JCAT -
A Java-Based Implementation of Collaborative Active Textbook

Vortrag im Seminar
Algorithmenanimation
im Sommersemester 1998

Wolfgang Holz
Lützowstraße 8
66119 Saarbrücken

Gliederung

1. Einführung

2. Die Sicht des Anwenders

3. Die Sicht des Autors

    - Festlegen der *interesting events*

    - Der Algorithmus

    - Ein View

    - Design einer Seite des Lehrbuches

4. Die Sicht des System-Entwicklers

1. Einführung

- Entwickelt von Marc H. Brown und Marc A. Najork
- JCAT ist ein auf JAVA basierendes System zum Entwickeln kollaborativer aktiver Lehrbücher für Algorithmen:
  - kollaborativ, da sich eine Gruppe von Leuten (Lehrer und Schüler) ein gemeinsames interaktives Experiment "teilt"
  - aktiv, da ein Leser mit Teilen des Lehrbuches interagieren kann
- Das JCAT Lehrbuch besteht aus Internet-Seiten mit aktiven (Applets) und passiven Teilen (Text, Bilder, Sound, etc.)
  - aktive Teile dienen der Darstellung interaktiver Algorithmenanimationen
  - passive Teile werden benutzt, um den Algorithmus konventionell zu beschreiben

- JCAT bietet die Möglichkeit mehrfache Views eines Algorithmus zu zeigen

- Alle Views werden simultan abgeglichen

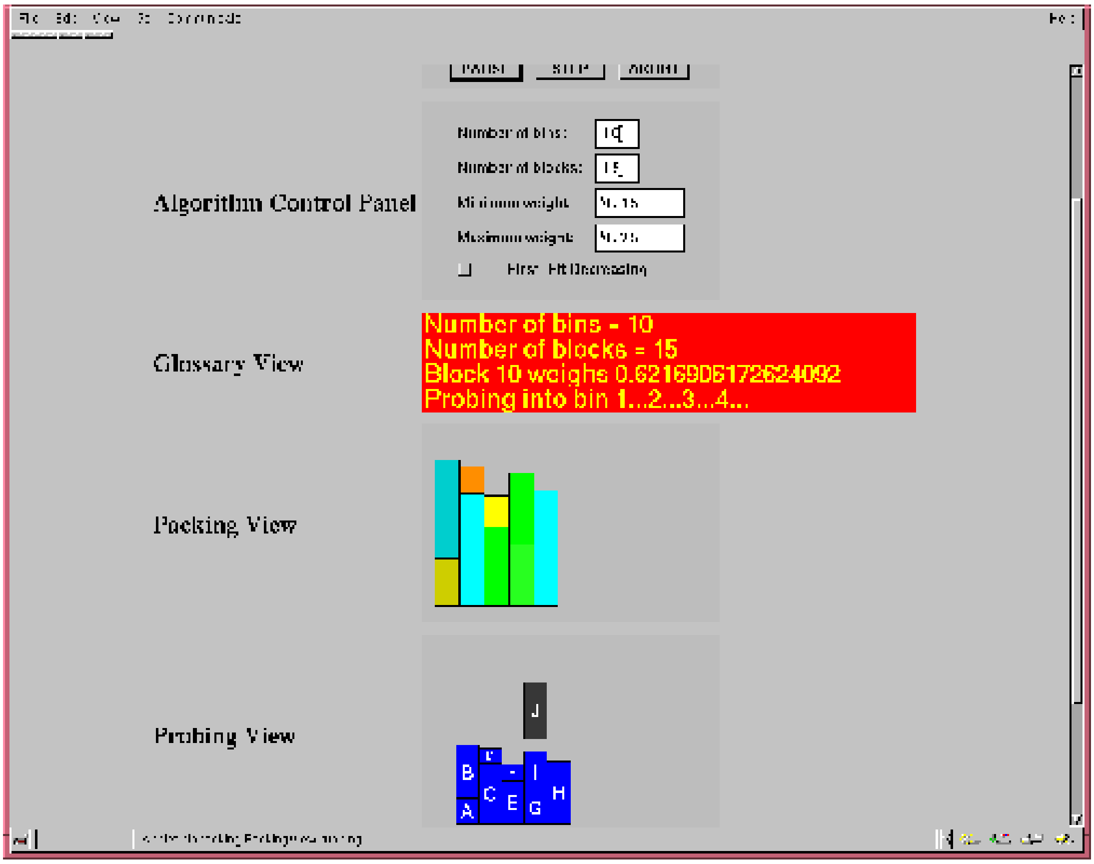
- JCAT basiert auf JAVA's RMI-Technologie, so daß auch Applets, die auf unterschiedlichen Systemen laufen, miteinander kommunizieren können

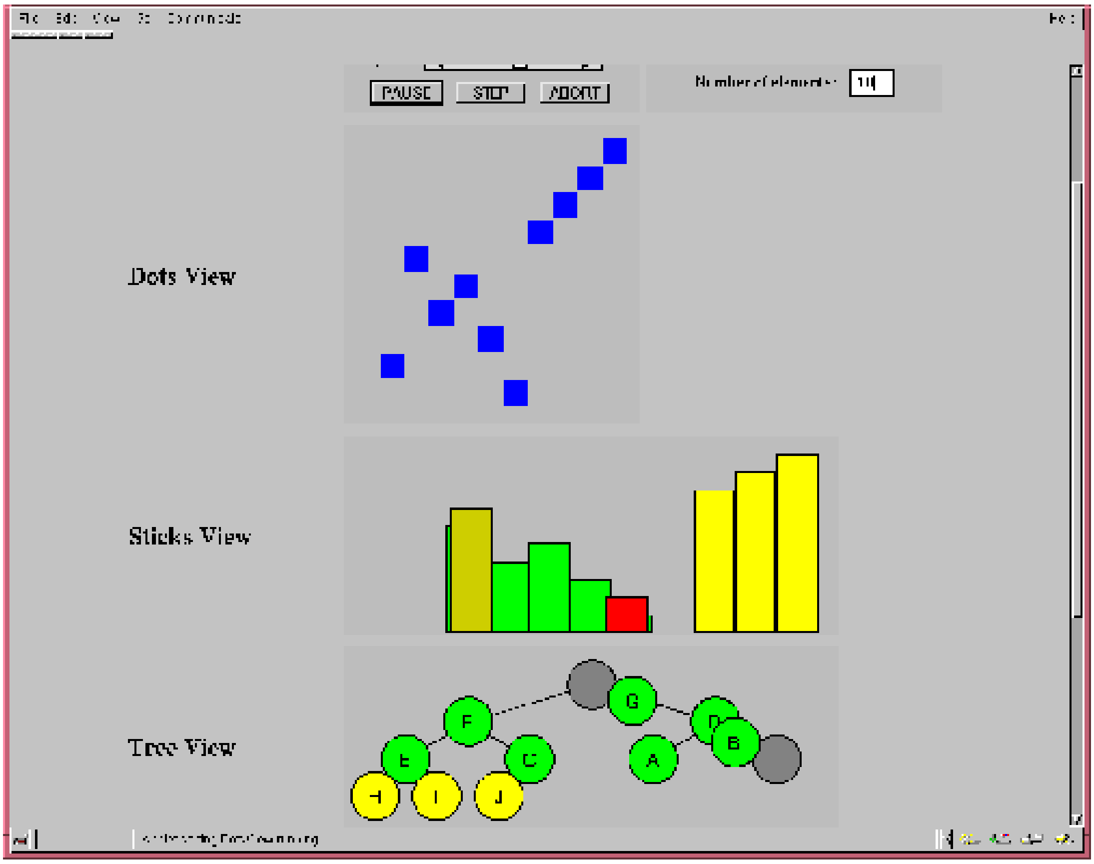
- Somit kann in einem elektonischen Klassenzimmer ein Lehrer die Animation kontrollieren und die Schüler können verschiedene Views des Algorithmus auf ihren Rechnern sehen

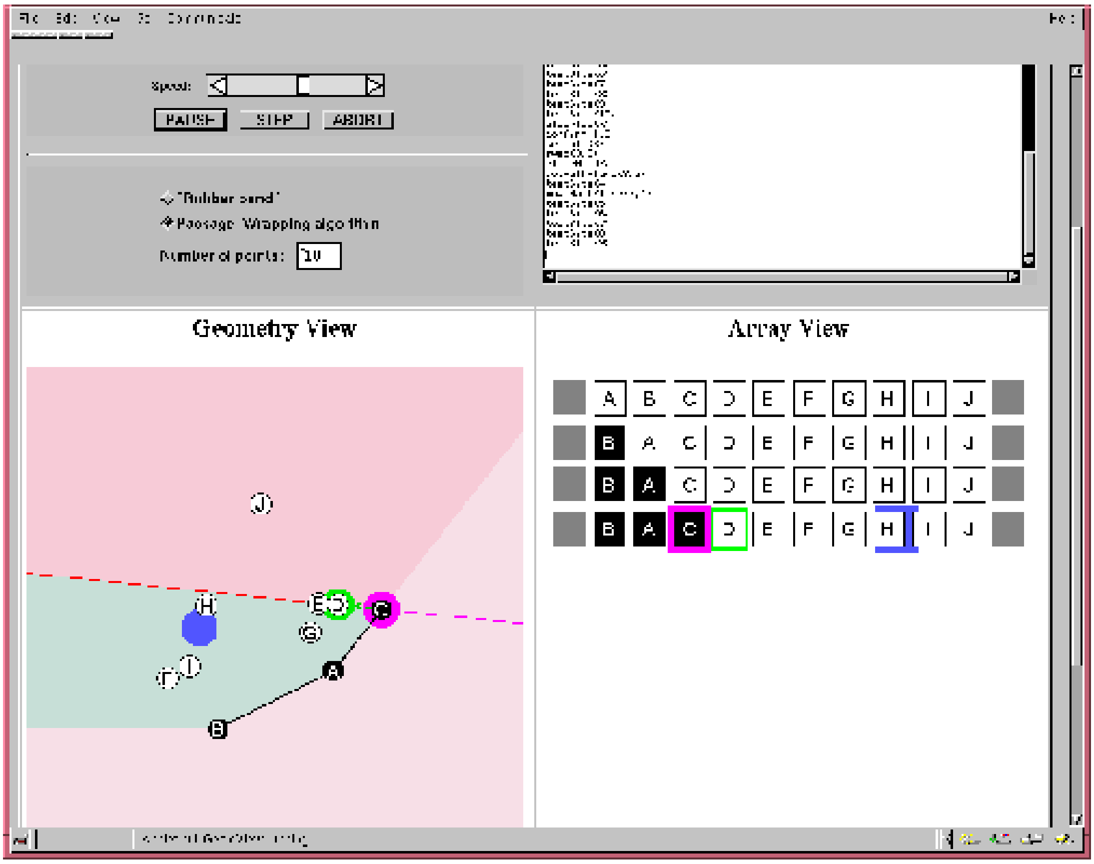
- Das konzeptionelle Modell zum Entwurf von Animationen basiert auf Balsas Idee der *interesting events (IE)*, um den Views die Operationen des Algorithmus mitzuteilen

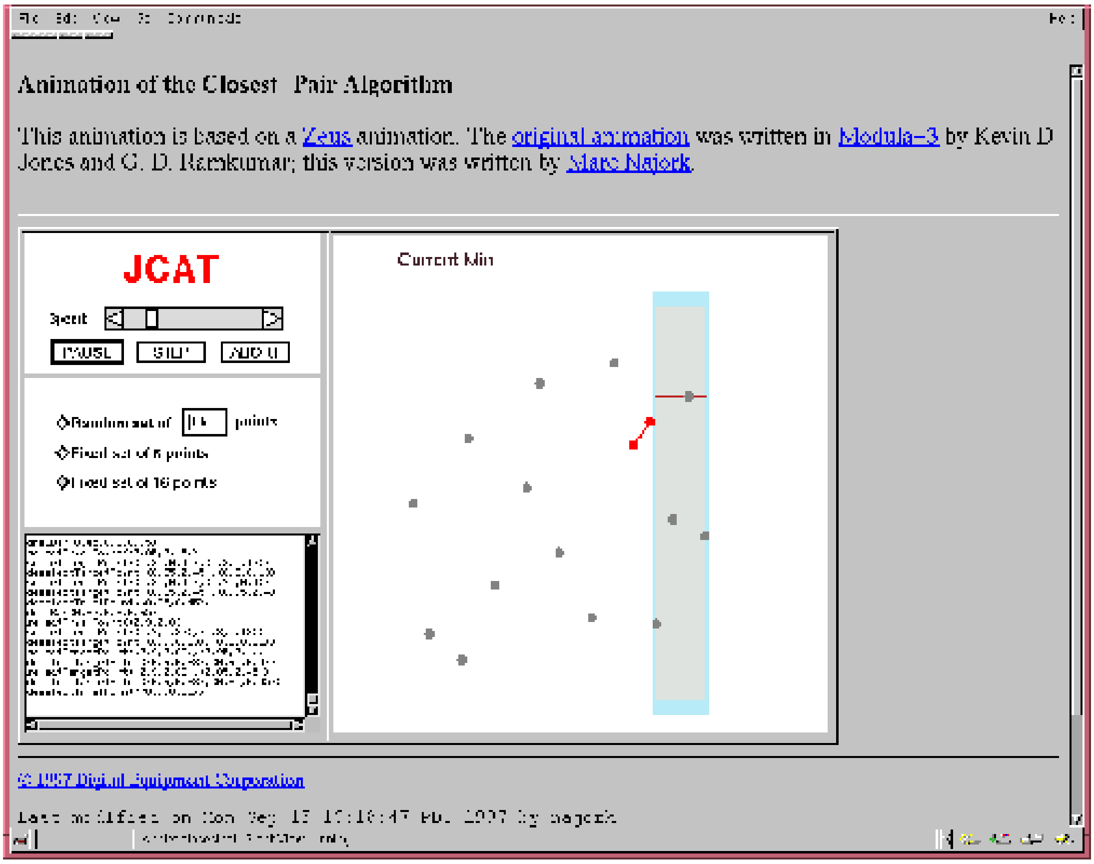
- JCAT ist die JAVA-Implementierung des CAT-Systems

2. Die Sicht des Anwenders




PAUSE
STEP
ABORT
Dots View
Sticks View
Tree View
F
G
D
E
C
A
B
H
I
J


Speed:
PAUSE
STEP
ABORT
"Rubber band"
Number of points:
10
Geometry View
Array View


Animation of the Closest Pair Algorithm
This animation is based on a Zeus animation. The original animation was written in Modula-3 by Kevin D Jones and G. D. Ramkumar; this version was written by Marc Najork.
JCAT
Current Min
Random set of
points
Fixed set of 8 points
Fixed set of 16 points

- Unbegrenzte Anzahl an Schülern kann den gleichen Algorithmus betrachten
- Alle Views werden synchronisiert
- Interaktive Kontrolle jedes Schülers über seine Views
- Möglichkeit die Views an eigene Bedürfnisse anzupassen
- Gleiche Dauer des Algorithmus auf unterschiedlichen Rechnern
- Auf langsameren Rechnern werden weniger Frames dargestellt (Unterschied zu reinen Multiplexern wie XMX oder Shared-X)

3. Die Sicht des Autors

Das System für die Animation von Algorithmen folgt Balsas Modell:

- Wichtige Punkte eines Algorithmus werden mit Prozeduraufrufen annotiert, die *interesting events* generieren

- Diese Events melden sich bei einem Event Dispatcher, der sie an alle registrierten Views weiterleitet

- Jeder View antwortet jedem *interesting event*, indem ein geeignetes Bild gezeichnet wird

Das Erstellen einer Seite eines JCAT-Lehrbuches besteht aus 4 Teilen:

3.1 Festlegen der *interesting events*

- *IE* werden als JAVA Interface spezifizert

```
public interface Binpacking {
  void setup(int numBins, int numBlocks);
  void newBlock(double wt);
  void probe(int bin);
  void pack();
}
```

- Der Präprozessor CATalyst leitet aus dem *IE*-Interface abstrakte Klassen für den Algorithmus und die Views ab

- Diese abgeleiteten Klassen liefern zusammen mit den Basisklassen von JCAT alle Kommunikations-Mechanismen zwischen Algorithmus und Views

3.2 Der Algorithmus

- Der Algorithmus ist ein Applet, das als Eingabe-Dialog erscheint

- Das Applet ist eine Unterklasse der abstrakten Algorithmen-Klasse, die von CATalyst generiert wurde

- Der Autor implementiert den Algorithmus, indem er die abstrakte Methode **algorithm** mit dem in Frage kommenden Algorithmus überschreibt.

- Zudem wird der Programmcode mit Aufrufen von Methoden der *IE* annotiert

- Diese Methoden werden von der Klasse BinpackingAlg geliefert

- Jede dieser Methoden erzeugt ein *Event*-Objekt, welches zum Kontroll-Feld des Lehrers weitergeleitet wird und von dort zu den Views

```
public class FirstFit extends BinpackingAlg {
  EntryField binFld, blockFld, minFld, maxFld;
  public void init () {
     ...
  }
  protected void algorithm () {
    int numBins = binFld.getInt();
    int numBlocks = blockFld.getInt();
    double min = minFld.getDouble();
    double max = maxFld.getDouble();
    setup(numBins, numBlocks);
    double totals[] = new double[numBins];
    for (int b = 0; b < numBlocks; b++) {
      double wt = Math.Random() * (max - min) + min;
      newBlock(wt);
      int bin;
      for (bin = 0; bin < numBins; bin++) {
        probe(bin);
        if (totals[bin] + wt <= 1.0) break;
      }
      if (bin == numBins) break;
      totals[bin] += wt;
      pack();
    }
  }
}
```

3.3 Ein View

- Ein View ist ein Applet mit einer Menge zusätzlicher Methoden, die mit den im *IE*-Interface definierten Methoden korrespondieren

- Unterklasse der abstrakten View-Klasse, die von CATalyst erzeugt wird

- Der Autor erzeugt einen konkreten View, indem er die Methoden, für die eine Animation gewünscht wird, überschreibt

```
public class ProbingView extends BinpackingView {
  GP gp = new GP();
  Vertex v;
  double currWt;
  double totals[];
  char id;
  int lastProbe;

  public void init() {
    super.init();
    add(gp);
  }

  public void setup(int numBins, int numBlocks) {
    id = 'A';
    totals = new double[numBins];
    gp.clear();
    gp.setWorld(-2.0, numBins + 1.0, 2.0, 0.0);
    gp.redisplay();
  }

  public void newBlock(double wt) {
    v = new Vertex(gp);
    v.setSize(1.0, wt);
    v.setPosition(-1.0, wt / 2.0);
    v.setColor(Color.darkGray);
    v.setBorder(0.01);
    v.setLabelColor(Color.white);
    v.setLabel(" " + id++);
    gp.redisplay();
    currWt = wt;
  }

  public void probe(int bin) {
    v.move(bin, totals[bin] + currWt / 2.0);
    gp.animate(1.0);
    lastProbe = bin;
  }

  public void pack() {
    totals[lastProbe] += currWt;
    v.setColor(Color.blue);
    gp.redisplay();
  }
}
```

- Die Klasse **GP**
  - High-level Animations-Paket, basierend auf Graphen:
    * Knoten mit Attributen: Position, Größe, Aussehen, Farbe, Randbreite, Label
    * Kanten mit Attributen: Farbe, Breite, Krümmung
    * Es gibt auch farbige Polygone, die aus einer Menge von Knoten bestehen
    * Knoten können repositioniert werden
    * Diese Repositionierung kann als Animation dargestellt werden
    * Wird ein Knoten bewegt, so werden alle Kanten und Polygone, die mit ihm assoziiert sind, mitbewegt
  - **GP** basiert auf einem Modula-3 Animations-Paket
  - Ist Staskos Tango-System nachgebildet

3.4 Design einer Seite des Lehrbuches

Alle Seiten des Lehrbuches werden in HTML gestaltet.

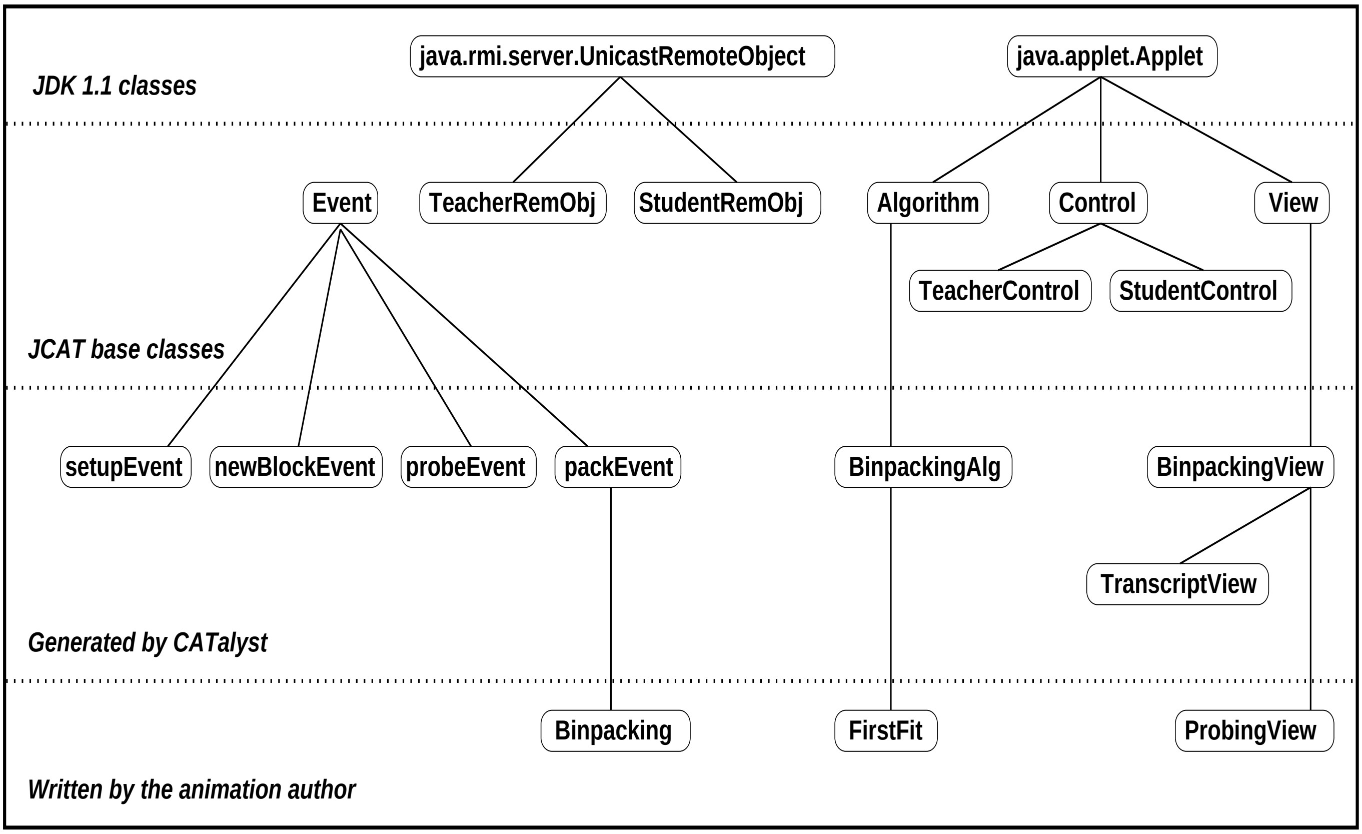
Beispiel:

```html
<html>
<head>
<title>Animation of Binpacking</title>
</head>
<body>
<h1>Animation of Binpacking</h1>
<hr>
<center>
<table>
<tr>
<td><b>JCAT control panel</b></td>
<td>
<applet CODE="jcat.FullControl.class" CODEBASE="../" WIDTH=300 HEIGHT=140>
<param name="localapplets" value="5">
</applet>
</td>
</tr>
<tr>
<td><b>Algorithm Control Panel</b></td>
<td>
<applet CODE="binpacking.FirstFit.class" CODEBASE="../" WIDTH=300 HEIGHT=200>
<param name=numBins value=10>
<param name=ffd value="NO">
</applet>
</td>
</tr>
<tr>
<td><b>Packing View</b></td>
<td>
<applet CODE="binpacking.PackingView.class" CODEBASE="../" WIDTH=300 HEIGHT=200>
</applet>
</td>
</tr>
</table>
</center>
</body>
</html>
```

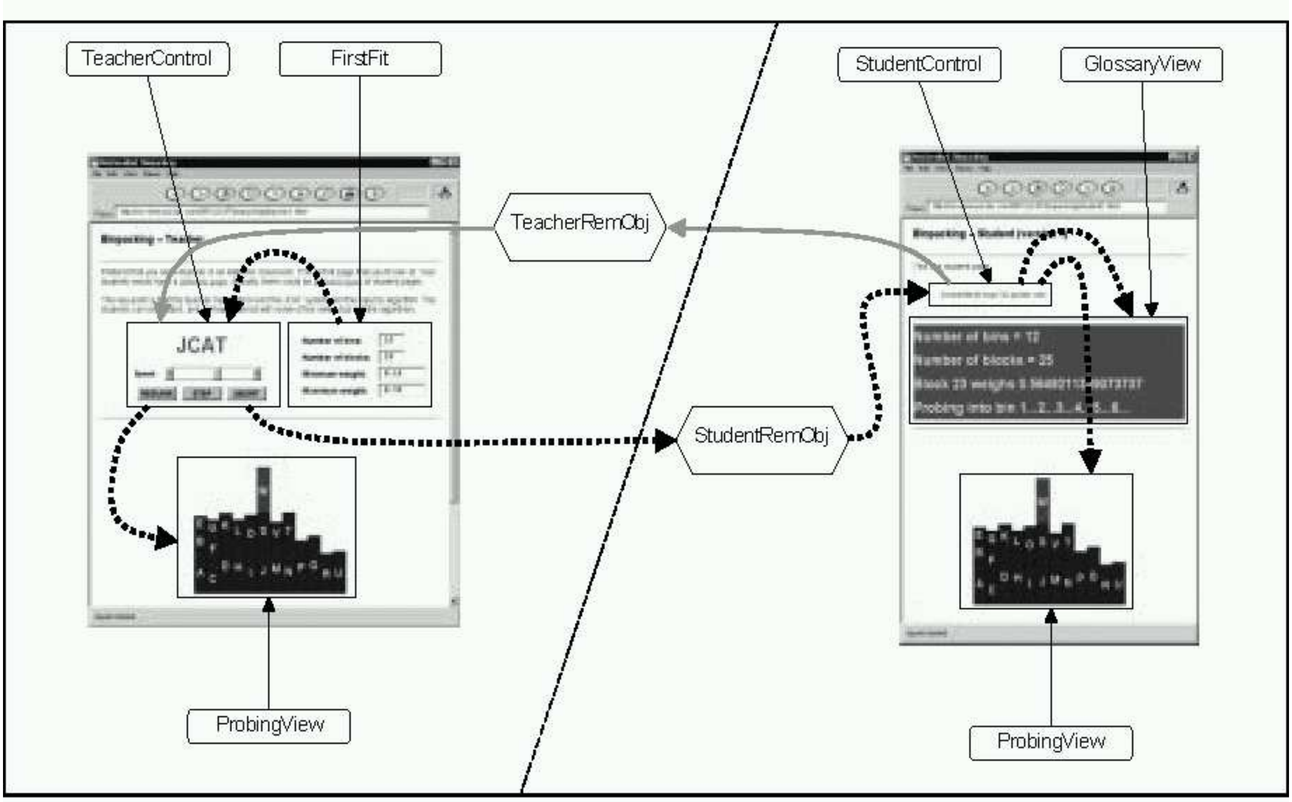
4. Die Sicht des System-Entwicklers

Das komplette JCAT-System besteht aus einer Sammlung von Klassen, die unabhängig von jeder einzelnen Animation, der GP Animations-Klasse und dem CATalyst Präprozessor sind.

Hierarchie der Klassen, die das **binpacking**-Beispiel bilden



Die Kommunikations-Pfade



Internet-Adresse:

http://www.research.digital.com/SRC/JCAT