5
1.2.3	Abgrenzung des Cluster-Monitors.....	6
1.3	Umfang, Ziele und Motivation dieser Arbeit	6
1.4	Abgrenzung dieser Arbeit.....	7
2	Anforderungsanalyse	8
2.1	Web-Tauglichkeit und volle Dynamik der Grafiken.....	8
2.2	Analyse der anzuzeigenden Daten	8
2.3	Benötigte Diagrammtypen	9
2.4	Benutzeranforderungen an die Diagrammtypen	10
2.4.1	Textuelle Darstellung.....	10
2.4.2	Liniendiagramm und Multiliniendiagramm	11
2.4.3	Bargraph	13
2.4.4	Kreisdiagramm.....	14
2.5	Anforderungen an das Protokoll	14
3	Analyse und Vergleich verfügbarer Produkte	15
3.1	Monitor-Systeme.....	15
3.2	Technologien zur Browser-basierten Anzeige dynamischer Grafiken	16
3.2.1	Flash und ActionScript.....	16
3.2.2	Scalable Vector Graphics	17
3.2.3	Java-Applets.....	18
3.2.4	Java 2D.....	19
3.2.5	Entscheidung für eine Darstellungsvariante	20
3.3	Technologien zur Übertragung von Daten.....	21
3.3.1	Type Length Value	21
3.3.2	Extensible Markup Language	21
3.3.3	Entscheidung für eine Übertragungstechnologie	22
3.4	Java Grafikbibliotheken.....	23
4	Architektur und Schnittstellen	26
4.1	Architektur	26
4.2	Schnittstellen.....	27
4.2.1	Schnittstelle: Framework – Darstellung.....	28

4.2.2	Schnittstelle: Darstellung – Cluster	29
4.3	Klassenhierarchie	29
4.4	Skalierbarkeit und Ressourcenbedarf	30
5	Implementierung	32
5.1	Schnittstelle zum Framework.....	32
5.2	Grafische Komponente	34
5.2.1	Diagrammtypen	34
5.2.2	Hilfsklassen.....	36
5.3	Kommunikations-Komponente.....	37
5.4	Testen der Darstellungskomponente	39
6	Fazit und Ausblick	41
6.1	Validierung der Anforderungen und Fazit.....	41
6.2	Mögliche Erweiterungen und Ausblick	42
7	Quellen- und Literaturverzeichnis	43
8	Anhang	45
8.1	Formale Definitionen der Schnittstellen.....	45
8.2	Vollständige Definition der Schnittstelle zum Framework	47

1 Einleitung

1.1 Das Cluster-System der Fachhochschule Bonn-Rhein-Sieg

Die Fachhochschule Bonn-Rhein-Sieg betreibt einen Rechen-Cluster zur Förderung computerwissenschaftlicher Arbeiten. Als Cluster wird ein Zusammenschluss mehrerer Recheneinheiten bezeichnet, die miteinander kommunizieren. Ein entsprechend entwickeltes Programm kann dabei auf mehreren Einheiten parallel laufen. Durch die gleichzeitige Bearbeitung einer Aufgabe kann so die Zeit bis zur Terminierung des Programms reduziert werden.

Der Cluster der Fachhochschule Bonn-Rhein-Sieg wird fachbereichsübergreifend für „Chemiesimulationen, Gitterberechnungen, Entwicklung paralleler Algorithmen, Modellierung, biomedizinische Berechnungen [FAC-O], übersetzt aus dem Englischen, A. d. V.“ genutzt. Des Weiteren wird der Cluster innerhalb des „Vertically Integrated Optical Testbed for Large Applications in DFN“ (VIOLA) [RÖS-06] genutzt. VIOLA implementiert eine optische Testumgebung unter anderem für Tests von Netzwerkgeräten und -architekturen. Mehr Informationen über VIOLA finden sich auf der Webseite des Projekts [RÖS-06].

Der Cluster besteht aus sechs Rechenknoten, die jeweils mit vier Opteron-Prozessoren als symmetrisches Multiprozessorsystem betrieben werden und einem Cluster-Server, der als Datei-Server fungiert und die Verbindung zum Hochschulnetzwerk herstellt. [FAC-O]a, übersetzt aus dem Englischen, A. d. V.] Die sechs Rechenknoten sind zusätzlich zum Gigabit-Ethernet über ein Myrinet – ein für Parallelcluster geeignetes Hochleistungsnetzwerk – verbunden. Mehr Informationen über den Cluster und dessen Architektur sind im Internet zu finden ([FAC-O]), [FAC-O]a)).

1.2 Beschreibung des Projekts „Cluster-Monitor“

Für den in Kapitel 1.1 beschriebenen Cluster soll eine Monitor-Applikation entwickelt werden. Mit dieser soll dem Benutzer die Möglichkeit gegeben werden, die Auswirkungen seines Programmes auf die verschiedenen Ressourcen des Cluster-Systems zu beobachten. Dazu zählen beispielsweise Prozessorauslastung, Netzwerk- oder Festplattendurchsätze.

1.2.1 Zielsetzung und Motivation

Der Anwender des Monitor-Systems soll im Optimalfall durch die Beobachtung der Auswirkungen seines Programms auf den Cluster Engpässe erkennen und diese gegebenenfalls eliminieren können. Weiterhin erhält der Administrator des Cluster-Systems einen Überblick über die aktuelle Auslastung der einzelnen Rechenknoten.

Besonderen Wert wurde bei der Monitor-Applikation auf benutzerseitige Plattformunabhängigkeit gelegt. Bei der großen Anzahl von Forschungsprojekten, für die der Cluster eingesetzt wird (siehe Kapitel 1.1), herrscht ein hoher Grad an Heterogenität bei den Clients. Die Benutzerkomponente läuft daher vollständig im Browser, damit sie von jedem Web-fähigen Computer aus leicht zugreifbar ist.

Ein weiterer wichtiger Punkt ist die Aktualisierungsrate. Bei der Prozessorauslastung ist es z. B. erforderlich, dass je nach Laufzeit eines Programms eine geeignete Aktualisierungsrate gewählt werden kann.

Letztes wichtiges Ziel der Monitor-Applikation ist die Erweiterbarkeit und Austauschbarkeit ihrer verschiedenen Komponenten (siehe Kapitel 1.2.2). Es muss z. B. eine einfache Möglichkeit geben, neue Messwerte zu erfassen und anzuzeigen, falls der Cluster für Anwendungen eingesetzt wird, in denen bisher nicht erfasste Leistungsdaten wichtig sind. Es wird daher darauf geachtet, dass die Applikation modular ist und alle Module bzw. Komponenten über weitestgehend standardisierte Schnittstellen kommunizieren. Dadurch lässt sich nicht nur jede Komponente leicht um weitere Funktionen ergänzen, sondern sogar vollständig durch eine neue Komponente ersetzen.

1.2.2 Komponenten des Cluster-Monitors

Das Projekt Cluster-Monitor lässt sich in drei Komponenten gliedern, die in jeweils einer Bachelor-Arbeit realisiert werden (siehe Abbildung 1):

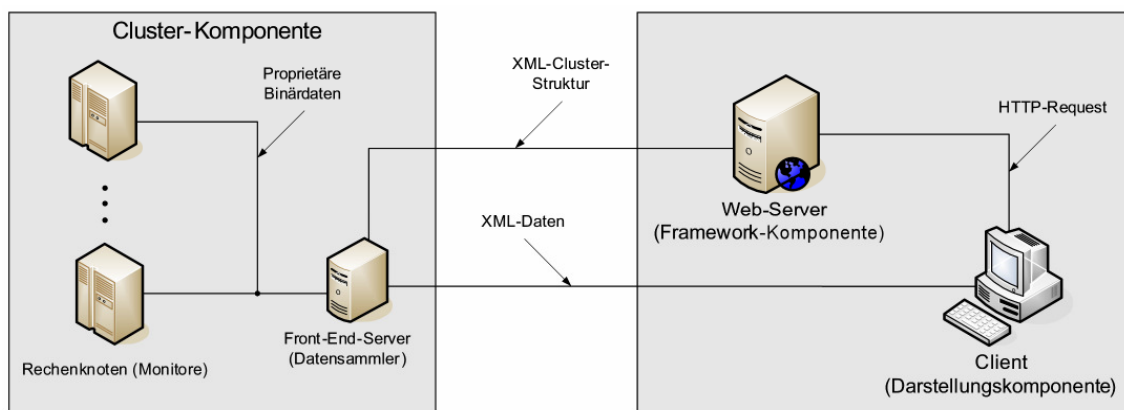


Abbildung 1: Gesamtübersicht des Cluster-Monitors [KLE-06, überarbeitet]

1. Die Cluster-Komponente beschäftigt sich mit der Sammlung, Aufbereitung und Bereitstellung der Leistungsdaten des Cluster-Systems. Das heißt zunächst werden die Daten von den einzelnen Rechenknoten gesammelt und gegebenenfalls aufbereitet (z. B. zu Durchschnittswerten zusammengefasst). Im Anschluss daran stellt der so genannte „Datensammler“ die Daten zur Weiterverarbeitung

bereit. Die Cluster-Komponente wird in einer anderen Bachelor-Arbeit realisiert [KLE-06].

2. Der Hauptbestandteil der Benutzerkomponente ist das so genannten Web-Framework. Über das Framework ist der Benutzer in der Lage, den Monitor zu bedienen, also z. B. auszuwählen, welche Komponenten er auf welchen Knoten beobachten möchte. Die Realisierung der Benutzerkomponente wird in einer anderen Bachelor-Arbeit beschrieben [JUN-06].
3. Die Darstellungskomponente beschäftigt sich mit der Frage, wie die Leistungsdaten des Clusters sinnvoll innerhalb des Frameworks darzustellen sind. Die dritte Komponente wird in dieser Arbeit realisiert, ihre Aufgaben werden in Kapitel 1.3 genauer erläutert.

1.2.3 Abgrenzung des Cluster-Monitors

Es gibt keine Funktionalität, mit der sich ältere Leistungsdaten abfragen ließen, die z. B. in einer Datenbank vorgehalten wären. Die Monitor-Applikation ist daher kein Tool zum Abrechnen von Ressourcen-Verbrauch.

Auch ist nicht vorgesehen, dass Benachrichtigungen verschickt werden, wenn z. B. ein Rechenknoten ausfällt. Die Monitor-Applikation eignet sich daher nicht als Tool zur Überwachung eines Clusters.

1.3 Umfang, Ziele und Motivation dieser Arbeit

Diese Arbeit umfasst die Realisierung der Darstellungskomponente des Cluster-Monitors. Dazu wird ein Toolkit bereitgestellt, welches es ermöglicht, zeitabhängige Messwerte geeignet (z. B. in Diagrammform) darzustellen.

Aus dem vorgesehenen Einsatz für den Cluster-Monitor (siehe Kapitel 1.2) ergibt sich insbesondere die Anforderung der Darstellung der Leistungsdaten in einem Browser. Weiterhin müssen die Leistungsdaten möglichst zeitnah angezeigt werden und die Grafiken damit voll dynamisch sein. Eine voll dynamische Grafik ist auch nach ihrer Erstellung in der Lage, weitere Daten anzunehmen und anzuzeigen. Der Hintergrund liegt darin, dass auch Messwerte angezeigt werden sollen, die nach der Initialisierung der Grafik entstehen. Das angestrebte Intervall zur Abfrage neuer Leistungsdaten beträgt 250 Millisekunden. Diese Messwerte müssen dem Benutzer zeitnah angezeigt werden. Es ist daher praktisch nicht möglich, die Grafiken dynamisch auf einem Webserver zu erstellen und diese danach an den Browser zu übertragen.

Um eine möglichst übersichtliche und verständliche Darstellung der verschiedenen Messdaten zu ermöglichen, werden verschiedene Darstellungsformen realisiert. Welche Darstellungsformen sinnvoll sind, wird in Kapitel 2.3 erläutert.

Um einen universellen Einsatz auch über das Projekt Cluster-Monitor hinaus problemlos zu ermöglichen, wird das Toolkit so aufgebaut, dass alle Schnittstellen möglichst auf standardisierten Technologien basieren. Auf der einen Seite befindet sich die Schnittstelle zum Framework des Monitors bzw. allgemein zum Browser, worüber die Diagramme aufgerufen und konfiguriert werden können. Auf der anderen Seite ist die Schnittstelle zur Abfrage neuer Messwerte vorhanden. Beide Schnittstellen werden in Kapitel 4.2 ausführlich beschrieben.

1.4 Abgrenzung dieser Arbeit

Das Toolkit stellt kein Benutzer-Interface bereit. So existieren zwar Möglichkeiten der Konfiguration, z. B. mittels Parameterübergaben oder Methodenaufrufen, allerdings werden hierfür keine Schaltflächen oder andere Elemente einer Benutzeroberfläche zur Verfügung gestellt. Auch gibt es keinen Algorithmus, der sinnvolle Darstellungsformen für jeweilige Messwerte ermittelt. Dies liegt im Ermessen des Benutzers. Gleiches gilt für die Anordnung der Diagramme auf der Webseite. Die Benutzeroberfläche, über die der Benutzer die oben genannten Funktionen auswählen und konfigurieren kann, realisiert das Web-Framework der Monitor-Applikation [JUN-06].

Das Toolkit verfügt ebenso wenig über Funktionen, Werte in irgendeiner Form zusammenzufassen, z. B. den Mittelwert oder die Summe vieler Einzelwerte zu ermitteln, um dann nur diese Ergebnisse anzuzeigen. Dies ist nicht Teil der Darstellungskomponente, sondern wird bereits auf der Server-Seite vom Datensammler der Cluster-Komponente [KLE-06] durchgeführt. So müssen nur die akkumulierten Daten übertragen werden.

2 Anforderungsanalyse

In Kapitel 1.3 wurden bereits die wesentlichen Ziele der Darstellungskomponente festgelegt. Hierzu zählten die Aktualisierungsrate, die Notwendigkeit der vollen Dynamik und die Verwendung von standardisierten Techniken. Darauf aufbauend werden im Folgenden die genauen Anforderungen an die Darstellungskomponente erarbeitet.

Zur Analyse der benötigten Diagrammtypen wird zunächst untersucht, welche Messdaten vorliegen und zu welchem Zweck diese dargestellt werden sollen. Daraus ergibt sich eine Liste von Diagrammtypen, deren Anforderungen einzeln erläutert werden.

2.1 Web-Tauglichkeit und volle Dynamik der Grafiken

Da das Monitor-System Browser-basiert ist, müssen die Grafiken Web-tauglich sein, ein Browser muss das Grafikformat also darstellen können. Die Grafiken können dabei nicht dynamisch auf dem Server erzeugt und anschließend über das Hypertext Transfer Protokoll (http) an den Client gesendet werden. Dies liegt daran, dass Messwerte erst zur Laufzeit der Applikation ermittelt werden können. Die Grafiken müssen also Client-seitig gezeichnet werden, sobald aktuelle Werte vorliegen. Die angestrebte Aktualisierungsrate beträgt dabei 250 Millisekunden.

2.2 Analyse der anzuzeigenden Daten

Folgende Leistungsdaten können von der Cluster-Komponente abgefragt werden (nicht alle wurden implementiert):

- Prozessorauslastung
- Speicherauslastung
- Netzwerkdurchsatz
- Festplattendurchsatz
- Speicherverbrauch laufender Prozesse

Für Programme, die die Rechenleistung vieler Prozessoren des Clusters benötigen, soll es die Möglichkeit geben, Daten akkumuliert anzuzeigen. Dies soll ebenso für die 24 Prozessoren des Clusters der Fachhochschule Bonn-Rhein-Sieg wie auch (über diesen Cluster hinaus gedacht) für Cluster-Systeme mit mehreren hundert Prozessoren gelten. Das heißt viele Einzeldaten werden zu einem Datenwert über eine Aggregatsfunktion zusammengefasst, z. B. zum Maximum, dem Durchschnitt oder der Summe. Die Aggregatsfunktionen werden innerhab der Cluster-Komponente analysiert und realisiert [KLE-06].

Leistungsdaten für eine Monitor-Applikation lassen sich, bezogen auf die unterschiedlichen Anforderungen an die Visualisierung, wie folgt klassifizieren:

- Daten, bei denen nur der aktuelle zeitabhängige Wert von Interesse ist
- Daten, deren zeitlicher Verlauf von Interesse ist
- Daten, bei denen die Aussagekraft in ihrer Relation zu anderen Daten liegt

Zu den ersten beiden Punkten zählen beispielsweise die Prozessorauslastung oder der Netzwerkdurchsatz während der Ausführung eines Programms. Ein Beispiel für den letzten Punkt ist eine Übersicht über alle Prozesse und deren Arbeitsspeicherverbrauch.

Welcher Diagrammtyp die Daten am anschaulichsten visualisiert, hängt also nicht nur von der zu überwachenden Komponente, sondern auch von dem Ziel der Analyse ab. Dies äußert sich darin, ob z. B. nur ein einzelner Wert genau abgelesen werden soll oder ob der zeitliche Verlauf, die Tendenz von Werten oder die Relation zu anderen Werten von Interesse ist. Daher kann der Benutzer die Darstellungsform aus dem Web-Framework heraus auswählen [JUN-06].

2.3 Benötigte Diagrammtypen

Große Mengen von Messdaten lassen sich oft am anschaulichsten in Grafiken darstellen; insbesondere wenn nicht jedes einzelne Datum von Bedeutung ist, sondern auf die übersichtliche Darstellung des zeitlichen Verlaufs von Daten oder deren Relation zu anderen Daten Wert gelegt wird. Ansonsten würde statt einer Grafik eher auf eine tabellarische Darstellungsform zurückgegriffen. [RIE-75]

Für jede Klasse von Daten ist daher in Verbindung mit dem Analyseziel ein Diagrammtyp besonders geeignet, einen bestimmten Sachverhalt schnell zu vermitteln.

Liegt statistisches Zahlenmaterial wie beispielsweise Prozessorauslastung oder Datendurchsatz vor, so weist das Liniendiagramm – auch dadurch, dass auf der Abszisse meist die Zeit aufgetragen wird – eine verständliche Semantik auf [RIE-75]. Für Daten, deren zeitlicher Verlauf dargestellt werden soll, um die Tendenz der Messdaten zu erfassen, eignet sich daher das Liniendiagramm, da hier auch eine gewisse Historie angezeigt wird.

Liegen aggregierte Daten vor, z. B. die minimale, maximale und durchschnittliche Auslastung von acht Prozessoren, ist ein Multiliniendiagramm eine angemessene Darstellungsform. Wie bei dem einfachen Liniendiagramm sieht man hier deutlich die Tendenz der einzelnen Werte. Zusätzlich lassen sich hier die angezeigten Werte leicht vergleichen, da alle Linien in ein Koordinatensystem aufgetragen werden.

Fordert das Analyseziel, nicht die Tendenz von Leistungsdaten zu erfassen, sondern dient es dem Zweck, einen einzelnen aktuellen Messwert abzulesen, so

lassen sich die Leistungsdaten in einem Balken- oder Stabdiagramm veranschaulichen. Dieser Diagrammtyp gilt als leicht einprägsam [RIE-75]. Auch für diesen Diagrammtyp wären Prozessorauslastung und Durchsatzdaten mögliche Datenquellen.

Wenn ein Messwert möglichst präzise abgelesen werden soll und kein Vergleich zu anderen Messdaten hergestellt werden muss, bietet die rein textuelle Ausgabe eine Alternative zum Bargraph. Hierbei wird bewusst auf grafische Elemente verzichtet, um nicht vom eigentlichen Messwert abzulenken.

Den letzten Typ von Leistungsdaten bilden Gliederungszahlen, die im Vergleich zueinander dargestellt werden sollen, beispielsweise der Speicherverbrauch laufender Prozesse. Im Kreissektorendiagramm (auch Kreisdiagramm oder Kuchen-diagramm) gelten Gliederungszahlen schon nach kurzer Betrachtungsdauer als gut abschätzbar [RIE-75]. Das Kreissektorendiagramm bietet sich allerdings nur an, sofern die Daten nicht schnelllebig sind, sich also nicht sekundlich so grundlegend ändern, dass das gesamte Diagramm inklusive der Legende neu gezeichnet werden müsste.

2.4 Benutzeranforderungen an die Diagrammtypen

Hauptanforderung an alle Diagramme ist, ein für den Analysezweck geeignetes Mittel darzustellen, die Informationen möglichst schnell und ohne Fehlinterpretationen zu erfassen. Daher wurde bewusst auf eine dreidimensionale Darstellung verzichtet. Beispielsweise vergleicht der Betrachter bei einem Balkendiagramm, bei dem keine zweidimensionalen Balken, sondern dreidimensionale Säulen dargestellt werden, intuitiv die Volumina und nicht mehr die effektiven Unterschiede der einzelnen Werte [RIE-75]. Bei einem Liniendiagramm würde die Projektion eines dreidimensionalen „Bandes“ auf das zugrunde liegende Koordinatensystem das genaue Ablesen von Werten erschweren. Bei einer zweidimensionalen Linie träte dieses Problem nicht auf.

Die einzelnen Anforderungen an die unterschiedlichen Diagrammtypen werden im Folgenden erläutert.

2.4.1 Textuelle Darstellung

Die textuelle Darstellung ist eine sehr einfache Form der Datenanzeige. Sie ist sinnvoll, wenn z. B. nur einzelne Werte unter Beobachtung stehender Ressourcen von Bedeutung sind, diese jedoch möglichst genau abgelesen werden müssen. Auch soll diese Darstellungsform möglichst wenig Platz beanspruchen, sodass viele Werte untereinander angezeigt werden können. Die textuelle Darstellung erfolgt hierbei zeilenweise, gegebenenfalls in tabellarischer Form. In der linken Zelle befindet sich die Bezeichnung der Ressource, in der rechten Zelle der jeweilige Wert. Eine

Tabelle entsteht, wenn aus dem Web-Framework heraus (siehe [JUN-06]) mehrere dieser textuellen Darstellungen untereinander angezeigt werden.

Für einen Beobachter sind die Daten bei dieser Darstellungsform allerdings schwer zu erfassen, vor allem beim Vergleich vieler Daten. In diesem Fall wären andere (grafische) Diagrammtypen besser geeignet.

2.4.2 Liniendiagramm und Multiliniendiagramm

Beim Liniendiagramm werden üblicherweise auf der Abszisse die Zeit und auf der Ordinate die Werte aufgetragen. Hiervon wird auch beim Cluster-Monitor nicht abgewichen, damit der Benutzer eine gewohnte Darstellung vorfindet und dadurch in der Lage ist, die dargestellte Information schnell zu erfassen.

Das Liniendiagramm zeigt für einen eingeschränkten Zeitraum die historische Entwicklung eines Wertes als Polygonzug an. Ist dieser Zeitraum überschritten, werden die ältesten Werte nicht mehr angezeigt. Es findet keine logarithmische oder andere Skalierung der zeitlichen Achse statt. „Es ist nicht statthaft, die Abszisse teilweise zu verzerren [RIE-75]“, dadurch wäre das Diagramm sehr viel schwerer ablesbar.

Abbildung 2 zeigt ein beispielhaftes Liniendiagramm, in dem die Prozessorauslastung in Prozent angezeigt wird. Die gestrichelten Linien zeigen jeweils den minimal bzw. maximal erreichten Wert an. Die hellgraue durchgezogene Linie ist eine Hilfslinie zur besseren Ablesbarkeit der Werte der Ordinate.

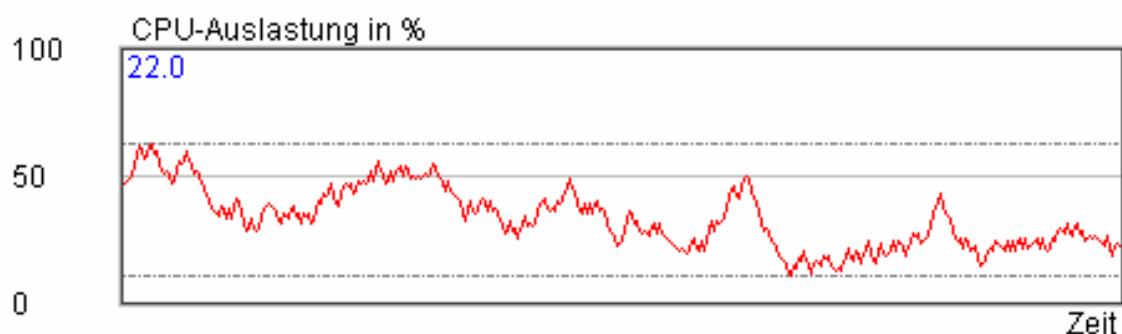


Abbildung 2: Beispiel für ein Liniendiagramm

Der maximale Wert für die Y-Koordinate (in Abbildung 2 liegt dieser bei 100) wird bei der Initialisierung des Liniendiagramms vom Browser bzw. Benutzer vorgegeben. Wird dieser Wert überschritten, soll das Koordinatensystem des Diagramms automatisch so skalieren, dass alle Werte innerhalb der Grafik sichtbar sind (Stauchung des Koordinatensystems). Befinden sich über einen längeren Zeitraum alle Werte nur noch in der unteren Hälfte des Diagramms, soll das Koordinatensystem so skalieren (hier: dehnen), dass auch die obere Hälfte der Grafik ge-

nutzt wird. Dadurch wird eine bessere Ablesbarkeit ermöglicht. Die Notwendigkeit der Skalierung liegt ursächlich darin, dass bei der Initialisierung nicht immer der maximale Messwert feststeht. Beispielsweise liegt der maximale (theoretisch erreichbare) Messwert für den Durchsatz einer Gigabit-Ethernet-Karte bei einem Gigabit pro Sekunde. Wenn die Anwendung allerdings nur einen Bruchteil des maximal möglichen Datendurchsatzes erreicht, ergibt sich eine fast waagerechte Linie, die nur im unteren Bereich des Diagramms verläuft.

Das Skalierungsverfahren soll wie folgt ablaufen: Übersteigt ein Messwert den Darstellungsbereich des Diagramms, wird die Skala des Diagramms unmittelbar verdoppelt. So wird garantiert, dass immer alle Werte visualisiert werden können. Durch die Methode der Verdopplung wird versucht, einen möglichst stabilen Zustand zu erreichen, bei dem der Wert in naher Zukunft nicht erneut den darstellbaren Bereich verlässt und dadurch eine weitere Skalierung angestoßen wird. Wenn zu oft skaliert wird, erschweren die ständigen Änderungen der Skala das intuitive Ablesen von Werten erheblich. Aus demselben Grund wird auch nur dann das Koordinatensystem gedehnt, wenn sich nach einem längeren Zeitraum (mindestens über die einfache, maximal über die doppelte Breite des Diagramms) alle Werte nur noch in der unteren Diagrammhälfte befinden. Bei einer Aktualisierung des Diagramms viermal pro Sekunde und einer Breite von 300 Pixeln würde sich das Koordinatensystem frühestens alle 75 Sekunden, spätestens alle 150 Sekunden dehnen. Die Werte entstehen dadurch, dass pro Sekunde vier Pixel für neue Werte benötigt werden: $300 / 4 = 75$ bzw. bei doppelter Breite $2 \cdot 75 = 150$.

Um den aktuellsten Wert immer sofort erfassen zu können, wird dieser zusätzlich in textueller Form direkt im Diagramm ausgegeben (blauer Wert in Abbildung 2, Seite 11). Des Weiteren soll es möglich sein, Minimal- und Maximallinien einzublenden, die jeweils den minimal bzw. maximal erreichten Wert repräsentieren, damit auch diese schnell abgelesen werden können.

Um den Vergleich verschiedener Werte – z. B. des geringsten und höchsten Durchsatzes von vier Festplatten – zu ermöglichen, wird ein weiterer Diagrammtyp, das Multiliniendiagramm, benötigt. Vom Grundsatz her entspricht dieses Multiliniendiagramm dem Liniendiagramm, allerdings soll es gleichzeitig mehrere Polygonzüge darstellen können. Um zu gewährleisten, dass auch bei vielen Polygonzügen die Übersichtlichkeit nicht verloren geht, sollen hier keine Minimal- und Maximallinien angezeigt werden können. Aus dem gleichen Grund entfällt auch innerhalb des Diagramms die Ausgabe aller aktuellen Werte in Textform. Die unterschiedlichen Polygonzüge sollen in verschiedenen Farben angezeigt, ferner soll eine entsprechende Legende erstellt werden.

2.4.3 Bargraph

Der Bargraph (Balkengraf) besteht aus einem einzelnen horizontalen Balken, der von links „gefüllt“ wird, vergleichbar mit einem Balkendiagramm, das nur einem Balken enthält. Der Vergleich verschiedener Werten wird hier möglich, indem aus dem Web-Framework heraus [JUN-06] mehrere Balkengrafen untereinander erzeugt werden. Auch bei diesem Diagrammtyp soll es die Möglichkeit geben, den minimal oder maximal erreichten Wert – hier durch eine vertikale Linie – zu kennzeichnen. Lastspitzen lassen sich hierdurch besonders gut erkennen. Der Maximalwert wird beim Aufruf der Grafik vom Browser bzw. Benutzer gesetzt. Übersteigt der anzuzeigende Wert diesen Maximalwert, soll die Skala des Balkens skaliert, in diesem Fall also gestaucht werden. Abbildung 3 zeigt einen Balkengrafen, der den Netzwerkdurchsatz in Kilobyte pro Sekunde anzeigt. Wenn der Wert von 189 KB/s auf 208 KB/s steigt (was im Bargraph aus Abbildung 3 nicht mehr anzeigbar wäre), soll die Skala des Balkengrafen automatisch auf die doppelte Breite skalieren (siehe Abbildung 4). Analog zum Liniendiagramm soll auch die entsprechende Dehnung stattfinden.

Die Skalierungsfunktion soll – wie bei jedem skalenbasierten Diagrammtyp – auch deaktivierbar sein. Dies ist vor allem dann wichtig, wenn die Werte mit jenen anderer Balkengrafen verglichen werden sollen, da sich der Vergleich von Werten verschiedener Balkengrafen mit unterschiedlichen Skalen äußerst schwierig gestaltet. Dies wird in Abbildung 3 und Abbildung 4 deutlich, in denen der untere offensichtlich kleinere Balken (208) einen größeren Wert repräsentiert als der obere größere Balken (189).

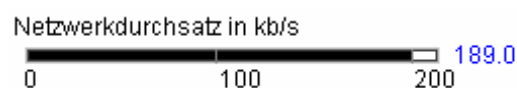


Abbildung 3: Bargraph vor der Skalierung

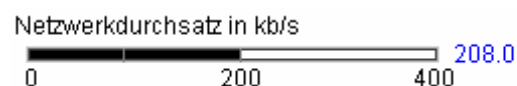


Abbildung 4: Bargraph nach der Skalierung

2.4.4 Kreissektorendiagramm

Das Kreissektorendiagramm besteht aus einem Kreis, der in verschiedenfarbige Segmente eingeteilt ist, und aus der dazugehörigen Legende. Anhand eines Maximalwertes werden die Datenwerte der einzelnen Segmente prozentual umgerechnet und einer entsprechenden Kreisfläche zugewiesen. Ein Kreissektorendiagramm gehört zu den Darstellungstypen, die über eine Legende verfügen müssen, welche die einzelnen Kreissegmente betitelt. Da sich bei jeder Änderung der Legende der Betrachter erneut im Diagramm orientieren muss, ist diese Darstellungsform nicht für schnelllebige Daten geeignet. Daher ist hier die Aktualisierungsrate zum Neuzeichnen des Diagramms höher anzusetzen als bei anderen Diagrammtypen.

Das Kreissektorendiagramm wurde im Rahmen dieser Bachelor-Arbeit nicht implementiert, Architektur und Schnittstellen (siehe Kapitel 4) sind aber dafür ausgelegt, sodass eine nachträgliche Integration in den Cluster-Monitor einfach zu realisieren ist.

2.5 Anforderungen an das Protokoll

Die wichtigsten Anforderungen an das Protokoll sind Flexibilität und Erweiterbarkeit. Das Protokoll muss so erweiterbar gehalten werden, dass es auch nachträglich möglich ist, andere Messdaten zu erfassen und über dieses Protokoll zu übertragen. Dazu muss das Protokoll so flexibel sein, dass nahezu beliebige Datentypen und Datenstrukturen übertragen werden können. Ebenso gehört zur Flexibilität, dass sowohl die Server-Seite, also der Datensammler, als auch die Client-seitige Anwendung ausgetauscht werden können.

Im Widerspruch dazu steht zunächst die Anforderung, das Protokoll möglichst leichtgewichtig in Bezug auf den Overhead auszulegen. Messwerte können bis zu viermal pro Sekunde angefordert werden, was bedeutet, dass eine effiziente Kommunikation hinsichtlich Latenz und Bandbreite gewährleistet sein muss. Dennoch liegt die Priorität bei der Flexibilität, sodass ein für dieses Ziel angemessener Overhead toleriert wird. In Kapitel 3.3 werden verschiedene Möglichkeiten der Datenübertragung aufgezeigt. In Kapitel 3.3.3 wird schließlich ein Konsens gefunden, der beide Anforderungen berücksichtigt.

3 Analyse und Vergleich verfügbarer Produkte

Dieses Kapitel beschäftigt sich mit der Analyse bereits existierender Lösungen, sowohl für den Cluster-Monitor als auch speziell für die Darstellungskomponente.

Zunächst wird der Frage nachgegangen, ob auf vorhandene Monitor-Systeme zurückgegriffen werden kann. Dabei werden diese klassifiziert und kurz die Vor- und Nachteile aufgezeigt.

3.1 Monitor-Systeme

Vorhandene Monitor-Systeme lassen sich unter dem Gesichtspunkt der plattform-unabhängigen Darstellung grob in zwei Kategorien einteilen. Auf der einen Seite stehen Browser-basierte Anwendungen. Für die Aktualisierung der Grafiken im Browser wird bei ihnen immer die gesamte Webseite, mindestens aber das Bild zur Darstellung der Werte, neu geladen. Auf der anderen Seite gibt es Desktopanwendungen, die, teilweise speziell für eine Plattform kompiliert, auch nur auf dieser laufen.

Nagios [GAL-06] und Ganglia [GAN-06], als Beispiele für Browser-basierte Anwendungen, erstellen keine voll dynamischen Grafiken. Deutlich wird dies in den Online-Demonstrationen der beiden Produkte. Beispielsweise in der Nagios-Demonstration von NETWAYS [NET-06] oder in der Ganglia-Demonstration der Berkeley University of California [BER-06]. Bei beiden muss die Aktualisierung der Webseite entweder durch den Anwender angestoßen werden oder dies geschieht automatisiert über ein Meta-Refresh – also ein HTML-Tag, der den Browser dazu veranlasst, die Seite in einem vorgegebenen Zeitabstand neu zu laden.

Als Beispiel für eine Desktopanwendung sei hier LLVIEW des Zentralinstituts für angewandte Mathematik des Forschungszentrums Jülich [FOR-05] genannt. Hierbei treten die Aktualisierungsprobleme, die z. B. bei Nagios oder Ganglia existieren, nicht auf. Die Grafiken können mehrmals pro Sekunde aktualisiert werden, da sie nicht komplett neu erstellt und von einem Webserver geladen werden müssen. Nachteilig für unsere Anforderungen an die Benutzerkomponente (siehe Kapitel 1.3) ist allerdings, dass der Benutzer die Anwendung zunächst auf jedem Client installieren muss.

Fazit ist, dass bei den vorhandenen Browser-basierten Anwendungen die Aktualisierung der Diagramme, bedingt durch das Laden der Webseite vom Webserver, viel Zeit kostet und damit beispielsweise im Sekundentakt nicht mehr praktikabel ist. Bei den Desktopanwendungen hingegen ist eine Installation auf allen Clients nötig, von denen aus die Anwendung bedient werden soll. Daher fiel die Entscheidung, eine eigene Monitor-Applikation zu erstellen.

3.2 Technologien zur Browser-basierten Anzeige dynamischer Grafiken

Da die Entscheidung getroffen wurde, eine eigene Monitor-Applikation zu entwickeln, werden in diesem Kapitel verschiedenen Möglichkeiten evaluiert, voll dynamische Grafiken auf Webseiten anzuzeigen. Dabei werden die Grundlagen der unterschiedlichen Technologien wie Flash, Scalable Vector Graphics und Java-Applets beleuchtet und verglichen, wobei besonders auf die Erfüllung der in Kapitel 1 genannten Ziele und Anforderungen an den Cluster-Monitor geachtet wird.

3.2.1 Flash und ActionScript

Eine Möglichkeit, dynamische Grafiken in Browsern anzuzeigen, ist das Produkt Flash der Firma Macromedia. Flash beruht auf vektorbasierten Grafiken und unterstützt insbesondere animierte Grafiken, so genannte Flash-Filme. Dabei kann Flash nicht nur zur Darstellung einfacher Filme wie „Intros“ auf Webseiten genutzt werden, wofür es ursprünglich gedacht war. Vielmehr lassen sich heutzutage – vor allem dank ActionScript, einer für Flash entwickelten Skriptsprache – ganze Internetauftritte und sogar Browser-basierte Anwendungen programmieren. [MÜL-03]

Ein Vorteil von Flash ist, dass die auf Client-Seite benötigte Anwendung zum Abspielen von Flash-Dateien kostenlos zu beziehen ist. Dieser Flash-Player ist auch als Browser-Plugin verfügbar, sodass Flash-Dateien direkt im Browser abgespielt werden können. Dabei gibt es Browser-Plugins für alle gängigen Betriebssysteme und Browser, so z. B. den Internet Explorer für Windows, Firefox für Linux oder Safari für MacOS. Ein weiterer Vorteil von Flash ist, dass es im Vergleich zu anderen dynamischen Grafikformaten weit verbreitet und damit auch zukunftsicher ist.

Mit der Entwicklung von ActionScript als Erweiterung für Flash sind einige neue Funktionen hinzugekommen. So lassen sich beispielsweise durch eine integrierte XML-Schnittstelle sehr bequem XML-Dateien parsen und erstellen (XML, Extensible Markup Language, siehe Kapitel 3.3.2). Übliche Bildaktualisierungsraten bei Flash-Filmen liegen je nach Anwendung bei bis zu 25 Bildern pro Sekunde, das bedeutet alle 40 Millisekunden ein neues Bild. Dies unterbietet sogar deutlich die angestrebte Bildaktualisierungsrate von 250 Millisekunden. Flash ist außerdem sehr flexibel: Es lassen sich beliebige geometrische Formen mit Flash zeichnen.

Erster großer Nachteil an Flash ist die fehlende Multithread-Fähigkeit. Wenn beispielsweise neue Daten im XML-Format vorliegen und ausgewertet werden sollen, stoppt die Flash-Animation so lange, bis der XML-Parser seinen Durchlauf beendet hat und die Daten ausgewertet wurden. Erst dann kann die Grafik wieder aktualisiert werden. Das Problem, dass nicht mehrere Animationen gleichzeitig laufen

können, lässt sich über den Umweg des Multithread-fähigen Betriebssystems umgehen. Startet man mehrere Flash-Animationen in einem Browser, so wird für jeden Flash-Film ein eigener Flash-Player gestartet. Um den Vergleich mit Java aufzunehmen: In Java sind Threads vollständig in die Sprache integriert, wodurch der Browser nur eine Java Virtual Machine starten muss. In dieser Virtual Machine können jedoch mehrere Applets (also Java-Programme) laufen, sofern diese Multithread-fähig programmiert wurden. Es muss nicht für jedes Applet eine eigene Virtual Machine gestartet werden, was ressourcenschonender ist.

Zweiter großer Nachteil von Flash ist, dass es sich nicht um eine „richtige“ Programmiersprache handelt. Die Entwicklung des Flash-Films geschieht in einer Flash-spezifischen Entwicklungsumgebung wie „Flash Professional 8“ [ADO-06] von Macromedia. Auch die unkomplizierte Flash-Datei liegt nicht in für Menschen lesbarer Form vor. Mit dem später hinzugekommenen ActionScript sind die Möglichkeiten zu programmieren erweitert worden. Jedoch handelt es sich auch bei ActionScript „nur“ um eine Skriptsprache, die nicht den Funktionsumfang einer vollwertigen Programmiersprache aufweist.

3.2.2 Scalable Vector Graphics

Scalable Vector Graphics (SVG) bietet eine weitere Möglichkeit, dynamisch Bilder auf Webseiten darzustellen. Erstmals veröffentlicht wurde SVG im Jahr 2001 auf Empfehlung des World Wide Web Consortium (W3C). Sie entstanden aus dem Bedürfnis heraus, Bilder für das Internet so zu gestalten, dass diese möglichst schnell zu laden sind, also wenig Speicherplatz benötigen. Daher kam der Gedanke, von den vergleichsweise speicherintensiven Bitmap-Grafiken weg und hin zu den Vektorgrafiken zu gehen. SVG ermöglicht diesen Schritt durch die Definition von Vektorobjekten in XML-Dateien. Der Quellcode einer Scalable Vector Graphic ist daher von Menschen les- und veränderbar. Alle Vektorobjekte in Summe ergeben schließlich das Bild. Eine der Besonderheiten an SVG ist, dass die Vektorobjekte auch nachträglich veränderbar sind. So lässt sich beispielsweise eine bereits gezeichnete Linie verschieben. Dieses Konzept nennt sich bei SVG Animation und ist definiert als Änderung eines Objektes über die Zeit. [PEA-03, übersetzt aus dem Englischen, A. d. V.]

SVG bietet einige Vorteile: Es gibt eine Vielzahl vordefinierter geometrischer Objekte und Figuren, die verwendet werden können. Diese haben die Gemeinsamkeit, automatisch geglättet zu werden, das heißt eine diagonale schwarze Linie wird beispielsweise mit grauen Pixeln weichgezeichnet. Des Weiteren lassen sich relativ einfach grafische Effekte wie Farbverläufe oder Transparenzeffekte erzeugen. Bei zweckmäßiger Anwendung sehen die Grafiken qualitativ sehr hochwertig aus. Auch vorteilhaft ist die Möglichkeit, ein virtuelles Koordinatensystem (eine so genannte viewBox) über die Grafik zu legen, was z. B. bei der Darstellung

eines Liniendiagramms hilfreich ist. Verwendet man eine solche viewBox, so ist nicht nur das Bild skalierbar, sondern getrennt davon auch das Koordinatensystem der viewBox.

SVG hat nicht nur Vorteile, sondern bringt auch einige Nachteile mit sich. Da SVG – im Vergleich zu herkömmlichen Bildformaten fürs Internet – noch neu ist, befindet diese Technologie nach Meinung des Verfassers aus nachfolgenden Gründen noch in der Erprobungsphase. So unterstützt der aktuelle Firefox 1.5 zwar grundsätzlich SVG-Dateien, allerdings noch keine Animationen. Der Internet Explorer 6.0 kann ohne Plugin keine SVG-Dateien anzeigen, es existiert jedoch ein Plugin von Adobe, das Animationen unterstützt. Dieses Plugin gibt es bisher nur für den Internet Explorer. Ein weiterer Nachteil von SVG besteht darin, dass die Animationen in SVG-Dateien zunächst nur statisch sind, das heißt in der SVG-Datei muss bereits die gesamte Animation definiert sein. Die voll dynamische Änderung von Bildern – also die Definition von Animationen zur Laufzeit – lässt sich nur mithilfe einer Skriptsprache wie JavaScript realisieren, nur mit XML ist dies nicht möglich. Auch diese Funktionalität wird derzeit nur vom Internet Explorer unterstützt.

Fazit: SVG besticht zwar durch seine ansprechende Optik und lässt sich bequem erlernen, doch die Möglichkeiten, voll dynamische Grafiken zu gestalten, sind sehr begrenzt. Dies liegt vor allem daran, dass SVG letztlich nur eine Beschreibungssprache ist. Bindet man JavaScript ein, erweitern sich die Möglichkeiten auf die Ebene der Skriptsprachen; auch hier stehen aber nicht alle Möglichkeiten zur Verfügung, die mit einer vollwertigen Programmiersprache vorlägen. Da sich die Technologie noch nicht im Internet etabliert hat, ist SVG derzeit als Exot zu betrachten. Ob SVG sich durchsetzen und auch in einigen Jahren noch von Browsern unterstützt wird, bleibt abzuwarten.

3.2.3 Java-Applets

Applets sind Programme, die in der Programmiersprache Java geschrieben werden. Sie können entweder als eigenständige Programme agieren oder in Webseiten eingebunden werden. In beiden Fällen ist Voraussetzung, dass auf dem Computer auf dem das Applet angezeigt werden soll, die Java-Laufzeitumgebung installiert ist. Auch wenn das Applet von einem Webserver geladen wurde und als Teil einer Webseite im Browser dargestellt wird, so wird es doch von der lokalen Java Virtual Machine ausgeführt. [SUN-06, übersetzt aus dem Englischen, A. d. V.]

Ein Nachteil von Applets liegt darin, dass es ohne Vorkenntnisse schwieriger zu erlernen ist als beispielsweise Macromedia Flash oder Scalable Vector Graphics. Bei den beiden Letztgenannten handelt es sich um Skript- bzw. Auszeichnungssprachen, Applets hingegen werden in einer vollwertigen Programmiersprache geschrieben.

Genau hier liegen allerdings auch die vielen Vorzüge von Applets: Bedingt durch die Tatsache, dass Applets auf der Programmiersprache Java basieren, gelten viele der Vorteile von Java auch für Applets. Nicht nur die Java-Laufzeitumgebung ist für alle gängigen Betriebssysteme kostenlos zu beziehen, auch Plugins sind für die meisten Browser vorhanden. Aufgrund der großen Beliebtheit von Java kann sogar davon ausgegangen werden, dass es auf vielen Systemen bereits installiert ist. Es hat sich auf dem Markt etabliert und ist daher zukunftssicher. Vorteilhaft ist weiterhin, dass Threads voll in die Sprache Java integriert sind. Das bedeutet, dass Java Threads kennt und in einer Virtual Machine mehrere Threads parallel ausgeführt werden können [BER-05]. Der Browser kann so Ressourcen sparen, denn er muss nur eine Virtual Machine starten, in der mehrere Applets gleichzeitig ausgeführt werden können.

Zu den weiteren Vorzügen Javas zählt, dass es viele vordefinierte Klassen und Methoden gibt, die auch speziell bei der Applikation des Clustermonitors sinnvolle Verwendung finden. Dazu zählen beispielsweise die Socket-basierte Kommunikation und die Möglichkeit, XML-Streams über vorhandene Parser zu verarbeiten.

3.2.4 Java 2D

Java 2D erweitert die Funktionalität des Abstract Windowing Toolkit (AWT) und ermöglicht dem Programmierer dadurch, komplexere geometrische Formen und Figuren zu zeichnen [SUN-01, übersetzt aus dem Englischen, A. d. V.]. Das Spektrum reicht dabei von einfachen Dingen wie einer gestrichelten Linie bis hin zu komplizierten Operationen wie affinen Transformationen. Eine affine Transformation ist die Abbildung eines Vektorraumes auf einen anderen. Das Besondere daran ist, dass parallele Linien bei dieser Art der Transformation parallel bleiben. Java kennt affine Transformationen (in der Klasse *AffineTransform*) und stellt unter anderem Funktionen zum Verschieben, Skalieren und Rotieren bereit. In einem Applet lassen sich affine Transformationen auf grafische Objekte anwenden und dadurch die Koordinatenpunkte des Grafikobjekts in „Pixelkoordinaten“ zur Anzeige im Browser umwandeln [AUD-03, übersetzt aus dem Englischen, A. d. V.]. Für ein Applet mit Java 2D-Funktionalität hat dies den Vorteil, dass man z. B. das Koordinatensystem des Applets – bei dem der Nullpunkt links oben liegt – so spiegeln und verschieben kann, bis ein in der Mathematik übliches Koordinatensystem entsteht. Punkte, Linien und andere Zeichenobjekte können dann in gewohnter Weise auf das transformierte Koordinatensystem gezeichnet werden, ohne dass für jedes einzelne Objekt umständliche Umrechnungen angestellt werden müssten. Da Transformationen sich auch zur Laufzeit anwenden lassen, können sogar bereits erstellte Zeichenobjekte neu skaliert werden. Dies ist z. B. in einem Liniendiagramm sinnvoll, wenn neu darzustellende Werte den Zeichenbereich verlassen. In diesem Fall wird das Koordinatensystem so gestaucht, dass alle Werte wieder darstellbar sind, ohne die betroffenen Linien einzeln bearbeiten zu müssen.

3.2.5 Entscheidung für eine Darstellungsvariante

In Tabelle 1 sind die verschiedenen Alternativen für die Darstellungskomponente gegenübergestellt. Die Tabelle dient nicht dem vollständigen Vergleich der Technologien, sondern ist speziell ausgerichtet auf die in Kapitel 1 genannten Anforderungen des Cluster-Monitors.

	Flash und ActionScript	Scalable Vector Graphics und JavaScript	Java-Applets
Browser	Das Flash-Plugin ist für alle gängigen Browser vorhanden	Firefox ohne Animation, Internet Explorer nur mit Plugin von Adobe	Das Java-Plugin ist für alle gängigen Browser vorhanden
Betriebssysteme	Ein Player ist für alle gängigen Betriebssysteme vorhanden	Mit Animationen nur Windows, da das Plugin nur für den Internet Explorer vorhanden ist	Die Java-Laufzeitumgebung läuft auf allen gängigen Betriebssystemen
Verbreitung	Stark verbreitet	Wenig verbreitet	Stark verbreitet
Thread-fähig	Nein	Nein	Ja
XML-Parser	Mit ActionScript	Mit JavaScript	Zum Beispiel mit Simple API for XML (SAX)
Sockets	XMLSocket mit ActionScript	Nein, aber mit JavaScript geht XML über HTTP	Ja

Tabelle 1: Gegenüberstellung analysierter Darstellungsvarianten

Zusammenfassend lässt sich über Flash in Verbindung mit ActionScript sagen, dass es weit verbreitet ist und für alle gängigen Browser entsprechende Plugins existieren. Flash ist aber nicht Multithread-fähig, was sich unter anderem ressourcenbelastend auf den Prozessor auswirkt, da für jeden Flash-Film ein eigener Flash-Player gestartet werden muss. Die Scalable Vector Graphics haben noch nicht den Verbreitungs- und Entwicklungsgrad erreicht, den man sich für eine produktiv laufende Anwendung wünschen würde. Die für die Monitor-Applikation benötigten Funktionen wären nur mittels eines Plugins möglich, das es nur für den Internet Explorer gibt, was die Applikation auf Windows-Plattformen beschränken würde. Dies wäre eine Einschränkung, die im heterogenen Forschungsumfeld des Cluster-Systems (siehe Kapitel 1.2.1) kontraproduktiv wirken würde.

Die Entscheidung fiel daher auf Java-Applets. Die plattformunabhängige und Multi-thread-fähige Programmiersprache Java ist weit verbreitet, Plugins sind für alle gängigen Browser verfügbar. Die programmiertechnischen Möglichkeiten werden durch ein gut dokumentiertes Application Programming Interface (API) unterstützt. Auch bietet Java mit Java 2D und dem Simple API for XML (SAX) einfache Möglichkeiten, auch komplexere Grafiken zu erstellen und XML-Dateien elegant auszuwerten.

3.3 Technologien zur Übertragung von Daten

In diesem Kapitel werden verschiedene vorhandene Technologien zur Übertragung von Daten betrachtet und verglichen. Besonderes Augenmerk wird hierbei auf die in Kapitel 2.5 genannten Anforderungen an das Übertragungsprotokoll zwischen Darstellungskomponente und Cluster-Komponente gelegt. Dazu zählt z. B. die Austauschbarkeit der Komponenten, wodurch sämtliche für Programmiersprachen spezifischen Lösungen, so z. B. das direkte Verschicken von Java-Objekten, entfallen.

3.3.1 Type Length Value

Bei Type Length Value (TLV) handelt es sich um ein Kodierungsverfahren, Daten in Kommunikationsprotokollen zu verschicken. Die einzelnen Datensätze werden wie folgt kodiert: Zunächst wird die Art der Daten als *type* kodiert. Bei der Monitor-Applikation könnten z. B. der Datendurchsatz für einen bestimmten Datenträger, die CPU-Auslastung für eine konkrete CPU oder die akkumulierten Daten für zwei Netzwerkinterfaces eines Knotens als *type* kodiert werden. Danach wird die Länge der eigentlichen Daten als *length* kodiert. Zuletzt folgen die Daten als *value*. Diesem folgt unmittelbar der nächste Datentyp; es gibt kein Trennzeichen, da über das Feld *length* die Länge der Daten bekannt ist.

Großer Vorteil der TLV-Kodierung ist, dass sie mit wenig Overhead auskommt und daher mit einer hohen Durchsatzeffizienz gekennzeichnet ist. Es werden nur die nötigsten Informationen übertragen, wodurch auch das Parsen der Informationen relativ schnell geht. Nachteilig ist allerdings, dass es vergleichsweise unflexibel ist und heute kaum noch Verwendung findet, da es in vielen Bereichen von XML (siehe Kapitel 3.3.2) abgelöst wurde. Dennoch handelt es sich um einen Standard, der sich in Protokollen wie dem Simple Network Management Protocol (SNMP) für den Informationsaustausch oder auf der deutschen Krankenkassenkarte (2006) als Kodierungsverfahren für die Informationsspeicherung etabliert hat.

3.3.2 Extensible Markup Language

Bei der Extensible Markup Language (XML) handelt es sich um einen Standard zum Austausch von Informationen. XML ist dabei so flexibel, dass nahezu beliebige

Daten – sofern sich diese in einer baumartigen Struktur serialisieren lassen – in das XML-Format konvertiert werden können. Der intelligente Vorteil von XML ist, dass nicht nur die reinen Datenwerte, sondern auch die Semantik und Struktur dieser Werte mitkodiert werden. Dadurch kann auch auf maschineller Seite sehr flexibel gearbeitet werden.

Ein weiterer Vorteil von XML ist, dass es als Datenaustauschformat relativ bekannt ist und daher auch in Zukunft noch als solches in Anwendungen implementiert sein wird. Des Weiteren gibt es für viele gängige Programmiersprachen Parser, die XML-Dateien einlesen und auswerten können. Dadurch wird zum einen der Aufwand gespart, den die Entwicklung eines proprietären Formats kosten würde, zum anderen entfällt die Entwicklung eines entsprechenden Parsers für dieses Format. XML ist als Format zum Austausch zwischen Anwendungen gedacht. An ein solches Format wird grundsätzlich nicht der Anspruch gestellt, es müsse in für Menschen lesbarer Form vorliegen. XML ist jedoch für Menschen lesbar, was ebenso vorteilhaft wie nachteilig sein kann.

Ein klarer Nachteil besteht darin, dass durch XML relativ viel Overhead verursacht wird, um Informationen für Menschen lesbar zu machen. Werden die Daten ausschließlich von Programmen ausgewertet, ist dies eine ungenutzte Eigenschaft, die aber trotzdem Ressourcen kostet. Bei einer rein maschinellen Verarbeitung wäre eine kompaktere Kodierung möglich. Beispielsweise werden für den Wert 53757 in XML fünf Ziffern gespeichert und damit (je nach Zeichencode) fünf Bytes benötigt. Binär ließe sich der Wert in die Bitfolge `11010001 11111101` kodieren, was einer Größe von zwei Bytes entspräche. Damit für Menschen auch die Semantik des Wertes erkennbar ist, werden in XML die Daten in einer hierarchischen Struktur aufgebaut. Diese Struktur erzeugt weiteren Overhead. Um deutlich zu machen, dass es sich um eine deutsche Postleitzahl (PLZ) handelt, würde man den Wert z. B. in einen PLZ-Tag einschließen: `<plz>53757</plz>`. Dadurch kämen in diesem Beispiel noch einmal elf Bytes hinzu. Insgesamt würden also 16 Bytes übertragen werden, für einen Wert, der binär kodiert (wie in TLV) nur ein Achtel des Speicherplatzes, nämlich zwei Bytes, benötigen würde. Außer Acht gelassen wurden in diesem Beispiel Header-Informationen, die sowohl bei TLV als auch bei XML nötig wären.

3.3.3 Entscheidung für eine Übertragungstechnologie

Tabelle 2 zeigt einen Vergleich von Type Length Value und der Extensible Markup Language bezogen auf die Kriterien Overhead, Flexibilität, Aktualität und Integration in Programmiersprachen.

	Type Length Value	Extensible Markup Language
Overhead	Sehr wenig Overhead	Sehr viel Overhead
Flexibilität	Starre Konstruktion	Hoher Flexibilisierungsgrad
Aktualität und Integration	Gilt als überholt, Parser sind vorhanden	Gilt als aktuelle Technologie, es gibt Parser für viele Programmiersprachen inklusive Java

Tabelle 2: Gegenüberstellung analysierter Übertragungstechnologien

Zusammenfassend lässt sich sagen, dass der große Vorteil des Type-Length-Value-Formates – der geringe Overhead – heute nicht mehr ausschlaggebend zum Tragen kommt. Sowohl die Bandbreite der Netzwerke als auch die Speicherkapazitäten entwickeln sich stetig weiter, sodass bei den geringen Datenmengen, die bei der Monitor-Applikation übertragen werden, dieser Punkt nicht mehr ins Gewicht fällt. In folgendem Rechenbeispiel wird von einer Übertragung mittels XML ausgegangen. Berechnet werden die Nutzdaten für die Monitor-Applikation; außer Acht gelassen wird der Overhead, der durch die unteren Schichten wie das Transmission Control Protocol entsteht. Geht man davon aus, dass ein darzustellender Wert in 200 Bytes auszudrücken ist (nach oben aufgerundet) und maximal viermal pro Sekunde 40 unterschiedliche Werte angezeigt werden sollen, so entsteht ein zu übertragendes Volumen von $(200 \cdot 40 \cdot 4 =)$ 32.000 Byte pro Sekunde, was weniger als 2 Megabyte (MB) pro Minute entspricht. Ein Volumen, welches problemlos über aktuelle Netzwerke zu übermitteln ist.

Die Entscheidung fiel daher auf die Extensible Markup Language. Die ausschlaggebenden Argumente hierfür sind ihr hoher Grad an Flexibilität und Erweiterbarkeit sowie die Tatsache, dass es sich um einen etablierten Standard handelt. Ein weiterer Punkt ist die Austauschbarkeit der verschiedenen Komponenten der Monitor-Applikation. Ein Datenstrom im XML-Format kann gegebenenfalls auch von externen Programmen erzeugt und gelesen werden.

3.4 Java Grafikbibliotheken

In Kapitel 3.2.5 wurde sich für Java-Applets als zu verwendendes Visualisierungsmittel entschieden, in Kapitel 3.3.3 für XML als zu verwendendes Datenübertragungsformat. In diesem Kapitel werden daher verfügbare Bibliotheken von Java-Applets analysiert, wobei insbesondere auf die volle Dynamik und die XML-Schnittstelle geachtet wird.

Vorhandene Java-Klassen-Bibliotheken zur Darstellung verschiedener Diagrammtypen sind sowohl kommerziell als auch frei verfügbar. Eine umfangreiche Über-

sicht [W10-06] und ausführliche Funktionsbeschreibungen zu den einzelnen Produkten sind im Internet zu finden.

Der Vorteil, bereits vorhandene Bibliotheken zu nutzen, liegt darin, dass sich mit wenigen Code-Zeilen Diagramme erzeugen lassen, die bereits fertig formatiert und beschriftet sind. Da Funktionalität und Eigenschaften bei den meisten Bibliotheken identisch sind, wird im Folgenden nur auf eine Bibliothek – SpiceCharts von reconn.us [REC-05] – näher eingegangen. Danach können Analogien zu weiteren Produkten wie Swiftchart von Swiftchart Limited [SWI-O] oder easycharts von ObjectPlanet AS [OBJ-06] leicht hergestellt werden. Abschließend wird noch das Produkt Real-Time Graphics Tools for Java der Quinn-Curtis Inc. [QUI-06a] betrachtet.

Bei SpiceCharts handelt es sich um eine Bibliothek von Java-Klassen, die Diagrammtypen wie Kreissektoren-, Balken- oder Liniendiagramme beinhalten. Wie bei nahezu allen vergleichbaren Produkten sind auch hier die Diagramme nicht voll dynamisch. Diese Bibliothek geht davon aus, dass die Werte beim Laden der Webseite bzw. bei der Initialisierung des Applets einmalig (z. B. aus einer Datenbank) geholt werden, sich diese aber während der Anzeige der Webseite nicht mehr ändern. Deutlich wird dies, wenn man sich den Quellcode der beispielhaften Einbindung eines 2D-Liniendiagramms auf eine Webseite ansieht [REC-05a]. Bei einer Änderung bleibt nur die Möglichkeit, das gesamte Diagramm zu verwerfen und neu zu zeichnen. Da für die Monitor-Applikation jedoch bis zu viermal pro Sekunde neue Werte dargestellt werden sollen, würde das Neuzeichnen eines Diagramms in dieser Geschwindigkeit zu einem schnellen Flackern führen. Um das Diagramm dynamisch aktualisierbar zu machen, müsste der Quellcode angepasst werden. Da für SpiceCharts die Applets nur in kompilierter Form vorliegen, sind nachträgliche Änderungen am Code nicht möglich. Viele freie Bibliotheken, so auch SpiceCharts, ergänzen die erstellten Diagramme um eine „Werbezeile“. SpiceCharts ist daher als Bibliothek nicht geeignet.

Hinsichtlich weiterer Bibliotheken seien hier beispielhaft Swiftchart und easycharts genannt. Wie in den Beispielen von Swiftchart [SWI-O]a) und easycharts [OBJ-06a] deutlich zu sehen ist, müssen auch hier alle zu zeichnenden Werte bereits bei der Initialisierung der Applets in Form von Koordinatenpunkten übergeben werden. Auch diese Bibliotheken sind daher, bedingt durch die fehlende Funktionalität der voll dynamischen Grafiken, für den Cluster-Monitor nicht geeignet.

Die Real-Time Graphics Tools for Java der Quinn-Curtis Inc. erlauben Änderungen am Applet, auch nachdem es gezeichnet wurde, damit erfüllen sie das Kriterium der voll dynamischen Grafiken. Sie basieren auf der statischen Variante, den Chart2D for Java Charting Tools [QUI-06]. Sie sind erheblich umfangreicher und beinhalten neben Balken- und Liniendiagrammen z. B. auch Zeigerdiagramme

(ähnlich einem Tachometer) und erweiterte Funktionen wie das Gruppieren verschiedener Wertequellen in ein einzelnes Diagramm. Diese Bibliothek verfügt nicht über eine XML-Schnittstelle. Die Verwendung standardisierter Technologien ist jedoch ein wichtiges Kriterium des Cluster-Monitors (siehe Kapitel 1.2.1 und Kapitel 1.3).

Auch für andere Bibliotheken gilt bezüglich Änderungen am Code, insbesondere nicht vorhandene Schnittstellen und volle Dynamik betreffend, dass sie nicht einfach zu implementieren sind: Damit ein Applet dynamisch aktualisiert werden kann, muss dies von vornherein in seiner Architektur vorgesehen sein. So muss es die Möglichkeit geben, Applets zur Laufzeit neu zu zeichnen; es muss möglich sein, Daten zur Laufzeit des Applets z. B. über eine Socket-Verbindung abzufragen, und das Applet muss Multithread-fähig sein, wenn mehrere Applets gleichzeitig ausgeführt werden sollen. Dies sind Kriterien, die bereits in die Planung der Architektur des Applets einfließen sollten. Eine nachträgliche Erweiterung gestaltet sich als schwierig, da die Schnittstellen für oben genannten Funktionen vollständig fehlen.

Fazit ist, dass keine der analysierten verfügbaren Bibliotheken alle genannten Anforderungen und Ziele vollständig abdeckt. Daher ist es erforderlich, ein eigenes Toolkit zu entwickeln.

4 Architektur und Schnittstellen

In diesem Kapitel wird zunächst die Struktur der Monitor-Applikation aus Kapitel 1.2.2 aufgegriffen, präzisiert und die Verteilung der einzelnen Komponenten auf physikalische Maschinen erläutert. Der zweite Teil beschäftigt sich mit den Schnittstellen der Darstellungskomponente zur Framework- und Cluster-Komponente. Im dritten und vierten Abschnitt wird näher auf die Architektur der Darstellungskomponente eingegangen, indem Klassenhierarchie und Ressourcenbedarf dargestellt werden.

4.1 Architektur

Die Monitor-Applikation besteht aus drei großen Komponenten. Abbildung 5 zeigt die Verteilung der Komponenten auf physikalische Maschinen.

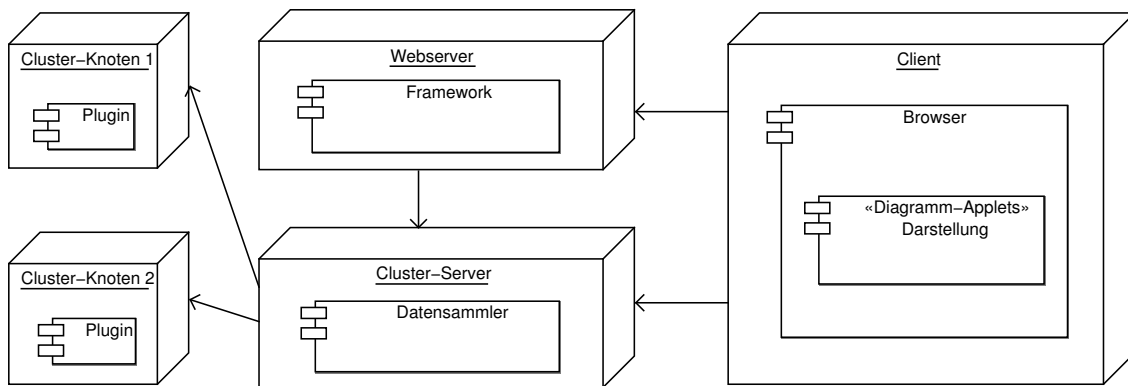


Abbildung 5: Verteilungsdiagramm des Cluster-Monitors (UML 2)

- Die Cluster-Komponente (siehe [KLE-06]) läuft auf dem Cluster-Server und den Cluster-Knoten. Auf den einzelnen Knoten werden die verschiedenen Daten wie Prozessorauslastung oder Netzwerkdurchsatz von so genannten Plugins gemessen. Der auf dem Server laufende Teil (der so genannte Datensammler) sammelt und verarbeitet die Rohdaten der einzelnen Knoten und stellt sie aufbereitet der Darstellungskomponente zur Verfügung. Er ist daher der für diese Arbeit relevante Teil.
- Die Framework-Komponente (siehe [JUN-06]) stellt die Oberfläche der Monitor-Applikation als Browser-basiertes Framework zur Verfügung. Das Framework läuft auf einem Webserver und kommuniziert mit dem Datensammler, um z. B. zu erfragen, welche Plugins mit welchen Aggregatsfunktionen angeboten werden und welche derzeit aktiv laufen.
- Die Darstellungskomponente besteht aus den verschiedenen Applets, die der Benutzer über das Framework ausgewählt hat. Die Applets laufen innerhalb

des Browsers auf dem Client und zeigen die Messdaten z. B. in Diagrammform an. Jedes Applet wendet sich dabei selbstständig an den Datensammler, um von dort die aktuellen Messdaten abzufragen.

Genauer betrachtet findet die folgende Kommunikation zwischen den einzelnen Komponenten statt: Der Benutzer startet das Framework der Monitor-Applikation in seinem Browser über die entsprechende Webseite. Das Framework fragt nun vom Datensammler die Cluster-Architektur ab. Anhand dieser wird ein Baum auf der Webseite generiert, der die Cluster-Architektur repräsentiert und dem Benutzer die verschiedenen Monitoring-Plugins zur Auswahl anbietet. Wählt der Benutzer ein Plugin aus, so erzeugt das Framework das passende Diagramm-Applet. Die genaue Architektur der Web-Applikation und die Interaktion mit dem Benutzer ist einer anderen Bachelor-Arbeit [JUN-06] zu entnehmen. Das Applet wendet sich nun seinerseits an den Datensammler, um die aktuellen Messwerte abzufragen (siehe Abbildung 6). Die Abbildung zeigt die Schnittstellen der drei Komponenten und die Kommunikation untereinander. Nicht dargestellt wird die Kommunikation des Datensammlers mit den einzelnen Cluster-Knoten, da diese für diese Arbeit nicht relevant ist. Die Kommunikation des Datensammlers ist in einer anderen Bachelor-Arbeit [KLE-06] beschrieben.

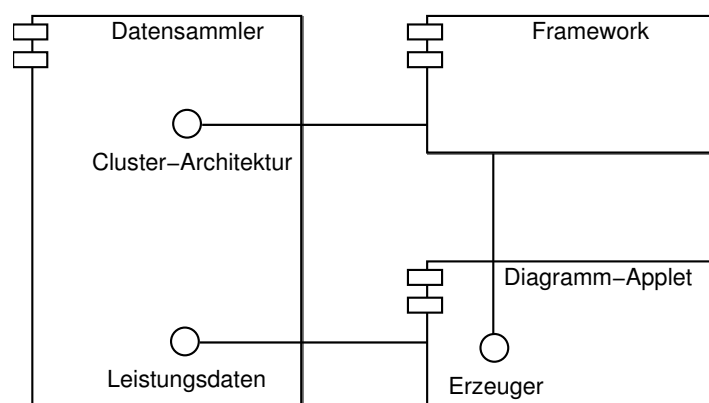


Abbildung 6: Komponentendiagramm (UML 2)

4.2 Schnittstellen

Da jede der in Kapitel 1.2.2 vorgestellten Komponenten getrennt in einer Bachelor-Arbeit entwickelt wird, ist eine enge Kommunikation und Abstimmung sehr wichtig. Im Folgenden wird daher zunächst definiert, welche Daten über die Schnittstellen der Darstellungskomponente übertragen werden. In Kapitel 5 werden die Schnittstellen schließlich realisiert.

4.2.1 Schnittstelle: Framework – Darstellung

Die Framework-Komponente muss anhand der über das Web-Interface getätigten Benutzereingaben verschieden konfigurierte Applets aufrufen [JUN-06]. Dabei muss vor allem der Typ des Diagramms ausgewählt werden können. Je nach erzeugtem Applet sind bei der Initialisierung verschiedene Konfigurationsparameter möglich. Auch zur Laufzeit des Applets können noch gewisse Veränderungen am Applet vorgenommen werden. Die Schnittstelle zum Framework ist unidirektional, das heißt das Framework startet und konfiguriert die Applets, letztere kommunizieren jedoch nicht aktiv mit der Framework-Komponente. Die formale Definition der Schnittstelle im XML-Schema-Format befindet sich im Anhang 8.1, die vollständige Schnittstelle im XML-Format inklusive aller möglichen Parametern im Anhang 8.2. Im Folgenden werden die verschiedenen Parameter klassifiziert und erläutert.

Für alle Diagrammtypen sind bei der Initialisierung die folgenden Parameter obligatorisch:

- Höhe und Breite des Applets
- Anzuzeigende Messwerte
- DNS-Name oder IP-Adresse des Servers (Datensammler)
- Portnummer, für die Verbindung zum Datensammler

Für diese Parameter sind keine Standardwerte möglich. Werden Höhe und Breite nicht angegeben, kann der Browser das Applet nicht korrekt anzeigen; fehlen DNS-Name oder Port des Datensammlers, kann keine Socket-Verbindung geöffnet werden; fehlt die Angabe des Messwerts, kann das Applet nicht wissen, was anzeigen werden soll.

Des Weiteren sollten die folgenden Parameter für jeden Diagrammtyp übergeben werden. Dies ist optional, denn bei Fehlen wird nach Möglichkeit auf Standardwerte zurückgegriffen oder es fehlt die entsprechende Funktionalität des Plugins.

- Beschriftung des Diagramms
- Aggregatsfunktion
- Abfrageintervall für den Datensammler

Für die implementierten Diagrammtypen Liniendiagramm, Multiliniendiagramm und Bargraph sind weitere optionale Parameter möglich:

- Maximal darstellbarer Wert, mit dem die Ordinate initialisiert wird
- Ein- oder Ausschalten der Skalierungsfunktion für die Diagrammskalen

Es ist auch möglich, zur Laufzeit Einfluss auf Applets zu nehmen. In Liniendiagramm und Bargraph gibt es die Möglichkeit, Hilfslinien ein- oder auszublenden, die den minimal oder maximal erreichten Wert anzeigen.

4.2.2 Schnittstelle: Darstellung – Cluster

Über das Framework wird der Darstellungskomponente (den Applets), mitgeteilt, welche Daten sie anzeigen sollen. Bei der Initialisierung der Applets werden diese Informationen an die Cluster-Komponente, also den Datensammler, weitergegeben. Des Weiteren müssen auch das Abfrageintervall und die Aggregatsfunktion übermittelt werden, damit der Datensammler die Informationen korrekt aufbereiten kann und die Server-seitigen Plugins korrekt konfiguriert werden können.

Den aktuellen Wert erfragt das Applet im festgelegten Abfrageintervall vom Datensammler. Dieser schickt als Antwort den aktuellen Wert an das anfragende Applet zurück.

Wird ein Applet geschlossen, teilt es dem Datensammler mit, dass keine weiteren Werte mehr übermittelt werden sollen und trennt danach die Verbindung. Das Schließen des Applets kann explizit vom Benutzer über das Framework initiiert werden aber auch implizit, wenn der Browser geschlossen oder die Webseite der Monitor-Applikation verlassen wird.

4.3 Klassenhierarchie

Abbildung 7 (Seite 30) zeigt die implementierten Diagrammtypen in ihrer Klassenhierarchie. Grundlage für alle Diagramm-Applets ist die abstrakte Klasse *KMR_Applet*. *KMR_Applet* ist direkt abgeleitet von *java.applet.Applet*, erweitert diese Klasse jedoch um zusätzliche Methoden, die von allen Diagramm-Applets verwendet werden können. Da mehrere Applets gleichzeitig in einer Virtual Machine laufen sollen, müssen alle Applets als Thread implementiert werden. Java unterstützt keine Mehrfachvererbung, weswegen die Klasse *KMR_Applet* nicht von *java.lang.Thread* abgeleitet werden kann, sondern stattdessen das Interface *java.lang.Runnable* implementieren muss.

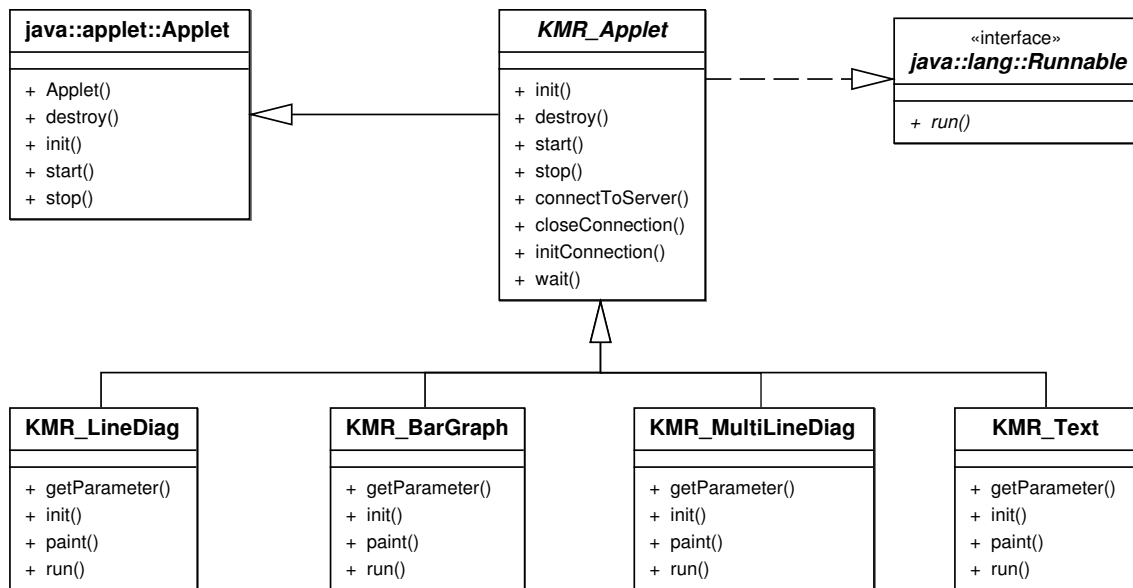


Abbildung 7: Klassendiagramm (UML 2)

4.4 Skalierbarkeit und Ressourcenbedarf

Das Verteilungsdiagramm (Abbildung 5, Seite 26) zeigt die Verbindung des Clients mit dem Datensammler, worüber die Messdaten an die Applets transportiert werden. Um komplizierte Kommunikation und Synchronisation zwischen den Diagramm-Applets zu vermeiden, agiert jedes Applet völlig autark, weiß also nichts von der Existenz weiterer Applets. Dies hat aber auch den Nachteil, dass jedes Applet eine eigene Socket-Verbindung öffnen muss, um aktuelle Daten abzufragen (siehe auch Implementierung, Kapitel 5.3). Daher wird für jedes Applet ein Port belegt, was bei den heute im Internet üblichen Transportprotokollen eine Grenze von maximal 65.535 Applets bedeutet. Allerdings ist dies keine realistische Grenze, da schon ab einer Anzahl von über 100 Diagrammen fraglich ist, ob es für den Benutzer nicht übersichtlicher wäre, verschiedene Werte akkumulieren zu lassen. Beispielsweise wird für die Abfrage von drei akkumulierten Werten aus 100 einzelnen im Multiliniendiagramm nur ein Socket benötigt.

Alternativ wäre auch möglich gewesen, dass ein dediziertes Applet die Kommunikation mit dem Datensammler regelt und die jeweiligen Messwerte an die entsprechenden Diagramm-Applets weiterleitet. Vorteilhaft wäre hierbei das Öffnen nur eines Sockets, wodurch die Konfiguration von Firewalls vereinfacht werden würde, wenn das Monitor-System z. B. über das Internet genutzt werden soll. Diese Vorgehensweise wurde aber aus oben genannten Gründen nicht gewählt.

Der Ressourcenbedarf des Clients wurde auf zwei Systemen mit den Prozessoren AMD Athlon und Intel Pentium 4 gemessen. Dabei wurde eine Anwendung mit 20 komplizierten Diagrammtypen (also keinen rein textuellen Ausgaben) im Firefox-Browser gestartet und eine Aktualisierungsrate von 250 Millisekunden festgelegt.

Der Arbeitsspeicherverbrauch belief sich dabei auf weniger als 10 MB. Verglichen mit Java Virtual Machine und Firefox, die jeweils ca. 20 MB benötigen, ist dieser Wert bei den heutigen Arbeitsspeichergrößen im Gigabyte-Bereich nicht ausschlaggebend.

Um die Rechenknoten bzw. das gesamte Cluster-System möglichst wenig zu belasten und dadurch das Messergebnis weitestmöglich unverfälscht zu erhalten, werden alle rechenintensiven Grafikoperationen auf der Client-Seite durchgeführt. Die Applets bewirken daher allerdings eine starke Prozessorauslastung auf dem Client: Auf den genannten Computern belasten die 20 gleichzeitig laufende Diagramm-Applets den Prozessor mit bis zu 80 %.

5 Implementierung

In diesem Kapitel wird beschrieben, wie die in Kapitel 4 vorgestellte Architektur umgesetzt wurde. Als Erstes wird die Schnittstelle zum Framework erläutert. Der zweite Abschnitt behandelt die Implementierung der Applets selbst inklusive einiger Hilfsklassen. Zuletzt wird die Kommunikationskomponente beschrieben, insbesondere das Protokoll zur Kommunikation mit dem Datensammler.

5.1 Schnittstelle zum Framework

Die Monitor-Applikation läuft auf der Client-Seite vollständig Browser-basiert. Die Framework-Komponente erzeugt daher je nach Benutzereingaben HTML-Code, welcher ein speziell konfiguriertes Applet aufruft.

Jeder Diagrammtyp ist in einer eigenen Klasse implementiert, sodass das Framework über den Dateinamen des Applets steuern kann, welcher Diagrammtyp angezeigt werden soll. Die Höhen- und Breitenangaben werden als Attribute des *applet*-Tags übergeben. Anhand dieser Attribute wird einerseits vom Browser der entsprechende Platz auf der Webseite für das Applet reserviert. Auf der anderen Seite fragt das Applet diese Informationen vom Browser ab, um anhand derer die korrekten Ausmaße des Koordinatensystems berechnen zu können.

Alle anderen Parameter (seien es die obligatorischen Informationen über den Datensammler, zusätzliche Informationen über Abfrageintervall und Aggregatsfunktion oder diagrammspezifische Parameter) werden mittels HTML-Tags „übergeben“. Hierzu stellt HTML die Möglichkeit bereit, innerhalb eines *applet*-Tags beliebig viele *parameter*-Tags zu verwenden. Der *parameter*-Tag kennt die beiden Attribute *name* und *value*. So lassen sich dem Applet bei der Initialisierung Name-Wert-Paare übergeben, die das Verhalten eines bestimmten Diagrammtyps steuern können. Welcher Diagrammtyp welche Parameter akzeptiert ist in einer XML-Datei hinterlegt, die vom Framework ausgelesen wird. Die formale Spezifikation anhand eines XML-Schemas ist im Anhang 8.1 hinterlegt, die vollständige Schnittstellendefinition im Anhang 8.2. Im Folgenden werden die diagrammspezifischen Parameter beispielhaft für das Liniendiagramm in XML-Form gezeigt:

```
<?xml version="1.0"?>
<applet>
  <diagram type="line">
    <classname>diagramApplets.KMR_LineDiag.class</classname>
    <width>50</width><width>100</width><width>150</width><width>200</width>
    <width>300</width><width>400</width><width>500</width><width>600</width>
    <width_add>50</width_add>
    <default_width>400</default_width>
    <height>10</height><height>20</height><height>50</height>
    <height>100</height><height>200</height><height>300</height>
    <height_add>20</height_add>
    <default_height>100</default_height>
    <parameter name="description"
      description="Beschriftung der Y-Achse"/>
    <parameter name="aggregate" description="Eine Aggregatsfunktion" />
    <parameter name="ids" description="Eindeutige ID(s) fuer ein Plugin" />
  </diagram>
</applet>
```

```

<parameter name="interval"
  description="Intervall in Millisekunden bis zur naechsten Abfrage" />
<parameter name="coord_max" description="Maximal darstellbarer Wert" />
<parameter name="show_max"
  description="Ein-/Ausschalten der Hilfslinie fuer den Maximalwert" />
<parameter name="show_min"
  description="Ein-/Ausschalten der Hilfslinie fuer den Maximalwert" />
<parameter name="scalable"
  description="Skalierbarkeit ein- oder ausschalten" />
<parameter name="server_name"
  description="IP-Adresse oder DNS-Name des Datensammlers" />
<parameter name="port"
  description="Port, auf dem der Socket geoeffnet werden soll" />
</diagram>
</applet>

```

Über den *classname*-Tag erfährt das Framework Paket- und Dateinamen des Applets. Anhand der verschiedenen *width*- und *height*-Tags kann dem Benutzer erlaubt werden, die gewünschte Größe des Diagramms auszuwählen. Er kann den Zeichenbereich des Liniendiagramms z. B. 200x500 Pixel groß wählen, Standardwerte sind in *default_width* und *default_height* hinterlegt. Das Framework muss *width_add* bzw. *height_add* Pixel addieren, die z. B. für die Achsenbeschriftungen verwendet werden.

Alle Plugins haben sind die Parameter *ids*, *server_name*, *port*, *aggregate* und *interval* gemeinsam. Über den Parameter *ids* wird dem Applet mitgeteilt, welche Daten anzuzeigen sind (Plugin-ID). Dies ist entweder ein einzelner Integer-Wert oder, im Fall von akkumulierten Daten, eine mit Komma getrennte Liste von Integer-Werten. Das Applet selbst wertet diese Information nicht aus, muss sie aber beim Verbindungsaufbau dem Datensammler (siehe Kapitel 5.3) mitteilen, damit dieser die richtigen Daten liefert. Über die Parameter *server_name* und *port* lässt sich der Socket des Datensammlers definieren, den das Applet zum Verbindungsaufbau nutzen soll. Der Name des Servers kann dabei ein DNS-Name oder die IP-Adresse sein. Der Parameter *aggregate* ermöglicht die Definition einer Aggregatsfunktion. Wie bei der Plugin-ID wertet das Applet diesen Parameter nicht aus, sondern leitet die Information nur an den Datensammler weiter. Erlaubte Werte sind *NONE*, wenn keine Akkumulierung gewünscht ist (Standard), *SUM* für die Summe, *AVG* für den Durchschnitt, *MIN* und *MAX* für Minimum respektive Maximum. Über den Parameter *interval* wird das Abfrageintervall angegeben, in dem das Applet aktuelle Werte vom Datensammler abfragen soll. Anzugeben ist der Wert in Millisekunden; der Standardwert beträgt 250 Millisekunden.

Für das Liniendiagramm können außerdem die Parameter *description*, *coord_max*, *show_min*, *show_max* und *scalable* angegeben werden. Über *description* lässt sich das Diagramm beschriften. Hier sollte nach Möglichkeit auch die Einheit (z. B. %) angegeben sein, damit die Achse des Diagramms korrekt beschriftet werden kann. Über *coord_max* kann der beim Starten des Applets zu verwendende Höchstwert für die Ordinate angegeben werden. Dieser muss sinnvoll gewählt sein, damit ein einfaches Ablesen der Werte möglich ist. Bei einer Angabe in Prozent sollte dieser

Wert z. B. 100 betragen. Die Parameter *show_min* und *show_max* steuern, ob die Minimal- und Maximal-Hilfslinien beim Start des Applets ein- oder ausgeblendet sein sollen. Ist der Parameter *scalable* auf *false* gesetzt, wird eine Skalierung des Applets unterbunden. Dies kann sinnvoll sein, wenn man genau weiß, dass die Messwerte einen bestimmten Wertebereich nie verlassen werden oder wenn Daten verglichen werden sollen (siehe Kapitel 2.4.3).

Anhand dieser Informationen erstellt das Framework z. B. folgenden HTML-Code für ein Liniendiagramm mit dem Dateinamen *KMR_LineDiag.class* im Paket *diagramApplets* mit einer Breite von 450 und einer Höhe von 120 Pixeln. Dieses Liniendiagramm soll die zum Durchschnitt aggregierten Plugins mit den IDs 200, 201, 202 und 203 alle 500 Millisekunden vom Server *cluster.fh-brs.de* auf Port 8080 abfragen:

```
<applet code="diagramApplets.KMR_LineDiag.class" width="450" height="120">
  <param name="server_name" value="cluster.fh-brs.de">
  <param name="port" value="8080">
  <param name="interval" value="500">
  <param name="coord_max" value="100">
  <param name="ids" value="200,201,202,203">
  <param name="aggregate" value="avg">
</applet>
```

5.2 Grafische Komponente

Wie dem Klassendiagramm (Abbildung 7, Seite 30) zu entnehmen ist, sind allen Klassen die Methoden *init*, *run*, *paint* und *getParameter* gemein. Die *init*-Methode wird einmalig vom Browser beim Start des Applets aufgerufen und übernimmt dementsprechende Initialisierungsarbeiten. Die *run*-Methode wird als Hauptmethode verwendet. Sobald der Thread des Applets nach der Initialisierung gestartet wurde, wird innerhalb der *run*-Methode eine Schleife gestartet, die bis zur Zerstörung des Threads bzw. Applets Werte vom Datensammler abfragt und anschließend durch Aufruf der *paint*-Methode anzeigt. Die *paint*-Methode wird auch vom Browser aufgerufen, wenn ein Applet z. B. durch ein anderes Fenster verdeckt war. Daher werden in den *paint*-Methoden keine wertverändernde Anweisungen durchgeführt, sondern nur Vorhandenes zum Zeichnen gelesen. Die *getParameter*-Methode liest die in Kapitel 5.1 beschriebenen Parameter ein, die von Browser bzw. Framework gesetzt wurden.

5.2.1 Diagrammtypen

Die Klassen *KMR_LineDiag* und *KMR_MultiLineDiag* implementieren die Funktionalität eines Liniendiagramms und die eines Multiliniendiagramms. Sie sind beide abhängig von der Klasse *KMR_Polygon* (Abbildung 8, Seite 35). Diese Klasse repräsentiert einen Polygonzug innerhalb eines Liniendiagramms und besteht daher aus einem Array von Linien-Objekten. Sie implementiert das Interface *java.util.Iterator*, angelehnt an das Muster des Iterators [GAM-97], sodass sich alle Einzellinien des

Polygonzuges bequem abfragen lassen. Die Schreiboperation (Methode: *addSingleLine*) auf das Array ist dabei als Queue implementiert, sodass die jeweils ältesten Linien durch neue überschrieben werden.

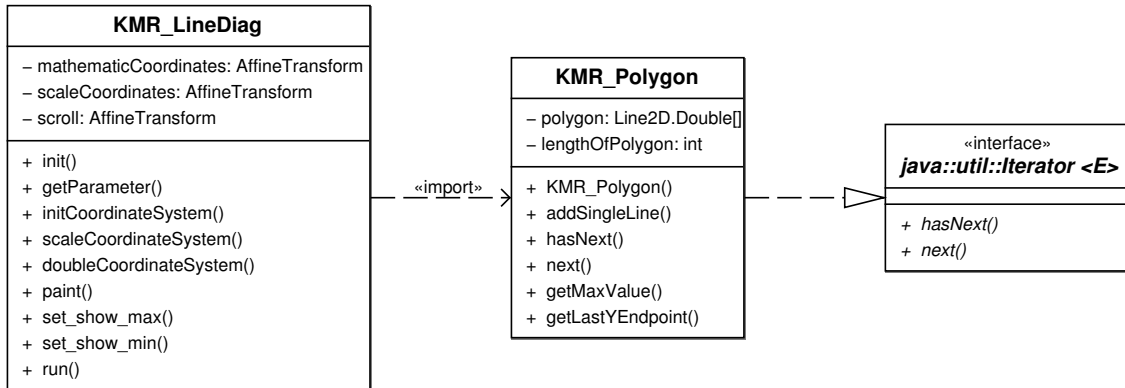


Abbildung 8: Klassendiagramm für das Liniendiagramm (UML 2)

Das Liniendiagramm funktioniert wie folgt: Die aktuell abgefragten Werte vom Datensammler werden in Linienform gebracht und anschließend zu einem Polygonzug-Objekt zusammengefasst. Der Polygonzug wird auf ein *Graphics2D*-Objekt aufgetragen, auf das zuvor verschiedene affine Transformationen ausgeführt wurden. Die erste Transformation (*mathematicCoordinates*) verschiebt und spiegelt das Applet-Koordinatensystem so, dass ein in der Mathematik übliches kartesisches Koordinatensystem entsteht, in dem steigende X- und Y-Werte nach rechts bzw. nach oben aufgetragen werden. Die zweite Transformation (*scaleCoordinates*) skaliert das Koordinatensystem in der Form, dass alle Werte darstellbar sind; sie staucht es z. B. so, dass ein Pixel fünf Einheiten der Ordinate entspricht, wenn das Applet eine Höhe von 100 Pixeln hat und derzeit alle Werte unter 500 liegen. Diese Funktion implementiert die Methoden *initCoordinateSystem* beim Start des Applets und weiterhin die Methoden *scaleCoordinateSystem* und *doubleCoordinateSystem*, welche die Y-Dimension des Koordinatensystems zur Laufzeit des Applets halbieren bzw. verdoppeln. Die letzte Transformation (*scroll*) verschiebt für jeden neu erhaltenen Wert das Koordinatensystem um einen Pixel nach links. Dadurch entsteht der Eindruck, der Polygonzug wandere von rechts nach links durch das Applet. Die Methoden *set_show_min* und *set_show_max* steuern das Ein- und Ausblenden der Hilfslinien für den minimal bzw. maximal erreichten Wert. Sie können zur Laufzeit des Applets vom Browser aufgerufen werden, z. B. über JavaScript (siehe [JUN-06]).

Die Klasse *KMR_Bargraph* implementiert die Funktion eines Balkendiagramms mit nur einem Balken. Der vom Datensammler abgefragte Wert wird durch einen Balken repräsentiert, der je nach Messwert mehr oder weniger farblich ausgefüllt

wird. Wie beim Liniendiagramm erfolgt die Skalierung des Balkens über affine Transformationen.

Die textuelle Darstellung von Werten ist in der Klasse *KMR_Text* realisiert worden. Da mit Java 2D keine Mittel zur Strukturierung von Benutzeroberflächen (wie z. B. *Layout-Manager*) zur Verfügung stehen, wird sowohl die statische Beschriftung als auch der dynamische Messwert immer an feste Koordinatenpunkte geschrieben. Dadurch entsteht bei mehreren Text-Applets, die untereinander angezeigt werden, der optische Eindruck einer Tabelle.

5.2.2 Hilfsklassen

Es wurden verschiedene Hilfsklassen implementiert, z. B. zum Parsen von XML-Dateien. Zu den XML-Hilfsklassen gehören unter anderem die Klassen *KMR_XMLHandler*, die den XML-Stream des Datensammlers mit Messdaten auswertet, und die Klasse *KMR_XMLHandlerMeta*, welche Statusmeldungen des Datensammlers interpretiert. Letztere wird daher nur beim immer gleich laufenden Verbindungsaufbau benötigt, der in *KMR_Applet* implementiert wird (siehe Kapitel 5.3). Die Klassen *KMR_XMLHandler* wird in jedem Diagramm-Applet zur Auswertung der Messdaten verwendet. Außerdem werden Methoden zur Abfrage einer Menge von Datenwerten als *java.util.Vector*, oder einzelner Datenwerte bereitstellt.

Für den XML-Parser wurde auf das Simple API for XML Parsing (SAX) zurückgegriffen. Dieser wurde entworfen, um das schnelle Verarbeiten der Daten zu ermöglichen [ULL-05].

In der ersten prototypischen Implementierung eines Diagramm-Applets wurde ein eigener kleiner XML-Parser entwickelt, der im Wesentlichen nur aus *String.split*-Methoden bestand. Im Laufe des Projektfortschritts stellte sich jedoch heraus, dass die XML-Struktur der zu übertragenen Daten immer komplexer wurde, sodass der eigene Parser durch den SAX-Parser ersetzt wurde. Eine Alternative wäre das Document Object Model (DOM) gewesen. Bei dieser Parsing-Methode wird zunächst aus der gesamten XML-Datei ein Objekt im Speicher erzeugt. Auf diesem Objekt können danach Operationen durchgeführt werden. Diese Funktionalität wird jedoch hier nicht benötigt, weshalb der einfachere SAX-Parser verwendet wurde.

Eine weitere Hilfsklasse ist *CM_Styles*. In dieser werden Farben, Strichstärken und Schriftarten für die Anwendung „Cluster-Monitor“ definiert. Da alle Diagrammtypen auf die hierin definierten Stilelemente zurückgreifen, entsteht ein einheitliches Aussehen aller Diagramme.

Abbildung 9 (Seite 37) zeigt im UML-Klassendiagramm Vererbungshierarchie und Abhängigkeiten aller implementierten Klassen. Alle Klassen, die einen XML-

Handler instanziiieren, benötigen auch die Klasse *KMR_XMLSAXParser*; die vier Diagramm-Applets benötigen *CM_Styles*. Beide Beziehungen wurden aus Gründen der Übersichtlichkeit ausgelassen.

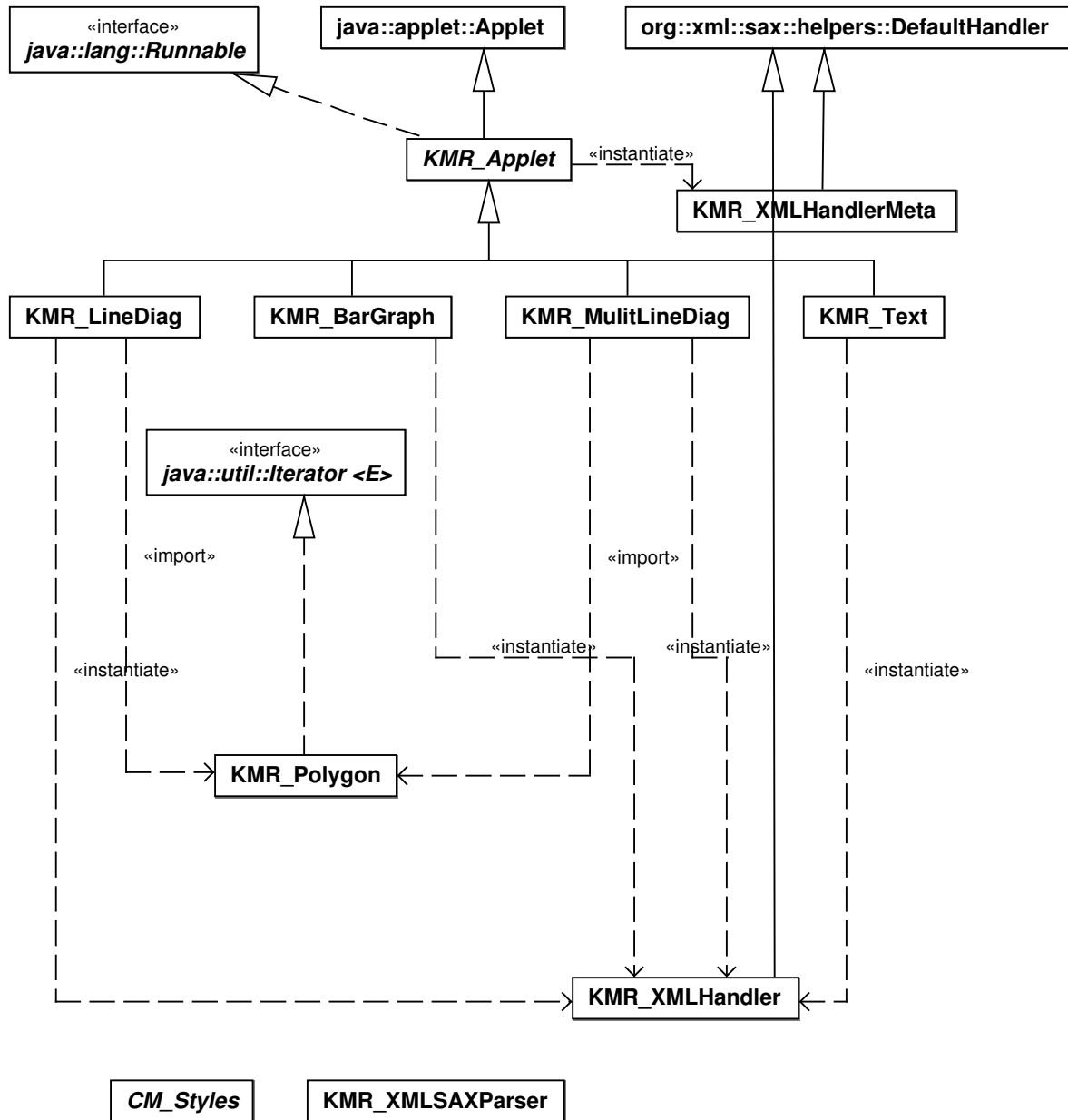


Abbildung 9: Klassendiagramm aller implementierten Klassen (UML 2)

5.3 Kommunikations-Komponente

Bei der Konzeption des Protokolls zur Kommunikation zwischen der Darstellungskomponente (den Applets) und der Cluster-Komponente (dem Datensammler) wurde besonderer Wert auf Flexibilität gelegt (siehe Anforderungen in Kapitel 1.2.1). Daher wurde als Strukturierungsmittel für die zu übertragenen

Informationen in Kapitel 3.3.3 die Extensible Markup Language (XML) gewählt. In der Gesamtarchitektur werden die gemessenen Leistungsdaten an zwei Stellen über das Netz gesendet: beim ersten Mal zwischen den einzelnen Knoten und dem Datensammler, beim zweiten Mal zwischen Datensammler und Client (siehe Abbildung 1, Seite 5). Die erste Übermittlung findet innerhalb des Cluster-Systems statt, wobei davon ausgegangen wird, dass die Einzelkomponenten des Cluster-Systems über ein eigenes und sicheres Netz, wenn nicht sogar ein spezielles Hochleistungsnetz, verbunden sind. Daher wurde an dieser Stelle als Transportprotokoll das unsichere und schnelle User Datagram Protocol (UDP) verwendet. Diese Kommunikation wird näher in einer anderen Bachelor-Arbeit behandelt [KLE-06]. Was für ein Netzwerk den Datensammler mit dem Client verbindet, kann nicht vorausgesagt werden. Möglich wäre sogar eine Übertragung über das Internet. Daher setzt das Protokoll an dieser Stelle direkt auf eine Socket-Verbindung über das sicherere (aber auch langsamere) Transmission Control Protocol (TCP) auf. Es sieht die in Abbildung 10, Seite 40 und die im Folgenden beschriebenen Schritte vor.

Zunächst erzeugt der Benutzer über die Webseite ein Diagramm-Applet.

1. Bei der Initialisierung wird anhand der im Framework angegebenen Parameter ein XML-Stream zusammengesetzt und übertragen. Beispiel: Alle 250 Millisekunden erfolgt Aktualisierung der Daten des Plugins mit der ID 42 ohne Aggregatsfunktion:

```
<?xml version="1.0"?>
<protocol type="init">
  <interval>250</interval>
  <aggregate>NONE</aggregate>
  <id>42</id>
</protocol>
```

2. Der Datensammler antwortet mit einer XML-Datei, die Metainformationen enthalten kann. Die Antwort kann *ok* sein, wenn die Anfrage korrekt gestellt wurde, oder *not_ok*, falls die Initialisierungsnachricht ungültig war.
3. Ist die Verbindung hergestellt, kann das Applet in einer Schleife eine weitere Meta-Nachricht mit der Information *get* schicken, die den aktuellen Wert vom Datensammler erfragt.
4. Der Datensammler antwortet mit einer XML-Nachricht des Typs *value*, beliebig viele Datenmuster (*sample*) können enthalten sein. Ein Datenmuster fasst dabei alle zu einem Messwert gehörenden Informationen (*value*) zusammen. Beispielsweise ist in einem Multiliniendiagramm jeder Polygonzug ein Datenmuster, bestehend aus einem einzelnen Messwert; bei einem Kreissektorendiagramm beschreibt ein Datenmuster den Sektor des Kreises, z. B. mittels einer Beschriftung und einer Prozentangabe. Hierzu ein Beispiel:

```

<?xml version="1.0"?>
<protocol type="values">
  <sample time="316364400">
    <value datatype="double">12.5</value>
    <value datatype="String">Sektor 1</value>
  </sample>
  <sample time="316364413">
    <value datatype="double">87.5</value>
    <value datatype="String">Sektor 2</value>
  </sample>
</protocol>

```

Das Applet kann nach dem Empfang eines Wertes beliebig viele weitere Anfragen stellen.

5. Wird das Applet beendet, schickt dieses eine Meta-Nachricht mit der Information *end* an den Datensammler und beendet aktiv die Verbindung. Der Datensammler kann das Plugin daraufhin deaktivieren, wenn nicht noch von anderen Applets darauf zugegriffen wird.

Die formale Definition der drei Protokolltypen *init*, *meta* und *value* im XML-Schema-Format befindet sich im Anhang 8.1. Dort sind auch die jeweiligen Restriktionen angegeben, was z. B. die Länge von Zeichenketten oder gültige Wertebereiche betrifft.

5.4 Testen der Darstellungskomponente

Einzelne Methoden und Klassen wurden während und nach der Implementierung getestet. Besonders ist hervorzuheben, dass die Darstellungskomponente an sich schwer zu testen ist: Das Web-Framework, aus dem die Applets aufgerufen werden [JUN-06], und die Cluster-Komponente, die Werte hätte liefern können [KLE-06], wurden zeitgleich zur Darstellungskomponente umgesetzt. Aus diesem Grund wurden die beiden Komponenten simuliert und ein Demonstrationsmodus wurde entwickelt. Durch die Entwicklungsumgebung, mit der es möglich war, den Aufruf der Applets aus einem Browser inklusive der dazugehörigen Parameter durchzuführen, konnte das Framework simuliert werden. Für den Datensammler, also den Teil der Cluster-Komponente, mit dem die Applets kommunizieren, wurde eigens ein Server implementiert, der Zufallswerte erzeugt und diese über die definierte Schnittstelle (siehe Kapitel 4.2.2) im XML-Format zur Verfügung stellt.

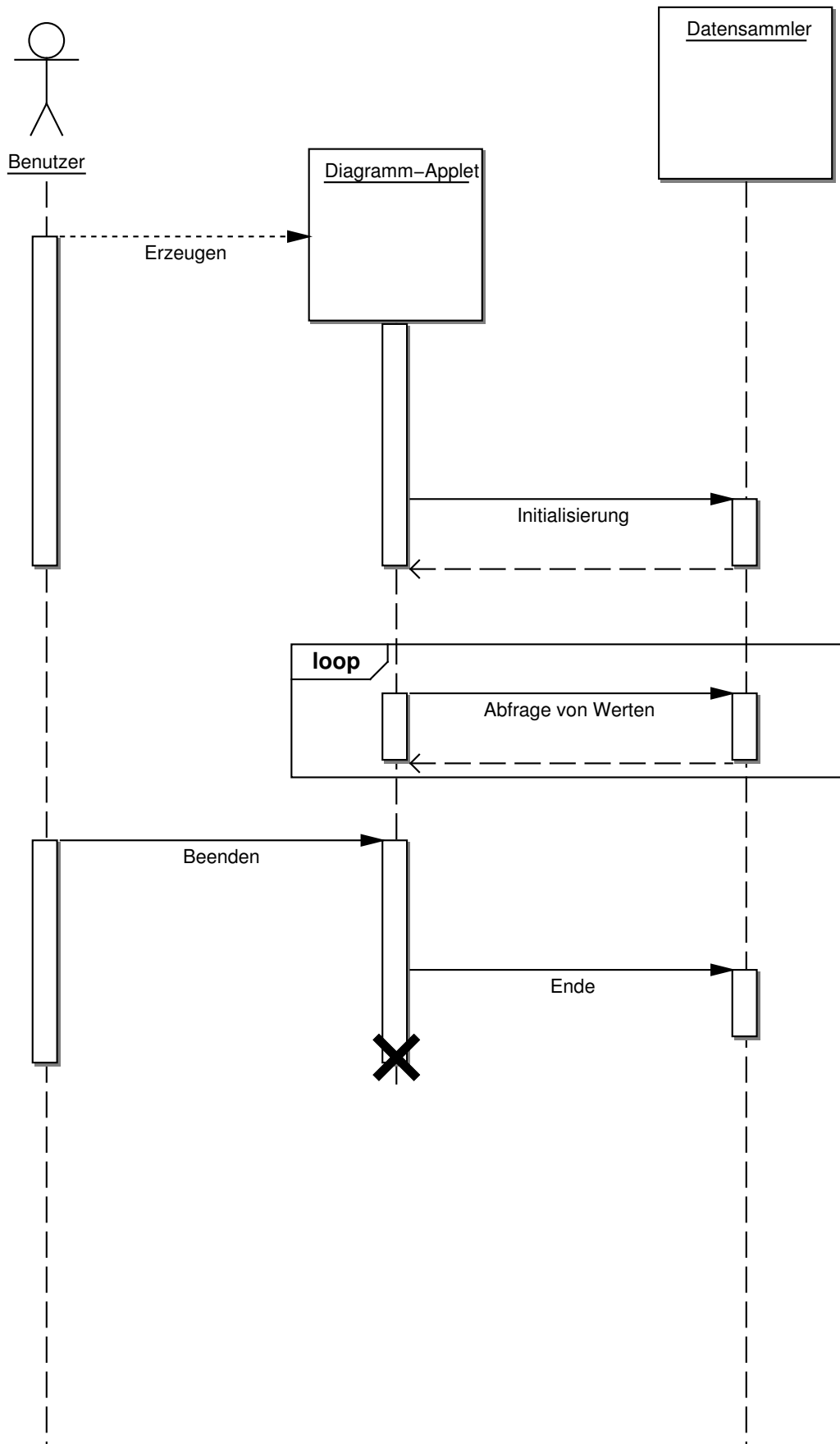


Abbildung 10: Sequenzdiagramm (UML 2)

6 Fazit und Ausblick

In diesem Kapitel wird die Arbeit zunächst zusammengefasst und Erreichtes betrachtet. Dazu werden die Ziele und Anforderungen aus Kapitel 1 aufgegriffen und ein Fazit gezogen. Zum Schluss werden Vorschläge für Erweiterungen implementierter Diagrammtypen sowie Vorschläge für die Entwicklung weiterer Diagrammtypen gegeben.

6.1 Validierung der Anforderungen und Fazit

Ziel der Arbeit war es, eine Bibliothek bereitzustellen, die es ermöglicht, zeitabhängiges Zahlenmaterial geeignet im Browser darzustellen. Dieses Ziel wurde erreicht. Dazu wurde auf die Technologie der Java-Applets in Verbindung mit Java 2D zurückgegriffen, mittels derer eine textuelle Darstellung und die Diagrammtypen Liniendiagramm, Multiliniendiagramm und Bargraph implementiert wurden. Die Architektur eines Kreissektorendiagramms wurde kurz erläutert, das Protokoll und die Schnittstellen wurden so flexibel gehalten, dass auch dieser Diagrammtyp leicht nachträglich implementiert werden kann. Damit sind geeignete Darstellungsformen für alle in Kapitel 2.2 vorgestellten Analyseziele entworfen worden.

Die angestrebte Aktualisierungsrate von 250 Millisekunden wurde durch die Applets erreicht. Damit stellt diese Bibliothek eine deutlich effektivere Möglichkeit zur Verfügung, dynamische Werte auf einer Webseite anzuzeigen, als dies durch eine Server-seitige Generierung von Bildern möglich wäre. Die Anforderung, die Grafiken voll dynamisch zu gestalten, wurde erfüllt.

Alle Schnittstellen basieren auf der Extensible Markup Language. Damit entsprechen sie einem zukunftssicheren Standard, es kann auch von anderen Anwendungen aus auf sie zugegriffen werden. Das Protokoll wurde so flexibel gehalten, dass der Entwicklung von zusätzlichen Diagrammtypen und Plugins nichts im Wege steht. Auch der vollständige Austausch einer Komponente wird dadurch ermöglicht.

Abschließend lässt sich sagen, dass alle Ziele und Anforderungen erfüllt wurden. Die Cluster-Komponente ist noch nicht fertiggestellt, sodass die endgültige Zusammenführung aller Komponenten zum Cluster-Monitor noch nicht durchgeführt werden konnte. Die Darstellungskomponente des Cluster-Monitors lässt sich über die definierte Schnittstelle jedoch problemlos aus dem Web-Framework aufrufen und funktioniert im Demonstrationsmodus mit dem implementierten Testserver.

6.2 Mögliche Erweiterungen und Ausblick

Alle Schnittstellen wurden so flexibel gestaltet, dass neben den Liniendiagrammen und dem Bargraph einfach weitere Darstellungsarten implementiert werden können. Denkbar wären hier z. B. die Implementierung eines Kreissektorendiagramms zur Darstellung von Gliederungsinformationen oder jene eines Zeigerdiagramms als alternative Darstellungsform zum Bargraph.

Ein weiterer Ansatz für mögliche Erweiterungen wäre die Beschriftung von Skalen. Beispielsweise bei den Liniendiagrammen oder beim Bargraph bestünde die Möglichkeit, die Skalen detaillierter zu berechnen. Dazu könnte ein Algorithmus entwickelt werden, der (abhängig von dem zur Verfügung stehenden Platz in Pixeln) entsprechende Zwischenwerte berechnet und diese, gegebenenfalls sogar mit Hilfslinien, in den Diagrammen anzeigt.

7 Quellen- und Literaturverzeichnis

- [ADO-06] Adobe Systems Incorporated: Flash Professional 8, <http://www.adobe.com/products/flash/flashpro/>, 2006
- [AUD-03] Austin, D.: Mathematical Graphics: Introduction to Java, <http://merganser.math.gvsu.edu/david/reed03/notes/chap4.pdf>, 2003
- [BER-05] Berrendorf, R.: Verteilte und parallele Systeme 2: Java Threads, 2005
- [BER-06] Berkeley University: Ganglia:: UC Berkeley Grid Report, <http://monitor.millennium.berkeley.edu/>, 2006
- [COY-02] Coy, C.; Bormann, C.: Java & Webapplikationen, SPC TEIA Lehrbuch Verlag, Berlin 2002
- [FAC-0] Fachhochschule Bonn-Rhein-Sieg: Platform for Scientific Computing at the University of Applied Sciences Bonn-Rhein-Sieg, <http://wr0.wr.inf.fh-bonn-rhein-sieg.de/wr/index.html>, Zugriff: 2006
- [FAC-0]a] Fachhochschule Bonn-Rhein-Sieg: WR Cluster Hardware, <http://wr0.wr.inf.fh-bonn-rhein-sieg.de/wr/hardware/hardware.html>, Zugriff: 2006
- [FOR-05] Forschungszentrum Jülich: LLVIEW: graphical monitoring of LoadLeveler controlled cluster, <http://www.fz-juelich.de/zam/llview/>, 2005
- [GAL-06] Galstad, E.: About Nagios, <http://www.nagios.org/about/>, 2006
- [GAM-97] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns, Addison-Wesley Professional, 1997
- [GAN-06] Ganglia: Ganglia Monitoring System, <http://ganglia.sourceforge.net/>, 2006
- [JUN-06] Jung, M.: Web Framework zur Steuerung eines Cluster-Monitoringsystems, Fachhochschule Bonn-Rhein-Sieg, St. Augustin 2006
- [KLE-06] Kleinhans, J.: Verteilte Erfassung von Leistungsdaten für ein web-basiertes Cluster-Monitoringsystem, Fachhochschule Bonn-Rhein-Sieg, St. Augustin 2006
- [LIL-04] Lilley, C.; Jackson, D.: Scalable Vector Graphics (SVG) – About SVG, <http://www.w3.org/Graphics/SVG/About>, 2004
- [MÜL-03] Müller, R.: ActionScript für Programmierer, dpunkt.verlag, Heidelberg 2003
- [NET-06] NETWAYS: Nagios Demosystem, <http://www.netways.de/de/produkte/nagios/demosystem/>, 2006
- [OBJ-06] ObjectPlanet AS, easycharts, <http://www.objectplanet.com/easycharts/>, 2006
- [OBJ-06a] ObjectPlanet AS, easycharts - Plotter Chart With Many Plots, http://www.objectplanet.com/easycharts/examples/plotter_many.html, 2006

- [PEA-03] Pearlman, E.; House, L.: SVG for Web Developers, Pearson Education, New Jersey 2003
- [QUI-06] Quinn-Curtis Inc., QCChart2D for Java Charting Tools, <http://www.quinn-curtis.com/QCChart2DJavaProdPage.htm>, 2006
- [QUI-06b] Quinn-Curtis Inc., QCRTGraph Real-Time Graphics Tools for Java, <http://www.quinn-curtis.com/QCRTGraphJavaProdPage.htm>, 2006
- [REC-05] Reconn.us: SpiceCharts – Free Java Charts, <http://www.reconn.us/java.html>, 2005
- [REC-05a] Reconn.us: SpiceCharts – Beispielanwendung eines 2D-Liniendiagramms, <http://www.reconn.us/download/Line2D.zip>, 2005
- [RIE-75] Riedwyl, H.: Grafische Gestaltung von Zahlenmaterial, Haupt Berne 1975
- [RÖS-06] Rösler-Laß, M.: Projekt VIOLA, <http://www.viola-testbed.de/>, 2006
- [SUN-01] Sun Microsystems: Programmer's Guide to the Java 2D™ API, 2001
- [SUN-06] Sun Microsystems: Applets, <http://java.sun.com/applets/>, 2006
- [SWI-0J] Swiftchart Limited: Swiftchart to create your own chart and graph, <http://www.swiftchart.com/index.htm>, Zugriff: 2006
- [SWI-0Ja] Swiftchart Limited: Swiftchart - Chart and graph examples, http://www.swiftchart.com/example_2.htm, Zugriff: 2006
- [TAN-03] Tanenbaum, A.; van Steen, M.: Verteilte Systeme, Pearson Studium, München 2003
- [ULL-05] Ullenboom, C.: Java ist auch eine Insel, Galileo Press, 2005
- [W10-06] w100w: Graphs and Charts, http://w100w.com/english/java/applets/graphs_and_charts/, 2006

8 Anhang

8.1 Formale Definitionen der Schnittstellen

Schnittstelle: Darstellung – Framework (HTTP-Code und Parameter)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Definition simple elements -->
  <xs:element name="classname" type="xs:string" />
  <xs:element name="width" type="xs:positiveInteger" />
  <xs:element name="width_add" type="xs:nonNegativeInteger" />
  <xs:element name="height" type="xs:positiveInteger" />
  <xs:element name="height_add" type="xs:nonNegativeInteger" />
  <xs:element name="default_width" type="xs:positiveInteger" />
  <xs:element name="default_height" type="xs:positiveInteger" />

  <!-- Definition Attribute -->
  <xs:attribute name="type" type="xs:string" />
  <xs:attribute name="name" type="xs:string" />
  <xs:attribute name="description" type="xs:string" />

  <!-- Definition complex elements -->
  <xs:element name="parameter">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="description" type="xs:string"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="diagram">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="classname" />
        <xs:element ref="width" maxOccurs="unbounded" />
        <xs:element ref="width_add" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="default_width" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="height" maxOccurs="unbounded" />
        <xs:element ref="height_add" minOccurs="0" maxOccurs="1" />
        <xs:element ref="default_height" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="parameter" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute ref="type" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="applet">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="diagram" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Schnittstelle: Applet – Datensammler (Initialisierung der Verbindung)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Definition simple elements -->
  <xs:element name="interval" type="xs:positiveInteger" />
  <xs:element name="aggregate">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:maxLength value="4" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="id" type="xs:positiveInteger" />

  <!-- Definition Attribute -->
  <xs:attribute name="type" type="xs:string" fixed="init" />

  <!-- Definition complex elements -->
  <xs:element name="protocol">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="interval" />
        <xs:element ref="aggregate" maxOccurs="unbounded" />
        <xs:element ref="id" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute ref="type" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Schnittstelle: Applet – Datensammler (Übertragung von Metainformationen)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Definition simple elements -->
  <xs:element name="meta">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="ok"/>
        <xs:enumeration value="not_ok"/>
        <xs:enumeration value="get"/>
        <xs:enumeration value="end"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

  <!-- Definition Attribute -->
  <xs:attribute name="type" type="xs:string" fixed="meta" />

  <!-- Definition complex elements -->
  <xs:element name="protocol">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="meta" />
      </xs:sequence>
      <xs:attribute ref="type" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Schnittstelle: Applet – Datensammler (Übertragung von Werten)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Definition Attribute -->
  <xs:attribute name="type" type="xs:string" fixed="values" />
  <xs:attribute name="datatype" type="xs:string" />
  <xs:attribute name="time" type="xs:string" />

  <!-- Definition complex elements -->
  <xs:element name="value">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute ref="datatype" use="required" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="sample">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="value" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute ref="time" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="protocol">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sample" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute ref="type" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

8.2 Vollständige Definition der Schnittstelle zum Framework

```
<?xml version="1.0"?>

<applet>

<diagram type="line">

  <classname>diagramApplets.KMR_LineDiag.class</classname>

  <width>50</width>
  <width>100</width>
  <width>150</width>
  <width>200</width>
  <width>300</width>
  <width>400</width>
  <width>500</width>
  <width>600</width>
  <width_add>50</width_add>
  <default_width>400</default_width>

  <height>10</height>
  <height>20</height>
  <height>50</height>
  <height>100</height>
  <height>200</height>
  <height>300</height>
  <height_add>20</height_add>
  <default_height>100</default_height>
```

```

<parameter name="description"
  description="Beschriftung der Y-Achse" />
<parameter name="aggregate" description="Eine Aggregatsfunktion" />
<parameter name="ids" description="Eindeutige ID(s) fuer ein Plugin" />
<parameter name="interval"
  description="Intervall in Millisekunden bis zur naechsten Abfrage" />
<parameter name="coord_max" description="Maximal darstellbarer Wert" />
<parameter name="show_max"
  description="Ein-/Ausschalten der Hilfslinie fuer den Maximalwert" />
<parameter name="show_min"
  description="Ein-/Ausschalten der Hilfslinie fuer den Minimalwert" />
<parameter name="scalable"
  description="Skalierbarkeit ein- oder ausschalten" />
<parameter name="server_name"
  description="IP-Adresse oder DNS-Name des Datensammlers" />
<parameter name="port"
  description="Port, auf dem der Socket geoeffnet werden soll" />
</diagram>

<diagram type="bargraph">

  <classname>diagramApplets.KMR_BarGraph.class</classname>

  <width>50</width>
  <width>100</width>
  <width>200</width>
  <width>300</width>
  <width>400</width>
  <width>500</width>
  <width>600</width>
  <width_add>80</width_add>
  <default_width>200</default_width>

  <height>41</height>
  <height_add>0</height_add> <!-- Hoehe ist konstant -->
  <default_height>41</default_height>

  <parameter name="description" description="Beschriftung des Balkens" />
  <parameter name="aggregate" description="Eine Aggregatsfunktion" />
  <parameter name="ids" description="Eindeutige ID(s) fuer ein Plugin" />
  <parameter name="interval"
    description="Intervall in Millisekunden bis zur naechsten Abfrage" />
  <parameter name="coord_max" description="Maximal darstellbarer Wert" />
  <parameter name="show_max"
    description="Ein-/Ausschalten der Hilfslinie fuer den Maximalwert" />
  <parameter name="show_min"
    description="Ein-/Ausschalten der Hilfslinie fuer den Minimalwert" />
  <parameter name="scalable"
    description="Skalierbarkeit ein- oder ausschalten" />
  <parameter name="server_name"
    description="IP-Adresse oder DNS-Name des Datensammlers" />
  <parameter name="port"
    description="Port, auf dem der Socket geoeffnet werden soll" />
</diagram>

<diagram type="multiline">

  <classname>diagramApplets.KMR_MultiLineDiag.class</classname>

  <width>50</width>
  <width>100</width>
  <width>150</width>
  <width>200</width>
  <width>300</width>
  <width>400</width>
  <width>500</width>
  <width>600</width>
  <width_add>50</width_add>
  <default_width>400</default_width>

```

```

<height>10</height>
<height>20</height>
<height>50</height>
<height>100</height>
<height>200</height>
<height>300</height>
<height_add>30</height_add>
<default_height>100</default_height>

<parameter name="description"
  description="Beschriftung der Y-Achse" />
<parameter name="aggregate"
  description="Liste von Aggregatsfunktionen" />
<parameter name="ids" description="Eindeutige ID(s) fuer ein Plugin" />
<parameter name="interval"
  description="Intervall in Millisekunden bis zur naechsten Abfrage" />
<parameter name="coord_max" description="Maximal darstellbarer Wert" />
<parameter name="scalable"
  description="Skalierbarkeit ein- oder ausschalten" />
<parameter name="server_name"
  description="IP-Adresse oder DNS-Name des Datensammlers" />
<parameter name="port"
  description="Port, auf dem der Socket geoeffnet werden soll" />

</diagram>

<diagram type="textonly">

  <classname>diagramApplets.KMR_Text.class</classname>

  <width>300</width>
  <width_add>0</width_add> <!-- Breite ist konstant -->
  <default_width>300</default_width>

  <height>14</height>
  <height_add>0</height_add> <!-- Hoehe ist konstant -->
  <default_height>14</default_height>

  <parameter name="description" description="Beschriftung des Wertes" />
  <parameter name="aggregate" description="Eine Aggregatsfunktion" />
  <parameter name="ids" description="Eindeutige ID(s) fuer ein Plugin" />
  <parameter name="interval"
    description="Intervall in Millisekunden bis zur naechsten Abfrage" />
  <parameter name="server_name"
    description="IP-Adresse oder DNS-Name des Datensammlers" />
  <parameter name="port"
    description="Port, auf dem der Socket geoeffnet werden soll" />

</diagram>

</applet>

```

Eidesstattliche Erklärung

Name: Reineck, Karsten
Adresse: Bonner Logsweg 15
53123 Bonn

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

St. Augustin, 4. August 2006

Ort, Datum

Unterschrift