VisBenchmark
Eine Experimentier- und Visualisierungsumgebung für Benchmark-Probleme

Angelika Blauth

Inhaltsverzeichnis:

Literatur

---

1. Einleitung und Aufgabenstellung

In der Realität geschieht es häufig, daß Auswirkungen von Modifikationen an Algorithmen nicht mehr analytisch erfaßt werden können, sondern nur durch das Testen einer großen Anzahl von Benchmarkproblemen gemessen werden.

So wird z.B. im Bereich der automatischen Deduktion die einige Tausend Probleme umfassende TPTP-Benchmark-Sammlung eingesetzt, um verschiedene Theorembeweiser zu vergleichen und um den Nutzen von vorgeschlagenen Verbesserungen empirisch zu untersuchen.

Da dabei aber meist eine große Menge von Meßwerten anfällt, ist es nötig, die Daten geeignet zu visualisieren, um die vorliegenden Ergebnisse korrekt auswerten zu können.

Aufgabe dieses Fortgeschrittenenpraktikums war es also, eine geeignete Experimentier- und Visualisierungs-Umgebung zu konzipieren und zu implementieren, mit der komfortabel diese Problemstellung bearbeitet werden kann. Weiterhin sollte die Umgebung universell einsetzbar, portabel und frei konfigurierbar sein.

Die einzelnen Aufgabenstellungen wurden folgendermaßen gelöst:

- Die Visualisierung der einzelnen Daten erfolgt über ein Streudiagramm (engl. scatterplot), welches häufig bei statistischen Problemstellungen eingesetzt wird. Mit Hilfe eines Streudiagramms ist es sehr einfach möglich, die Meßdaten von zwei Verfahren zu vergleichen, die (fast) die gleichen Probleme bearbeitet haben.

- Da ein solches Streudiagramm aber nur statisch ist, ist diese Lösung für eine Experimentierumgebung noch nicht ausreichend. So will man vielleicht bei einigen Punkten wissen, welche Daten dahinter stecken (z.B. bei Ausreißern) oder auch einen Ausschnitt des Diagramms in der um einiges genaueren Tabellendarstellung betrachten. Deshalb sind Interaktionsmöglichkeiten im Programm eingebaut, mit denen der Benutzer (meist über Maus-Klicks) einzelne Daten in einer anderen Form betrachten kann (s. 2. Benutzerhandbuch).

- Um das Programm universell einsetzen zu können, werden alle benötigten Daten aus URLs ausgelesen. Dies hat den Vorteil, daß sowohl Daten aus Textdateien und aus Datenbanken über eine Schnittstelle zu erreichen sind und zudem diese Daten auf ganz unterschiedlichen Rechnern liegen können. Zusammen mit der Möglichkeit, das Programm individuell zu konfigurieren, kann man die Umgebung auch für ganz andere Problemstellungen heranziehen. Einziges Kriterium dabei ist, daß mindestens zwei numerische Meßreihen vorliegen (näheres dazu s. 2.2 Konfigurieren des Programms).

- Das Portabilitätskriterium wurde dadurch erreicht, indem die Umgebung in Java programmiert wurde. Es ist dadurch möglich, das Programm als selbständige Applikation auf nahezu jeder Platform oder als sogenanntes "Applet" aus einem java-fähigen WWW-Browser zu starten.

All diese Kriterien ermöglichen das Szenario einer weltweiten Anwendung. So kann z.B. eine Deduktions-Forschungsgruppe, welche an verschiedenen Orten arbeitet, ihre Performance-Meßwerte zu den TPTP-Problemen in einem einheitlichen Format über eine URL zur Verfügung stellen.Es besteht dann die Möglichkeit, mit Hilfe von VisBenchmark diese Daten, welche auf unterschiedlichen Rechnern liegen können, miteinander zu vergleichen.

Am Ende dieser Einleitung soll noch kurz darauf hingewiesen werden, wozu das Programm nicht dienen sollte: Es sollte eine Experimentierumgebung und kein Präsentationstool werden, mit dessen Hilfe die Plots z.B. (in verschiedenen Formaten) schön ausgedruckt werden können oder ein Programm, das eine möglichst genaue Darstellung der Punkte im Streudiagramm ermöglicht. Diese Aufgaben erledigen schon andere Programme in sehr guter Qualität (wie z.B. gnuplot oder PowerPoint).

---

2. Benutzerhandbuch

Dieses Benutzerhandbuch soll mit dem Programm vertraut machen. Zuerst werden die einzelnen Programmkomponenten besprochen; dann wird (durch ein fortlaufendes Beispiel unterstützt) gezeigt, wie das Programm individuell konfiguriert werden kann.

2.1 Arbeiten mit VisBenchmark

2.1.1 Installation

Es gibt zwei Möglichkeiten, das Programm aufzurufen:

1. **Als Applet.** In dieser Version kann die Umgebung einfach mit Hilfe eines Java-fähigen WWW-Browsers (z.B. Netscape 3.0 oder Microsoft Internet Explorer) von der WWW-Seite http://www.pms.informatik.uni-muenchen.de/~blauth gestartet werden.

2. **Als Applikation.** Dazu muß der Programmcode visBenchmark.zip bzw. visBenchmark.tar.gz in ein Verzeichnis kopiert und entpackt werden. Außerdem muß auf dem eigenen Rechner das JDK Version 1.0 installiert sein. Das Programm wird dann mit dem Kommando

   visBenchmark

   aufgerufen. (VisBenchmark ist eine Batch-Datei, welche die Umgebung mit den richtigen Parametern startet).

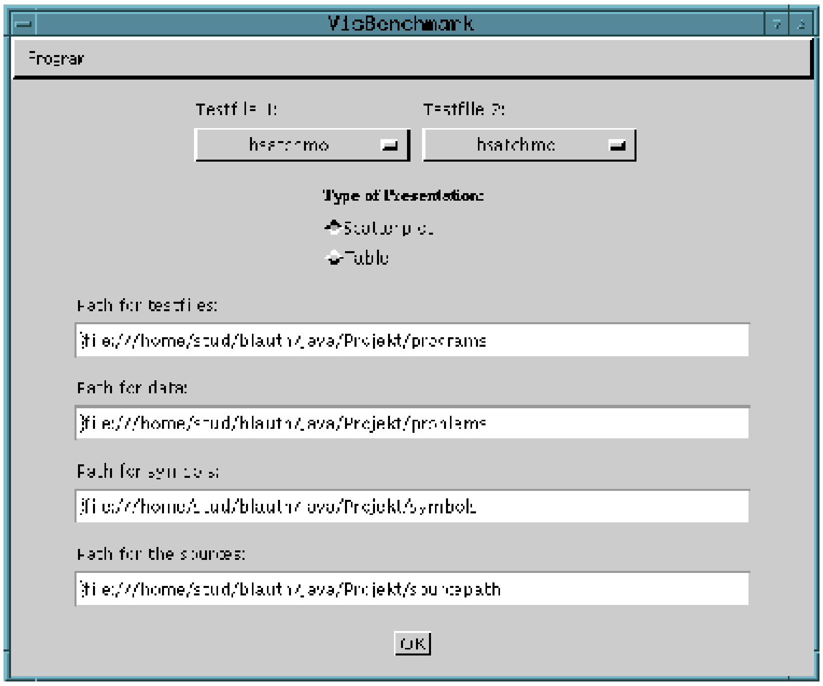
In beiden Fällen wird die Umgebung mit den Standardeinstellungen, d.h. als Experimentierumgebung für die TPTP-Benchmarksammlung aufgerufen. Sollen andere Probleme damit bearbeitet werden, muß das Programm umkonfiguriert werden (s. 2.2 Konfigurieren des Programms).

Es folgen nun noch einige Hinweise die bei der Entscheidung helfen sollen, welche Programmversion man aufrufen sollte.

Will man das Programm einfach nur kurz testen, ist sicherlich die Applet-Version zu bevorzugen, da hierbei kein Installationsaufwand vorhanden ist. Außerdem ist diese Version natürlich zu bevorzugen, wenn man keine Möglichkeit hat, Java auf dem eigenen Rechensystem zu installieren.

Ansonsten sollte man die Applikations-Version installieren, da sie um einiges robuster ist. Sie muß auf jeden Fall aufgerufen werden, wenn man das Programm von einem Rechner aus starten will, der keine Internet-Verbindung hat. Denn nur in dieser Version ist es möglich, Konfigurationsdateien mit lokalen URLs (*file:// ...*) anzugeben. Im Applet ist dies wegen den Sicherheitsvorkehrungen der WWW-Browser nicht erlaubt.

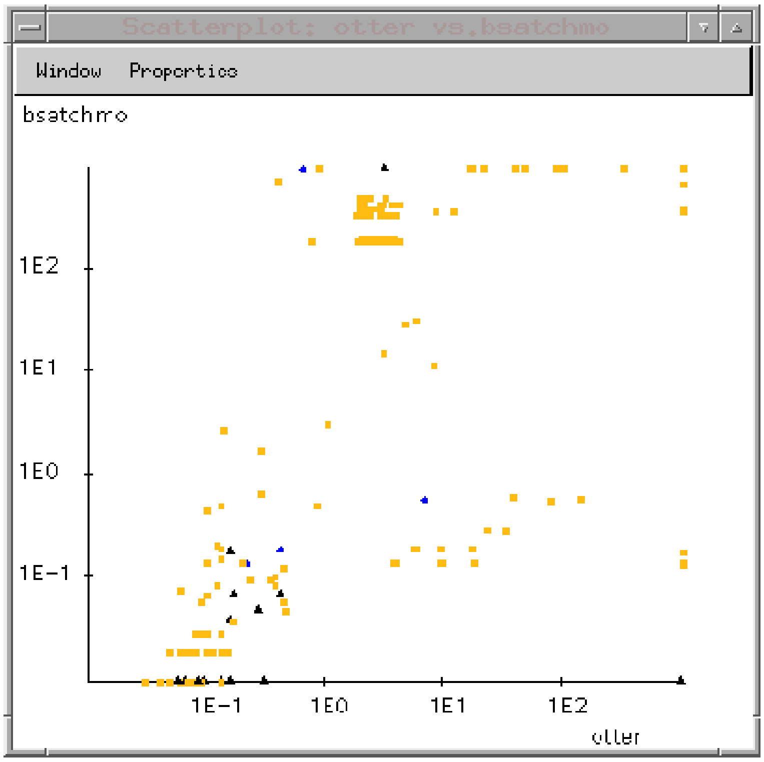
2.1.2 Haupt-Fenster



Im Hauptfenster, das beim Start des Programms erscheint, können folgende Einstellungen vorgenommen werden:

- Über eine Auswahlliste werden die beiden Verfahren gewählt, welche miteinander verglichen werden sollen.
- Als Darstellungsart kann Streudiagramm (scatterplot) oder Tabelle (table) gewählt werden.
- Die Pfadangaben für die Konfigurationsdateien können neu gesetzt werden. Diese Einstellungen sind aber nur für den aktuellen Programmlauf gültig. Will man die Pfade permanent anders setzen, müssen diese in der Start-HTML-Seite (Appletversion) oder in der Batch-Datei (Applikations-Version) angegeben werden. Eine nähere Beschreibung dieser Einstellungen findet man unter 2.2 Konfigurieren des Programms. Die Pfadangaben müssen gültige URL-Adressen sein, wobei in der Applet-Version wiederum nur HTTP-Adressen (http:// ...) erlaubt sind.
- Durch Klick auf "OK" wird der Scatterplot oder die Tabelle mit den aktuellen Werten angezeigt.

2.1.3 Arbeiten mit dem Scatterplot

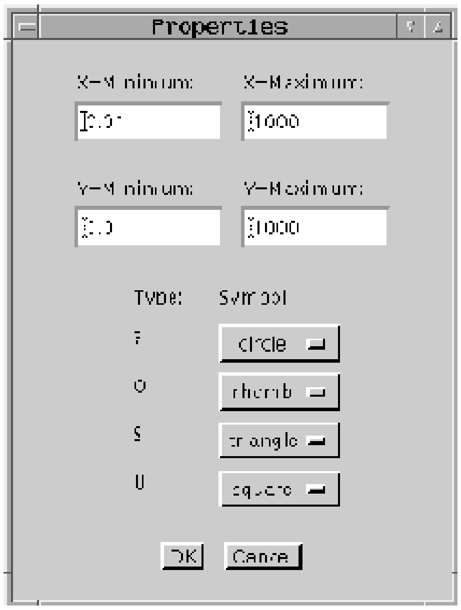


Der Scatterplot ist folgendermaßen aufgebaut: An jeder Achse wird ein zu testendes Verfahren im logarithmischen Maßstab aufgetragen. Die Daten werden an den Schnittpunkten angezeigt, wobei nur solche berücksichtigt werden, die von beiden Verfahren bearbeitet wurden. Dabei werden für verschiedene Klassen von Daten eigene Symbole angezeigt (näheres dazu s. 2.2 Konfigurieren des Programms). Wird der Scatterplot vom Haupt-Fenster aus aufgerufen, wird automatisch die geeignete Größe berechnet, um alle Daten anzeigen zu können.

Neben den allgemeinen Möglichkeiten ein Fenster zu verändern (wie z.B. Ändern der Größe, Minimieren des Fensters, usw.), bestehen folgende Interaktionsmöglichkeiten im Scatterplot:

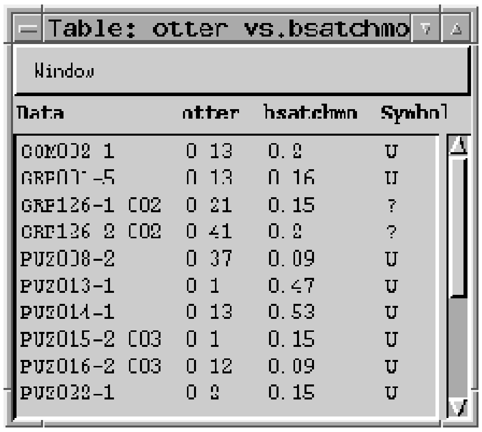
- Durch Ziehen der Maus mit gedrückter linker Maustaste kann ein Bereich des Scatterplots ausgewählt werden.
    - Beim Loslassen der Maustaste erscheinen die Daten, die im ausgewählten Bereich liegen, in der Tabellendarstellung (s. 2.1.4. Arbeiten mit der Tabelle).
    - Wird beim Loslassen der Maus gleichzeitig die Shift-Taste gedrückt, erscheint der ausgewählte Ausschnitt in einem neuen Scatterplot-Fenster. Dadurch wird ein (nicht-stufenloses) Zoomen ermöglicht. Die etwas komplizierte Methode (eigentlich war dafür das Ziehen mit der rechten Maustaste gedacht) rührt daher, daß bei Java wegen der Portabilität der Programme nur eine Maustaste unterstützt wird.

Mit Hilfe des Menüpunktes "Properties - Change Properties" können die Eigenschaften des aktuellen Scatterplots verändert werden:



- Die Bereichsgrenzen der Achsen können neu angegeben werden. Dadurch kann man einen bestimmten Ausschnitt vergrößert darstellen.
- Zudem kann jeder Daten-Klasse ein neues Symbol zugeordnet werden.
- Durch Klicken auf "OK" treten die Änderungen in Kraft, durch "Cancel" wird das Fenster geschlossen, ohne die vorgenommenen Einstellungen zu berücksichtigen.

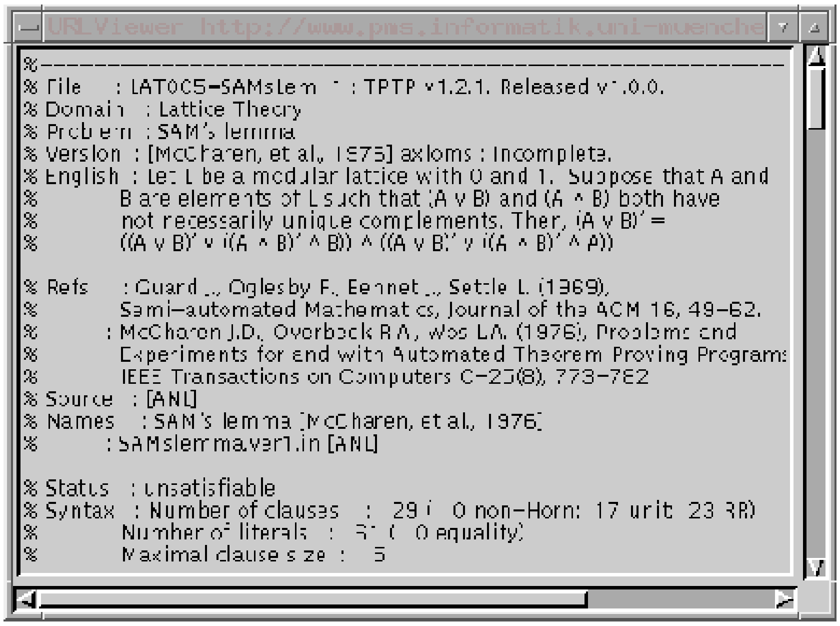
2.1.4 Arbeiten mit der Tabelle



Die Tabelle ist folgendermaßen aufgebaut: In der ersten Spalte stehen die Namen der Daten, gefolgt von den Meßwerten dieser Daten bei den beiden Verfahren. In der letzten Spalte schließlich ist noch die Daten-Klasse angegeben.

Wird die Tabelle vom Haupt-Fenster aufgerufen, werden alle Daten angezeigt, die zumindest von einem Verfahren bearbeitet wurden. Wurde ein Datum nicht behandelt, ist dies mit "--" gekennzeichnet.

Durch Anklicken einer Tabellenzeile erscheinen nähere Informationen zu diesem Datum in einem neuen Fenster.



2.2 Konfigurieren des Programms

Es ist möglich, das Programm individuell zu konfigurieren und somit an vielfältige Problemstellungen anzupassen. Im folgenden wird der Aufbau der vier Konfigurationsdateien, der einzelnen Zeitreihen-Dateien und das Einbinden dieser in das Programm erläutert. Ein fortlaufendes Beispiel soll dabei die auszuführenden Schritte verdeutlichen.

Die Problemstellung für das Beispiel lautet folgendermaßen: *Es soll getestet werden, ob ein Zusammenhang zwischen den einzelnen Ergebnissen der Informatik I - und Informatik II- Klausuren besteht. Dies zu prüfen ist möglich, da die Zusammensetzung der Teilnehmer beider Klausuren ähnlich ist.*

2.2.1 Aufbau der Konfigurationsdateien

Die vier Konfigurationsdateien sind bis auf eine Ausnahme alle nach dem gleichen Schema aufgebaut: Jede Zeile besteht aus zwei Spalten, welche untereinander durch Tabulatoren (TAB-Taste) getrennt sind. Die Namen und Pfadangaben der Konfigurationsdateien sind frei wählbar.

1. **Verfahren** (Testfiles): In dieser Datei sind die zu untersuchenden Verfahren und die URLs zu ihren Zeitreihen-Dateien (s.2.2.2 Aufbau der Zeitreihen-Dateien) angegeben. Der Aufbau der Datei lautet folgendermaßen:

```
<Verfahren1>       <URL1>
<Verfahren2>       <URL2>
     ...            ...
```

Im Beispiel würde die Datei also folgendermaßen aussehen:

```
Info I   http://www.pms.informatik.uni-muenchen.de/~blauth/example/InfoI
Info II  http://www.pms.informatik.uni-muenchen.de/~blauth/example/InfoII
```

2. **Daten** (Data): Darin steht die Liste aller Daten, die von zumindest einem Verfahren bearbeitet wurden, zusammen mit der Daten-Klasse. Die Klasse hilft dabei, die einzelnen Daten in verschiedene Kategorien aufzuteilen und mit jeweils anderen Symbolen anzuzeigen. Der Aufbau der Datei lautet also folgendermaßen:

```
<Data1>    <Klasse1>
<Data2>    <Klasse2>
  ...          ...
```

Im Beispiel besteht diese Datei also aus den Matrikelnummern und den Semesterangaben der einzelnen Studenten.

```
150374400376    1
220770387263    5
080769332656    1
141271462180    3
    usw.
```

3. **Symbole** (Symbols): Diese Datei enthält die Symbole zu den einzelnen Daten-Klassen. Die Daten der jeweiligen Klassen werden dann im Scatterplot mit den hier angegeben Symbolen aufgetragen.

   Als Symbole stehen zur Verfügung: **square** (Quadrat), **circle** (Kreis), **triangle** (Dreieck) und **rhomb** (Raute).
   Der allgemeine Aufbau der Datei lautet:

```
<Klasse1>    <Symbol>
<Klasse2>    <Symbol>
 ...            ...
```

Im Beispiel:

```
1       square
3       triangle
5       circle
```

4. **Quelle** (Sources): Diese Konfigurationsdatei bildet die oben erwähnte Ausnahme. Sie besteht aus genau zwei Zeilen mit je einer Spalte. Hier werden die Pfadangaben der Quell-Dateien zu den einzelnen Daten mit Hilfe von regulären Ausdrücken in der Perl-Syntax angegeben. In den Quell-Dateien sind dabei nähere Angaben zu den Daten enthalten.

   In der ersten Zeile steht dabei der reguläre Ausdruck, mit welchem gematcht werden soll. Der zu matchende String ist dabei implizit immer der jeweilige Name des Datums (wie z.B. ein TPTP-Problem, oder die Matrikelnummer). Dabei muß dieser Ausdruck, oder auch Teile davon in Klammern eingeschlossen werden.

   In der zweiten Zeile ist dann ein String anzugeben, der teilweise durch den gematchten String ersetzt wird. Hier kann man mittels \0, \1, ... die in Klammern gesetzten Ausdrücke der ersten

Zeile referenzieren (\0 steht für den 1.Ausdruck, \1 für den 2., usw.).
Soll in diesem String das Zeichen \ selbst angegeben werden, ist ihm ein zweites \ voranzustellen (\\).

Die Pfadangaben der Quell-Dateien in der vorgestellten Weise anzugeben, scheint auf den ersten Blick recht kompliziert zu sein. Denn es hätte theoretisch eine Datei gereicht, in der jedes Problem mit dem dazugehörigen Quell-Pfad angegeben ist.
Diese Art der Darstellung hätte aber einige Nachteile mit sich gebracht.

Die Pfadangaben sind größtenteils redundant, da sie sich meist nur in wenigen Verzeichnissen unterscheiden. Will man aber den gemeinsamen Pfadanteil der Daten ändern, muß man dazu die gesamte Liste durchgehen.

Außerdem müßten bei diesem Lösungsansatz die einzelnen Pfade als Strings komplett in das Programm eingelesen werden, was zu längeren Wartezeiten geführt hätte (die Pfadangaben sind meist relativ lang).

Im Beispiel sind die Quell-Daten im Verzeichnis

```
http://www.pms.informatik.uni-muenchen.de/~blauth/example
```

unter den jeweiligen Matrikelnummern gespeichert. Darin sind Angaben wie z.B. Name des Studenten, erreichte Punkte bei den einzelnen Aufgaben oder ähnliches enthalten.
Somit würde die Datei folgendermaßen aussehen:

```
([0-9]+)
http://www.pms.informatik.uni-muenchen.de/~blauth/example/\0
```

2.2.2 Aufbau der Zeitreihen-Dateien

Die Zeitreihen-Dateien sind genauso aufgebaut wie der Großteil der Konfigurationsdateien: Es sind in jeder Zeile zwei Spalten enthalten, die durch Tabulator (TAB) voneinander getrennt sind. In der ersten Spalte ist dabei der Datenname anzugeben und in der zweiten Spalte der dazugehörige numerische Wert.

```
<Data1>      <Zahl1>
<Data2>      <Zahl2>
  ...           ...
```

Dabei ist zu beachten, daß diese Dateien an der Adresse stehen müssen, welche in der Datei Verfahren angegeben ist und daß die Datennamen mit denen in der Datei Data übereinstimmen.

Im Beispiel könnte die Datei zur InfoI-Klausur folgendermaßen aussehen:

```
150374400376    3
220770387263    1.7
080769332656    5
  usw.
```

Diese Datei muß an der angegeben Adresse, also unter

```
http://www.pms.informatik.uni-muenchen.de/~blauth/example/InfoI
```

zu finden sein.

2.2.3 Einbindung in das Programm

Die Konfigurationsdateien können auf zwei Arten in das Programm eingebunden werden:

1. Um sie bei **einem** Programmlauf zu verwenden, können die Pfadangaben der einzelnen Konfigurationsdateien im Hauptfenster gesetzt werden (s. 2.1.2 Haupt-Fenster).

2. Um sie permanent ins Programm einzubinden, ist folgendes zu tun:

    - **Applikation**: Die URLs der Konfigurationsdateien müssen im Batchfile *visBenchmark* angegeben werden, und zwar in folgender Reihenfolge:
      1.Verfahren, 2. Daten, 3. Symbole, 4. Quellen.

    - **Applet**: Die URLs müssen hier in der HTML-Seite angegeben werden, die das Programm startet (VisBenchmark.html). Sie sind in der selben Reihenfolge anzugeben wie in der Applikationsversion des Programmes. Dazu muß diese Datei auf den eigenen Rechner kopiert und aufgerufen werden, da ihr Inhalt sonst nicht geändert werden kann.

---

3. Entwurf und Implementierung

Dieses Kapitel soll einige interessante Aspekte der Programmentwicklung darstellen. So wird die Methode der objektorientierten Entwicklung, mit deren Hilfe das Programm entworfen wurde, nämlich die Booch-Methode, kurz vorgestellt. Es wird dabei davon ausgegangen, daß der Leser mit den wichtigsten Begriffen der Objektorientierung vertraut ist. Zudem wird die dadurch erhaltene Klassenstruktur erläutert und es wird kurz auf die Programmierung unter Java eingegangen.

3.1 Kurzer Überblick über die Booch-Methode

Die Beschreibung der Booch-Methode folgt im wesentlichen der Darstellung in [1]. Da die Methode relativ komplex ist, wird hier nur auf die für die Entwicklung des Programms wichtigsten Schritte eingegangen. Wer sich näher damit beschäftigen möchte, sei auf [2] verwiesen.

Booch beschreibt die Programmentwicklung als sicheren Pfad vom Problem zum objektorientierten Programm. Um dies zu erreichen, gibt es einen Rahmen, der *Makroprozeß* genannt wird. Dieser setzt sich aus den Phasen *Konzeption*, *Analyse*, *Design*, *Implementierung* und *Wartung* zusammen. In jeder dieser Phasen finden mehrere *Mikroprozesse* statt, welche die tägliche Arbeit des Programmierers darstellen. Zur einheitlichen Darstellung der Entwurfsarbeit gibt es unterschiedliche Diagramme.

Im folgenden soll die Analysephase näher untersucht werden.

Am Anfang der Analysephase soll nur eine grobe Klassenstruktur erstellt werden. Um wichtige Klassen zu finden, kann man z.B. nach der *Pflichtenheftanalyse* vorgehen. Dabei durchsucht man die Anwendung nach greifbaren Dingen (in Visbenchmark wurden hierbei z.B. die Klassen Scatterplot und

Table gefunden). Die erhaltenen Klassen stellt man dann in einem *Klassendiagramm* zusammen.

Als nächstes gilt es Verbindungen zwischen den Klassen herzustellen und das Klassendiagramm zu konkretisieren. Die Verbindungen können hierbei verschiedenster Art sein, wobei es für jeden Typ im Diagramm eine Darstellungart gibt. So gibt es die Vererbungsbeziehung (*kind-of*) und die Aggregation (*part-of*). Bei der Aggregation unterscheidet man, ob eine Klasse als Wert oder als Referenz in der anderen Klasse enthalten ist. Eine Referenz auf eine Klasse ist dann nötig, wenn mehrere Objekte auf eine Instanz zugreifen müssen. Zudem gibt es noch die Verwendungs-Beziehung (*link*), die besteht, wenn eine Klasse in einer Methode der anderen Klasse als Über- oder Rückgabeparameter eingesetzt wird.

Im *Zustandsdiagramm* wird untersucht, in welche Zustände eine Klasseninstanz versetzt werden kann, und welche Ereignisse dafür nötig sind. Diese können dann durch Methoden oder Attribute realisiert werden.

Im *Objektdiagramm* treten dann die Instanzen in Aktion: Hier wird der Ablauf einzelner Aktionen (wie z.B. der Weg, der nötig ist, einen Scatterplot anzuzeigen) durchdacht. Wichtig ist hierbei, daß die Objekte zusammenarbeiten und so den Fluß des Systems erhalten.

Es ist wichtig zu erwähnen, daß die einzelnen Diagramme nicht in einer bestimmten Reihenfolge bearbeitet werden müssen, sondern durcheinander betrachtet werden. So findet man z.B. im Objektdiagramm wieder neue Akteure, welche dann im Klassendiagramm wieder genauer zu betrachten sind, usw.

Im Kern besteht die Methode also darin, von einer relativ hohen Abstraktionsebene durch wiederholte Ausführung der einzelnen Schritte zu einer niedrigen Abstraktionsebene zu gelangen, welche dann eine einfache Implementierung erlauben soll.

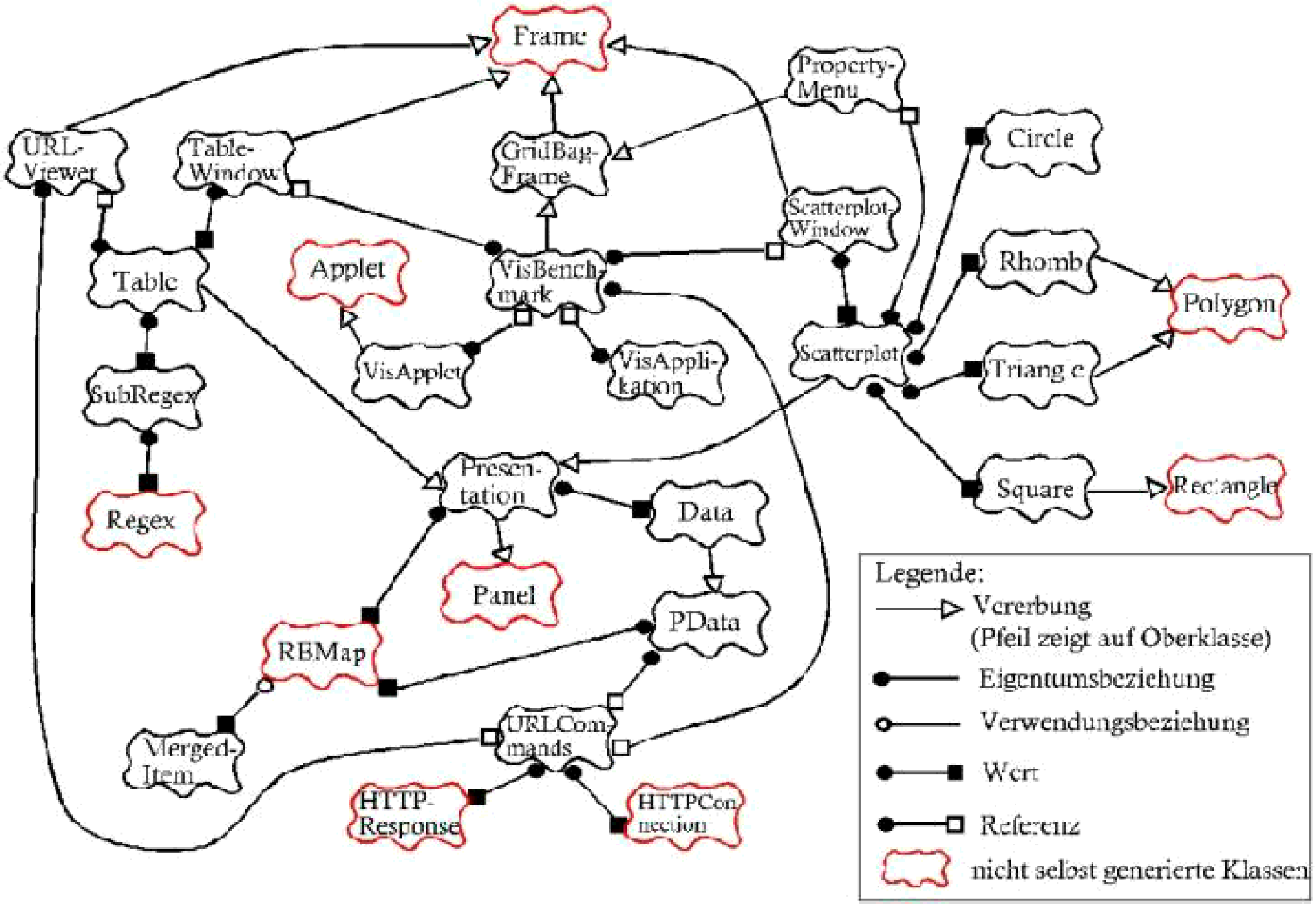
3.2 Klassenstruktur von VisBechmark

Das Programm wurde mit Hilfe der Booch-Methode entwickelt, wobei diese aber nicht zu jedem Schritt herangezogen wurde.
So wurden die oben erläuterten Schritte zwar alle betrachtet, aber es wurden z.B. nicht alle Abläufe durch Objektdiagramme untersucht, sondern nur wenige wichtige. Durch dieses Vorgehen wurden Klassen, wie Scatterplot, Table, Presentation, PData und Data gefunden. Außerdem half die Analyse von Abläufen dabei, die Struktur des Programms zu durchleuchten. Die Entwicklung des Programms dagegen komplett mit der Booch-Methode zu erledigen, hätte meiner Meinung nach zu einem Overhead geführt, da VisBenchmark doch ein relativ leicht zu überschauendes Projekt ist.

Es wurde aber viel Wert darauf gelegt, daß einzelne Klassen auch in anderen Programmen leicht zu verwenden sind.

Da es in der objektorientierten Programmierung guter Stil ist, "das Rad nicht neu zu erfinden", sind im Programm einige Klassenstrukturen (*packages*) von anderen Entwicklern enthalten, nämlich collections, ein Paket zur effizienten Speicherung von Datenstrukturen, HTTPClient, ein Paket zum Herstellen von HTTP-Verbindungen und pat, ein Paket, welches die Nutzung von regulären Ausdrücken unter Java ermöglicht.

All dies führt zu folgender Klassenstruktur:



Es folgt nun eine kurze Erläuterung der einzelnen Klassen. Eine genauere Beschreibung der Klassen findet man im Quell-Code des Programms.

- **Circle**, **Rhomb**, **Square** und **Triangle**: Diese Klassen stellen jeweils eine Darstellungsart von Punkten im Scatterplot dar.
- **Data**, **PData** und **RBMap** dienen der strukturierten Speicherung der Programmdaten.
  Data ist dabei eine Unterklasse von PData, die noch um einen String ergänzt ist. Es ist einfach möglich, weitere Unterklassen von PData für eigene Zwecke zu bilden.
  RBMap hingegen ist eine geordnete Map aus dem Package collections..
- **GridBagFrame** vereinfacht die Arbeit mit dem Java-Layoutmanager "gridbag".
- **MergedItem** erlaubt es, zwei Werte in einem RBMap-Element zu speichern. Es ist dazu eine eigene Klasse nötig, da RBMap als Werte nur Klassen annimmt.
- **Presentation**, **Scatterplot** und **Table** bilden die wichtigsten Klassen in VisBenchmark. Scatterplot

und Table stellen die beiden Visualisierungsmöglichkeiten dar, die das Programm bietet. Presentation ist dabei eine Oberklasse dieser beiden Klassen, in welcher alle Methoden und Attribute aufgelistet sind, die von beiden Darstellungsarten benötigt werden.

- **PropertyMenu** stellt das Fenster dar, welches dem Benutzer erlaubt, die Eigenschaften des Scatterplots anzuzeigen und zu verändern.

- **ScatterplotWindow** und **TableWindow** sind die Klassen der Fenster, in denen Scatterplot bzw. Table enthalten sind. Es wurden dazu eigens zwei neue Klassen angelegt, um die Darstellungsmöglichkeiten leicht in ein anderes Programm zu übernehmen.

- **SubRegex** ist eine Erweiterung von Regex (aus dem Paket pat), welche es erlaubt, Teile eines regulären Ausdrucks durch einen anderen zu ersetzen.

- **URLCommands** übernimmt die Aufgabe, Verbindungen zu URLs zu erstellen und diese einzulesen. Dazu werden im Fall von http-URLs die Klassen **URLConnection** und **URLResponse** aus HTTPClient benötigt.

- **URLViewer** ermöglicht es, den Inhalt einer beliebigen URL als Text in einem Fenster anzuzeigen.

- **VisApplet** und **VisApplication** sind die Dateien, die zum Starten von VisBenchmark aufgerufen werden. Dabei startet VisApplet die Applet-Version und VisApplication die Applikations-Version des Programms.

- **VisBenchmark** stellt das Haupt-Fenster der Anwendung dar. In ihm werden alle nötigen Initialisierungsarbeiten durchgeführt.

3.3 Interessantes zur Entwicklung mit Java

Hier sollen einige Vor- und Nachteile von Java aufgelistet werden, die mir bei der Programmentwicklung so aufgefallen sind. Die Liste erhebt daher keinen Anspruch auf Vollständigkeit und ist natürlich sehr subjektiv.

Zuerst zu den **Nachteilen**:

- Die Arbeit mit dem AWT (Abstract Windowing Toolkit) brachte einige Probleme mit sich:
  So laufen Programme, die das AWT benützen, unter HP-UX 9 überhaupt nicht und auch bei HP-UX 10 ist es nicht möglich, Dialoge zu erzeugen. Allerdings tauchen diese Probleme bei anderen Platformen (Windows 95, Linux) nicht auf.

- Zudem erweist sich die Arbeit mit dem AWT als teilweise recht langwierig. So müssen die Elemente von Fenstern mit Hilfe von teilweise übereinandergelegten Layouts angeordnet werden, was einen recht langen Code zur Folge hat. Abhilfe würden hier visuelle Implementierungstools schaffen, welche zwar teilweise schon angeboten werden, aber noch nicht so ganz ausgereift sind.

- Auch fehlt die Möglichkeit, Methoden als Übergabeparameter zu verwenden.

- Das größte Manko (aber zugleich auch der größte Vorteil) von Java ist aber die Portabilität: Da das Programm interpretiert wird, ist es natürlich nicht besonders schnell. Das führt dazu, daß man bei einer großen Menge von Daten sehr schnell an die Grenzen des Programms kommt, was z.B. bei einer Entwicklung unter C++ nicht der Fall wäre.

Die **Vorteile**:

- Die in Java eingebaute Speicherbereinigung (*garbage collection*) erleichtert die Programmierung ungemein. Dieses Konzept, das z.B. in der funktionalen Programmierung schon längst verwendet wird, nimmt dem Programmierer die Speicherverwaltung des Programms ab. Es ist also im Gegensatz zu C++ nicht mehr nötig, Dekonstruktoren zu den Objekten selbst anzugeben. Da nicht mehr überlegt werden muß, wann ein Dekonstruktor aufzurufen ist, werden hierbei eventuelle Programmierfehler vermieden.

- Zudem ermöglicht das Fehlen von Zeigerarithmetik (*Pointerarithmetic*) eine rasche Entwicklung, da die Verwendung von zu vielen Zeigern bei Sprachen wie C++ schnell unübersichtliche Programme ergibt, was wiederum zu schwer auffindbaren Fehlern führt.

- Das Konzept des *Exception-Handlings* zwingt den Programmierer dazu, im Programm auftretende Laufzeitfehler zu berücksichtigen (z.B. bei I/O-Operationen). Dies erhöht die Stabilität von Programmen.

- Ein weiterer großer Vorteil ist die leichte Erlernbarkeit der Sprache. Wenn man C oder C++ kennt, fällt der Umstieg ziemlich leicht, aber auch sonst dürfte es wegen der einfachen Sprachkonzepte keine Schwierigkeiten geben.

---

4. Ausblick

Die Visualisierungsumgebung ist erst ein Prototyp und deswegen noch ausbaufähig.

So kann die Darstellungsart noch verfeinert werden, indem stufenloses Zoomen eingebaut wird, oder Probleme, die bei einem bestimmten Verfahren nicht terminieren, gesondert angezeigt werden. Außerdem könnten auch noch Klassen von URLViewer abgeleitet werden, welche auch die Darstellung von Nicht-Text-Dateien ermöglichen.

Eine andere Möglichkeit besteht darin, noch zusätzliche statistische Meßwerte wie Mittelwert, Korrelationskoeffizient, usw. hinzuzufügen, um die dargestellten Ergebnisse besser auswerten zu können. Es ist dabei auch vorstellbar, die Umgebung dann in ein bestehendes statistisches System zu integrieren.

Ein Problem bei der Weiterentwicklung des Programms ist aber sicherlich, daß sich die Java-API noch in Entwicklung befindet. So wurde bei der Umstellung auf Java 1.1 ein neues Event-Handling-Konzept eingeführt, was zur Folge hat, daß unter 1.0 entwickelte Programme, welche das AWT benutzen, umzuschreiben sind. Zwar gibt sich Sun bei der Dokumentation dazu viel Mühe, dieses Problem genau zu erläutern, und Hilfestellungen zu geben (siehe How To Convert Programs to the 1.1 AWT API), der

Umbau kompletter Anwendungen dürfte aber erheblichen Aufwand verursachen (nach [3]).

Anhang: Zusammenfassung der Unterschiede zwischen den Programmversionen

Eigenschaft	Applet	Applikation
Programm-Start	Konfig.-Dateien werden aus HTML-Seite ausgelesen	Konfig.-Dateien werden aus Batch-Datei ausgelesen
URLs	müssen http:// ... Adressen sein	können auch file:// ... Adressen sein
Stabilität	Probleme mit Netscape 3.0 unter Linux, ansonsten stabiler Programmablauf	stabil

Literatur

[1] Verena Müsken u. Thomas J. Schult, Objektkunst, c't 4/96, S. 330 - 342.

[2] Grady Booch, Objektorientierte Analye und Design, Addison-Wesley, 1994.

[3] Claudia Piemont, Klappe, die 1.1.ste, c't 6/97, S. 364 - 371.

[4] G. Sutcliffe, C.B. Suttner and T. Yemenis, The TPTP problem library, in A. Bundy, editor, Proceedings of the 12th International Conference on Automated Deduction, number 814 in Lecture Notes in Artificial Intelligence, pages 252-266, Springer-Verlag, 1994.

[5] David Flanagan, Java in a Nutshell - a Desktop Quick Reference for Java Programmers, O'Reilly & Associates, 1996.