The Mayday Processing Framework

A Graphical Processing Framework for Mayday

MPF

A processing framework for Mayday

Florian Battke 2006

Motivation

Anforderungen

Aufbau

- *einfache Filter*
- *Filteroptionen*
- *Applicator*
- *MaydayDataObj.*
- *Beispiel*
- *komplexe Filter*
- *Designer*

Übersicht

Vorführung

Motivation

- Übliche Pipeline bei Microarray-Experimenten
  - Normalisierung
  - Imputation
  - Log Transformation
  - Herausfiltern von uninteressanten Genen
- Wiederkehrende Arbeitsabläufe
- Mayday bietet bislang keine einheitliche Möglichkeit, diese Aufgaben durchzuführen und zu automatisieren.

Anforderungen

- Möglichst allgemeines Framework
- Erweiterbar durch Plugins
- Anwendung auf mehrere ProbeLists (batch)
- Zusammenstellen von Filtern zu „Pipelines“
- Speichern von Pipelines und Einstellungen
- Auch für Nicht-Progammierer verständlich

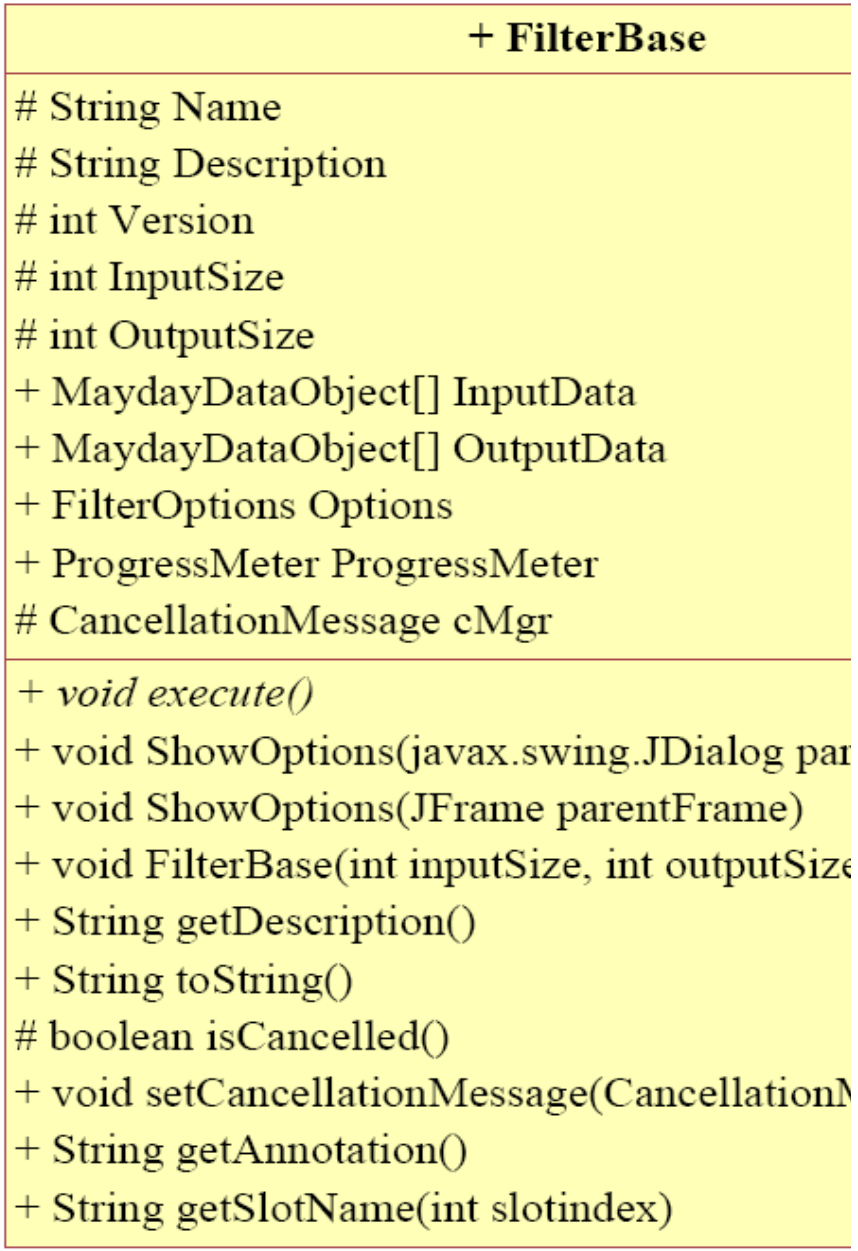
Aufbau

- „Einfache“ Filter
  - abgeleitet von FilterBase
  - leicht zu implementieren
- Applicator
  - Schnittstelle zwischen Mayday und den Filtern
- „Komplexe Filter“
  - aus einfachen Filtern zusammengesetzt
  - Filtergraph definiert Reihenfolge
  - Optionen können nach aussen präsentiert werden
- Designer
  - GUI für die Erstellung komplexer Filter

Aufbau: Einfache Filter

Grundlegende Bestandteile

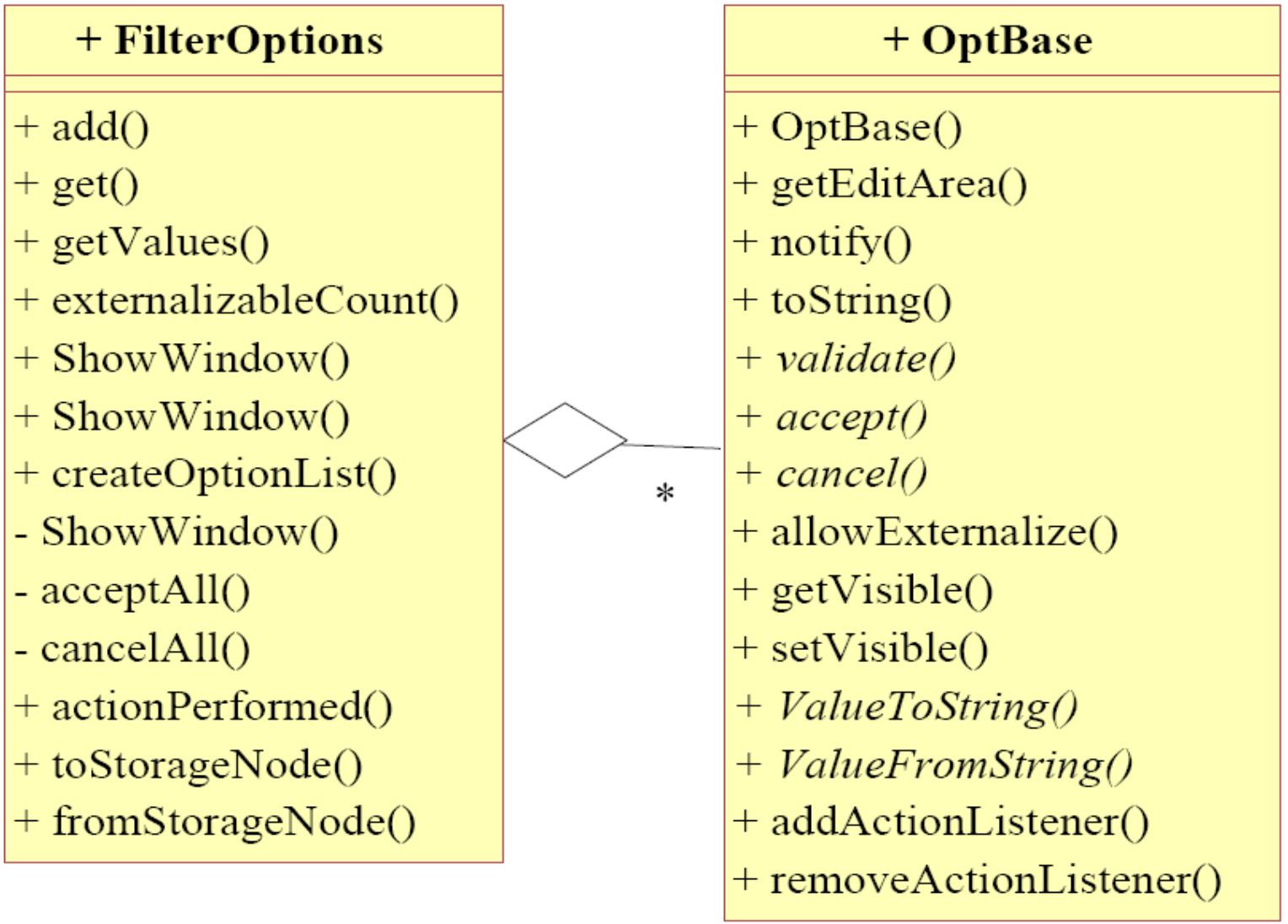
- Name des Filters
- Beschreibung
- Eingänge/Ausgänge
- Optionen
- Funktion



Aufbau: Filteroptionen

- Abgeleitet von OptBase
- Einfach zu implementieren
- Bereits implementiert sind: Bool, Int, Double, String, DropDown-Liste, ...
- Options-Objekte
  - enthalten den Namen und Wert einer Option, einen Defaultwert sowie eine Beschreibung
  - erzeugen GUI-Objekte zum Setzen der Option
  - prüfen Benutzereingaben auf Korrektheit
  - Konvertieren Optionswert<-->String zum Speichern und Laden von Einstellungen
- FilterOptions verwaltet Optionslisten und erzeugt Dialogfenster für alle Optionen

Aufbau: Filteroptionen



Aufbau: Applicator

- Wird vom Mayday PluginManager gestartet
- Drei Schritte:
  - Auswahl des anzuwendenden Filters
  - Zuordnung der Eingabedaten auf die Eingänge des Filters (wenn mehr als ein Eingang vorhanden ist)
  - Setzen von Filteroptionen
- Ruft den gewählten Filter *n mal* auf
- Gibt Feedback über aktuellen Stand (%, Fehler)
- Räumt bei Fehlern im Speicher auf
- Bietet Zugang zum Designer

Aufbau: MaydayDataObject

- Kapselt eine ProbeList
- Stellt sicher, daß Veränderungen
  - nicht die ursprünglichen Daten betreffen
  - entweder ganz oder gar nicht in Mayday auftreten
  - für die Programmierer von Filtern einfach durchzuführen sind
  - möglichst wenig Speicher verwenden
- Kümmert sich um
  - Einfügen und Löschen von Probes
  - Erzeugen eindeutiger Namen
  - Annotieren neuer/geänderter Probes
  - Weiterleitung von Daten zwischen Subfiltern

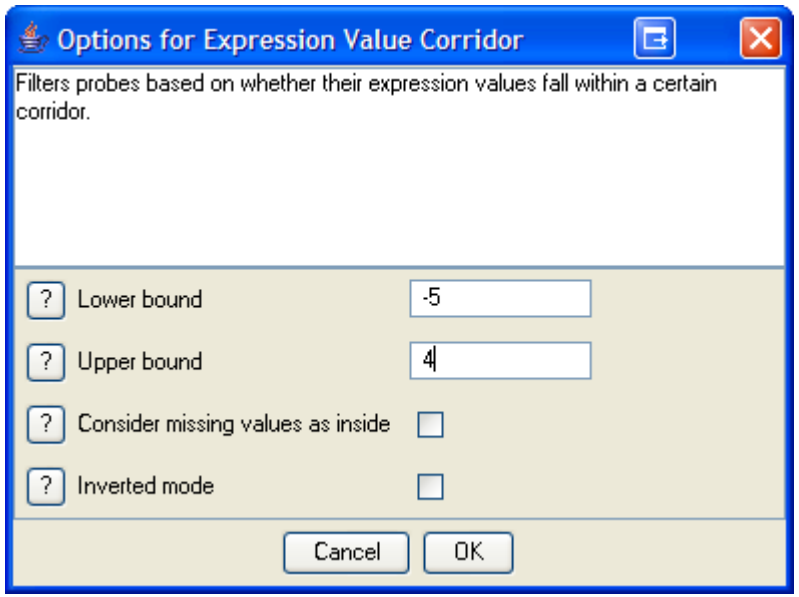
Beispiel: Einfacher Filter (1)

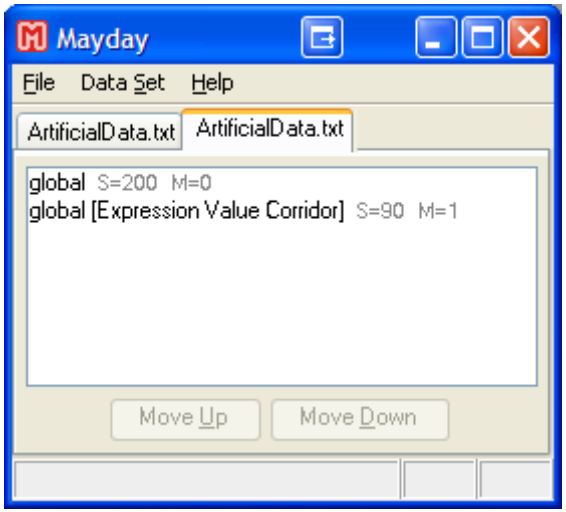
```
package mayday.MPF.Filters;

import java.util.Vector;
import mayday.MPF.FilterBase;
import mayday.MPF.OptionTypes.OptBoolean;
import mayday.MPF.OptionTypes.OptDouble;
import mayday.core.Probe;

/** @author Florian Battke */
public class ExpValCorridor extends FilterBase {

    OptDouble minVal = new OptDouble(
            "Lower bound",
            "The lowest value still in the corridor",
            0.0);

    OptDouble maxVal = new OptDouble(
            "Upper bound",
            "The highest value still in the corridor",
            100.0);

    OptBoolean invert = new OptBoolean(
            "Inverted mode",
            "Select this option to discard probes that fall inside the corridor "
            + "instead of keeping them",
            false);

    OptBoolean nullInside = new OptBoolean(
            "Consider missing values as inside",
            "Select this option to treat missing values as falling into the "
            + "corridor as opposed to lying outside of it.",
            false);
```

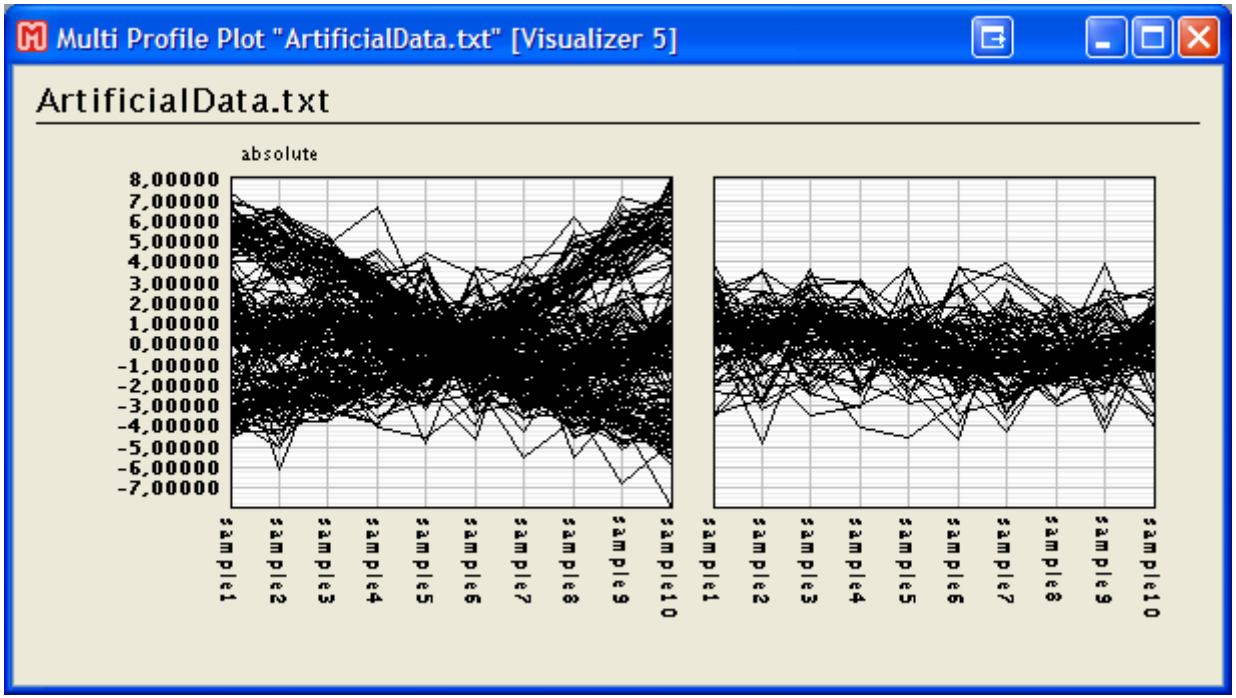
Beispiel: Einfacher Filter (2)

```
public ExpValCorridor() {
    super(1,1);
    Name="Expression Value Corridor";
    Description="Filters probes based on whether their expression values fall"
            + " within a certain corridor.";
    Options.add(minVal);
    Options.add(maxVal);
    Options.add(nullInside);
    Options.add(invert);
}

private boolean checkCriteria(Probe pb) {
    boolean isInside = true;
    for (int i=0; i!=pb.getNumberOfExperiments() && isInside; ++i) {
        Double d = pb.getValue(i);
        if (d==null)
            isInside &= nullInside.Value;
        else
            isInside &= (d<=maxVal.Value) && (d>=minVal.Value);
    }
    return isInside;
}

public void execute() {
    OutputData[0]=InputData[0];
    for (Probe pb : OutputData[0]) {
        boolean keep = checkCriteria(pb);
        if (invert.Value) keep=!keep;
        if (!keep) OutputData[0].remove(pb);
    }
}
```

Beispiel: Ergebnis







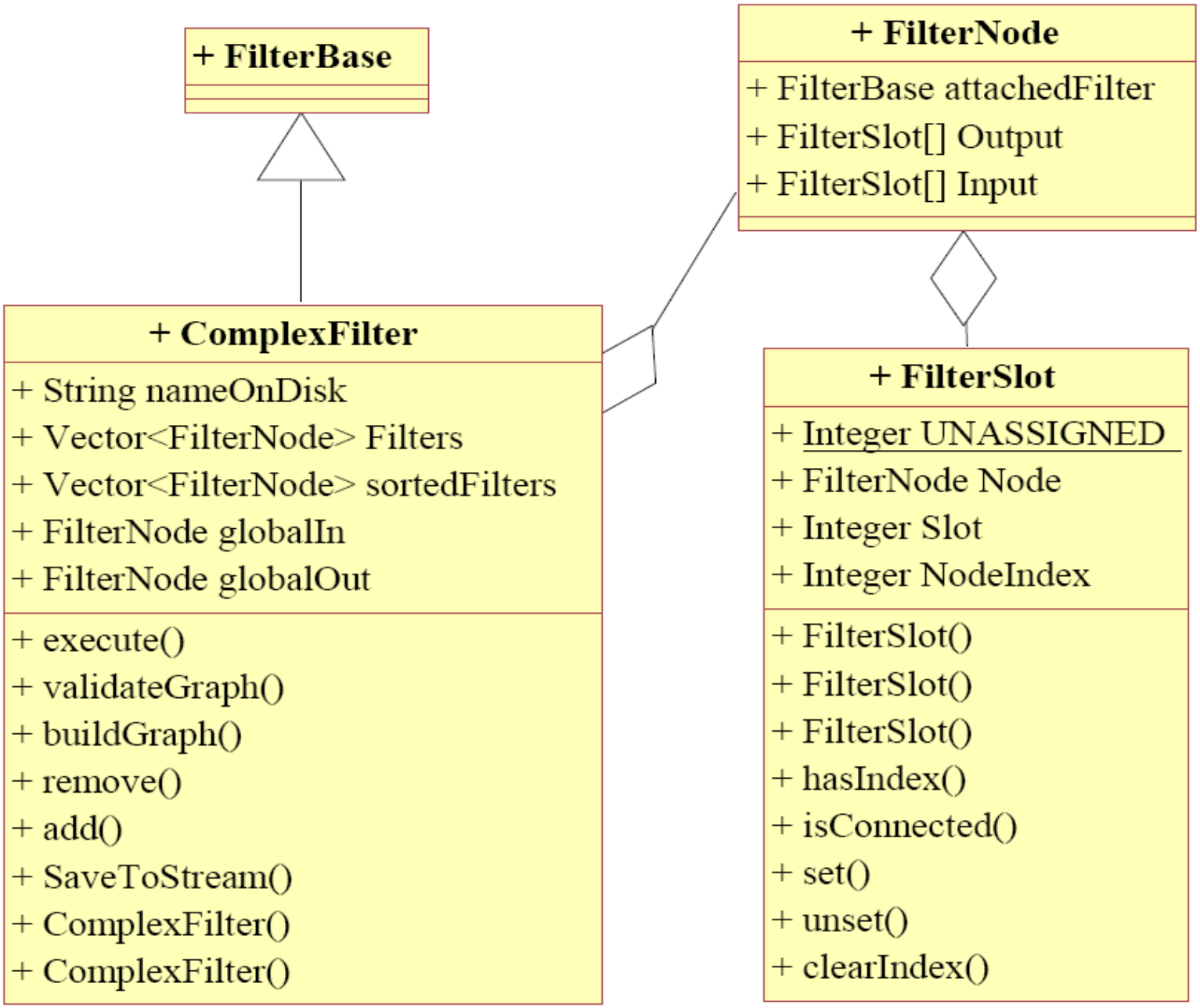
MPF
A processing framework for Mayday

Florian Battke 2006

Aufbau: ComplexFilter

- Verwaltet einen Graphen aus FilterNodes
- Prüft den Graphen auf Zyklen und fehlende Verbindungen
- Lädt und speichert den Graphen
- Enthält Voreinstellungen für die Subfilter
- Berechnet die Ausführungsreihenfolge der Subfilter
- Präsentiert eine Auswahl der Subfilter-Optionen nach außen
- Kann wiederum in einen anderen komplexen Filter integriert werden

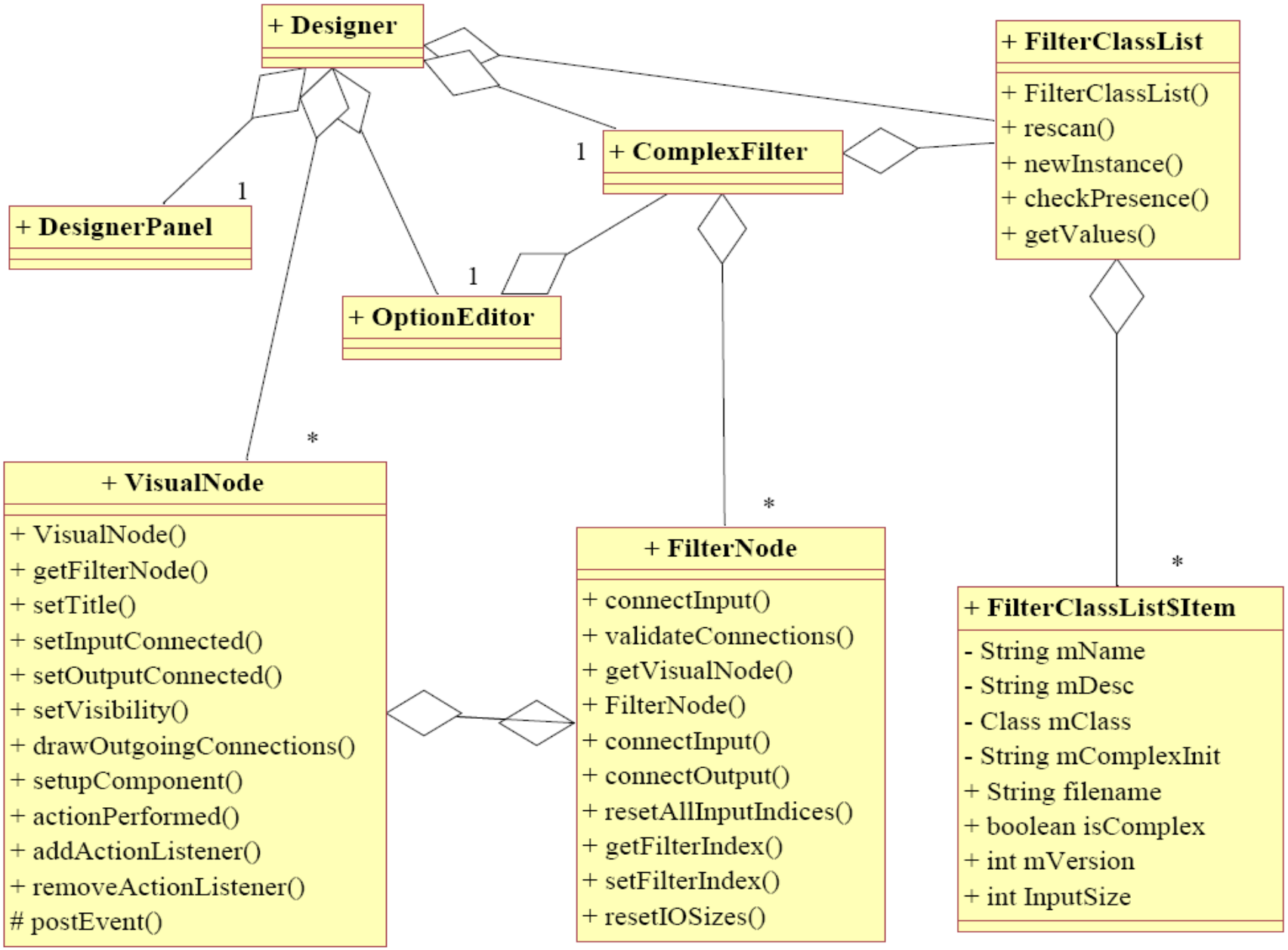
Aufbau: Komplexe Filter



MPF
A processing framework for Mayday

Florian Battke 2006

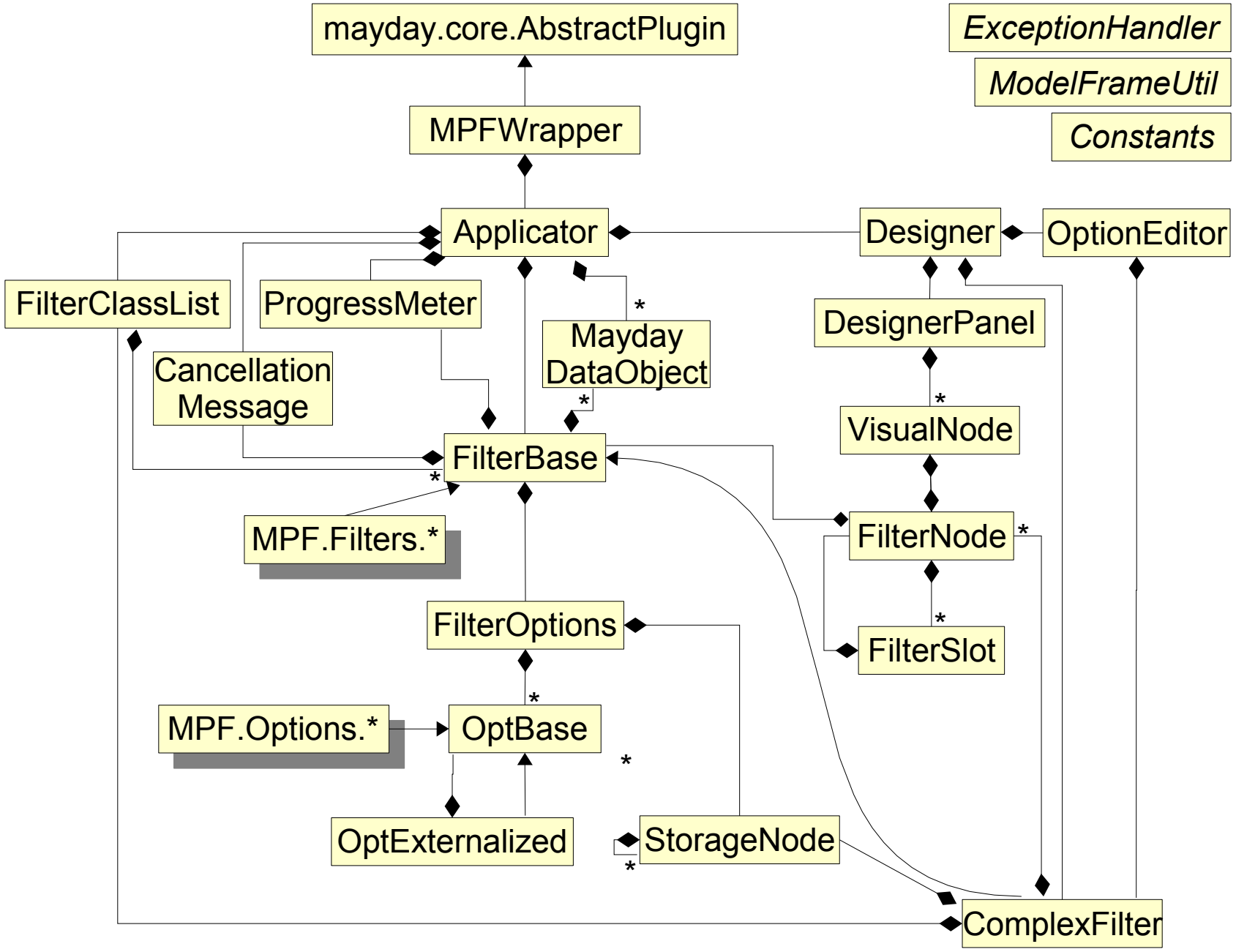
Aufbau: Designer



MPF
A processing framework for Mayday

Florian Battke 2006

(Un-)Übersicht über alle Klassen



(+weitere interne Klassen)

MPF
A processing framework for Mayday

Florian Battke 2006

Übersicht

- Bislang vorhandene Filter
  - Basic Imputation
  - Basic Normalization
  - Log Transformation
  - Filter on MIOs (String, Int, Double)
  - Expression Value Corridor
- Noch zu implementieren
  - Wrapper für R-Skripte (verwendet RPlugin)

Vorführung

- Log Transformation
  - Behandlung von Werten <=0
- Erstellen einer Pipeline
  - Imputation zum Ersetzen der Fehlstellen durch Konstante Werte
- Option externalisieren
- Neugewonnene Pipeline anwenden