

# WPF-Animationsframework

---

Projektdokument

**FHNW**

**Projekt IP 5**

**Herbstsemester 2009**

**Roger Fankhauser / Cedric Hollenstein**

**Versions Nr.:** 1.0

**Autor:** Cedric Hollenstein /  
Roger Fankhauser

**Datum Letzte Revision:** 31.01.2010

**Dokument Status:** Final

**File:** Projektdokument.doc

## Revisionen

Version	Datum	Kommentar	Autor
0.1	22.09.2009	Initialversion	Cedric Hollenstein
0.2	28.09.2009	Erste Anforderungen hinzugefügt	Roger Fankhauser
0.3	13.10.2009	Architektur-Design hinzugefügt	Roger Fankhauser
0.4	06.01.2010	Schwierigkeiten und Lösungsansätze	Cedric Hollenstein
0.5	10.01.2010	Analyse bestehender Lösungen, Konzept	Cedric Hollenstein
0.6	12.01.2010	Erstellen einer Datenstruktur	Cedric Hollenstein
1.0	31.01.2010	Fertigstellen des Dokumentes	Roger Fankhauser

# Inhaltsverzeichnis

<b>REVISIONEN</b>	<b>I</b>
<b>INHALTSVERZEICHNIS</b>	<b>II</b>
<b>1. EINLEITUNG</b>	<b>1</b>
1.1 Ausgangslage.....	1
1.2 Projektauftrag.....	1
1.3 Auftraggeber.....	1
1.4 Projektmitglieder.....	1
<b>2. ANFORDERUNGEN ENTWICKLUNG</b>	<b>1</b>
2.1 Animationsframework.....	1
2.2 Animationen.....	1
2.2.1 Array / Liste.....	1
2.2.2 Binärbaum.....	1
<b>3. ARCHITEKTUR-DESIGN</b>	<b>2</b>
3.1 Trennung Business Logik / User Interface.....	2
3.2 Trennung Algorithmen-Analyse / Animationen.....	2
<b>4. ANALYSE BESTEHENDER LÖSUNGEN</b>	<b>3</b>
4.1 Ausgangslage.....	3
4.2 BALSa Familie (BALSa, BALSa II, Zeus).....	3
4.3 TANGO Familie (TANGO, XTANGO, POLKA, Samba, JSamba).....	4
4.4 JELiot.....	5
4.5 Flashdance.....	6
4.6 Konklusion.....	8
<b>5. KONZEPT</b>	<b>10</b>
5.1 Einleitung.....	10
5.2 Erzeugen und „Unterdrücken“ von Ereignissen.....	11
5.3 Detailgrad.....	12
<b>6. TECHNISCHE SCHWIERIGKEITEN UND LÖSUNGSANSÄTZE</b>	<b>13</b>
6.1 Verwendung von System.Array.....	13
6.2 Auslesen von Variablennamen.....	13
6.3 Auslesen von Variablennamen.....	14
6.4 Erstellen einer animierten Array Klasse.....	14
6.5 Abfangen von Events aus den Animationsklassen.....	14
6.6 WPF Animationen.....	15
6.6.1 Auslagern von UI Elementen in XAML.....	15
6.6.2 Doppelte Referenz auf XAML-Objekte.....	15
6.6.3 Geschwindigkeitsregelung Animationen.....	16
<b>7. FRAMEWORKAUFBAU</b>	<b>17</b>
7.1 Allgemein.....	17
7.2 Namespace und Klassenübersicht.....	17
7.3 Wichtige Klassen.....	18
7.3.1 AnimArray.....	18
7.3.2 AnimInt.....	18
7.3.3 EventProcessor.....	18
7.3.4 Animator.....	18
7.3.5 Array-, List, Tree-Animator.....	18
7.3.6 AnimationEquation.....	18
7.3.7 DisplayElement.....	18
7.3.8 Array-, List-, Tree-Element.....	18
<b>8. IMPLEMENTATION EINES ALGORITHMUS</b>	<b>19</b>
8.1 Allgemein.....	19
8.2 Beispiel Binäre Suche.....	19
8.3 Beispiel Quicksort.....	20
<b>9. IMPLEMENTATION EINER DATENSTRUKTUR</b>	<b>21</b>
9.1 Allgemein.....	21
9.2 Beispiel Array.....	21

<b>10.</b>	<b>EINBINDEN UND STARTEN EINES ANIMIERTEN ALGORITHMUS</b>	<b>24</b>
10.1	Klassenaufbau.....	25
<b>11.</b>	<b>PROJEKTMANAGEMENT</b>	<b>26</b>
11.1	Allgemein.....	26
<b>12.</b>	<b>EINGESETZE TOOLS</b>	<b>28</b>
12.1	Microsoft Visual Studio.....	28
12.2	Microsoft Team Foundation Server.....	28
12.3	Microsoft Sandcastle.....	28
<b>13.</b>	<b>REFLEXION/LESSONS LEARNED</b>	<b>29</b>
<b>14.</b>	<b>QUELLENANGABEN</b>	<b>30</b>

# 1. Einleitung

## 1.1 Ausgangslage

Im Rahmen des Projekts Algoria entwickelt das IMVS eine neuartige Tablet-PC Applikation, die den Tablet-PC des Dozenten zum virtuellen Whiteboard werden lässt. Datenstrukturen sollen auf herkömmliche Art skizziert und entsprechende Algorithmen darauf angewendet werden können. Die Applikation Algoria erkennt die skizzierten Datenstrukturen mit Methoden der künstlichen Intelligenz, bildet sie logisch im Hauptspeicher nach und stellt sie zudem verschönert dar. Dadurch entsteht die Möglichkeit, typische Algorithmen (wie beispielsweise das Sortieren eines Arrays oder das Einfügen in einen Baum) auf die skizzierte Datenstruktur anzuwenden und den Ablauf des Algorithmus schrittweise darzustellen bzw. zu animieren. So kann sich der Dozent auf die Erklärung konzentrieren und wird nicht durch das Skizzieren an der Wandtafel abgelenkt.

## 1.2 Projektauftrag

Ziel dieser Arbeit ist es, ein Framework zu entwerfen und zu implementieren, welches auf einfache Art ermöglicht, diverse Algorithmen (sortieren, einfügen, löschen, mischen, konvertieren in andere Datenstrukturen, usw.) zu animieren. Da die Algoria Applikation per Plug-ins erweiterbar sein wird, muss auch das Framework ein späteres hinzukommen von neuen Algorithmen unterstützen (Callbacks). Zudem sollen sich die Animationen intuitiv oder physikalisch sinnvoll verhalten. Beim Einfügen in einen Baum soll beispielsweise zuerst Platz geschaffen und anschliessend alle Nodes an die richtigen Positionen verschoben werden.

## 1.3 Auftraggeber

Christoph Stamm  
Institut für Mobile und Verteilte Systeme  
Steinackerstrasse 5  
5210 Windisch  
+41 56 462 47 44  
[christoph.stamm@fhnw.ch](mailto:christoph.stamm@fhnw.ch)

Beat Walti  
Institut für Mobile und Verteilte Systeme  
Steinackerstrasse 5  
5210 Windisch  
+41 56 462 44 11  
[beat.walti@fhnw.ch](mailto:beat.walti@fhnw.ch)

## 1.4 Projektmitglieder

Cedric Hollenstein  
Weidweg 5c  
5702 Niederlenz  
+41 79 665 97 95  
[cedric.hollenstein@students.fhnw.ch](mailto:cedric.hollenstein@students.fhnw.ch)

Roger Fankhauser  
Cheiblerrain 9  
5610 Wohlen  
+41 76 509 50 86  
[roger.fankhauser@students.fhnw.ch](mailto:roger.fankhauser@students.fhnw.ch)

## 2. Anforderungen Entwicklung

### 2.1 Animationsframework

Vor-/Nachteile von Animationsinformationen entweder in Operatoren oder in separaten Code.

	Operatoren	Separater Code
Vorteile	- Saubere Trennung von Algorithmen und Darstellung	- Die Logik kann direkt im jeweiligen Algorithmus aus implementiert werden
Nachteile	- Grösserer Programmieraufwand - Gemeinsame Operationen der Algorithmen müssen gefunden werden - ev. Spezialfälle berücksichtigen, die nur einmal gebraucht werden	- Algorithmen werden unnötig aufgebläht - mehr Framework-Kenntnisse erforderlich zum Hinzufügen von Algos

### 2.2 Animationen

Mindestens folgende Operationen müssen bei der jeweiligen Datenstruktur animiert werden:

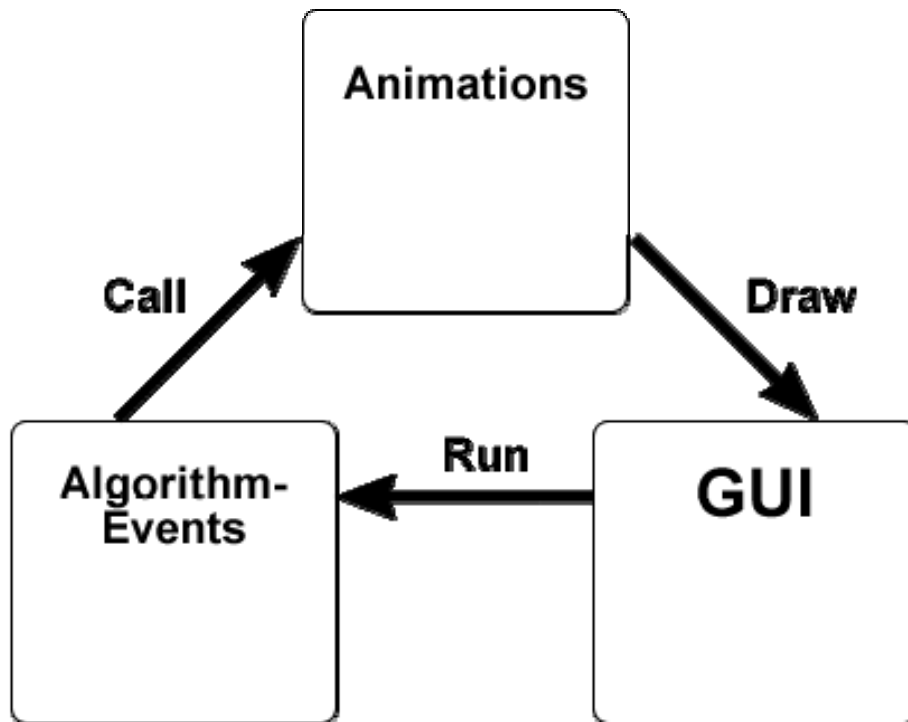
#### 2.2.1 Array / Liste

- Suchen
- Einfügen / Ändern
- Löschen
- Sortieren / Vertauschen

#### 2.2.2 Binärbaum

- Suche

### 3. Architektur-Design



#### 3.1 Trennung Business Logik / User Interface

Die Trennung zwischen der Business Logik und dem User Interface wird mit Hilfe der Markup-Sprache XAML erreicht, die bereits nativ mit WPF geliefert wird. In einem ausgelagerten XAML-File werden die Animationen bzw. die UI-Objekte in ihrem Aussehen definiert.

#### 3.2 Trennung Algorithmen-Analyse / Animationen

Diese Trennung wird anhand mehrerer Interfaces erreicht. In diesen stehen die verschiedenen Aktionen, die von einem Algorithmus ausgelöst werden können. Eine Animator-Klasse implementiert diese und erstellt danach die erforderlichen Animationen. Mithilfe vieler Event-Argumente, die von der Methode mitgeliefert werden, können die erforderlichen Parameter zum Animieren abgerufen werden.

## 4. Analyse bestehender Lösungen

### 4.1 Ausgangslage

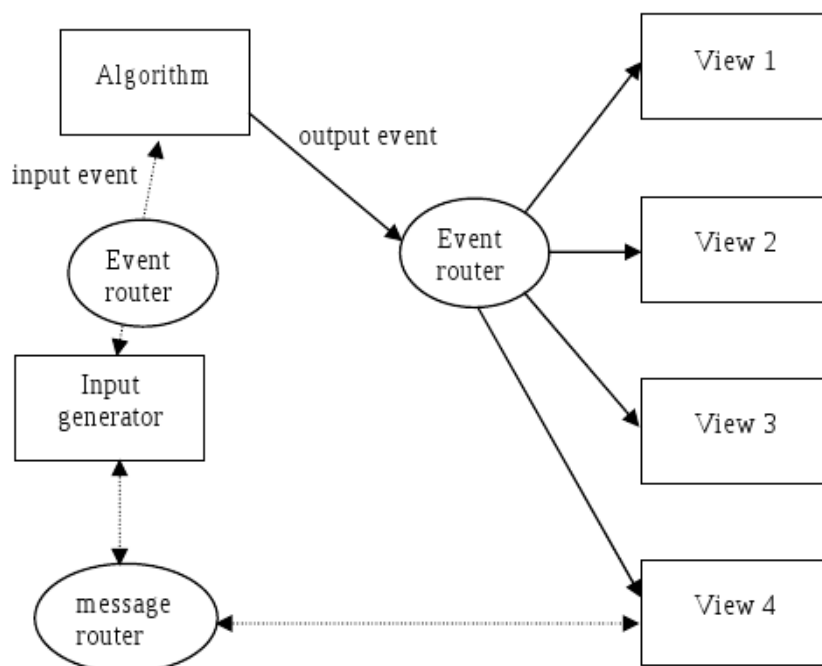
Bevor mit dem Design der Lösung begonnen wurde, wurden erst einmal vorhandene Lösungen analysiert und miteinander verglichen. Dabei wurde der Aufbau der Lösung, das Aussehen sowie die Trennung von Algorithmus- und Animationscode berücksichtigt. Die so gewonnenen Erkenntnisse flossen danach in den Entwurf und die Implementation des WPF Animationsframeworks ein.

Schon in den Sechzigerjahren gab es ein Bedürfnis den Ablauf von Algorithmen graphisch darzustellen. Dies führte zur Erstellung verschiedener Animationsframeworks, wobei zwei Frameworkfamilien deren Wurzeln in die 80er Jahre zurückreichen, besonders herausstechen. Dies sind einerseits die Applikationen der BALSFA Familie, andererseits jene der TANGO Familie. Das Interesse richtete sich hauptsächlich darauf, wie bei diesen Programmen aufgrund des Ausgangsalgorithmus die Befehle für die Animationen generiert werden. Insgesamt wurden vier verschieden aufgebaute Systeme untersucht.

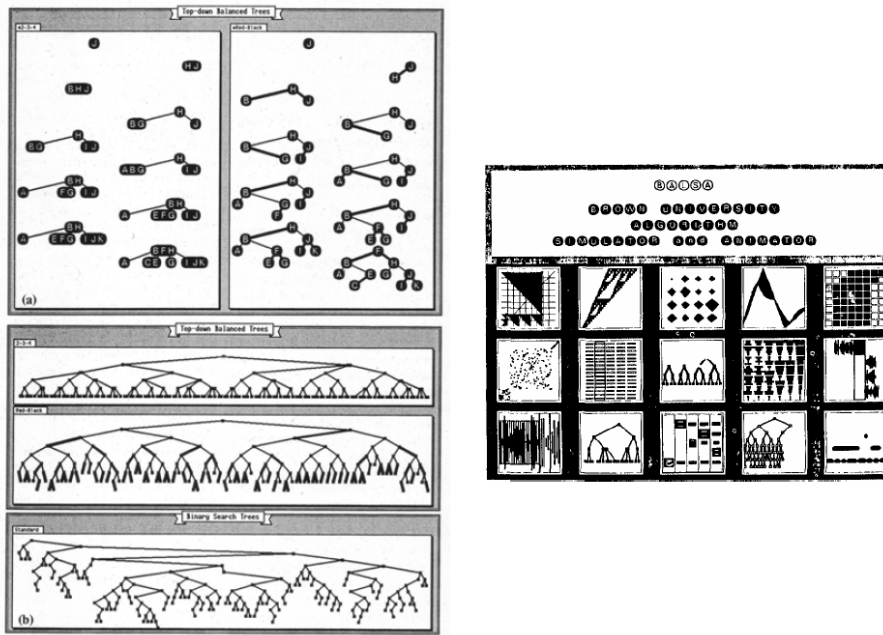
### 4.2 BALSFA Familie (BALSFA, BALSFA II, Zeus)

- Konzept von interessanten Ereignissen („Interesting Events“)
- Liste von zu überwachenden Ereignissen die verarbeitet und visualisiert werden (Änderungen von Werten, swaps)
- Algorithmus enthält Anweisungen für Animationssystem
- BALSFA II enthält die Module InputEvent und OutputEvent welche für zu animierende Ereignisse benutzt werden müssen (OutputEvent.Swap(i, i+1))

Architektur:



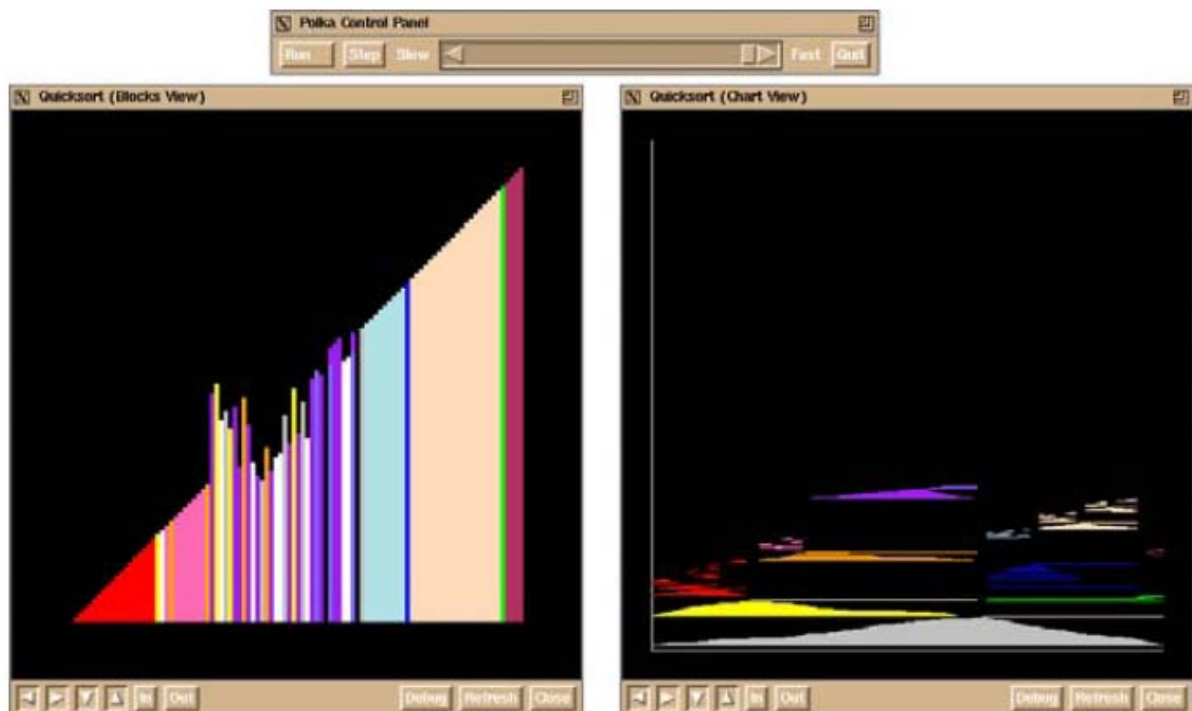
Darstellung:

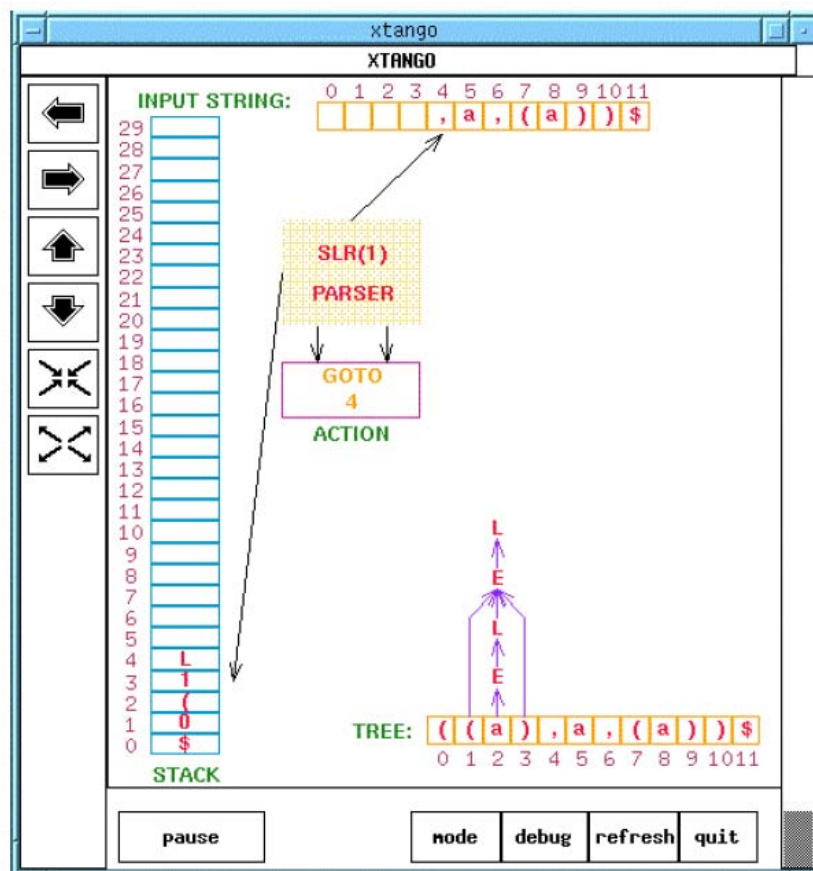
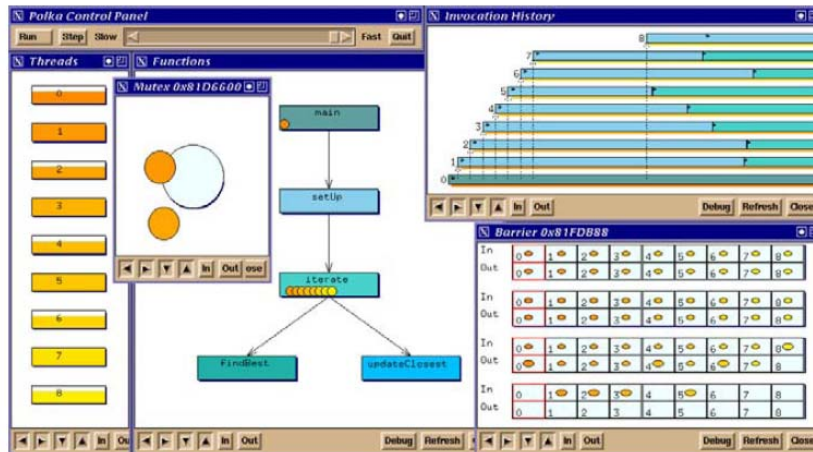


#### 4.3 TANGO Familie (TANGO, XTANGO, POLKA, Samba, JSamba)

- Definiert 4 Typen von Datenobjekten: Locations, Images, Paths und Transitions
- Es wird ein File mit einem TANGO Script erstellt, welches abgespielt werden kann
- Der Algorithmus löst Events aus welche von einem Animator dargestellt werden

Darstellung:

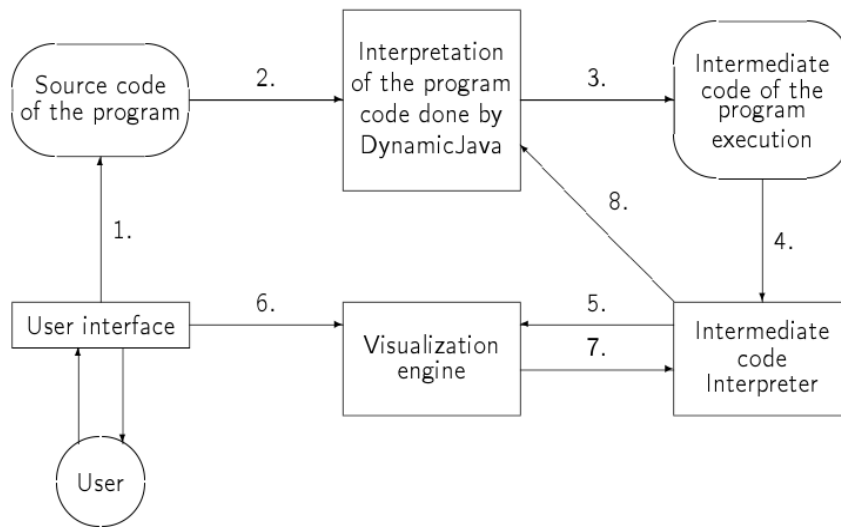




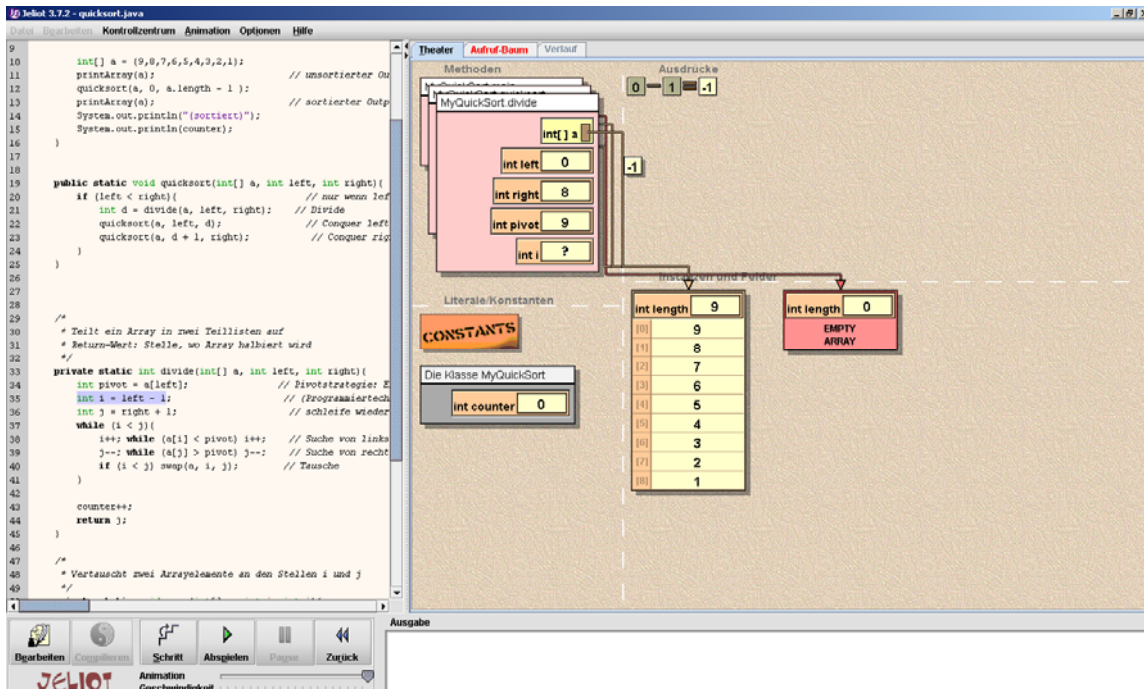
#### 4.4 JElilot

- Basiert auf selbstanimierenden Datentypen
- Algorithmus und Animationscode sind vollständig getrennt, der Animationscode wird zur Laufzeit generiert
- Benutzt den DynamicJava Java Interpreter um MCode zu generieren welcher die Animationen steuert

Architektur:



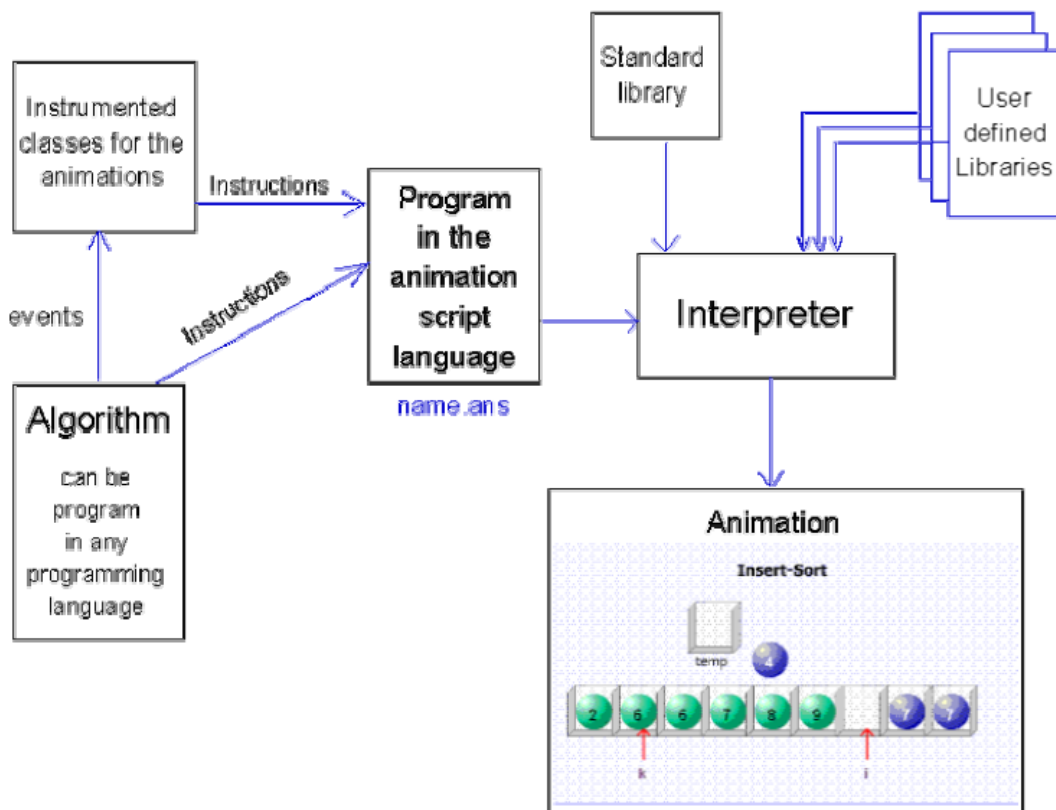
Darstellung:



#### 4.5 Flashdance

- Spielt ein generiertes Script ab
- Das Script wird erzeugt, in dem die Informationen für die Animation bei der Ausüfung des Algorithmus in eine Datei geschrieben wird. Der Ursprüngliche Algorithmus wird danach nicht mehr benötigt.

Architektur:



Scripterzeugung:

```

f = open('quicksort.ans', 'w')
f.write("&prog_text=\n")

S = [12,5,13,8,9,1,3,10,14,4,7,6,15,2,11]

for i in range(0,len(S),1):
    f.write("new Oval o%s %s 300 %s %s \n" %
           (S[i],20+i*30,15+S[i],15+S[i]*5))
qsort(S)
f.close()
  
```

```

import random

def qsort( A ):
    quicksort(A,0,len(A)-1)

def quicksort(A,low,high):
    if low < high:
        m = partition(A,low,high)
        quicksort(A,low,m-1)
        quicksort(A,m+1,high)

def partition(A,low,high):
    pivot = A[high]
    i = low-1
    for j in range(low,high):
        if A[j]<=pivot:
            i = i+1
            swap(A,i,j)
    swap(A,i+1,high)
    return i+1
  
```

```
def swap(A,i,j):
    temp = A[i]
    A[i] = A[j]
    A[j] = temp
    f.write("swap o%s o%s rect 0\n" % (A[i],A[j]))

def generate(A,low,high):
    i=low
    while i<high:
        A[i]=random.randrange(1,400)
        i=i+1
```

- Flashdance Script (Instructions)

**new** *object-type object-name x y width height [label] [colour]*  
 special primitive *object-types*:  
 View *view-name x y width height [label] [ background-colour]*  
 String *object-name x y width height string-text [colour] [style]*  
 Line *object-name x1 y1 x2 y2 [line-width] [colour]*  
 Rectangle *object-name x y width height [line-width] [colour]*  
 Oval *object-name x y width height [line-width] [colour]*

**remove** *object-name1*

**removeAll**

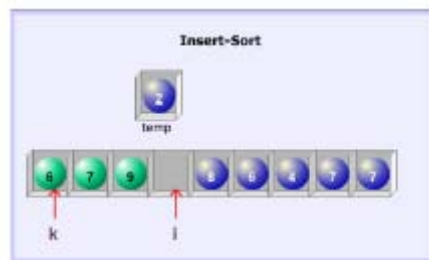
**change** *object-name property1 value1 property2 value2*

**exchange** *object-name object-type [x y width height] [ label]*

**moveTo** *object-name1 x1 y1*

**animTo** *object-name1 x1 y1]*

- Darstellung



#### 4.6 Konklusion

Die Generierung der Animationsbefehle wird bei den betrachteten Animationsframeworks auf verschiedene Arten gelöst:

1. Aufrufen von Funktionen zur Erstellung der Animation im Algorithmus
2. Schreiben eines Skripts mit den Anweisungen zur Anzeige der Animation
3. Parsen des Algorithmus und generieren der Animation aufgrund gefundener Befehle

Variante 1 hat dabei den Vorteil, dass der Algorithmus nicht analysiert werden muss, da die Animation über explizite Anweisungen erzeugt wird. Negativ ist, dass der Code des Algorithmus mit demjenigen für die Animation vermischt ist und dass für die Erstellung der Animation Kenntnisse über die Schnittstelle vorhanden sein müssen. Aufrufen der Funktionen für die Animation im Algorithmus Code zur Laufzeit

Variante 2 wird auf die selbe Art und Weise implementiert wie Variante 1, hier werden die Funktionen für die Generierung der Animation aber dazu verwendet, einen vom Algorithmus komplett unabhängigen Animationscode zu erstellen. Dieses Skript kann danach ausgeführt werden, ohne dass der ursprüngliche Algorithmus benötigt wird.

Variante 3 erfordert keinerlei zusätzlichen Code zur Animierung im eigentlichen Algorithmus, erfordert aber einen Parser, welcher den Code nach zu animierenden Anweisungen durchsucht und danach die Befehle für die Animation generiert.

Alle untersuchten Programme sind in der Lage beliebige Algorithmen zu animieren, es gibt in keinem eine Beschränkung auf Algorithmen die auf eine Datenstruktur angewendet werden oder Ähnliches.

Keines der untersuchten Programme setzt auf animierbare Operatoren, es kann also nicht auf eine bereits existierende Implementation zurückgegriffen werden.

Aufgrund dieser Punkte haben wir uns entschieden eine Lösung zu entwickeln die wie Balsa jeweils bei interessanten Operationen einen Event generiert. Das Kapitel „Konzept“ beschreibt genauer welche Events definiert wurden und wie diese jeweils erzeugt werden.

## 5. Konzept

### 5.1 Einleitung

Die Algorithmen für das Framework sollen möglichst ohne Änderungen bzw. zusätzlichen Code für die Animationen erstellt werden können. Da ein Herauslesen von Events und Variablennamen bei Verwendung der üblichen Typen nicht möglich ist, mussten in diesem Punkt einige Einschränkungen gemacht werden. Konkret bedeutet dies, dass bei der Erstellung animierter Algorithmen einige Spezielle Klassen zum Einsatz kommen. Sämtliche Datenstrukturen (Array, Liste, Binärbaum), Elemente in den Datenstrukturen sowie animierte „Zeigervariablen“ mussten durch eigene Klassen ersetzt werden, welche die nötigen Events generieren. Es wurde darauf Wert gelegt, dass sich die Anim-Klassen möglichst wie die Typen verhalten die sie ersetzen, eine vollständig gleiche Handhabung ist jedoch nicht möglich. Diese Dokument beschreibt diese eigenen Typen und deren Methoden und wie sie eingesetzt werden müssen.

Die ganze Animation von Algorithmen beruht auf dem Konzept von „Ereignissen“, wobei ein Ereignis eine „interessante“ Operation innerhalb der Ausführung des Algorithmus darstellt. Folgende Ereignisse wurden bei der Analyse verschiedener Algorithmen gefunden und definiert:

- Inserts
- Deletes
- Swaps
- Vergleiche
- Änderungen von „Zeigern“
- Wert gefunden

Es gibt drei verschiedene Arten von Klassen welche Ereignisse generieren:

- Datenstrukturen wie Array, Liste und Baum
- Datentypen wie AnimInt
- „Zeiger“ wie IndexInt

Die verschiedenen Arten von Klassen erzeugen jeweils verschiedene Arten von Ereignissen.

Die Ereignisse werden wo immer möglich im Hintergrund erzeugt, ohne dass der Programmierer sich während des Entwickeln des Algorithmus Gedanken um die Animation machen muss. Liefert diese Implementation noch nicht das gewünschte Ergebnis, so kann durch gezielte Tweaks am Code eine Veränderung der Animation erreicht werden.

Animierte Datenstruktur:

Animierte Datenstrukturen erzeugen jeweils Events, wenn eine Änderung am Inhalt der Datenstruktur vorgenommen wurde. Konkret passiert dies bei der Initialisierung der Datenstruktur, beim Einfügen, Löschen sowie Überschreiben eines Wertes. Des Weiteren löst auch jeder Lesezugriff auf ein Element innerhalb der Datenstruktur einen Event aus.

Datentyp:

Auf den Datentyp Klassen erzeugen sämtliche Vergleichsoperatoren einen Vergleichsevent. Dies wurde implementiert indem sämtliche Vergleichsoperatoren auf der Klasse überladen wurden.

Zeiger:

Zeiger werden jeweils mit ihrer aktuellen Position und Ihrem Namen angezeigt, wenn ein Zugriff auf ein Element in der Datenstruktur erfolgt. Sie können aber auch selber bei einer Änderung ihres Wertes einen Event erzeugen und so ihre aktuelle Position anzeigen.

## 5.2 Erzeugen und „Unterdrücken“ von Ereignissen

Methoden welche ein Ereignis erzeugen sind jeweils in zwei verschiedenen Varianten implementiert. Einerseits in einer Variante die einen `IndexInt` entgegennimmt und andererseits in einer Variante welche einen normalen Integer erwartet. Während Variante eins ein Ereignis erzeugt und die Position des übergebenen `IndexInt` sowie dessen Namen anzeigt, findet die Operation bei Methode „im Hintergrund“ ohne Darstellung einer Animation statt.

```
public T this[IndexInt index]
{
    get
    {
        if (index.Value >= elements.Length || index.Value < 0)
            throw new IndexOutOfRangeException();

        OnAnimationEvent(new IndexReadEventArgs(index));

        return elements[index.Value];
    }

    set
    {
        if (index.Value >= elements.Length || index.Value < 0)
            throw new IndexOutOfRangeException();

        value.Index = index.Value;
        elements[index.Value] = value;

        OnAnimationEvent(new IndexWriteEventArgs(index, elements[index.Value]));
    }
}
```

```
public T this[int index]
{
    get
    {
        if (index >= elements.Length || index < 0)
            throw new IndexOutOfRangeException();

        return elements[index];
    }
}
```

Indexer der `AnimArray` Klasse, oben mit Erzeugung eines Animationsevents, unten ohne.

### 5.3 Detailgrad

Der Detailgrad des Animationsablaufs lässt sich verstellen, wobei drei verschiedene Stufen möglich sind. Events die auf einer niedrigeren Detailstufe liegen werden jeweils auch angezeigt. Die verschiedenen Events sind in folgende Stufen unterteilt:

- Hoch  
Änderungen von IndexInt Variablen
- Mittel  
Lesen eines in einer Datenstruktur enthaltenen Elements
- Niedrig  
Ändern eines in einer Datenstruktur enthaltenen Elements  
Vertauschen von Elementen in einer Datenstruktur  
Einfügen eines Elements in eine Datenstruktur  
Löschen eines Elements aus einer Datenstruktur  
(Initialisierung des Algorithmus)  
(Ende eines Algorithmus)

Sämtliche Ereignisse die den Zustand der Datenstruktur ändern sind auf der niedrigsten Stufe und werden deshalb in jedem Fall angezeigt.

## 6. Technische Schwierigkeiten und Lösungsansätze

### 6.1 Verwendung von System.Array

Für die Implementation des Animierten Arrays wurde versucht eine eigene von System.Array abgeleitete Klasse zu erstellen. Wie sich herausstellte ist dies nicht möglich, MSDN liefert dazu folgende Erklärung:

Definition laut MSDN:

```
[SerializableAttribute]  
[ComVisibleAttribute(true)]  
public abstract class Array : ICloneable,  
    IList, ICollection, IEnumerable
```

MSDN Beschreibung:

Die Array-Klasse ist die Basisklasse für Implementierungen von Programmiersprachen, die Arrays unterstützen. Nur das System oder Compiler können jedoch explizit von der Array-Klasse ableiten. Benutzer sollten die von der Sprache zur Verfügung gestellten Arraykonstrukte verwenden.

Compilermeldung:

```
C:\Dokumente und Einstellungen\ch\Eigene Dateien\Visual Studio  
2008\Projects\projekt\WPF_Animationsframework\ImplementationTest\ImplementationTest\AnimArray.cs(8,1  
1): error CS0644: 'ImplementationTest.SubclassedArray' cannot derive from special class  
'System.Array'
```

Daraufhin wurde entschieden, das animierte Array als normale Klasse zu implementieren und einen Indexer zur Emulation eines Arrays zu verwenden.

MSDN Beschreibung:

Bei Indexern handelt es sich um ein syntaktisches Hilfsmittel, mit dem Sie Klassen, Strukturen und Schnittstellen erstellen können, auf die Clientanwendungen wie auf ein Array zugreifen können. Indexer werden am häufigsten in Typen implementiert, deren Hauptaufgabe darin besteht, eine interne Auflistung oder ein Array zu kapseln.

Dementsprechend wurde der animierte Array implementiert, ein internes, normales C# Array enthält die gespeicherten Elemente, auf welche über den Indexer zugegriffen werden kann.

### 6.2 Auslesen von Variablennamen

Die animierten Algorithmen sollen möglichst wenig zusätzlichen Code für die Animation benötigen, um möglichst nahe am ursprünglichen Code zu bleiben und die Animierung ohne zu großen Zeitaufwand zu ermöglichen.

Es stellten sich dabei folgende Probleme:

- Anzeigen von Zeigernamen
- Anzeigen von Zeigerpositionen
- Anzeigen von Vertauschungsoperationen
- Anzeigen von Einfügen/Löschen/Austauschen
- Anzeigen von Vergleichen

Die Animation erfolgt über eigene Datenstrukturen, wobei Array, LinkedList und AVL-Tree vom Framework zur Verfügung gestellt werden.

Die Zeigervariablen im Algorithmus sollen angezeigt und animiert werden, deshalb muss ihr Name bekannt sein. Ideal wäre ein automatisches Auslesen des Variablennamens aus dem Code, folgende Ansätze wurden für die Bestimmung der Variablennamen geprüft:

- Verwendung von Reflection
- Verwendung von Attributen
- Verwendung eines TypeDescriptors
- Verwendung einer Lambda Funktion
- Verwendung eines Indexers mit zusätzlichem Parameter
- Verwendung einer eigenen „IndexInt“ Klasse mit den erforderlichen Eigenschaften

Die Folgenden Klassen wurden schliesslich für die Erstellung von animierten Algorithmen implementiert:

- AnimArray, AnimList und AnimTree Klassen, welche Objekte vom Type AnimInt speichern und bei Änderungen (Einfügen/Löschen/Vertauschen) entsprechende Events auslösen.
- Eine AnimInt Struct welche bei Vergleichen mit einem anderen AnimInt einen entsprechenden Event auslöst.
- Eine IndexInt Struct welche es erlaubt einen Variablennamen zu setzen und Änderungen des Wertes zu verfolgen.

Das Verhalten dieser Typen wurde jeweils möglichst dem zu ersetzenden Element angeglichen um ein einfaches Austauschen zu ermöglichen.

### 6.3 Auslesen von Variablennamen

Da die Namen der Zeigervariablen („i“, „j“, etc.) angezeigt werden sollen, müssen die Namen der im Algorithmus verwendeten Variablen bekannt sein. Ursprünglich war geplant diese über Reflection auszulesen, wie sich herausgestellt hat ist dies aber nicht möglich, da die Namen der Variablen im Assembly nicht mehr vorhanden sind. Desweiteren ist es nicht möglich den Namen, welche die Variable bei der ursprünglichen Deklaration hatte auszulesen nachdem sie einer Methode übergeben wurde.

Um diese Probleme zu umgehen musste eine eigene Struct für die Zeiger erstellt werden, welche es erlaubt einen Namen zu setzen. Es wurde Wert darauf gelegt möglichst alle Eigenschaften eines normalen Integers zu reimplementieren, so ist es z. B. möglich eine Variable vom Typ IndexInt direkt einem normalen Integer zuzuweisen.

### 6.4 Erstellen einer animierten Array Klasse

Ursprünglich war es geplant System.Array als Basisklasse für die animierte Array-Klasse zu benutzen. Wie sich herausgestellt hat, ist es aber nicht möglich von System.Array abzuleiten, weshalb eine andere Lösung gefunden werden musste. Die animierte Array-Klasse ist nun als normale Klasse implementiert welche ein Array beinhaltet. Um den Zugriff auf die Elemente des inneren Arrays eines normalen Arrays anzugleichen wurde ein Indexer implementiert. Abgesehen von der Instantiierung lässt sich die animierte Array-Klasse wie ein gewöhnliches Array ansprechen und verwenden.

### 6.5 Abfangen von Events aus den Animationsklassen

Events können von mehreren verschiedenen Klassen ausgelöst werden, nämlich von IndexInt, AnimInt sowie den animierten Datenstrukturen. Für sämtliche Objekte dieser Klasse muss bei der Instanziierung ein Eventhandler gesetzt werden.

Dieses Problem wurde momentan gelöst, indem ein statischer Konstruktor den als Singleton implementierten EventProcessor als Eventhandler setzt.

## 6.6 WPF Animationen

### 6.6.1 Auslagern von UI Elementen in XAML

Die Extensible Application Markup Language (XAML) dient dazu, die Logik von der Darstellung zu trennen. Das XAML-File ist ein XML-basiertes Dokument, in dem man unter anderem angeben kann, wie ein UI-Element aussehen soll. Dies wird mit diversen XML-Tags und unzähligen Attributen zur Verfügung gestellt.

Die Animationen, die das Framework verlangt, sind dynamisch und sollen sich jeweils flexibel zur Laufzeit anpassen lassen. Da dieses dynamische Element im XAML nur beschränkt zur Verfügung steht, wurde der Ansatz so gewählt, dass alle grafischen Elemente nur programmier-technisch erstellt werden und so die volle Kontrolle über diese besteht.

Bald wurde jedoch klar, dass sich viele der Anzeigeelemente im Code wiederholen und zuweilen viele Linien Code benötigt wurden für ein einfaches grafisches Element.

Mit weiteren Erkenntnissen von XAML konnte eine Datenstruktur gefunden werden, die einfache Zugriffe auf definierte Elemente in einem XAML-File ermöglichten. Elemente und ganze Animationsabläufe können in einem sogenannten „ResourceDictionary“ gespeichert werden, auf das man, der Name deutet es bereits an, wie auf eine Dictionary-Datenstruktur zugreifen kann.

XAML wurde nun fast flächendeckend eingesetzt und Elemente bzw. Animationsabläufe dort ausgelagert, wo dies möglich war.

### 6.6.2 Doppelte Referenz auf XAML-Objekte

Sobald eine Zuweisung auf ein XAML-Objekt erfolgt ist, besteht eine Referenz darauf. Eine weitere Zuweisung auf dasselbe Objekt hat zur Folge, dass die Referenz neu gesetzt wird. Dies ist unproblematisch in Bezug auf Storyboards, da diese nach dem Auflösen wieder neu gesetzt werden können und die alte Referenz obsolet wird. Bei UI-Elementen ist dies jedoch unerwünscht, da Datenstruktur-Elemente mehrfach vorkommen müssen.

Ein Lösungsansatz dazu wäre das Klonen der Objekte. Dies ist jedoch nicht trivial, da es sich um verschachtelte Elemente handelt. Zudem fehlen den meisten UI-Elementen die cloning-Methoden. Diese Idee wurde daher schnell verworfen.

XAML bietet hierfür ein spezielles Attribut an, das jedoch erst nach längerem Suchen gefunden wurde. IntelliSense kennt dieses anscheinend jedoch auch nicht. Mit dem Attribut „x:Shared=‘false’“ kommt nun immer ein neues Objekt zurück, sobald man ein solches abrufen. Nun befindet sich unser Resource-Dictionary jedoch in einem eigenen XAML-File und verursacht bei diesem Attribut Probleme. Das Attribut steht nur zur Verfügung, wenn das Resource-Dictionary kompiliert ist. Wie man aber auf ein kompiliertes Resource-Dictionary zugreifen kann, konnte nicht heraus gefunden werden.

Folgender Lösungsansatz wurde schlussendlich gewählt: Display-Elemente werden direkt im Code erzeugt und nur die dazugehörigen Attribute im XAML definiert:

```
<ResourceDictionary x:Key="ArrayElement">
    <core:Double x:Key="Top">-75.0</core:Double>
    <core:Double x:Key="Left">200.0</core:Double>
    <core:Double x:Key="Width">50.0</core:Double>
    <core:Double x:Key="Height">50.0</core:Double>
    <core:Double x:Key="CornerRadius">10.0</core:Double>
    <core:Double x:Key="BorderThickness">2.0</core:Double>
</ResourceDictionary>
```

Mit dieser Methode können nun aber keine weiteren Attribute bestimmt werden, da nur diese bestehenden Attribute im Code berücksichtigt werden.

Dieser Lösungsansatz wurde jedoch nicht bei jedem Objekt so angewendet, da dies zu aufwändig und unflexibel geworden wäre. Dies bringt folgende Problematik mit sich: Bei einer hö-

heren Animationsgeschwindigkeit referenziert ein Storyboard zu einem Objekt, dass immer noch animiert wird. Da das Storyboard nun die momentane Position dieses Objektes ausliest und anschliessend um einen weiteren Wert animiert, kann es passieren, dass das Objekt an einer falschen Position steht. Passieren solche schnellen Zugriffe mehrmals hintereinander, verschlimmert sich die Fehlposition weiter.

### 6.6.3 Geschwindigkeitsregelung Animationen

Die Storyboard-Klasse in WPF sieht eine Geschwindigkeitsregelung der Animationen vor. Mit der Methode „SetSpeedRatio“ setzt man einen Faktor, um den die Animation beschleunigt werden soll. Es gibt keine feste Zahl, wie schnell eine Animation ablaufen soll. Dies ergibt sich aus der Tatsache, dass alle normalen Animationen interpoliert werden anhand eines Start- und Endwertes und der Dauer der Animation. Die einzige indirekte Möglichkeit um die Anfangsgeschwindigkeit festzulegen, besteht nun darin, die Dauer entsprechend anzupassen. Nachher kann man nur noch den Geschwindigkeitsfaktor anpassen.

Die Klasse „Storyboard“ lässt darauf schliessen, dass die Möglichkeit besteht, mehrere Storyboards hintereinander zu schalten und diese sequentiell ablaufen zu lassen. Animationen innerhalb eines Storyboards funktionieren grundsätzlich nach diesem Prinzip, allerdings muss man dort die Startzeit einer Animation entsprechend der Dauer der letzten Animation ausrichten. Storyboards jedoch starten unverzüglich, sobald diese gestartet wurden, ohne Rücksicht auf bereits gestartete Storyboards.

Für die Problemlösung dieses Verhaltens wurden einige Lösungsversuche ausprobiert, die schlussendlich jedoch kein befriedigendes Resultat hervor brachten.

- Ein On\_Completed-Event wird beim Beenden eines Storyboards ausgelöst. Im ersten Storyboard wurde nun in diesem Event anhand einer Queue das nächste Storyboard ausgelöst, das wiederum sein nächstes Storyboard ausgelöst hat. Dies wird solange wiederholt, bis die Queue abgearbeitet ist. Dieses Prinzip funktionierte soweit gut.
- Alle erzeugten Storyboards wurden in eine Liste eingefügt. Sollte sich die Geschwindigkeit ändern, wird durch die Liste hindurch iteriert und überall die Geschwindigkeit neu gesetzt.
- Ein übergeordnetes Storyboard wurde erstellt, das weitere Storyboards in sich aufnehmen kann. So sollte ein schnelles Anpassen des Geschwindigkeitsfaktors ermöglicht werden, da dies nur noch im übergeordneten Storyboard getan werden musste.
- Ein weiterer Versuch war der Aufruf einer Methode mit der die Startzeit bzw. die Animationsdauer der Storyboards beeinflusst werden konnte. Die Animationsdauer sollte bei jeder Änderung der Geschwindigkeit entsprechend geändert werden und anschliessend die Startzeit der nachfolgenden Storyboards angepasst werden.
- Ausserdem wurde noch nach anderen Lösungsansätzen gesucht, die jedoch alle nicht das eigentliche Problem beseitigen konnten.

Das sequentielle Aufrufen von den Storyboards konnte auf diverse Arten gelöst werden. Alle Lösungsansätze beseitigten jedoch das Hauptproblem nicht. Sobald man die Geschwindigkeit der Animation änderte, beschleunigten sich die momentan laufenden Animationen korrekt. Die nachfolgenden Animationen jedoch verhielten sich nicht korrekt. Die Animationsdauer wurde zwar korrekt gesetzt, jedoch hatte die Geschwindigkeit keinen Einfluss auf die Startzeit der Animation, so dass diese dennoch verzögert gestartet wurden. Das Problem besteht weiterhin, ein möglicher Lösungsansatz wurde nicht gefunden.

Die Geschwindigkeit kann dennoch indirekt beeinflusst werden. Der EventProcessor löst Events in einem bestimmten Intervall aus. Dieser Intervall kann angepasst werden, dies ist jedoch eine suboptimale Lösung, da so die Animationen schnell durcheinander geraten, wenn die Animationsgeschwindigkeit zu hoch sein sollte (siehe obiges Problem). Ausserdem kann die Geschwindigkeit so nicht live geändert werden.

## 7. Frameworkaufbau

### 7.1 Allgemein

In diesem Kapitel werden die einzelnen Elemente des Frameworks vorgestellt und beschrieben. Eine genauere Beschreibung der einzelnen Klassen und deren Methoden findet sich in der Code Dokumentation. Es soll hier nur eine grobe Übersicht gegeben werden.

### 7.2 Namespace und Klassenübersicht

Namespace	Klassen
<b>AnimationFramework</b>	
<b>AnimationFramework.Animation</b>	<ul style="list-style-type: none"> <li>• AnimationEquations</li> <li>• Animator</li> <li>• DisplayElement</li> </ul>
<b>AnimationFramework.Animation.Array</b>	<ul style="list-style-type: none"> <li>• ArrayAnimator</li> <li>• ArrayElement</li> </ul>
<b>AnimationFramework.Animation.List</b>	<ul style="list-style-type: none"> <li>• ListAnimator</li> <li>• ListElement</li> </ul>
<b>AnimationFramework.Animation.Tree</b>	<ul style="list-style-type: none"> <li>• TreeAnimator</li> <li>• TreeElement</li> </ul>
<b>AnimationFramework.Framework</b>	<ul style="list-style-type: none"> <li>• AlgorithmInfo</li> <li>• AlgorithmRunner</li> <li>• IAnimDataStructure</li> <li>• IEventProcessor</li> <li>• EventProcessor</li> <li>• Manager</li> </ul>
<b>AnimationFramework.Framework.Events</b>	<ul style="list-style-type: none"> <li>• AnimEventArgs</li> </ul>
<b>AnimationFramework.Framework.IndexEvents</b>	<ul style="list-style-type: none"> <li>• InitEventArgs</li> <li>• AlgorithmFinishedEventArgs</li> <li>• IndexReadEventArgs</li> <li>• IndexWriteEventArgs</li> <li>• SwapEventArgs</li> <li>• IndexValueChangedEventArgs</li> <li>• InsertEventArgs</li> <li>• RemoveEventArgs</li> </ul>
<b>AnimationFramework.Framework.SharedEvents</b>	<ul style="list-style-type: none"> <li>• CompareEventArgs</li> <li>• CompareValueEventArgs</li> <li>• CompareIndexValueEventArgs</li> </ul>
<b>AnimationFramework.Framework.ValueEvents</b>	<ul style="list-style-type: none"> <li>• InitEventArgs</li> <li>• AlgorithmFinishedEventArgs</li> <li>• IndexReadEventArgs</li> <li>• SwapEventArgs</li> <li>• InsertEventArgs</li> <li>• RemoveEventArgs</li> </ul>
<b>AnimationFramework.Types</b>	<ul style="list-style-type: none"> <li>• AnimArray</li> <li>• AnimInt</li> <li>• AnimList</li> <li>• AnimTree</li> <li>• IAnimatedElement</li> <li>• IndexInt</li> </ul>

## 7.3 Wichtige Klassen

### 7.3.1 AnimArray

Animiertes Array, das Elemente vom Typ `IAAnimatedElement` enthält und sich grösstenteils wie ein „normales“ Array verhält. Der einzige Unterschied ist die Art der Instanziierung und Initialisierung, welche wie bei einer herkömmlichen Klasse vorgenommen werden müssen. Bei der Initialisierung des Arrays sowie bei Zugriffen auf Array-Elementen werden Events ausgelöst.

### 7.3.2 AnimInt

Klasse für einen animierten Integer, wann immer ein `AnimInt` in einem Vergleich vorkommt, wird ein entsprechender Event generiert. `IndexInt`

Klasse für „Zeiger“ in Algorithmen, welche neben dem Integerwert auch einen Zeigernamen enthalten kann. Dieser Name wird bei einem Arrayzugriff im GUI angezeigt.

### 7.3.3 EventProcessor

Klasse welche sämtliche von `AnimArray`, `AnimInt` und `IndexInt` erzeugten Events entgegennimmt und verarbeitet. Der `EventProcessor` steuert wie detailliert die Animationen dargestellt werden sollen und gibt die Events danach zur Darstellung an das GUI weiter. Des Weiteren wird die Funktionalität für das Ändern der Geschwindigkeit des Animationsablaufs sowie das Pausieren und Schrittweise Durchsteppen des Algorithmus bereitgestellt.

### 7.3.4 Animator

Abstrakte Basisklasse, von der die `Array`, `List`, und `Tree-Animator`-Klassen ableiten. Hier sind Methoden und Klassenvariablen definiert, die von all diesen Klassen gebraucht werden können bzw. müssen. Z.B. werden hier die Elemente gespeichert, die nachher animiert werden.

### 7.3.5 Array-, List, Tree-Animator

Diese Klassen leiten von der abstrakten `Animator`-Klasse ab. Sie implementieren zudem entweder das `IAAnimationIndexHandler`- (`Array`, `List`) oder das `IAAnimationValueHandler`-Interface (`Tree`). Die Methoden, die von diesen Interfaces implementiert werden müssen, definieren wie die verarbeiteten Events animiert werden müssen. Mit diversen Event-Argumenten werden die erforderlichen Parameter für die Animation übergeben.

### 7.3.6 AnimationEquation

Enum-Klasse, die die verwendbaren `PennerDoubleAnimations` beinhaltet.

### 7.3.7 DisplayElement

Diese Klasse leitet von der `Border`-Klasse ab. Sie dient als Oberklasse der verschiedenen Datenstrukturelemente, die auf der Zeichenfläche animiert werden können.

In der Klasse selber befinden sich nur der Anzeigewert und eine Instanz zu einem `TextBlock`, in der der Wert dargestellt wird.

### 7.3.8 Array-, List-, Tree-Element

Diese Klassen leiten von der `DisplayElement`-Klasse ab und bestehen nur aus einem Konstruktor. Diesem wird ein Resource Dictionary übergeben, in dem Angaben wie Grösse und Position für das Element festgelegt sind. Im Resource-XAML-File können diese Werte geändert werden.

## 8. Implementation eines Algorithmus

### 8.1 Allgemein

Der auszuführende Algorithmus kann in einer beliebigen Klasse definiert werden, einzige Bedingung ist, dass die Signatur der Methode dem folgenden Delegate entsprechen muss:

```
- public delegate void AnimAlgorithm();
```

Ein bestehender Algorithmus kann auf einfache Art und Weise durch Austauschen der Datenstruktur sowie der Zeigervariablen durch vom Framework zur Verfügung gestellte Datentypen erreicht werden.

Das Framework bietet folgende Beispiialgorithmen an:

- Lineare Suche
- Binäre Suche
- Quicksort
- Heapsort
- Selectionsort
- Insertionsort

Die Animierten Versionen der binären Suche sowie von Quicksort werden in den folgenden Kapiteln vorgestellt und genauer beschrieben. Dabei wird gezeigt welche Anpassungen bei der Portierung gemacht werden müssen und wie der Animationsablauf beeinflusst werden kann.

### 8.2 Beispiel Binäre Suche

Das folgende Beispiel beschreibt welche Klassen und Methoden in einem Algorithmus verwendet werden müssen um den Korrekten Ablauf der Animation sicherzustellen. Dazu wird beschrieben, welche Events auf den jeweiligen Zeilen hinter den Kulissen ausgelöst werden.

```
public class AnimatedBinarySearch
{
    public void Run()
    {
        AnimInt value = new AnimInt(8);
        AnimArray<AnimInt> input = new AnimArray<AnimInt>(9, 2, 1, 5, 6, 8);
        IndexInt low = new IndexInt(0, "low");
        IndexInt high = new IndexInt(input.Length - 1, "high");

        while (low < high)
        {
            IndexInt mid = new IndexInt((high.Value + low.Value) / 2);

            if (input[mid] > value)
            {
                high = mid;
            }
            else if (input[mid] < value)
            {
                low = mid;
            }
            else
            {
                arr.FoundAt(mid, value);
                return;
            }
        }
    }
}
```

```
        arr.FoundAt(-1, value);  
        return;  
    }  
}
```

Der vollständige Algorithmus für die Binäre Suche.

### 8.3 Beispiel Quicksort

```
public class AnimatedQuickSort  
{  
    public void Run()  
    {  
        AnimArray<AnimInt> input = new AnimArray<AnimInt>(9, 2, 1, 5, 6, 8);  
        Quicksort(input, 0, input.Length - 1);  
    }  
  
    void Quicksort(AnimArray<AnimInt> input, IndexInt left, IndexInt right)  
    {  
        if (left < right)  
        {  
            int d = Divide(input, left, right);  
            Quicksort(input, left, d);  
            Quicksort(input, d + 1, right);  
        }  
    }  
  
    int Divide(AnimArray<AnimInt> input, IndexInt left, IndexInt right)  
    {  
        left.Name = "p";  
        AnimInt pivot = input[left];  
        IndexInt i = new IndexInt(left - 1, "i");  
        IndexInt j = new IndexInt(right + 1, "j");  
  
        while (i < j)  
        {  
            i++; while (input[i] < pivot) i++;  
            j--; while (input[j] > pivot) j--;  
            if (i < j)  
            {  
                input.Swap(i, j);  
            }  
        }  
  
        return j;  
    }  
}
```

Der vollständige Algorithmus für Quicksort.

## 9. Implementation einer Datenstruktur

### 9.1 Allgemein

Um eine Datenstruktur zu animieren muss diese das Interface `IAnimDataStructure` implementieren und einerseits Objekte von einem animierten Typ (`IAnimatedElement`) enthalten und andererseits bei einer Änderung der Datenstruktur einen entsprechenden Event auslösen. Datenstrukturen sind unterteilt in Indexbasierte und Wertbasierte Varianten.

### 9.2 Beispiel Array

Beispiel einer Animierten Datenstruktur anhand des animierten Arrays:

```
public class AnimArray<T> : IAnimDataStructure where T : IAnimatedElement
{
    public event AnimEvent AnimationEvent;
    private T[] elements;

    ...
}
```

Die Klasse generische Klasse `AnimArray` implementiert das Interface `IAnimDataStructure`, sie enthält Typen die wiederum das Interface `IAnimatedElement` implementieren. Sämtliche Animationsevents werden über den „AnimationEvent“ vom Type `AnimEvent` ausgelöst. Die im Array enthaltenen Werte werden in einem „normalen“ C# Array gespeichert.

```
public AnimArray(params T[] list)
{
    elements = list;
    InitializeArray();
}

public AnimArray(int size)
{
    elements = new T[size];
    InitializeArray();
}

public void Init(params T[] list)
{
    elements = list;
    InitializeArray();
}

private void InitializeArray()
{
    this.AnimationEvent =
        Manager.Instance.GetEventProcessor(Thread.CurrentThread.Name).QueueEvent;
    AnimationEvent(this, new InitEventArgs(this.Type, this.Length));
    SetIndex();
}
```

Es stehen zwei Konstruktoren zur Verfügung, beim einen kann eine beliebig viele Parameter zur Initialisierung übergeben werden, dem Anderen wird nur die gewünschte Grösse des Arrays mitgegeben. Die `Init` Methode erlaubt es bei einem bestehenden Array die enthaltenen Werte auszutauschen.

In der `InitializeArray` Methode `AnimationEvent` mit der `QueueEvent` des `EventProcessors` verbunden, dieser wird die Events unseres Arrays verarbeiten. Als nächstes wird gleich ein `InitE-`

vent erzeugt, welcher die Grösse und den Typ der Datenstruktur enthält. Schliesslich wird die Methode `SetIndex` aufgerufen um die Initialisierung des Arrays abzuschliessen.

```
private void SetIndex()
{
    for (int i = 0; i < Length; ++i)
    {
        elements[i].Index = i;

        OnAnimationEvent(new IndexWriteEventArgs(i, elements[i]));
    }
}
```

Die Methode `SetIndex` setzt den aktuellen Index auf den Elementen und übermittelt dem `EventProcessor` die Werte, mit welchen das `AnimArray` initialisiert wurde.

```
public T this[IndexInt index]
{
    get
    {
        if (index.Value >= elements.Length || index.Value < 0)
            throw new IndexOutOfRangeException();

        OnAnimationEvent(new IndexReadEventArgs(index));

        return elements[index.Value];
    }

    set
    {
        if (index.Value >= elements.Length || index.Value < 0)
            throw new IndexOutOfRangeException();

        value.Index = index.Value;
        elements[index.Value] = value;

        OnAnimationEvent(new IndexWriteEventArgs(index, elements[index.Value]));
    }
}
```

Um das Verhalten eines C# Arrays zu emulieren, wurde auf der `AnimArray` Klasse ein `Indexer` implementiert. Dieser erlaubt es über die bei Arrays üblichen eckigen Klammern einzelne Elemente zu lesen oder deren Wert zu ändern. Bei beiden Aktionen wird jeweils ein entsprechender Event erzeugt.

```
public T this[int index]
{
    get
    {
        if (index >= elements.Length || index < 0)
            throw new IndexOutOfRangeException();

        return elements[index];
    }
}
```

Es steht ein überladener `Indexer` für den Lesezugriff bereit, welcher einen normalen `int` entgegennimmt. Im Unterschied zum vorhergehenden `Indexer` erzeugt dieser keinen Event, was es ermöglicht die gewünschte Animation genauer zu steuern.

```
public void Swap(IndexInt i, IndexInt j)
{
    T temp = elements[i.Value];
    elements[i.Value] = elements[j.Value];
    elements[j.Value] = temp;

    OnAnimationEvent(new SwapEventArgs(i, j));
}
```

Die Swap Methode vertauscht die Elemente an den Positionen i und j und erzeugt den entsprechenden Event.

```
public void HighlightElement(IndexInt index)
{
    if (index >= elements.Length || index < 0)
        throw new IndexOutOfRangeException();

    OnAnimationEvent(new IndexReadEventArgs(index));
}
```

Falls ein Element des Arrays besondere Aufmerksamkeit verdient, so kann es über die HighlightElement Methode aktiviert werden. Es wird dabei dieselbe Animation ausgelöst wie bei einer Leseoperation.

```
protected void OnAnimationEvent(AnimEventArgs e)
{
    if (AnimationEvent != null)
    {
        AnimationEvent(this, e);
    }
}
```

Sämtliche Events die das animierte Array erzeugt werden von der OnAnimationEvent Methode ausgelöst.

```
public int Length
{
    get
    {
        if (elements == null)
        {
            return 0;
        }
        else
        {
            return elements.Length;
        }
    }
}
```

Wie bei einem normalen C# Array liefert das Length Property die Grösse des Arrays.

## 10. Einbinden und starten eines animierten Algorithmus

Um einen Algorithmus auszuführen sind neben der Methode welche den Algorithmus implementiert folgende Objekte nötig:

- Der Manager
- Ein passender Animator

Dem Manager wird die Methode übergeben, welche den zu animierenden Algorithmus enthält. Die Signatur der übergebenen Methode muss dabei dem folgenden Delegate entsprechen:

- `public delegate void AnimAlgorithm();`

Die Methode wird dem Manager beim Starten des Algorithmus übergeben. Die Algorithmen werden jeweils über die `CreateAlgorithm` Methode auf dem Manager erstellt und durch eine eindeutige Bezeichnung identifiziert. Die Ausführung des Algorithmus erfolgt in einem eigenen Thread, was das Ausführen verschiedener animierten Algorithmen zur selben Zeit erlaubt.

Nachfolgend der Code zur Ausführung eines Algorithmus anhand eines Beispiels mit einem animierten Array:

```
Manager animManager = Manager.Instance;  
  
AnimatedLinearSearch linearSearch = new AnimatedLinearSearch();  
IAnimationHandler arrayAnimator = new ArrayAnimator(canvas);  
animManager.CreateAlgorithm("LinearSearch", arrayAnimator);  
animManager.RunAlgorithm("LinearSearch", linearSearch.Run);
```

Danach kann die animierte Anzeige des Algorithmus gestartet werden:

```
Manager.Instance.Start("LinearSearch");
```

Pausieren des Algorithmus:

```
Manager.Instance.Pause("LinearSearch");
```

Zurücksetzen des Algorithmus:

```
Manager.Instance.Reset("LinearSearch");
```

Entfernen des Algorithmus:

```
Manager.Instance.RemoveAlgorithm("LinearSearch");
```

Standardmässig wird der `EventProcessor` automatisch erzeugt, es ist jedoch auch ein explizites Setzen eines eigenen `EventProcessors` möglich:

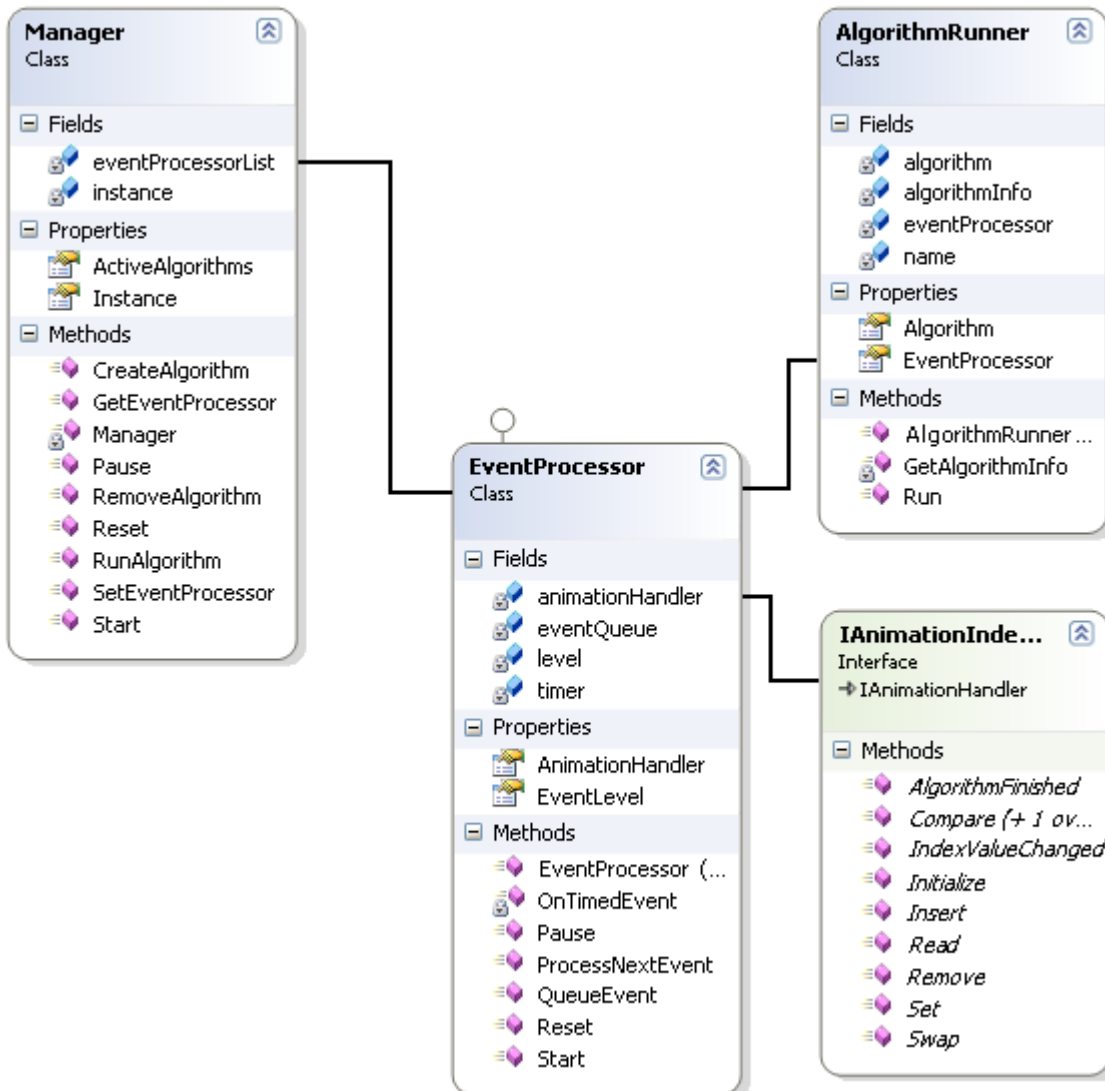
```
animManager.SetEventProcessor("LinearSearch",  
                               new EventProcessor(arrayAnimator));
```

Liste der aktiven Algorithmen:

```
String[] algorithms = animManager.ActiveAlgorithms;
```

## 10.1 Klassenaufbau

Folgendes Diagramm zeigt die Beziehungen zwischen Manager, AlgorithmRunner, EventProcessor und IAnimationHandler:



Übersicht mit Manager, AlgorithmRunner, EventProcessor und AnimationHandler

Im Manager wird ein EventProcessor erzeugt, welcher wiederum einen IanimationHandler besitzt, an den er die Events für die Anzeige weiterleitet. Der AlgorithmRunner wird bei der Ausführung des animierten Algorithmus erzeugt und führt diesen aus.



<b>Implementation Array</b>	3 Tage	10.11.09	12.11.09	Roger Fankhauser	31.01.10
<b>Implementation Liste</b>	3 Tage	13.11.09	17.11.09	Roger Fankhauser	31.01.10
<b>Implementation Tree</b>	3 Tage	18.11.09	20.11.09	Cedric Hollenstein	31.01.10
<b>Animationen abgeschlossen</b>	0 Tage	20.11.09	20.11.09		31.01.10
<b>Projektwoche</b>	5 Tage	30.11.09	04.12.09	Cedric Hollenstein, Roger Fankhauser	04.12.09
<b>Dokumentation API</b>	50.5 Tage	15.10.09	24.12.09	Cedric Hollenstein, Roger Fankhauser	31.01.10
<b>Dokumentations- / Reflexionphase</b>	24.5 Tage	24.12.09	31.01.10		31.01.10
<b>Dokumentation abgeschlossen</b>	0 Tage	24.12.09	24.12.09		31.01.10
<b>Abgabe</b>	0 Tage	24.12.09	24.12.09		31.01.10
<b>Präsentation</b>	8.5 Tage	24.12.09	05.01.10	Cedric Hollenstein, Roger Fankhauser	-
<b>Reserve</b>	16 Tage	06.01.10	31.01.10		31.01.10

Tabellarische Auflistung der Tasks

Wie man aus der Tabelle lesen kann, konnte der Zeitplan schon am Anfang nicht eingehalten werden. Mit dem Fortlauf des Projektes wurde die Verzögerung immer grösser, was schliesslich dazu führte, dass nicht alle Animationen realisiert werden konnten.

## 12. Eingesetzte Tools

### 12.1 Microsoft Visual Studio

Die offizielle Entwicklungsumgebung für die Programmierung in C#. Sämtliche Programmierarbeiten sowie die Verwaltung von Quelltext und Dokumenten erfolgten über Visual Studio. Da wir früher bereits mit Visual Studio gearbeitet hatten, war der Aufwand zur Einarbeitung minimal.

### 12.2 Microsoft Team Foundation Server

Die Verwaltung der Projektdokumente sowie des Quellcodes erfolgte über den von der FHNW zur Verfügung gestellten Team Foundation Server. Die Sourcecode-Verwaltung ist vollständig in Visual Studio integriert und erlaubt durch automatisches auschecken von bearbeiteten Dateien ein angenehmes Arbeiten.

### 12.3 Microsoft Sandcastle

Für die Erstellung der Dokumentation wurden die Kommentare aus dem Quelltext per Visual Studio in eine XML-Datei exportiert und von Sandcastle in eine HTML-basierte Dokumentation umgewandelt. Sandcastle wurde verwendet, weil die offizielle .NET Dokumentation von Microsoft auf dieselbe Weise erstellt wurde. Um die Konfiguration der Generierung zu vereinfachen wurde das GUI Programm „Sandcastle Help File Builder“ verwendet.

## 13. Reflexion/Lessons Learned

Das WPF-Animationsframework-Projekt war für uns beide interessant und äusserst fordernd. Algorithmen zu erkennen und diese dann zu animieren ist keine triviale Aufgabe, hinzu kommt, dass wir sehr wenig Programmiererfahrung mit C# und WPF hatten. Die meiste Zeit haben wir dazu aufgewendet, uns durch Bücher, durchs Internet und unzählige Foren zu lesen. Ausserdem war es vor allem im Bereich der Animationen recht aufwendig, geeignete Lösungen zu finden, da die meisten Dokumentationen nicht in die gewünschte Tiefe gegangen sind.

Frustrierend war die Tatsache, dass eigentlich trivial erscheinende Aufgaben zu richtigen Stundenfressern wurden. Hatte man sich mit diesen auseinander gesetzt und schlussendlich lösen können, kamen bereits die nächsten Hürden. Dies beruht sicherlich auch auf der mangelnden Erfahrung mit diesen neuen Technologien.

Trotz all dem konnten wir eine Menge über C#, WPF, Visual Studio und allgemein über die Funktionsweise von Algorithmen lernen. Das nächste .NET-Projekt wird uns beiden auf jeden Fall leichter fallen.

Dass wir das Projekt letztendlich trotz der aufgewendeten Zeit nicht erfolgreich abgeschlossen haben, hat verschiedene Gründe. Einerseits genügte die Art des Projektmanagements nicht und andererseits war die Erfahrung nicht genügend um wirklich effizient zu arbeiten. Hinzu kamen sicherlich auch einige Mängel in der Softwarearchitektur.

## 14. Quellenangaben

### Offizielle Seiten:

Microsoft MSDN:

<http://msdn.microsoft.com/en-us/library/default.aspx>

### Tutorials:

WPF Tutorial, Christian Moser:

<http://wpftutorial.net/Home.html>

### Andere Algorithmen-Frameworks / allg. Algorithmen:

Monash University:

<http://www.csse.monash.edu.au/~dwa/Animations/index.html>

Animating Recursive Algorithms Linda Stern, The University of Melbourne:

<http://imej.wfu.edu/articles/2002/2/02/index.asp>

John Morris, Auckland:

[http://www.cs.auckland.ac.nz/software/AlgAnim/alg\\_anim.html](http://www.cs.auckland.ac.nz/software/AlgAnim/alg_anim.html)

Sorting Algorithms:

<http://www.sorting-algorithms.com/>

Brian's Project Gallery

<http://www.brian-borowski.com/Software/Sorting/>

Dr. Nikolai Bezroukov, Sort Algo Efficiency:

<http://www.softpanorama.org/Algorithms/sorting.shtml>