

Prüfer: Prof. Dr. rer. nat. Kurt Rothermel

Betreuerin: Dr. rer. nat. Cora Burger

Betreuer: Dipl.-Inform. Rolf Mecklenburg

begonnen am: 1.08.1997

beendet am: 1.02.1998

CR-Nummer: C.2.2, C.2.4, H.5.1, H.5.2, K.3.1, K.3.2

Studienarbeit Nr. 1670

**" Erweiterung des JAVA-Baukastens zur  
Visualisierung von Protokollen: Weiter -  
entwicklung der Benutzerschnittstelle,  
Undo - Funktionalität "**

Papoulidis Konstantinos

Fakultät Informatik  
Institut für Parallele und Verteilte  
Höchstleistungsrechner  
Universität Stuttgart  
Breitwiesenstr. 20-22  
70565 Stuttgart

# Inhaltsverzeichnis I

<b>Kapitel 1 : Einleitung .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Aufgabenstellung .....	1
1.3 Überblick über die Studienarbeit .....	2
<b>Kapitel 2 : Anforderungen an den neuen Baukasten .....</b>	<b>3</b>
2.1 Beschreibung des alten Baukastens .....	3
2.2 Beschreibung des neuen Baukastens .....	4
2.3 Vergleich alter und neuer Baukasten .....	4
<b>Kapitel 3 : Gesamtentwurf des Baukastens .....</b>	<b>6</b>
3.1 Beschreibung über das Container-Management .....	6
3.2 Beschreibung der Visualisierungselemente .....	12
3.3 Beschreibung der Benutzerschnittstelle .....	13
3.3.1 " Button " - Panel .....	13
3.3.2 " TextArea " - Panel .....	13
3.3.3 " ImageString " - Panel .....	14
3.3.4 " Statuszeile " - Panel .....	15
3.4 Beschreibung der Undo-Architektur .....	16
3.4.1 Aufgaben der Klasse Undo .....	16
3.4.2 Aufgaben der Container Klassen .....	16
3.4.3 Aufgaben der Visualisierungselemente .....	17
3.4.4 Ablauf des Undo - Prozesses .....	17
3.4.5 Schematische Darstellung der Klassen und Stacks .....	17

## Inhaltsverzeichnis II

3.5 Beschreibung des Zeitmanagements .....	20
<b>Kapitel 4 : Entwurf des speziellen Teils .....</b>	<b>21</b>
4.1 Entwurf der Benutzerschnittstelle .....	21
4.2 Die Undo - Verwaltung der Klasse SDLContainer .....	24
4.2.1 Die Verwaltung der Zustandsinformationen .....	24
<b>Kapitel 5 : Implementierung des speziellen Teils .....</b>	<b>26</b>
5.1 Die Implementierung der Undo - Funktionalität der SDLKlassen ...	26
5.1.1 Undo - Informationen der Klasse SDLContainer .....	26
5.1.2 Undo - Informationen der Klasse SimpleSDLNode .....	28
5.1.3 Undo - Informationen der Klasse AnimatedSDLNode .....	30
5.1.4 Undo - Informationen der Klasse SimpleDigitalTimer .....	30
5.1.5 Undo - Informationen der Klasse AnimatedDigitalTimer .....	33
5.1.6 Undo - Informationen der Klasse SimpleSDLEdge .....	33
5.1.7 Undo - Informationen der Klasse FlashingSignal .....	35
5.1.8 Undo - Informationen der Klasse SimpleTravellingSignal .....	36
5.1.9 Undo - Informationen der Klasse AnimatedTravellingSignal .	40
5.2 Entwurfsänderung des " ImageString " - Panel .....	41
5.3 Lösung des Anpassungsproblem Image - Canvas .....	41
<b>Kapitel 6 : Zusammenfassung und Ausblick .....</b>	<b>43</b>
6.1 Zusammenfassung .....	43
6.2 Ausblick .....	43
<b>Kapitel 7 : Literatur .....</b>	<b>44</b>
<b>Anhang</b>	

## Abbildungsverzeichnis I

Abbildung 1: Time Sequence - Diagramm eines RPC .....	3
Abbildung 2: Graphendiagramm eines RPC .....	4
Abbildung 3: Überblick über neuen und alten Baukasten .....	5
Abbildung 4: logisches auf physisches Koordinatensystem .....	6
Abbildung 5: Logische Attribute eines Knotens .....	7
Abbildung 6: Logische Attribute einer Kante .....	7
Abbildung 7: Logische Attribute eines Signals .....	8
Abbildung 8: Klassenstruktur Container - Management .....	8
Abbildung 9: Klassenstruktur - Layout Visualisierungselemente .....	9
Abbildung 10: Event Trace - Diagramm - Layout durch den Container .....	10
Abbildung 11: Event Trace - Diagramm - Layout durch die Anwendung .....	10
Abbildung 12: Container - Klassenhierarchie .....	11
Abbildung 13: Klassenstruktur der Visualisierungselemente .....	12
Abbildung 14: " Button " - Panel .....	13
Abbildung 15: " TextArea " - Panel .....	14
Abbildung 16: " ImageString " - Panel .....	14
Abbildung 17: " Statuszeile " - Panel .....	15
Abbildung 18: Struktur der Klasse Undo .....	16
Abbildung 19: Undo Struktur der Klasse VisContainer .....	17
Abbildung 20: Undo - Ablauf als Event Trace Diagramm .....	18
Abbildung 21: Undo - Ablauf mit Stackdarstellung .....	19
Abbildung 22: Klassendiagramm Clock - Klasse und ihre Listener .....	20
Abbildung 23: " Button " - Panel .....	21
Abbildung 24: Änderung des " Pause " Button zu " Resume " .....	22
Abbildung 25: Vom Normalzustand in den " Stepping - Mode " .....	22
Abbildung 26: Auswirkungen bei gedrücktem " FastForward " - Button .....	23
Abbildung 27: SDL - Klassen .....	24
Abbildung 27: SDL - Stacks .....	25
Abbildung 28: Visualisierungsschritte der Klasse SDLContainer .....	26
Abbildung 29: Stackinhalt - SDL Container deaktiviert .....	27
Abbildung 30: Stackinhalt - SDL Container aktiviert .....	27
Abbildung 31: Visualisierung der Klasse SimpleSDLNode .....	28
Abbildung 32: Stackinhalt - SimpleSDLNode angezeigt .....	28
Abbildung 33: Stackinhalt - SimpleSDLNode deaktiviert .....	29
Abbildung 34: Stackinhalt - SimpleSDLNode aktiviert .....	29

## Abbildungsverzeichnis II

Abbildung 35: Stackinhalt - SimpleSDLNode gelöscht .....	30
Abbildung 36: Visualisierungsschritte SimpleDigitalTimer.....	30
Abbildung 37: Stackinhalt - SimpleDigitalTimer angezeigt .....	31
Abbildung 38: Stackinhalt - SimpleDigitalTimer gestartet und läuft noch .....	31
Abbildung 39: Stackinhalt - SimpleDigitalTimer gestartet und abgelaufen .....	32
Abbildung 40: Stackinhalt - SimpleDigitalTimer gestartet und gestoppt .....	32
Abbildung 41: Stackinhalt SimpleDigitalTimer gelöscht .....	33
Abbildung 42: Visualisierungsschritte der Klasse Simple SDLEdge .....	33
Abbildung 43: Stackinhalt - SimpleSDLEdge angezeigt .....	34
Abbildung 44: Stackinhalt - SimpleSDLEdge deaktiviert .....	34
Abbildung 45: Stackinhalt - SimpleSDLEdge aktiviert .....	35
Abbildung 46: Stackinhalt - SimpleSDLEdge gelöscht .....	35
Abbildung 47: Visualisierungsschritt der Klasse FlashingSignal .....	36
Abbildung 48: Stackinhalt - FlashingSignal angezeigt .....	36
Abbildung 49: Visualisierungsschritte der Klasse Simple TravellingSignal .....	37
Abbildung 50: Stackinhalt - Simple TravellingSignal angezeigt und noch unterwegs .....	37
Abbildung 51: Stackinhalt - SimpleTravellingSignal angezeigt und schon angekommen .....	38
Abbildung 52: Stackinhalt - SimpleTravellingSignal angezeigt, verfälscht und noch unterwegs .....	38
Abbildung 53: Stackinhalt - SimpleTravellingSignal angezeigt, verfälscht und schon angekommen .....	39
Abbildung 54: Stackinhalt - SimpleTravelling Signal angezeigt und verloren .....	39
Abbildung 55: Stackinhalt - SimpleTravellingSignal, verfälscht und verloren .....	40
Abbildung 56: Image - Canvas Anpassung .....	41

# **1. Einleitung**

## **1.1 Motivation**

Die Bereiche Rechnernetze, Verteilte Systeme und Kooperation in Verteilten Systemen basieren auf dem Austausch von Nachrichten zwischen Komponenten über einen Kommunikationskanal, wobei sich die Komponenten an ein bestimmtes Kommunikationsprotokoll halten. Die verschiedenen Ablaufvarianten eines Protokolls lassen sich sehr gut mit Hilfe einer Animation und der Möglichkeit zur Beeinflussung (Interaktion) durch den Benutzer (zum Beispiel Entfernen einer Nachricht vom Kanal) durchspielen.

Durch die Interaktion kann also der Benutzer in den Ablauf des Kommunikationsprotokolls eingreifen und verfolgen wie das Protokoll reagiert. Protokolle werden in der Lehre mit Momentaufnahmen erläutert, was nicht unbedingt zum Verständnis des Betrachters beiträgt. Die Bewegung von Bildern also Animation und die Möglichkeit des Eingreifens durch den Betrachter, trägt dazu bei, eine bessere Vorstellung für den Ablauf von Protokollen der scheinbar nicht zusammenhängenden Momentaufnahmen zu erhalten.

Also ein besseres Verständnis für Protokolle zu erlangen in der für ihn entsprechenden Geschwindigkeit.

Durch die Visualisierung der Protokolle kann der Benutzer (Vortragende als auch Zuschauer) sich auf die Funktionalität der Protokolle konzentrieren. Der Vortragende muß sich nicht mit der Darstellung des Protokolls befassen, denn dies übernimmt der "Baukasten".

Am Institut für Parallele und Verteilte Höchstleistungsrechner in der Abteilung Verteilte Systeme der Universität Stuttgart befaßt sich die Projektgruppe "ProtoVis" mit der Darstellung und Animation von Protokollen.

In einer bereits abgeschlossenen Arbeit wurde ein "Baukasten" in der Sprache Java entwickelt, der es gestattet, in einfacher Weise den Austausch von Nachrichten zwischen unterschiedlichen Prozessen zu visualisieren.

Aufgabe dieser Studienarbeit ist diesen Baukasten konzeptionell weiterzuentwickeln und die neuen Konzepte zu implementieren.

## **1.2 Aufgabenstellung**

Bei dieser Studienarbeit bestand die Aufgabenstellung aus sieben Teilaufgaben. Die erste Teilaufgabe war die Einarbeitung in die Programmiersprache Java. Dies war erforderlich, da vor der Bearbeitung der Studienarbeit keine Programmierkenntnisse über Java vorlagen. Weiterhin mußte sowohl die Einarbeitung in den Visualisierungsbaukasten von Peter W. Schurr als auch in das Themengebiet "Benutzerschnittstellen" erfolgen.

Als zweite Teilaufgabe wurde die Portierung des Visualisierungsbaukastens (Peter W. Schurr) auf die Java-Version 1.1.2 verlangt. In der Teilaufgabe drei sollten die aktuellen Fähigkeiten des Visualisierungsbaukastens zusammen mit seinen Defiziten beschrieben werden.

Bezüglich den Bedienelementen zur Manipulation des Animationsverhaltens wurde in Teilaufgabe vier die Verbesserung der Benutzerschnittstelle des Baukastens erwähnt.

Die fünfte Teilaufgabe verlangte eine Spezifikation und prototypische Implementation einer

"Undo" - Funktionalität zur Rücknahme bestimmter Visualisierungsschritte zu erstellen. Teilaufgabe Sechs forderte ein Demonstrationsapplet mit den neuen Funktion zu erzeugen. In der letzten Teilaufgabe wurde eine schriftliche Ausarbeitung, in der Motivation, Bewertung der Werkzeuge, Dokumentation von Konzepten, Implementierung und Testergebnisse verlangt wurde. Eine multimediale Präsentation und Vorführung der Ergebnisse der Arbeit, sollten zu einem späteren Zeitpunkt in einem Vortrag im Rahmen des VS-Kolloquiums erfolgen.

### **1.3 Überblick über die Studienarbeit**

In Kapitel 1 wird sowohl die Aufgabenstellung dieser Studienarbeit beschrieben als auch die dazugehörige Motivation erläutert. Bevor im Kapitel 3 auf die Beschreibung der Benutzerschnittstelle und "Undo" - Architektur eingegangen wird, werden als gemeinsamer Teil der beiden Studienarbeiten von Ralph Gorth mit dem Thema: " Erweiterung des Java-Baukastens zur Visualisierung von Protokollen: Erweiterung des Baukastens zur Visualisierung von SDL-Konstrukten " und Konstantinos Papoulidis mit dem Thema " Erweiterung des Java-Baukastens zur Visualisierung von Protokollen: Weiterentwicklung der Benutzerschnittstelle, Undo - Funktionalität " im Kapitel 2 die Unterschiede des alten und neuen Baukastens aufgezeigt. Es wird darauf hingewiesen, daß diese genannten Studienarbeiten eng miteinander verzahnt sind.

In Kapitel 3 wird der Gesamtentwurf des neu zu implementierenden Visualisierungsbaukastens dargelegt. Der Entwurf des speziellen Teils wird im Kapitel 4 beschrieben.

In Kapitel 5 werden Implementierungsdetails beschrieben. Die Zusammenfassung und ein Ausblick erfolgen im Kapitel 6. Das Literaturverzeichnis befindet sich in Kapitel 7.

Im Anhang werden die Methoden die das Anwendungsprogramm aufruft mit den dazugehörigen Parametern und ihrer Funktion aufgeführt.

Als Java - Entwicklungsumgebung wurde das Java Developer's Kit (JDK, Version 1.1) verwendet.

## 2. Anforderungen an den neuen Baukasten

### 2.1. Beschreibung des alten Baukastens

Der alte Baukasten von Peter Schurr ermöglicht die Visualisierung von Protokollen mit Hilfe von Time Sequence - Diagrammen. Dabei werden die Prozesse entlang einer Zeitachse von oben nach unten aufgetragen und die Nachrichtenübermittlung wird durch Pfeile vom sendenden zum empfangenden Prozeß in Richtung der fortschreitenden Zeit dargestellt. Auf eine Implementierung von Nachrichtenkanälen wurde verzichtet:

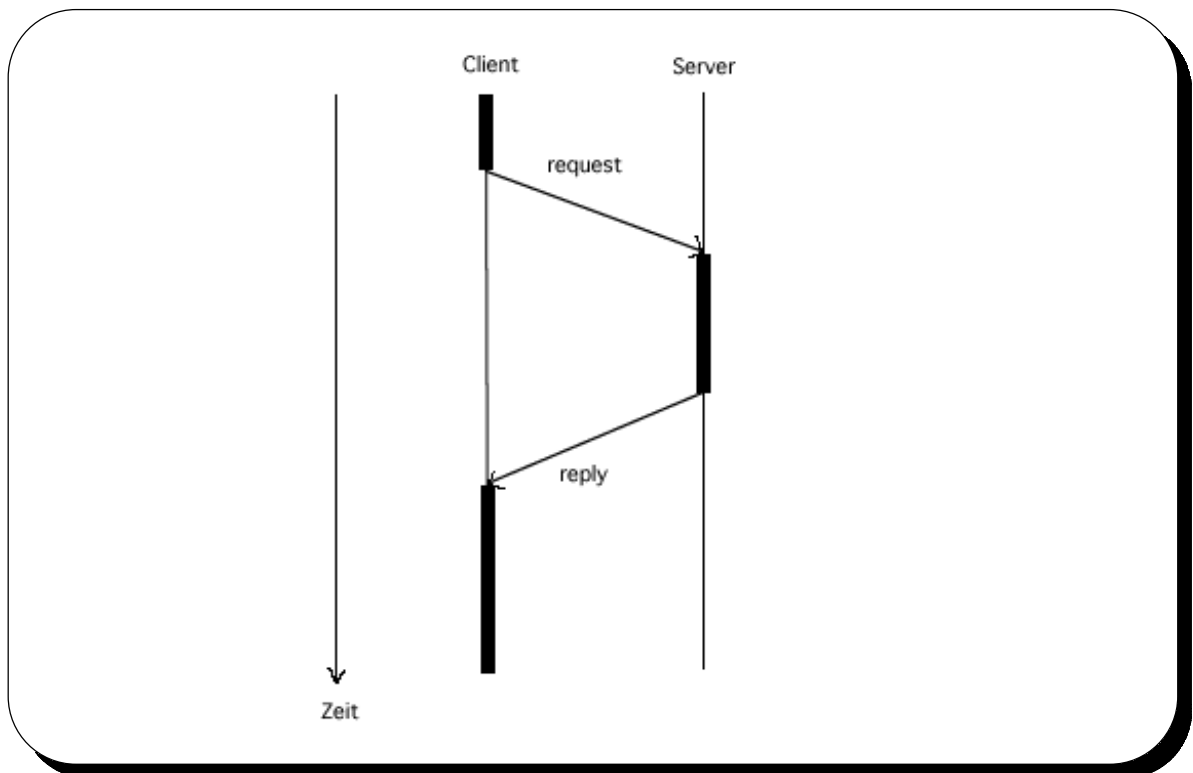


Abbildung 1 : Time Sequence - Diagramm eines RPC

Diese Darstellungsmethode hat den Vorteil, daß sie einfach am Bildschirm darzustellen ist, da Prozesse immer nebeneinander angeordnet sind und sich dadurch neue Prozesse parallel zur Zeitachse leicht einfügen lassen. Außer der leichten Darstellung hat sie den Vorteil, daß vergangene Nachrichtenübermittlungen im Blickfeld bleiben und dadurch auch längere Interaktionen von Prozessen sichtbar sind.

## 2.2. Beschreibung des neuen Baukastens

Der neue Baukasten ermöglicht dagegen die Visualisierung von Protokollen indem er eine graphenbasierte Lösung anbietet. Die Knoten sind dabei die Prozesse, die Zustandsverbindungen die Nachrichtenkanäle und Nachrichten werden durch Aktivitäten auf den Kanälen dargestellt:

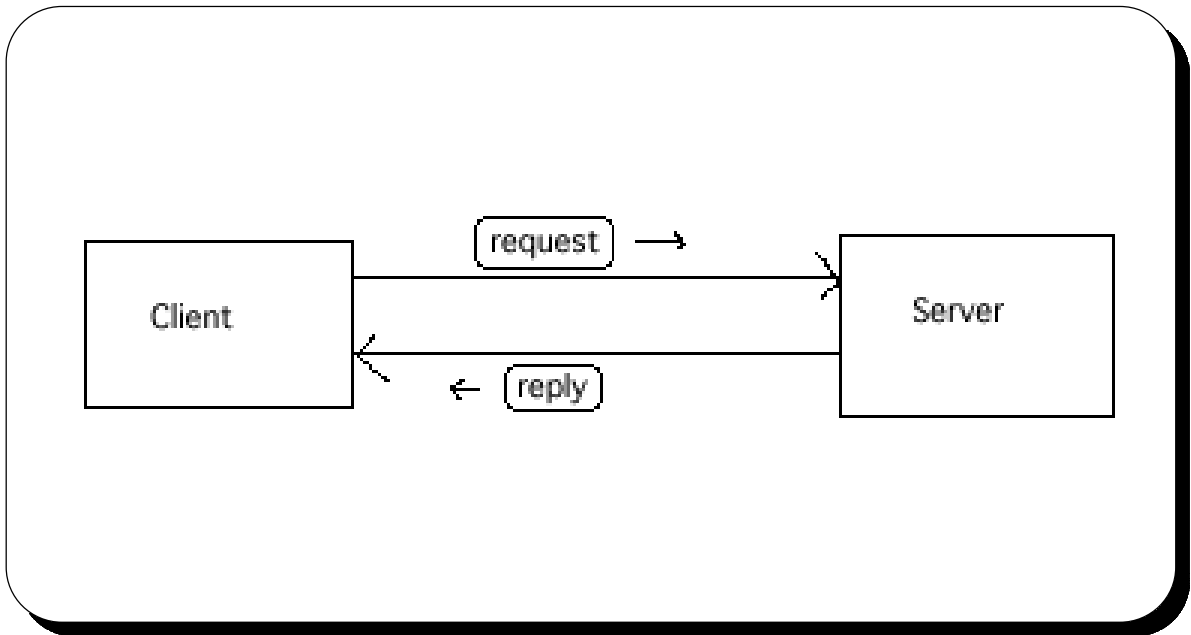


Abbildung 2 : Graphendiagramm eines RPC

Diese Methode wurde anhand der Protokollspezifikationsprache SDL (**Specification and Description Language**), die ebenfalls auf Graphen basiert, gewählt. Zusätzlich dazu bietet SDL noch eine Visualisierungsmethode, um endliche Automaten darzustellen. Diese Methode ist ebenfalls im Entwurf des Baukastens enthalten aber noch nicht implementiert

## 2.3. Vergleich alter und neuer Baukasten

Folgende Tabelle (auf der folgenden Seite) bietet einen kompakten Überblick über den alten und den neuen Ansatz des Visualisierungsbaukastens:

	alter Ansatz prozedural über Assistent	neuer Ansatz objekt - orientiert durch Klassenbibliothek
Art der Programmierschnittstelle		
Tiefe der Programmierschnittstelle	high-level durch Methoden für Broad-/Multicast	low-level durch umfassende Manipulationsmöglichkeiten
Visualisierungsmethode	Time Sequence	Graph
Fehlerfälle	Nachrichtenverlust und -verfälschung, Auftauchen von Nirvana - Nachrichten (diese beiden genannten Punkte beziehen sich auf beide Ansätze)	
Benutzerschnittstelle	ja, eingeschränkt	ja, erweitert
Geschwindigkeitssteuerung	ja	ja
Undo - Funktionalität	ja	ja
Erweiterbarkeit	schlecht	gut durch einfache Klassenhierarchie
Kanal - Konzept	nein	ja
Java - Version	1.0.2	1.1

Abbildung 3 : Überblick über neuen und alten Baukasten

## 3. Gesamtentwurf des Baukastens

### 3.1. Beschreibung des Container-Managements

#### Layout im logischen Koordinatensystem

Für die Darstellung der Protokollvisualisierung ist ein Container verantwortlich, der für das Layout der Visualisierungselemente zuständig ist. Da jedoch die Dimensionen der Darstellungsfläche am Bildschirm abhängt von i.) der Größe des Applets, das den Container samt Visualisierung darstellt ii.) dem Layout aller Oberflächenelemente im Applet iii.) und dynamischen Größenänderungen

- durch Hinzufügen und Entfernen von Oberflächenelementen
- und durch Änderungen des Browser - Fensters muß das Layout unabhängig von der physischen Auflösung in einem logischen Koordinatensystem stattfinden:

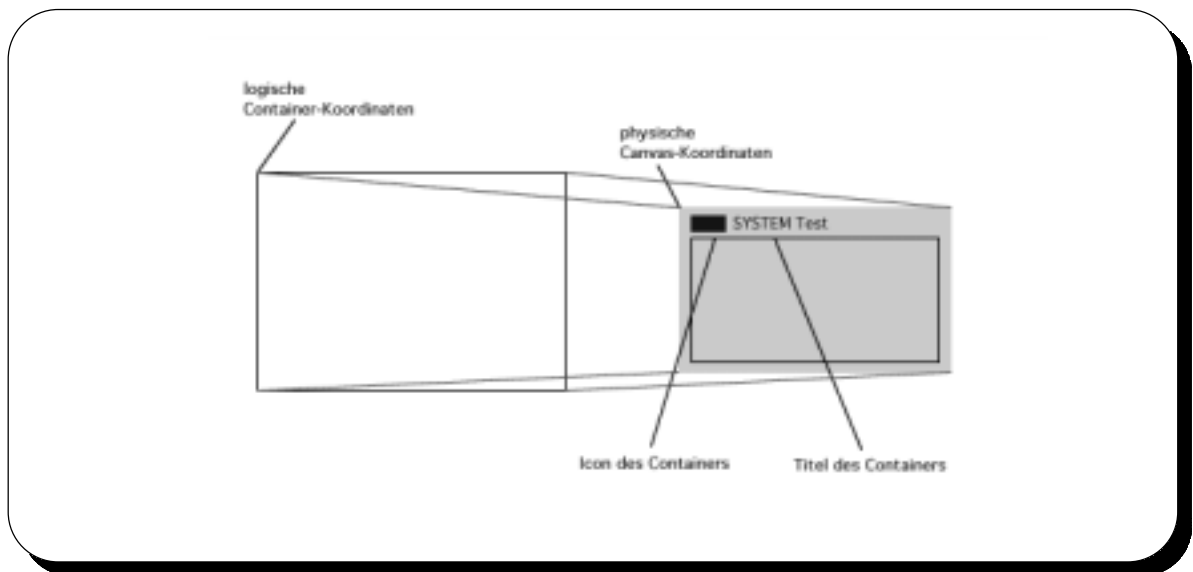


Abbildung 4 : logisches auf physisches Koordinatensystem

Die Dimensionen des logischen Koordinatensystems sind als Attribut des Containers frei wählbar.

#### Logische Größe der Visualisierungselemente

Die Größe der Visualisierungselemente hängt ab von i.) der Größe des Containers ii.) und der Anzahl der Elemente im Container. Deshalb müssen Position sowie Größe ebenfalls im logischen Koordinatensystem angegeben werden können.

## Knoten

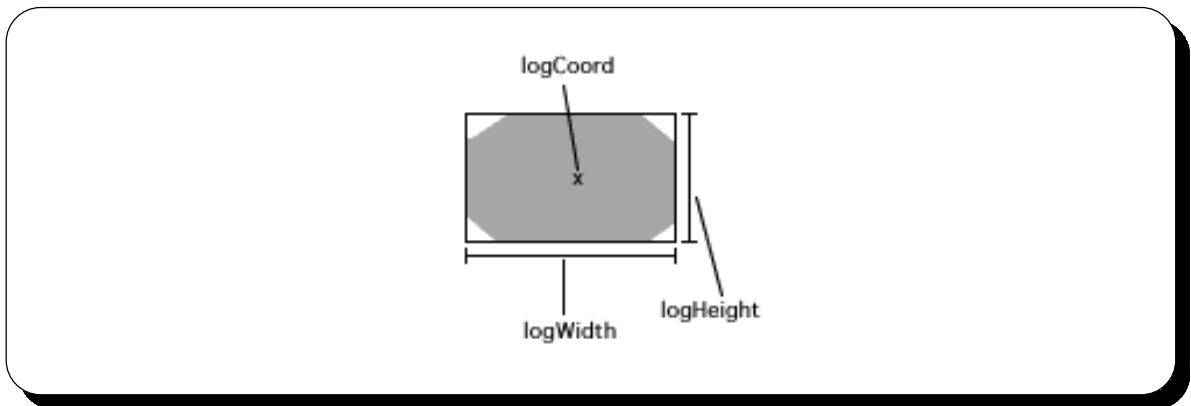


Abbildung 5 : Logische Attribute eines Knotens

Die Koordinaten eines Knotens sind die Koordinaten des Mittelpunkts des ihn umgebenden Rechtecks; die Größe des Knotens wird mittels der Breite und Höhe dieses Rechtecks angegeben.

## Kante

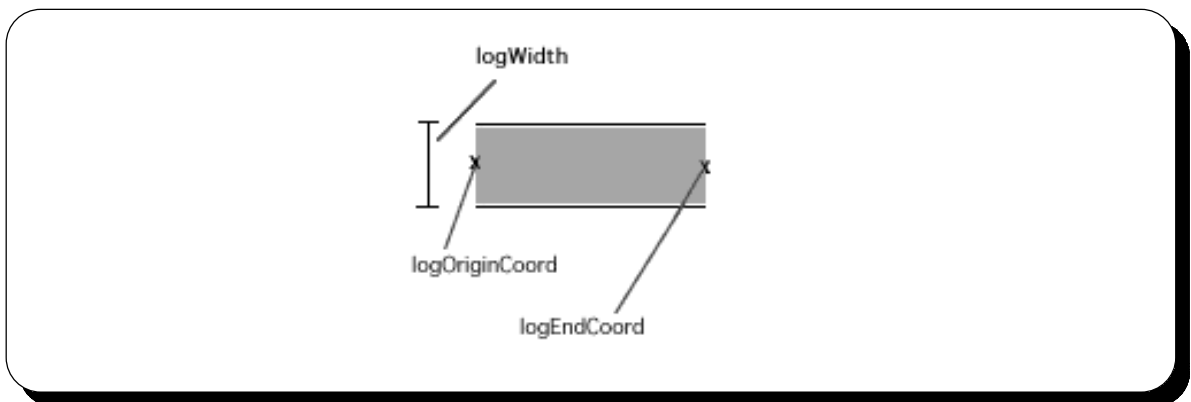


Abbildung 6 : Logische Attribute einer Kante

Die Größe einer Kante ist der die Kante umschließende Korridor; die Anfangs- und Endkoordinaten der Kante sind die Mittelpunkte des Korridors an den jeweiligen Enden.

## Signal

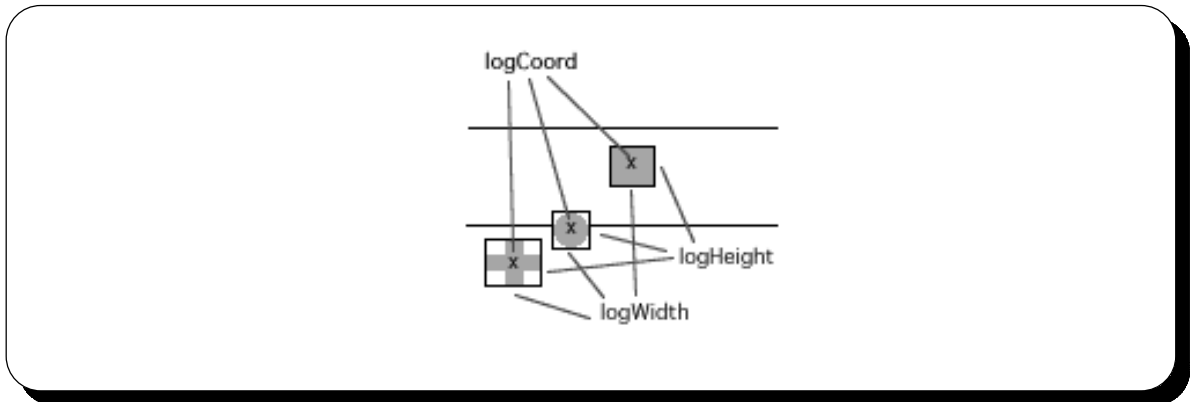


Abbildung 7 : Logische Attribute eines Signals

Bei Signalen, die eine " Teilchendarstellung " haben, d.h. als wanderndes Signal und nicht als aufblitzende Kante dargestellt werden, sind die Koordinaten die des Mittelpunkts des sie umgebenden Kreises; die Größe wird mittels des Radius dieses Kreises angegeben.  
Daraus ergibt sich folgende vorläufige Klassenstruktur für das Container - Management:

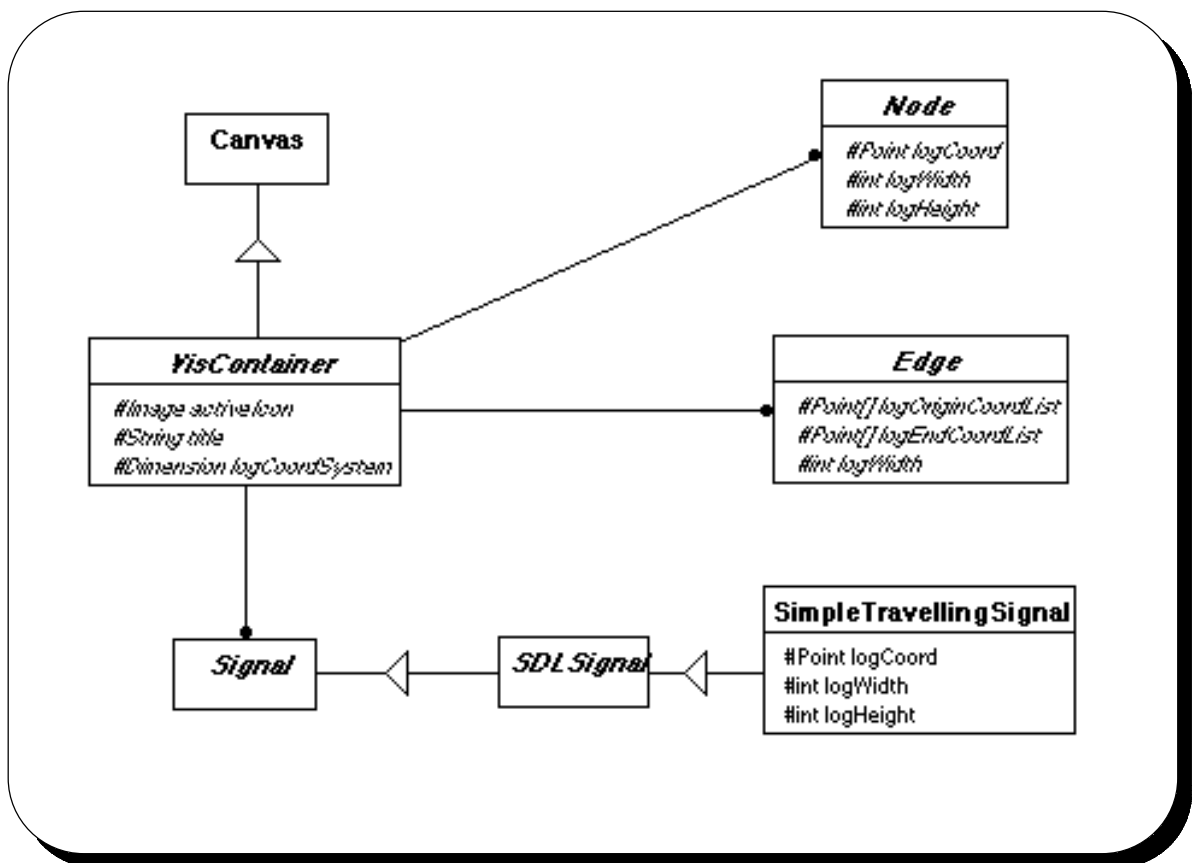


Abbildung 8 : Klassenstruktur Container - Management

## Layout der Visualisierungselemente

Das Layout, d.h. die Platzierung und Größe der Visualisierungselemente, übernimmt der Container in dem diese dargestellt werden. Das folgende Klassendiagramm zeigt die dafür nötigen zusätzlichen Attribute, Methoden und Klassen:

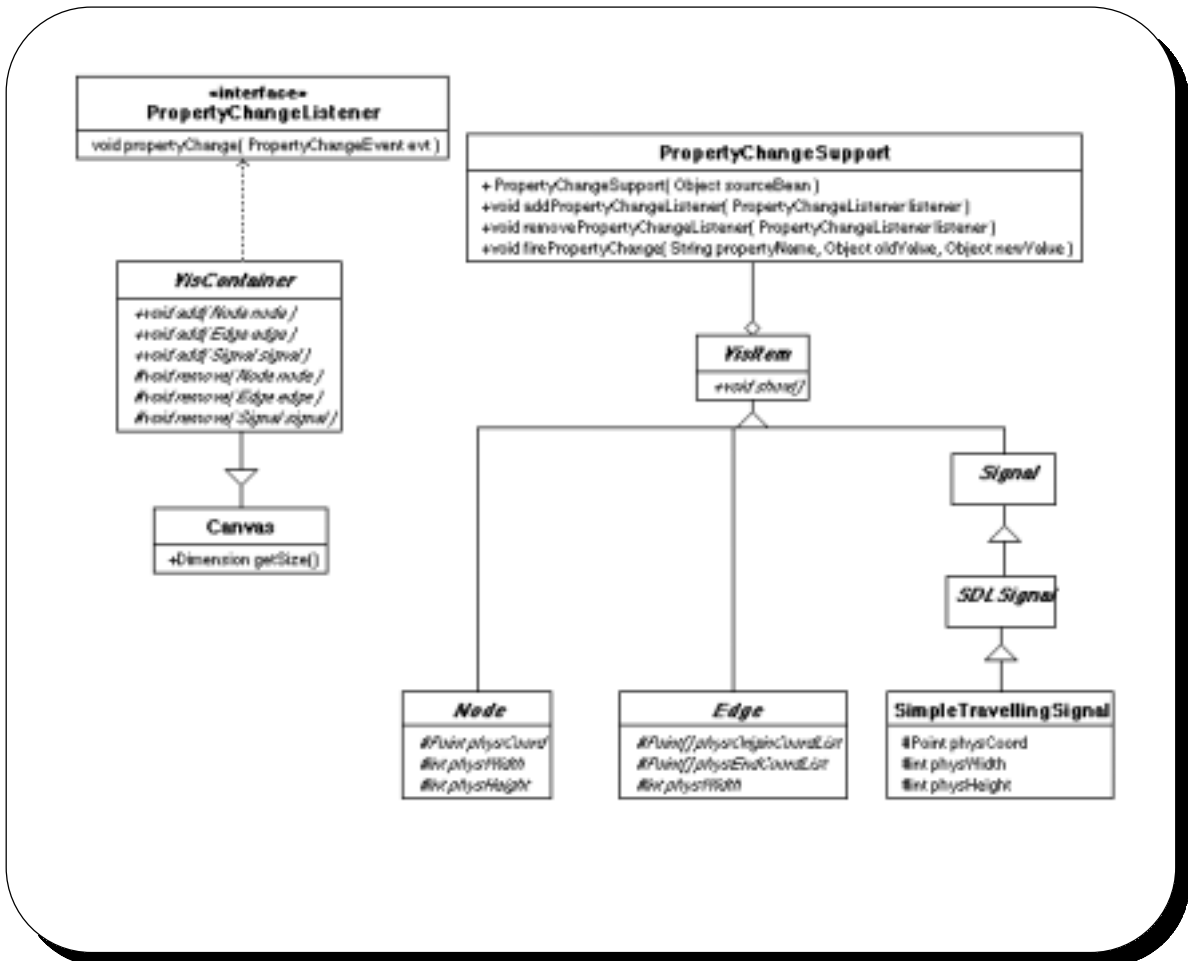


Abbildung 9 : Klassenstruktur - Layout Visualisierungselemente

## Layout durch den Container

Folgendes Event - Trace - Diagramm zeigt wie das Layout vom Hinzufügen zum Container bis zur Darstellung am Bildschirm abläuft:

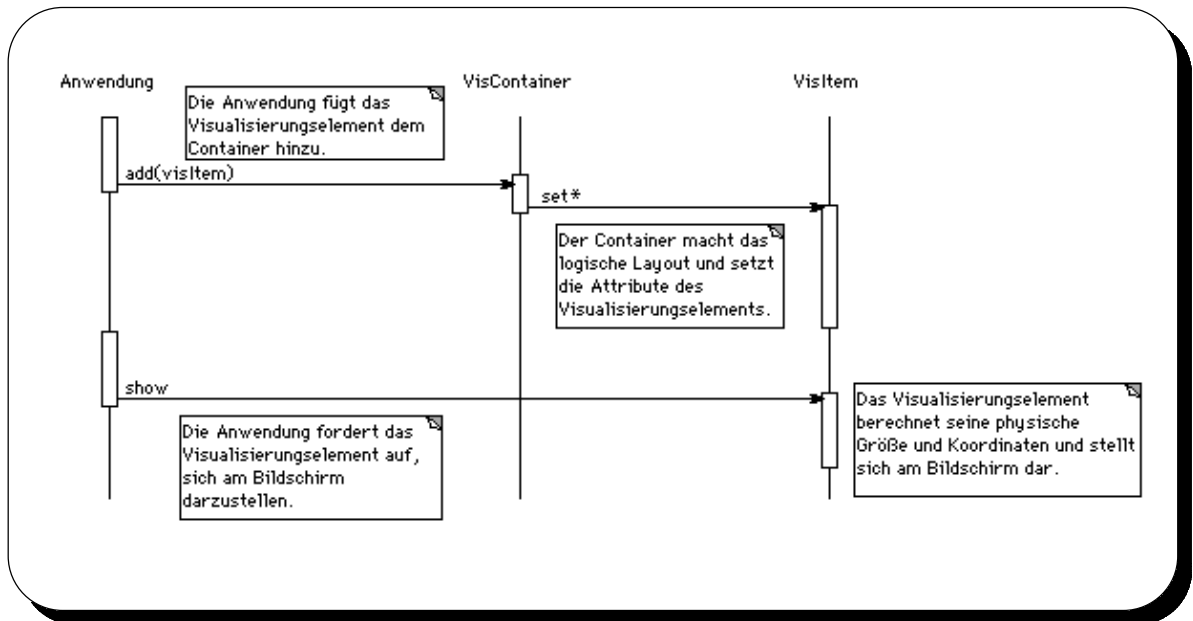


Abbildung 10 : Event Trace - Diagramm - Layout durch den Container

## Layout durch die Anwendung

Um der Anwendung, die den Baukasten benutzt, zu ermöglichen das automatische Layout des Containers zu modifizieren oder komplett selbst zu übernehmen, ist es auch möglich, daß die Anwendung die logischen Koordinaten selbst setzt. Folgendes Event-Trace-Diagramm zeigt ein Szenario bei Eingreifen der Anwendung:

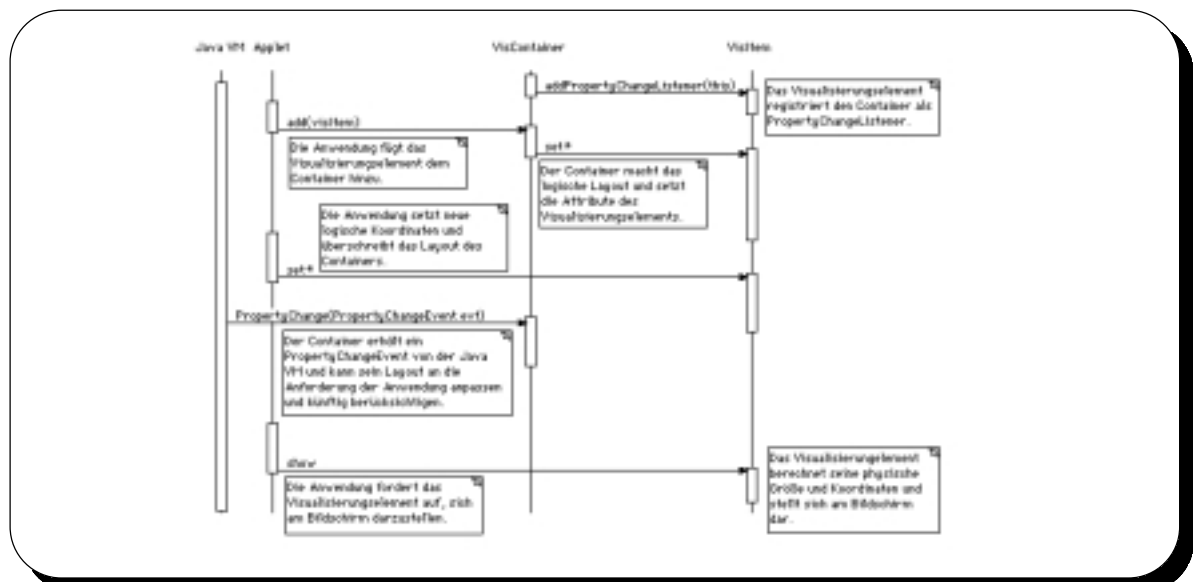


Abbildung 11 : Event Trace - Diagramm - Layout durch die Anwendung

## Container - Klassenhierarchie

Hier nun die gesamte Klassenhierarchie der Containerproblematik:

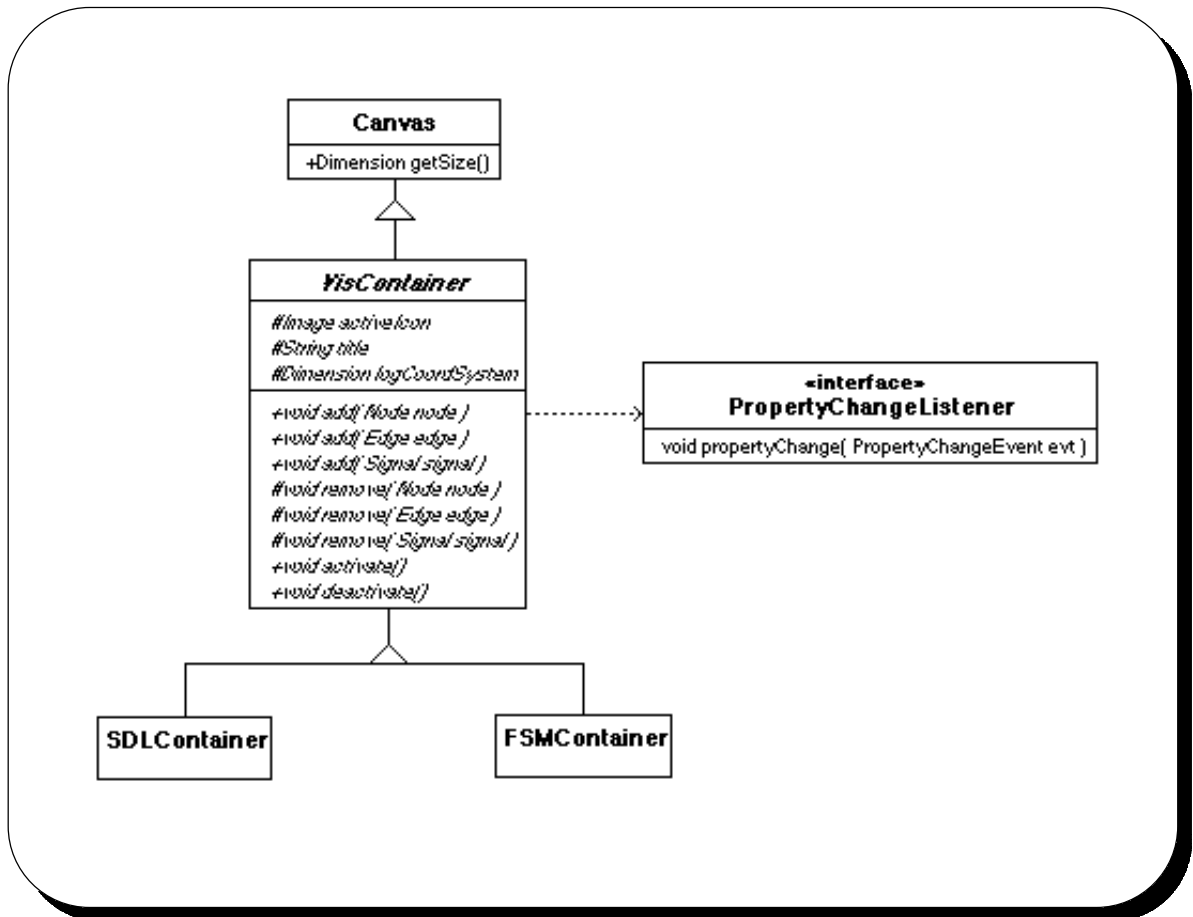


Abbildung 12 : Container - Klassenhierarchie

Die beiden Methoden `activate` und `deactivate` dienen dazu, den Container aktiv und inaktiv darzustellen, wenn das dazugehörige Visualisierungselement aktiv oder inaktiv dargestellt wird (zum Beispiel in einem Prozeßinteraktionsdiagramm wird ein Prozeß inaktiv gesetzt, dann muß der **FSMContainer**, der den Prozeß darstellt, ebenfalls inaktiv dargestellt werden können).

### 3.2. Beschreibung der Visualisierungselemente

Grundsätzlich werden 3 Kategorien von Elementen, die bei der Visualisierung von Protokollen zum Einsatz kommen, unterschieden:

- i.) Knoten, ii.) Kanten, die Knoten verbinden, iii.) Signale, die über Kanten laufen.
- Darauf aufbauend ergibt sich folgende Klassenstruktur:

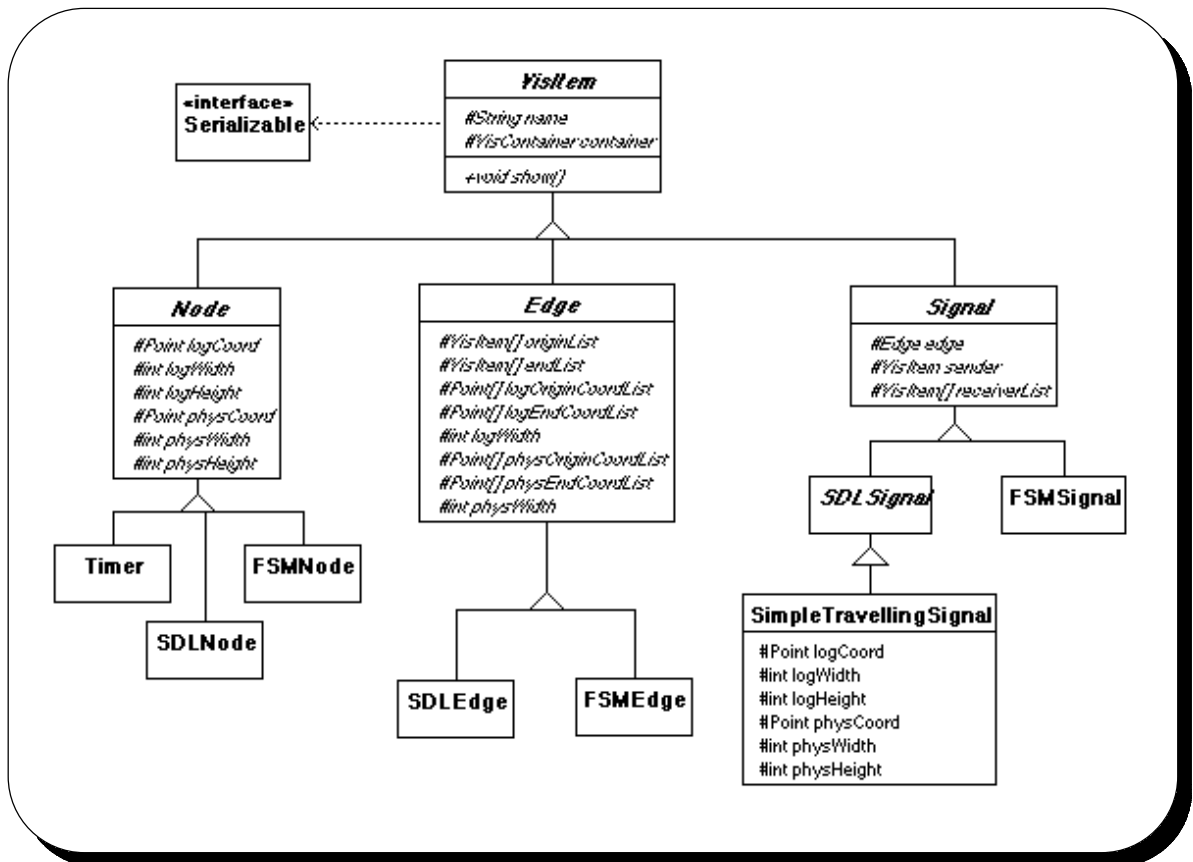


Abbildung 13 : Klassenstruktur der Visualisierungselemente

Diese 3 Kategorien werden wieder in zwei Gruppen aufgeteilt, die für die Visualisierung von SDL- und FSM - Strukturen (Fine State Machine) zuständig sind.

### 3.3 Beschreibung der Benutzerschnittstelle

Die Benutzerschnittstelle für den neu zu implementierenden Visualisierungsbaukasten, der auf eine graphenbasierte Lösung bei der Visualisierung von Protokollen ansetzt, wird durch die folgenden 4 Panels realisiert:

- a.) einem " Button " - Panel
- b.) sowie " TextArea " - Panel
- c.) als auch ein " ImageString " - Panel
- d.) und einem " Statuszeile " -Panel.

#### 3.3.1 " Button " - Panel

Das " Button " - Panel wird für die Interaktion des Benutzers (Betrachters) mit dem Anwendungsprogramm benötigt. Mit dessen Hilfe kann er auf die Animationsdarstellung (Visualisierung) Einfluß nehmen. Dadurch ist es möglich, ein besseres Verständnis für die meist komplexen Protokolle zu erreichen. Abbildung 14 zeigt einen Screenshot des implementierten " Button " - Panel, daß im Kapitel 4 Abschnitt 1 näher erläutert wird.

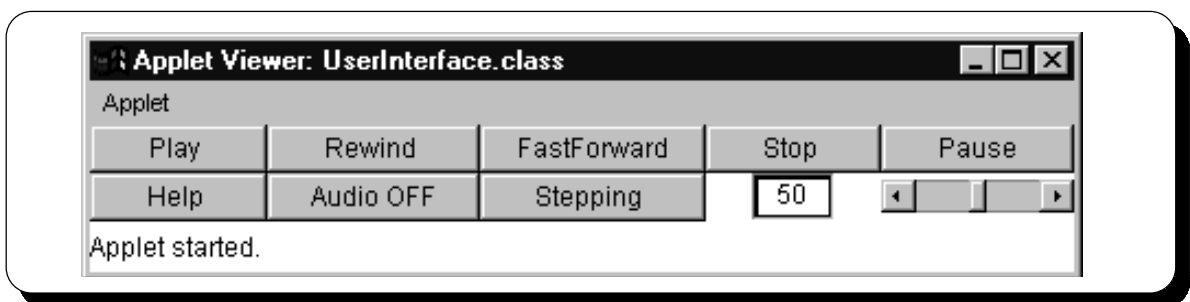


Abbildung 14 : " Button " - Panel

#### 3.3.2 " TextArea " - Panel

Ein " TextArea " - Panel (Textbereich) wird dem Anwendungsprogramm zur Verfügung gestellt um beliebige Informationen (zum Beispiel: ergänzende Informationen zu einem Protokoll während des Ablaufs) anzuzeigen.

Auf der folgenden Seite wird hierzu ein Beispiel gezeigt:

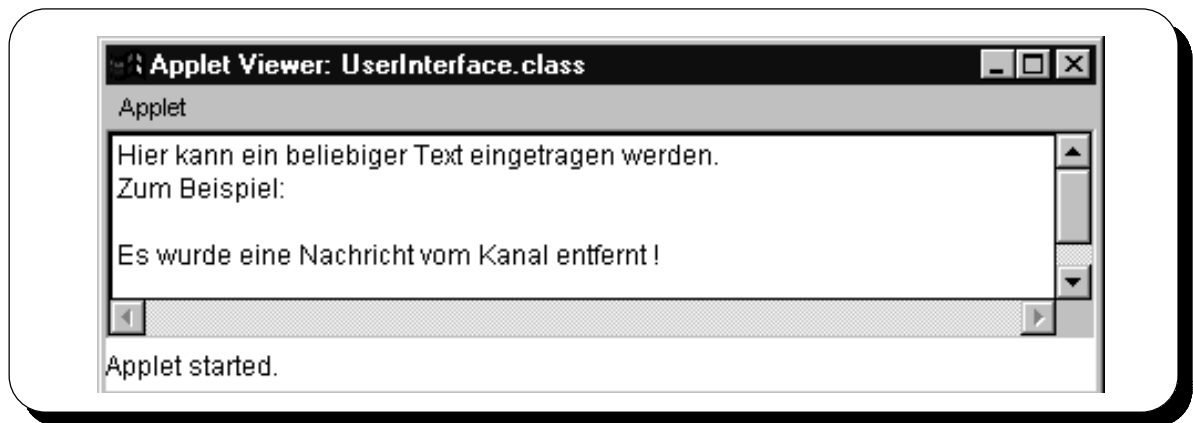


Abbildung 15 : " TextArea " - Panel

### 3.3.3 " ImageString " - Panel

Dieses " ImageString " - Panel das als " Legende " dient, wird der Anwendung zusätzlich angeboten, da in den Visualisierungscontainern übersichtshalber keine Beschriftungen für die Visualisierungselemente gewählt wurden und so die Elemente mit einem Icon und einer Beschriftung in dem " ImageString " - Panel eingetragen werden können.

Dadurch kann die Visualisierung verständlicher gemacht werden. Um dies zu verdeutlichen betrachten Sie hierzu folgendes Screenshot mit zufällig ausgewählten Icons und Beschriftungen:

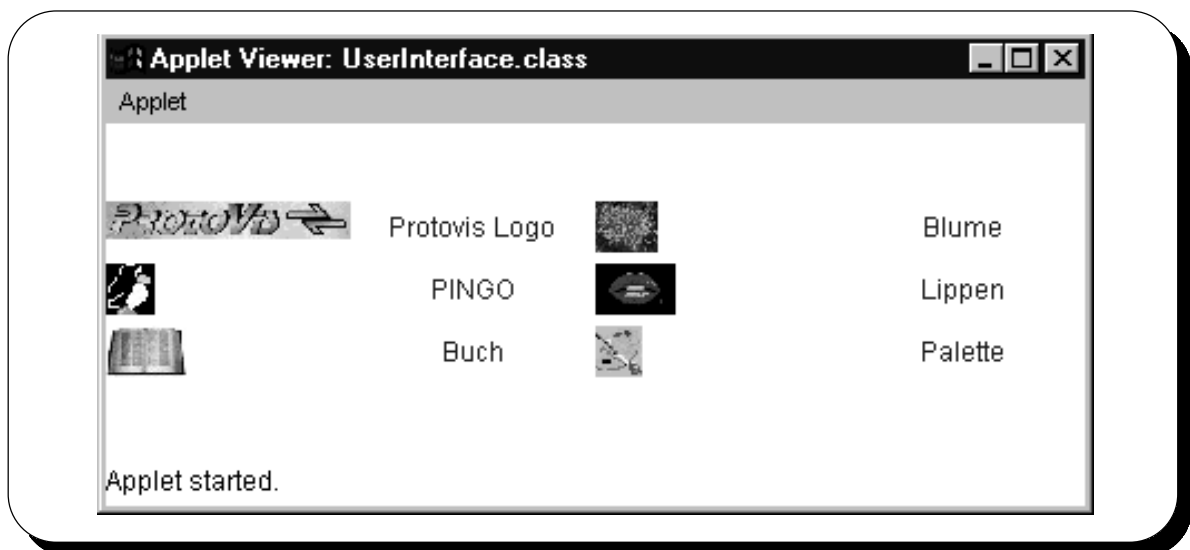


Abbildung 16 : " ImageString " - Panel

### 3.3.4 " Statuszeile " - Panel

Das " Statuszeile " - Panel wird zur Eintragung von Statusmeldungen verwendet.  
Hierzu folgendes Beispiel.

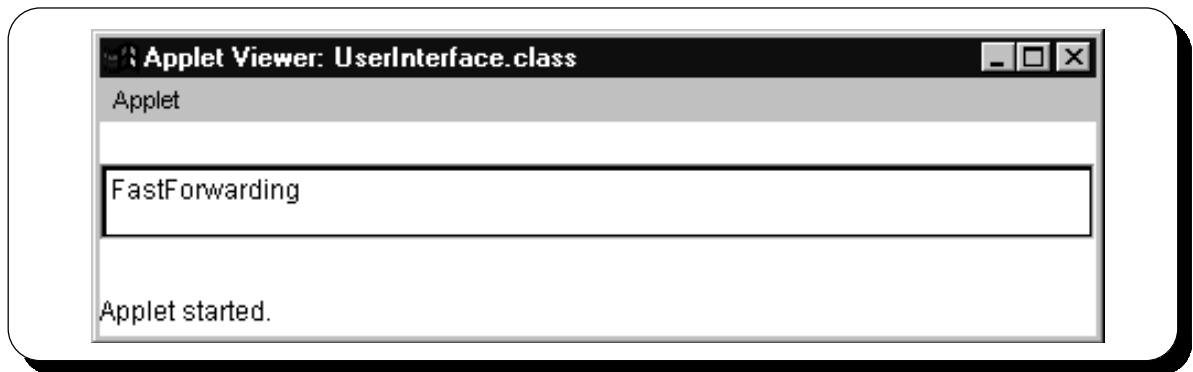


Abbildung 17 : " Statuszeile " - Panel

### **3.4. Beschreibung der Undo - Architektur**

Die Undo-Funktionalität des Baukastens wurde mittels einer 3-stufigen Hierarchie realisiert. An dem Undo von Visualisierungsschritten sind 3 Komponenten beteiligt:

- a.) Eine Klasse Undo, die die zentrale Undo - Verwaltung übernimmt.
- b.) Die Subklassen der Klasse VisContainer, die die Visualisierungsschritte der Visualisierungselemente übernehmen, die innerhalb dieses Containers dargestellt werden.
- c.) Die Subklassen der Klasse VisItem, die die Visualisierungselemente darstellen, welche ihre Zustandsänderungen an ihren Container weitermelden.

#### **3.4.1 Aufgaben der Klasse Undo**

Die Klasse Undo dient dazu, die Visualisierungsschritte, die in den Containern ablaufen zu sammeln und damit chronologisch zu ordnen. Damit ist ein Container von den anderen Containern entkoppelt und kann unabhängig von ihnen agieren. Die zentrale Klasse Undo sorgt dafür, daß Visualisierungsschritte in den verschiedenen Containern in der richtigen Reihenfolge registriert und auch wieder rückgängig gemacht werden. Dazu bietet sie eine Methode "register" an, mit deren Hilfe die Container einen Visualisierungsschritt unter der Angabe eines Bezeichners, der den Visualisierungsschritt kennzeichnet, anmelden können. Diese Information wird auf einen Stack gelegt und im Falle eines Undos wieder an den entsprechenden Container übergeben. Um ein Undo des Baukastens auszulösen, bietet die Klasse Undo der Anwendung eine Methode "undo" an. Der Rest erfolgt vollautomatisch. Folgendes Object Model-Diagramm verdeutlicht noch einmal die Struktur der Klasse Undo:

Abbildung 18 : Struktur der Klasse Undo

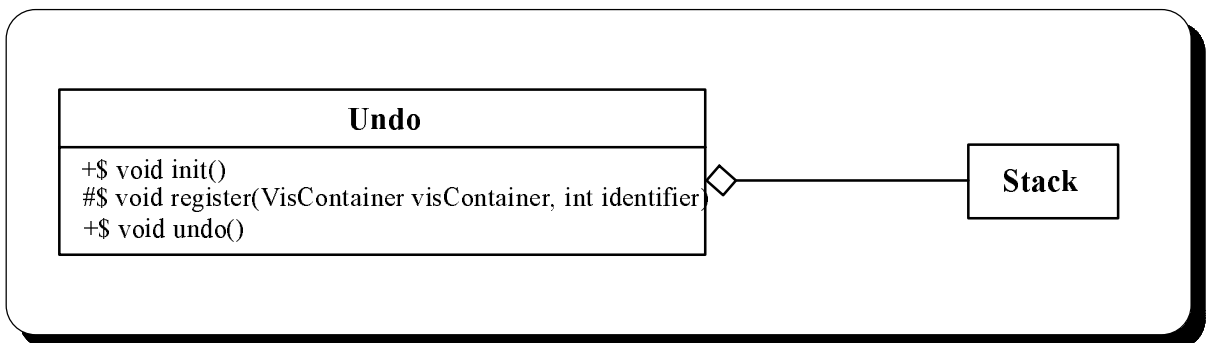


Abbildung 18 : Struktur der Klasse Undo

#### **3.4.2 Aufgaben der Container - Klassen**

Die Klasse VisContainer stellt die Struktur zur Verfügung, die deren Subklassen, die verschiedenen Container, implementieren müssen. Ein Container sammelt Rückmeldungen seiner Visualisierungselemente, speichert entsprechende Informationen auf einem Stack, bündelt diese zu Visualisierungsschritten und meldet diese mittels der Methode "register" an die zentrale Undo-Klasse. Der Container wird, im Falle eines Undo, durch Aufruf seiner "undo" Methode von der Klasse Undo dazu aufgefordert seinen letzten Visualisierungsschritt zurückzunehmen. Auf welche Art und Weise der Container Informationen seiner Visualisierungselemente sam-

melt, wie er sie zu Visualisierungsschritten bündelt und welche Bezeichner dafür vergeben werden bleibt jedem Container selbst überlassen und wird später anhand der konkreten Container erläutert.

Folgendes Object Model-Diagramm verdeutlicht noch einmal die Undo-Struktur der Klasse VisContainer:

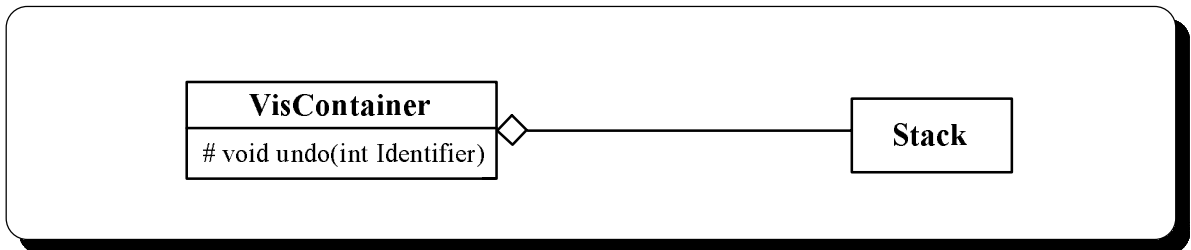


Abbildung 19 : Undo Struktur der Klasse VisContainer

### **3.4.3 Aufgaben der Visualisierungselemente**

Die Visualisierungselemente, d.h. die Subklassen der Klasse VisItem, müssen sämtliche Zustandsänderungen, die ihre Visualisierung betreffen an ihren Container übermitteln, damit er diese registrieren kann. Diese Interaktion zwischen den Visualisierungselementen und ihrem Container bleibt den beiden Beteiligten überlassen und wird später für die speziellen Visualisierungselemente und Container erläutert.

### **3.4.4 Ablauf des Undo - Prozesses**

Das in der Abbildung 20 (auf der folgenden Seite) dargestellte " Event Trace-Diagramm " zeigt die Interaktion der oben beschriebenen Komponenten.

### **3.4.5 Schematische Darstellung der Klassen und Stacks**

Zum Abschluß dieser Betrachtung folgt auch noch eine schematische Darstellung der beteiligten Klassen samt ihres Stackinhalts. Es soll klargemacht werden wie die Klasse Undo die Visualisierungsschritte aller beteiligten Container miteinander verzahnt. Betrachten Sie hierzu die Abbildung 21 (auf der darauffolgenden Seite).

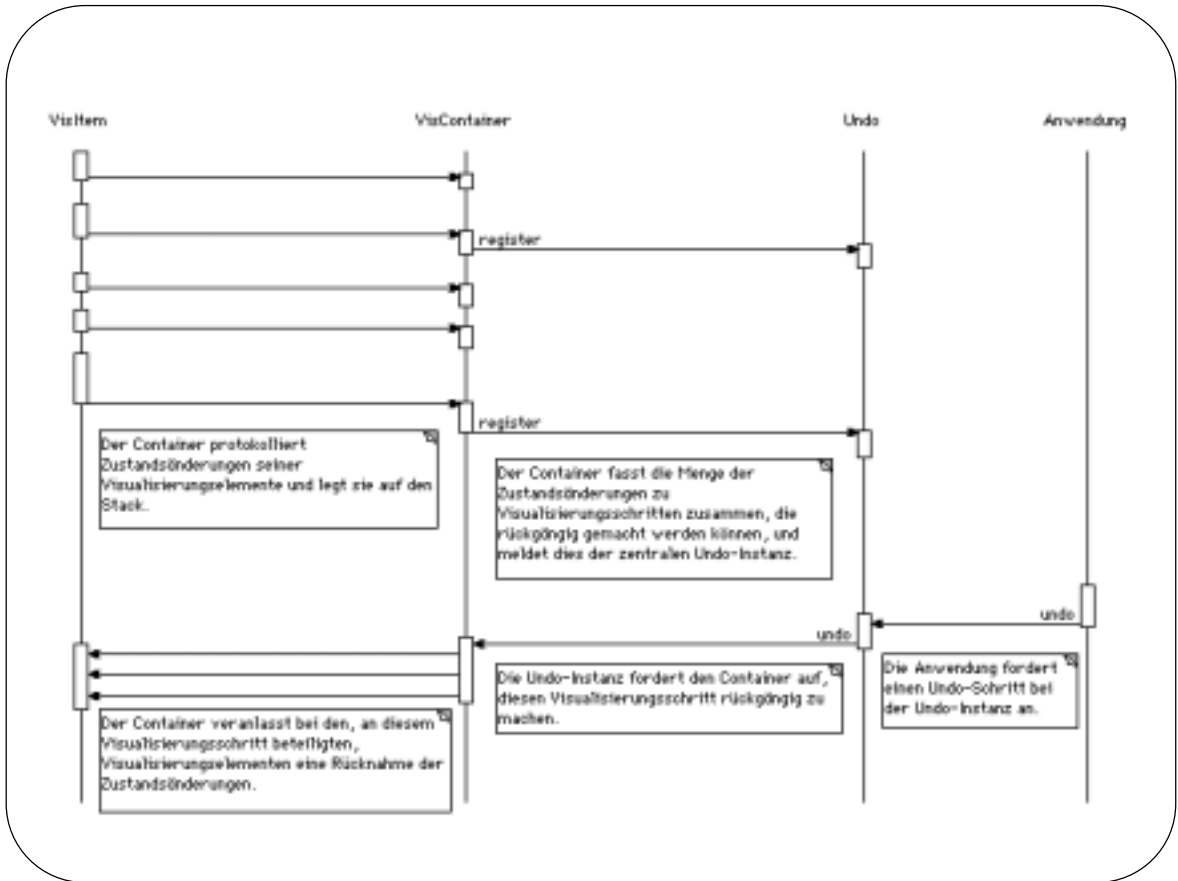


Abbildung 20 : Undo - Ablauf als Event Trace Diagramm

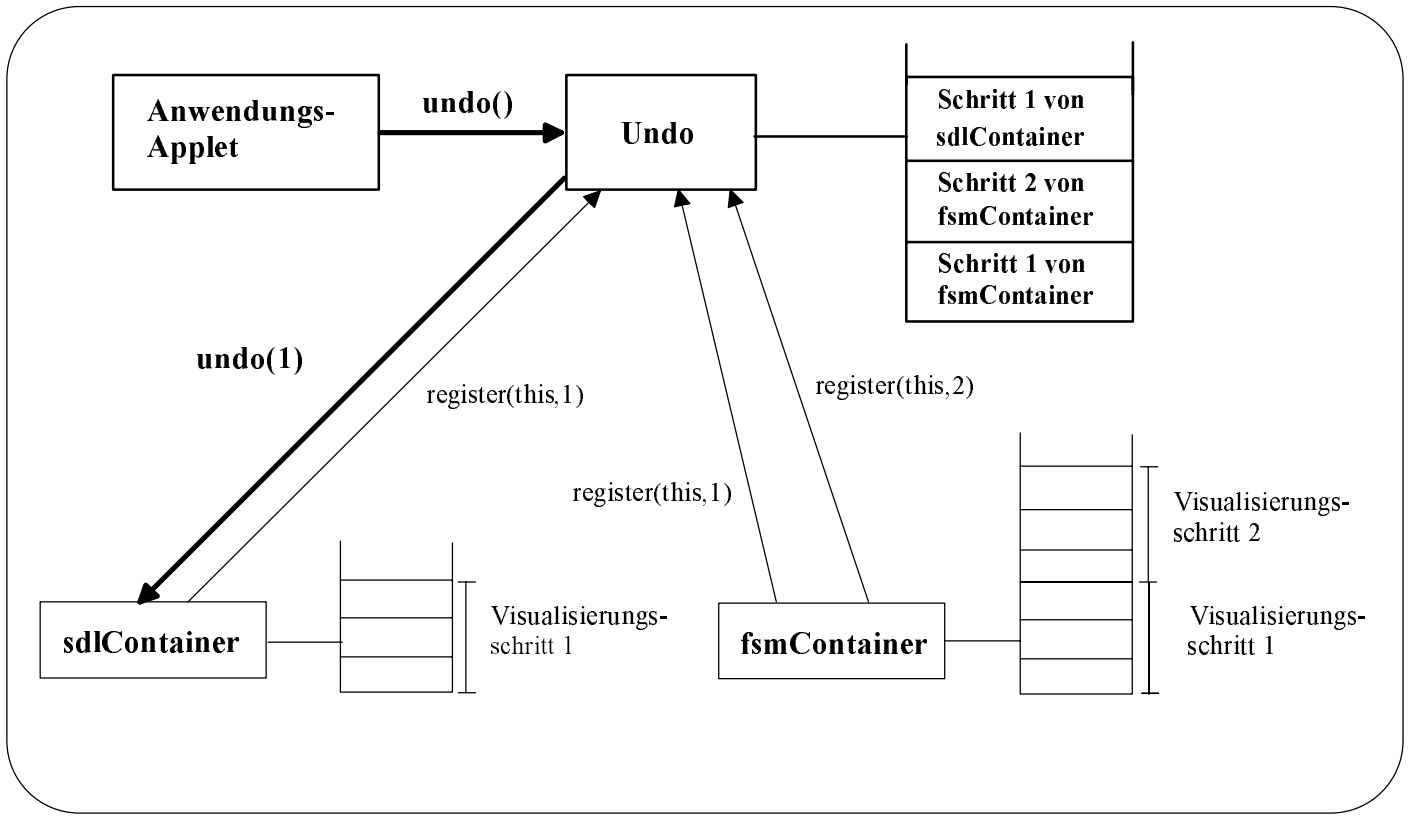


Abbildung 21 : Undo - Ablauf mit Stackdarstellung

### 3.5. Beschreibung des Zeitmanagements

Um die zeitlichen Abläufe innerhalb des Baukastens zu steuern wird eine Klasse Clock bereitgestellt. Diese Klasse verwaltet ein Property timeFactor, das angibt um welchen Faktor die Visualisierungszeit von der realen Zeit abweicht.

Beispiel:

Der Download einer 1 Kilobyte großen Datei über einen Kanal mit der Bandbreite 1MB/s dauert in Realzeit  $(1024 \text{ Byte}) / (1048576 \text{ Byte/s}) = 0,98 \text{ ms}$ . Diese kurze Zeitspanne kann aber nicht in Realzeit visualisiert werden; deshalb bewirkt ein Setzen des Properties timeFactor auf 1000, daß diese Dateiübertragung am Bildschirm in  $0,98 \text{ ms} * 1000 = 0,98 \text{ s}$  darstellt wird.

Die Klasse Clock implementiert das Property timeFactor als Bound-Property, sodaß interessierte Klassen (z.B. Timer- und Signal- Klassen) jede Änderung des Properties (zum Beispiel durch die Anwendung, um die Darstellung zu verlangsamen oder zu beschleunigen) sofort gemeldet bekommen und ihre eigene Visualisierung anpassen können. Das folgende Klassendiagramm veranschaulicht diese Struktur:

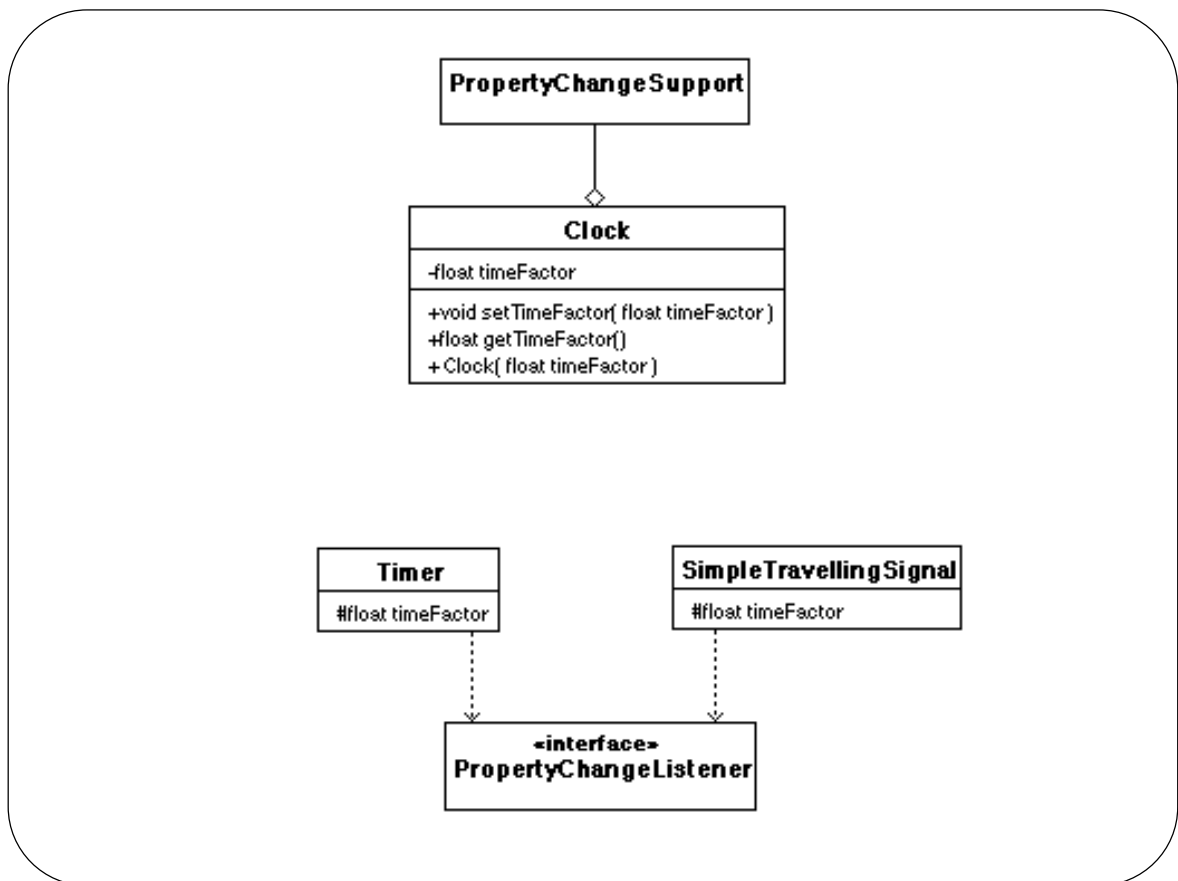


Abbildung 22 : Klassendiagramm Clock - Klasse und ihre Listener

## 4. Entwurf des speziellen Teils

### 4.1 Entwurf der Benutzerschnittstelle

Das Ziel des Entwurfs war, daß sich die Benutzerschnittstelle des Baukastens wesentlich angenehmer präsentieren sollte als die Benutzerschnittstelle des alten Baukastens. Um dieses Ziel zu erreichen wurden als erstes VCR - Kontrollen (Video-Casset-Recorder) verwendet. Unter VCR - Kontrollen werden Standardkontrollen wie " Play ", " Stop ", " Pause ", " Rewind (**Undo**) " und " FastForward " verstanden. Als Ausgangsbasis für die folgenden Erläuterungen dient Abbildung 23.

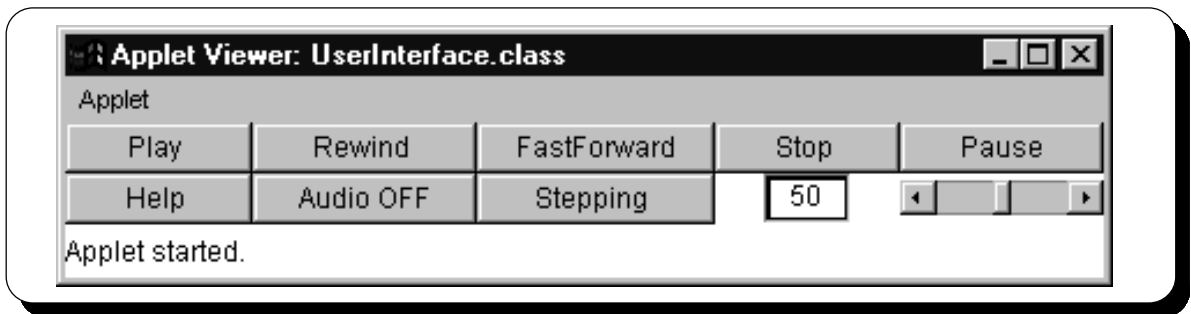


Abbildung 23 : " Button " - Panel

Wie zu sehen sind, sind diese Standardkontrollen übersichtshalber wie bei einem Kassettenrekorder angeordnet, somit können sie vom Benutzer leichter gemerkt werden. Verwendet wurde ein Panel mit Buttons zu diesem Zweck.

Es wird darauf hingewiesen, daß das Ziel dieser Studienarbeit, eine Benutzerschnittstelle zu entwerfen war. Wenn ein Button gedrückt wird, wird ein " Event " ausgelöst auf das die Anwendung entsprechend reagieren sollte. Beim drücken des " Play " - Buttons sollte der Animationsablauf beginnen. Mit " Rewind (Undo) " sollte die Möglichkeit bestehen Visualisierungsschritte zurückzunehmen. Um sich schnell nach vorwärts in den Animationsablauf zu bewegen wurde ein " FastForward " - Button hinzugefügt. Durch " Stop " sollte die Visualisierung beendet und durch " Pause " unterbrochen werden.

Wenn der " Pause " - Button gedrückt wird, verändert sich dessen Beschriftung zu " Resume " und in der " Statuszeile " erscheint die Meldung " Paused ". Entsprechende Meldungen gibt es bei den anderen genannten Buttons ebenfalls. Durch drücken von " Resume " wird die Animation fortgesetzt und dieser Button wechselt in seine alte Beschriftung " Pause " zurück. Zur verdeutlichung dient Abbildung 24.

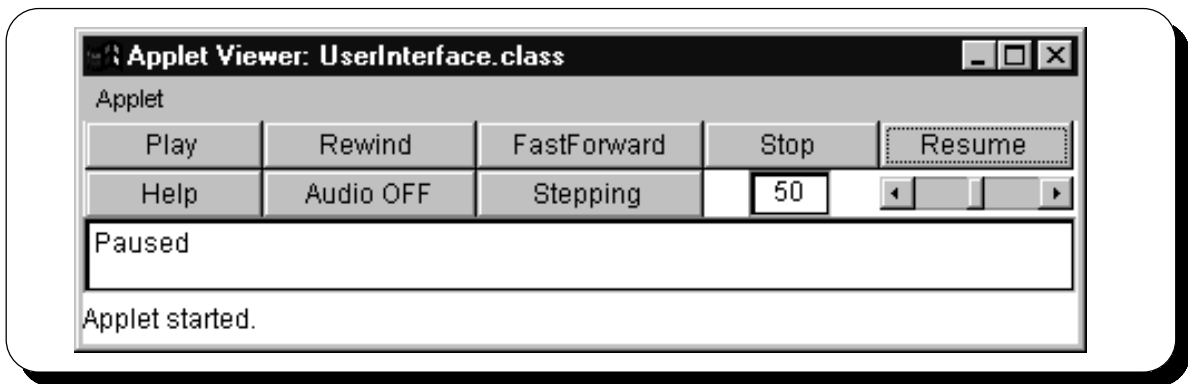


Abbildung 24 : Änderung des " Pause " Button zu " Resume "

Ein Button " Help " das noch hinzugefügt wurde sollte den Benutzer bei der Bedienung der Benutzerschnittstelle behilflich sein.

Da zu einem späteren Zeitpunkt auch noch Audio verwendet werden soll zur Animation (beispielsweise für kollidierende Nachrichten oder abgelaufene Timer) wurde ein Audio-Button hierfür bereitgestellt, da Ton ausgesprochen nervig sein kann. Beim Drücken des Audio OFF Buttons verändert sich die Beschriftung zu Audio ON.

Weiterhin unterstützt die Benutzerschnittstelle den Stepping - Modus der durch anklicken auf den " Stepping " - Button (im Normalzustand) in den " Stepping - Modus " wechselt d.h die Visualisierung wird bis zum nächsten Visualisierungsschritt fortgeführt und auf die Benutzereingabe gewartet und durch erneutes drücken des " Stepping - Modus " wird der nächste Visualisierungsschritt ausgeführt. Um in den Normalzustand zu gelangen wird auf den " Play " - Button gedrückt. Dabei wechselt die Beschriftung des " Stepping - Modus " - Buttons zu " Stepping " was den Normalzustand repräsentiert d.h kontinuierlicher Ablauf des Protokolls. Hierzu folgendes Bild:

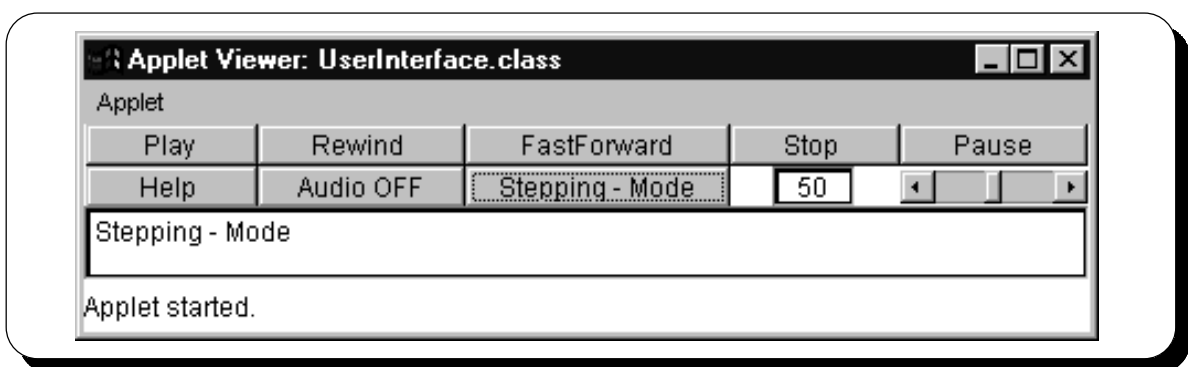


Abbildung 25 : Vom Normalzustand in den " Stepping - Mode "

Eine weitere Alternative die Darstellungsgeschwindigkeit vom Benutzer zu steuern ist: Optional ein Geschwindigkeitsregler der sich als horizontaler Regler an die Anordnung der anderen Buttons anpaßt. Weiterhin wurde ein " Textfield " für diesen Zweck bereitgestellt. Erlaubt werden beim Textfield Eingaben von Zahlen zwischen 1 und 99.

Im Normalzustand steht die Zahl 50 im Textfield. Bei der Eingabe einer Zahl in das Textfield, paßt sich der horizontale Regler der Zahl an. Entsprechend paßt sich bei einer Einstellung

des Reglers die Zahl im TextFeld an. Folgendes Bild soll dies verdeutlichen beim Drücken des " FastForward " - Button (Geschwindigkeitsregler auf höchsten Wert) :

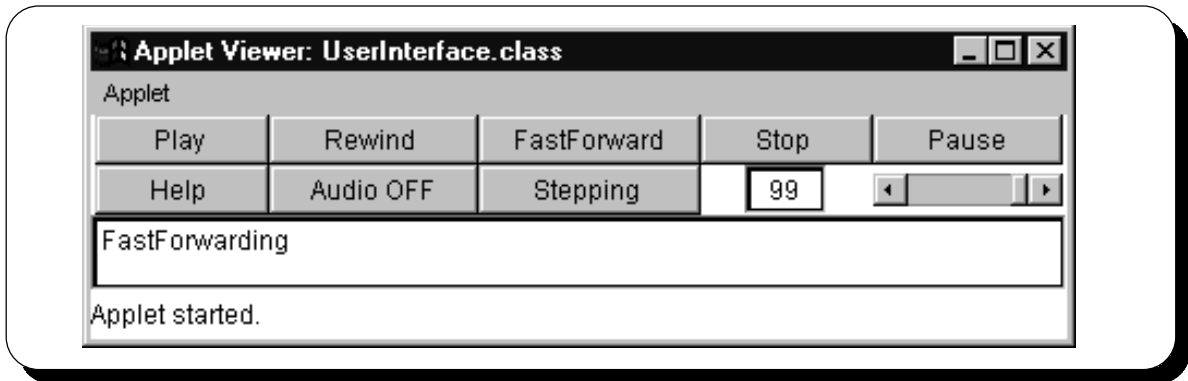


Abbildung 26 : Auswirkungen bei gedrücktem " FastForward " - Button

Mit der Hilfe der Standardkontrollen, " Stepping - Mode " , Geschwindigkeitsregler sollte es dem Benutzer möglich sein den Ablauf der Visualisierung zu steuern, also das Animationsverhalten zu manipulieren.

Weiterhin ist es möglich die Buttons auch in einer beliebigen Farbe und Schriftart darzustellen. Bezüglich dem " TextArea " - Panel und dem " ImageString " - Panel sollte die Möglichkeit bestehen nicht nur Informationen einzutragen sondern auch zu löschen. Prozesse können auch wieder verschwinden. Für das " ImageString " - Panel war vorgesehen sich dynamisch den Eintragungen anzupassen.

## 4.2. Die Undo - Verwaltung der Klasse SDLContainer

Die Übermittlung der Zustandsinformationen von den SDL-Visualisierungselementen zu ihrem entsprechenden SDLContainer erfolgt mittels Bound-Properties. Sobald sich die visuelle Darstellung eines Visualisierungselements ändert, werden diese Änderungen dem Container mittels PropertyChangeEvent gemeldet. Selbstverständlich verwaltet der Container auch seine eigenen Zustandsänderungen.

Das nachfolgende Object Model-Diagramm zeigt die dabei beteiligten Klassen:

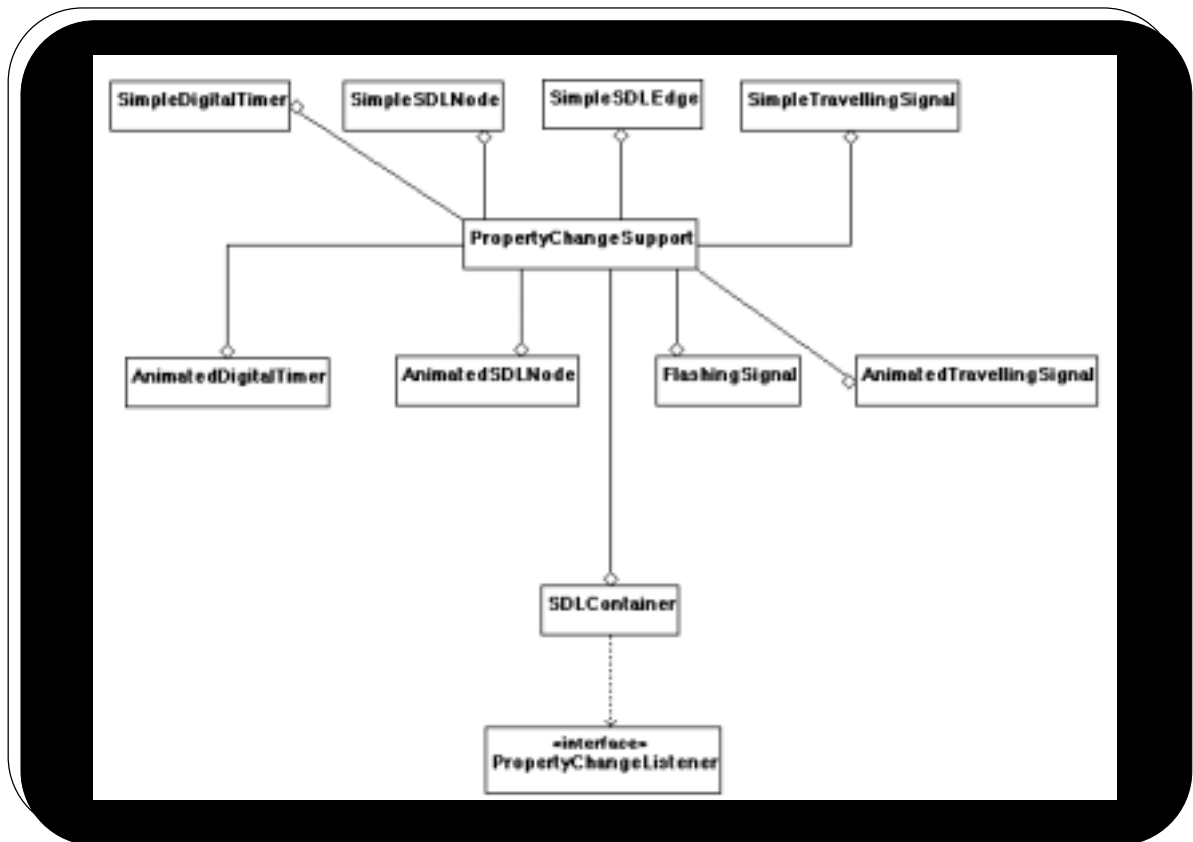


Abbildung 27 : SDL - Klassen

### 4.2.1 Die Verwaltung der Zustandsinformationen

Der SDLContainer verwaltet für jedes Visualisierungselement einen eigenen Stack, worauf er die PropertyChangeEvent speichert, die dieses Visualisierungselement ausgelöst hat. Er bildet daraus Gruppen von Events, die einen Visualisierungsschritt ausmachen. Diese Gruppen werden im nächsten Kapitel bei der Implementierung des Undo näher erläutert. Die Gruppen werden auf dem Stack durch ein null Objekt getrennt. Der SDLContainer adressiert diese Stacks mit Hilfe einer Hashtabelle, die er mit den Visualisierungselemente-Objekten direkt adressiert. Um die verschiedenen Visualisierungsschritte, symbolisiert durch Event-Gruppen auf den verschiedenen Visualisierungselemente-Stacks, in eine chronologische Reihenfolge zu bringen wird ein zentraler Stack verwendet. Auf diesen Stack werden Referenzen auf die Visualisierungselemente-Objekte gestapelt, die zuletzt einen Visualisierungsschritt vollzogen

haben. Diese Aufteilung erfüllt den selben Zweck wie die Aufteilung zwischen Undo-Klasse und Container-Klassen. Die Visualisierungsschritte werden getrennt nach Verursacher gespeichert und durch eine zentrale Instanz chronologisch geordnet. Auf dem zentralen Stack entspricht ein Eintrag einem Visualisierungsschritt, der auch dann sofort mittels der Methode register an die Klasse Undo weitergemeldet wird. Ein Bezeichner für diesen Visualisierungsschritt wird dabei nicht benötigt und deshalb wird 0 als identifier übergeben. Das folgende Diagramm zeigt diesen Sachverhalt noch einmal auf:

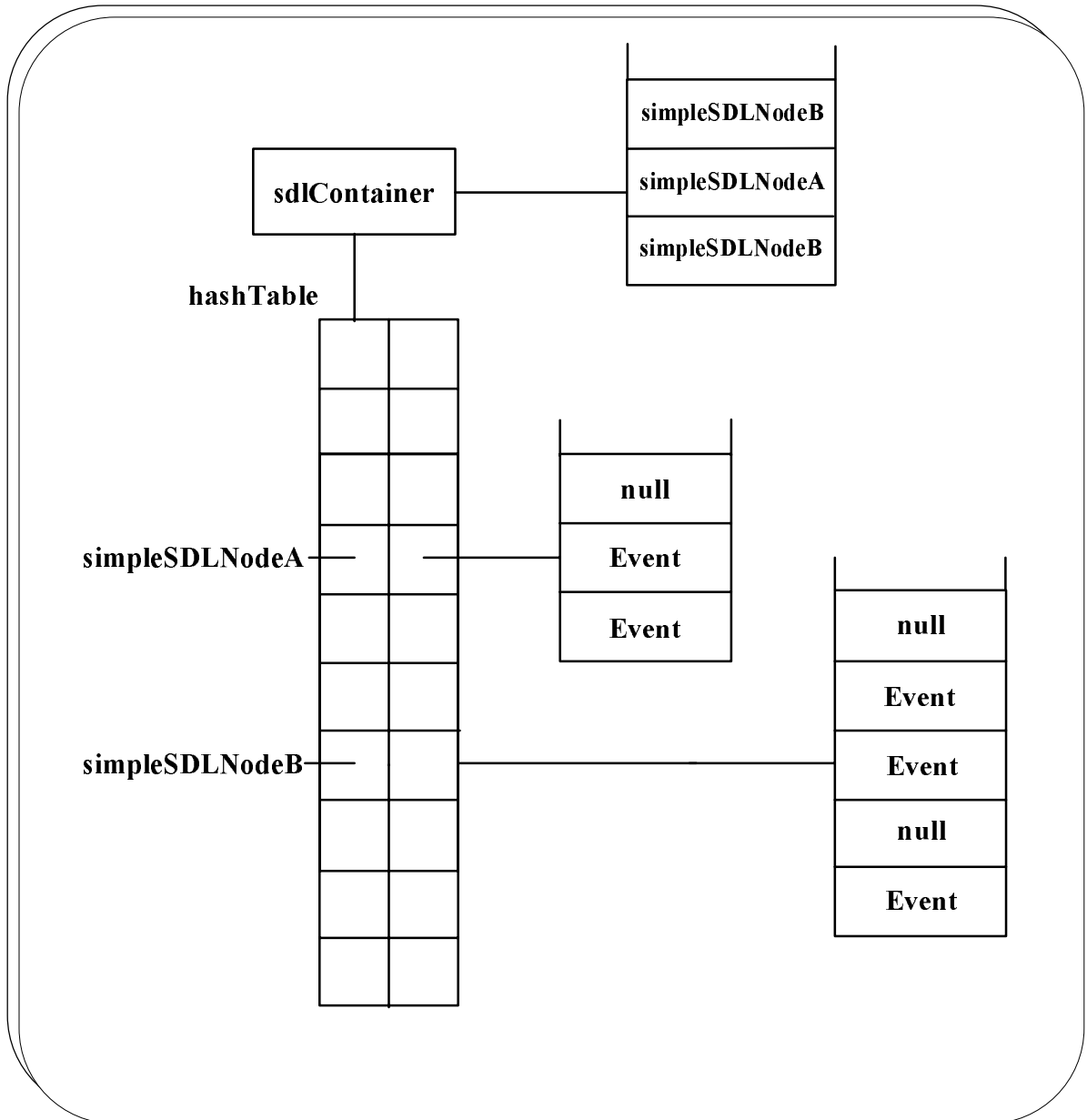


Abbildung 27 : SDL - Stacks

## 5. Implementierung des speziellen Teils

### 5.1 Die Implementierung der Undo - Funktionalität der SDLKlassen

Die Klasse SDLContainer filtert die Rückmeldungen seiner Visualisierungselemente und bildet daraus Visualisierungsschritte. Im folgenden wird dargestellt welche Methodenaufrufe von Visualisierungselementen (und des Visualisierungscontainers) einen Visualisierungsschritt ausmachen und durch welche Informationen auf den Stacks diese Visualisierungsschritte repräsentiert werden.

Durch diese Aufzeichnungen sind die Visualisierungsschritte, die auf dem Bildschirm sichtbar sind, und die Methodenaufrufe, die diese Visualisierungsschritte ausgelöst haben, vollständig bekannt und können durch entsprechende Rückabwicklung der Operationen in den Methoden leicht zurückgenommen werden. Im folgenden wird anhand von State - Diagrammen der SDL-Klassen gezeigt, durch welche Zustandsübergänge in den betroffenen Klassen ein Visualisierungsschritt gekennzeichnet ist und welche Informationen auf den betreffenden Stacks dabei mitprotokolliert werden.

Die Informationen auf dem Stack sind PropertyChangedEventArgs. In den Diagrammen sind folgende Informationen die diese Events liefern dargestellt:

Name des veränderten Properties, alter Wert des Properties, neuer Wert des Properties.

#### 5.1.1 Undo - Informationen der Klasse SDLContainer

Auf der Klasse SDLContainer sind 2 Visualisierungsschritte definiert:

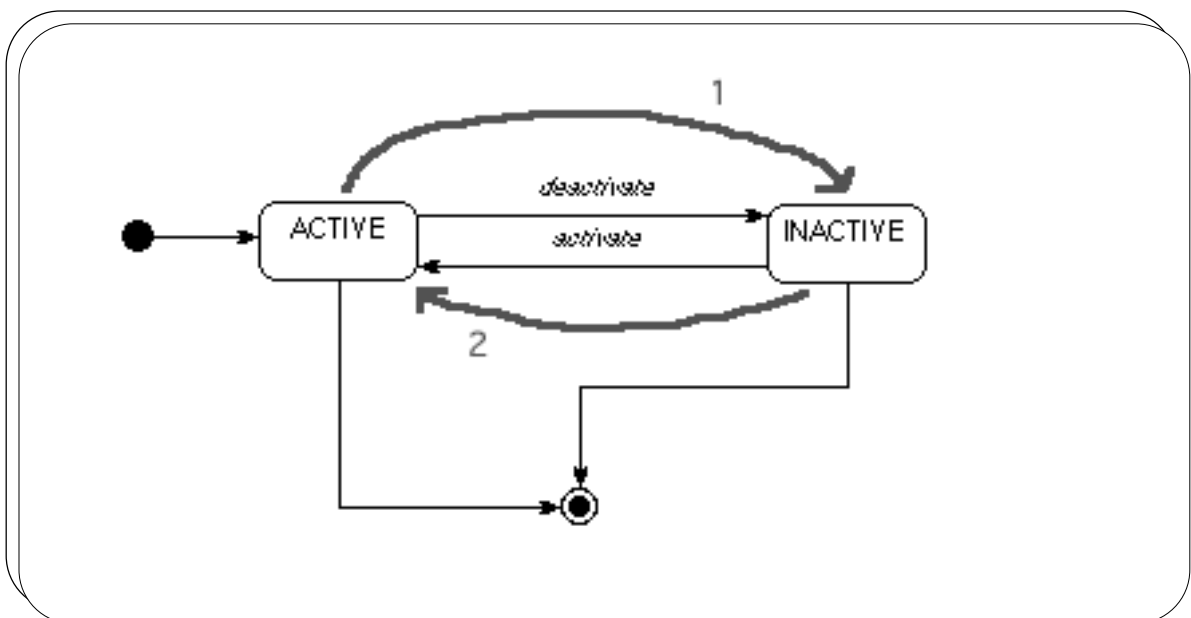


Abbildung 28 : Visualisierungsschritte der Klasse SDLContainer

Der Container wird deaktiviert (1). Auslösende Methodenaufrufe: `sdlContainer.deactivate()`  
Informationen auf dem Stack:

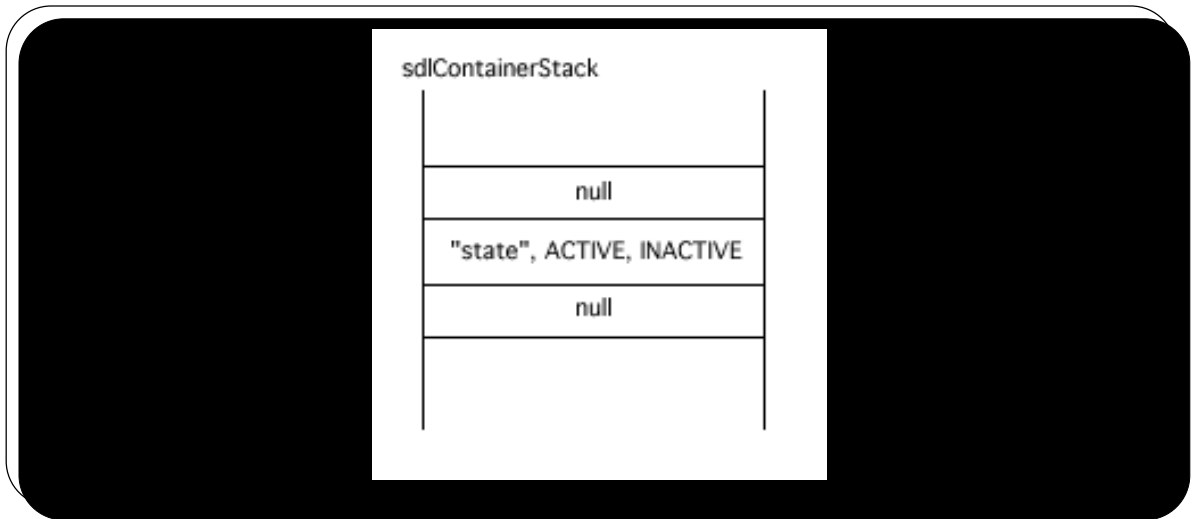


Abbildung 29 : Stackinhalt - SDLContainer deaktiviert

Der Container wird aktiviert (2). Auslösende Methodenaufrufe: `sdlContainer.activate()`  
Informationen auf dem Stack:

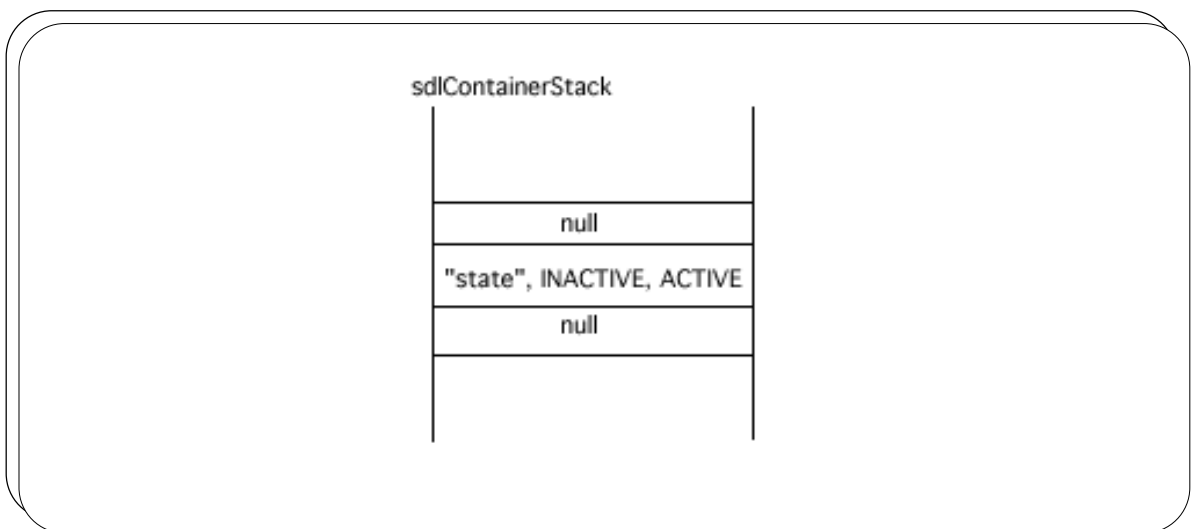


Abbildung 30 : Stackinhalt - SDLContainer aktiviert

### 5.1.2 Undo - Informationen der Klasse SimpleSDLNode

Auf der Klasse SimpleSDLNode sind 4 Visualisierungsschritte definiert:

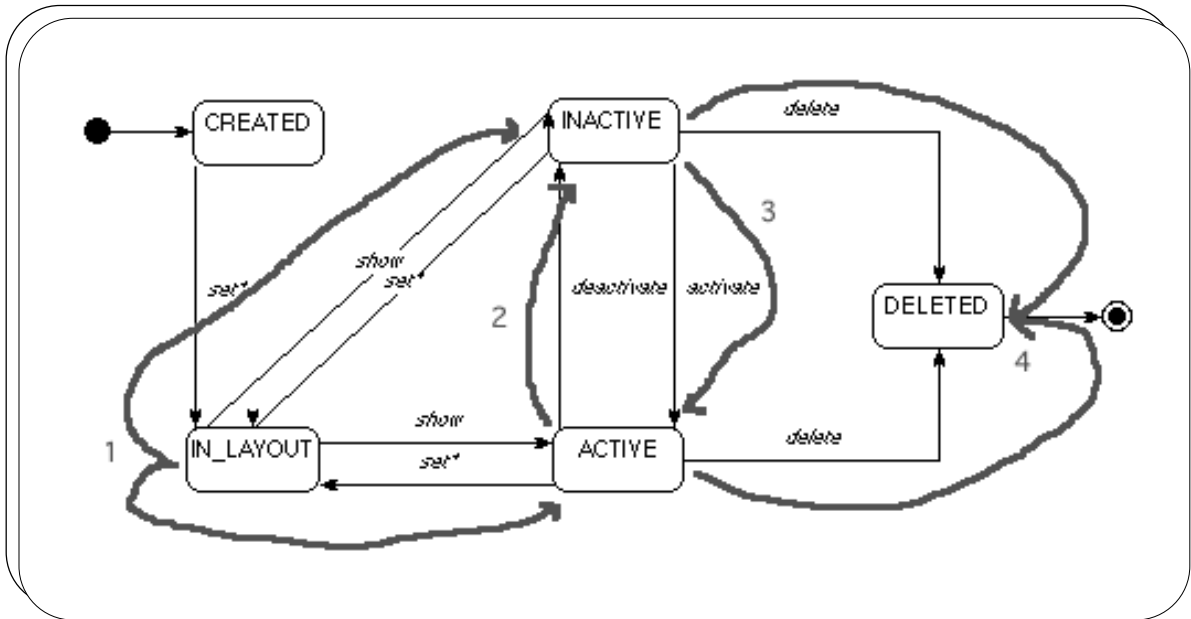


Abbildung 31 : Visualisierungsschritte der Klasse SimpleSDLNode

Der Knoten wird angezeigt (1). Auslösende Methodenaufrufe: `simpleSDLNode.set*` - `simpleSDLNode.show()`  
 Informationen auf dem Stack:

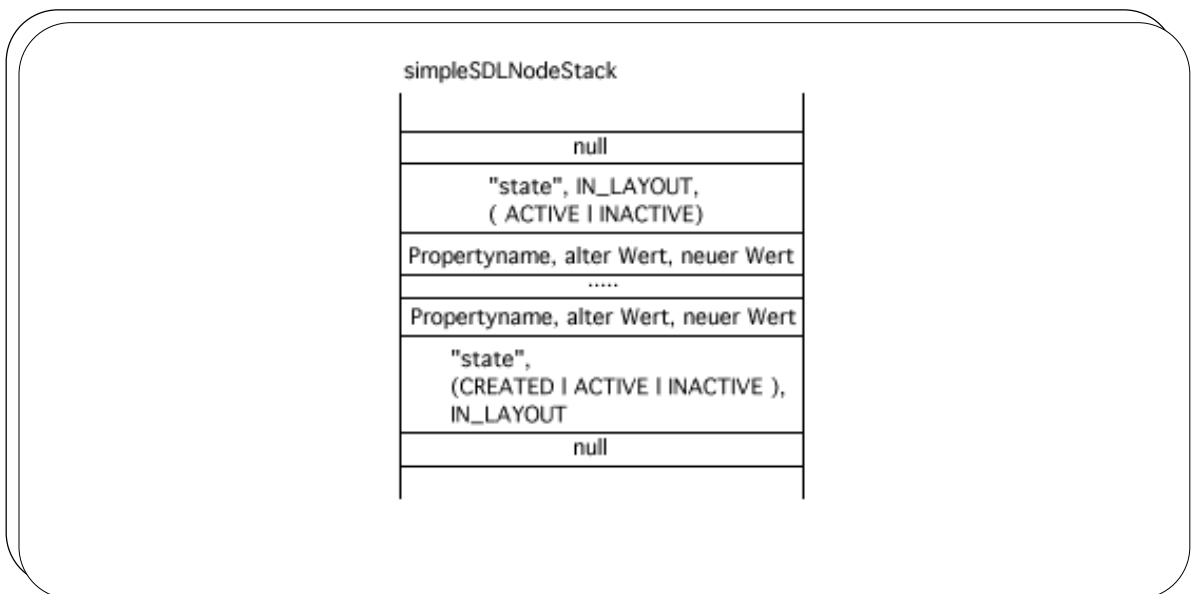


Abbildung 32 : Stackinhalt - SimpleSDLNode angezeigt

Der Knoten wird deaktiviert (2). Auslösende Methodenaufrufe: simpleSDLNode.deactivate()  
Informationen auf dem Stack:

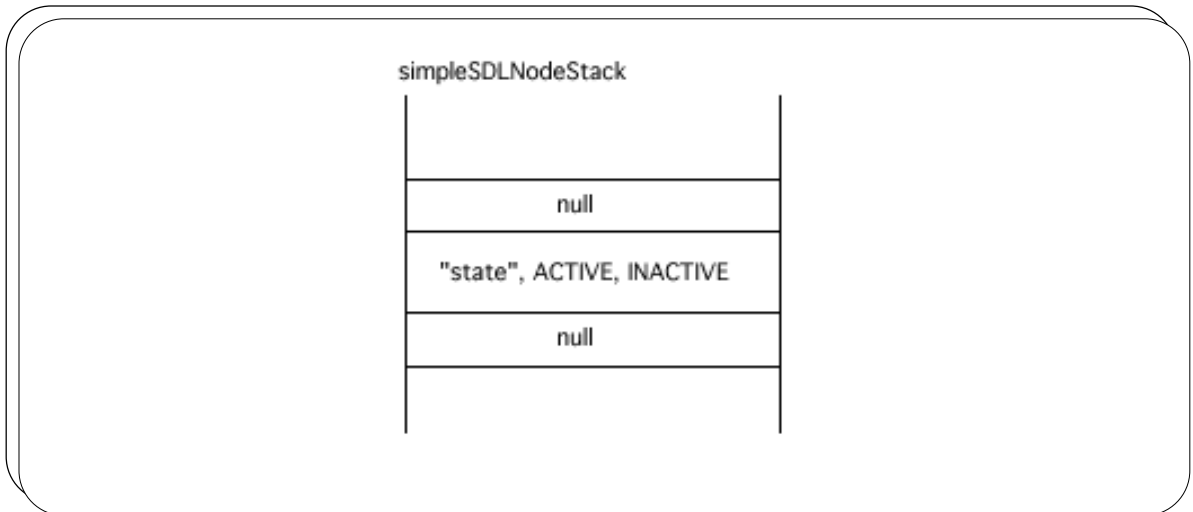


Abbildung 33 : Stackinhalt - SimpleSDLNode deaktiviert

Der Knoten wird aktiviert (3). Auslösende Methodenaufrufe: simpleSDLNode.activate()  
Informationen auf dem Stack:

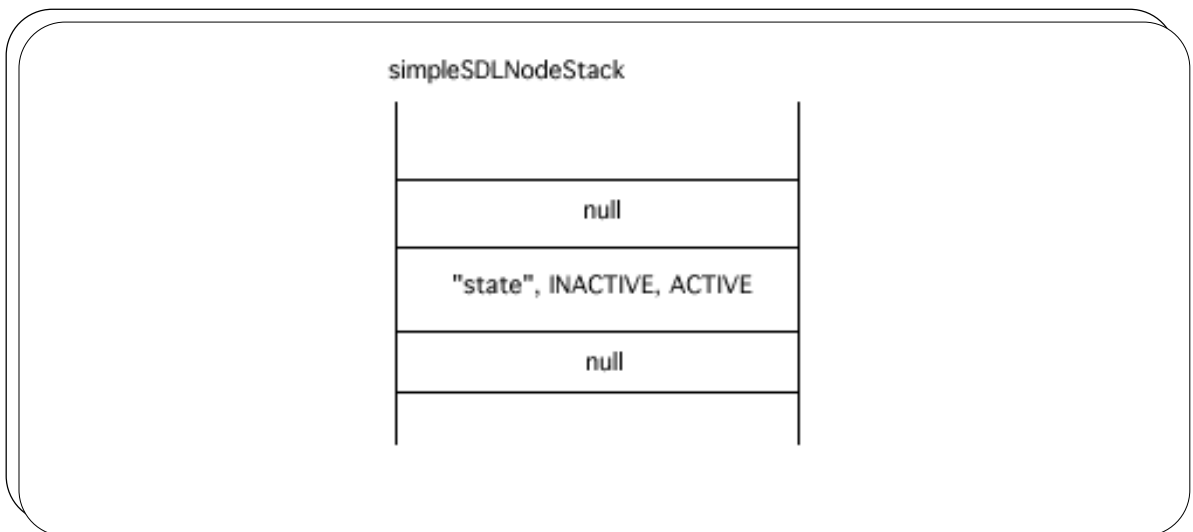


Abbildung 34 : Stackinhalt - SimpleSDLNode aktiviert

Der Knoten wird gelöscht (4). Auslösende Methodenaufrufe: simpleSDLNode.delete()  
Informationen auf dem Stack:

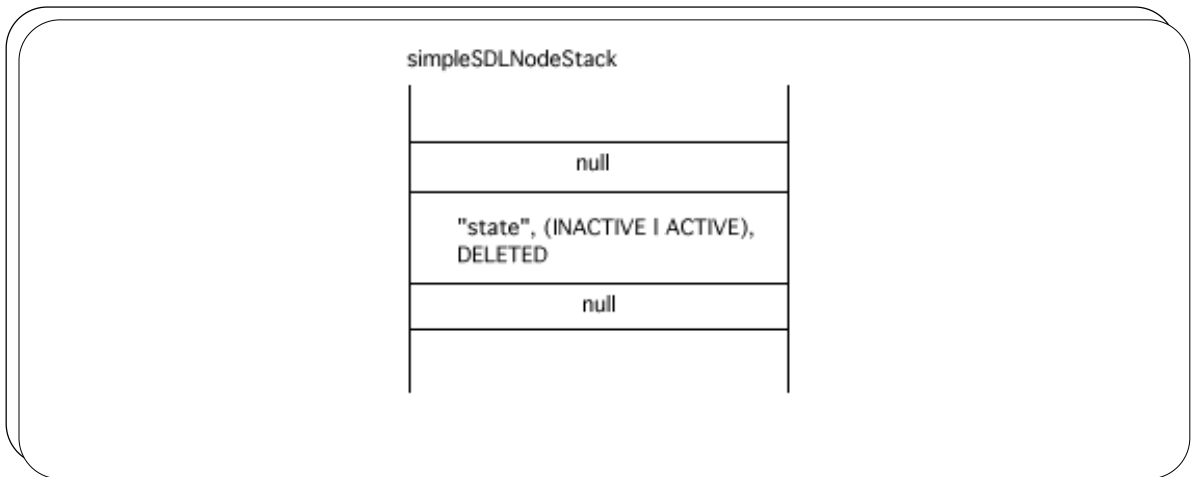


Abbildung 35 : Stackinhalt - SimpleSDLNode gelöscht

### **5.1.3 Undo - Informationen der Klasse AnimatedSDLNode**

Auf der Klasse AnimatedSDLNode sind die gleichen Visualisierungsschritte wie auf ihrer Oberklasse SimpleSDLNode definiert.

### **5.1.4 Undo-Informationen der Klasse SimpleDigitalTimer**

Auf der Klasse SimpleDigitalTimer sind 5 Visualisierungsschritte definiert:

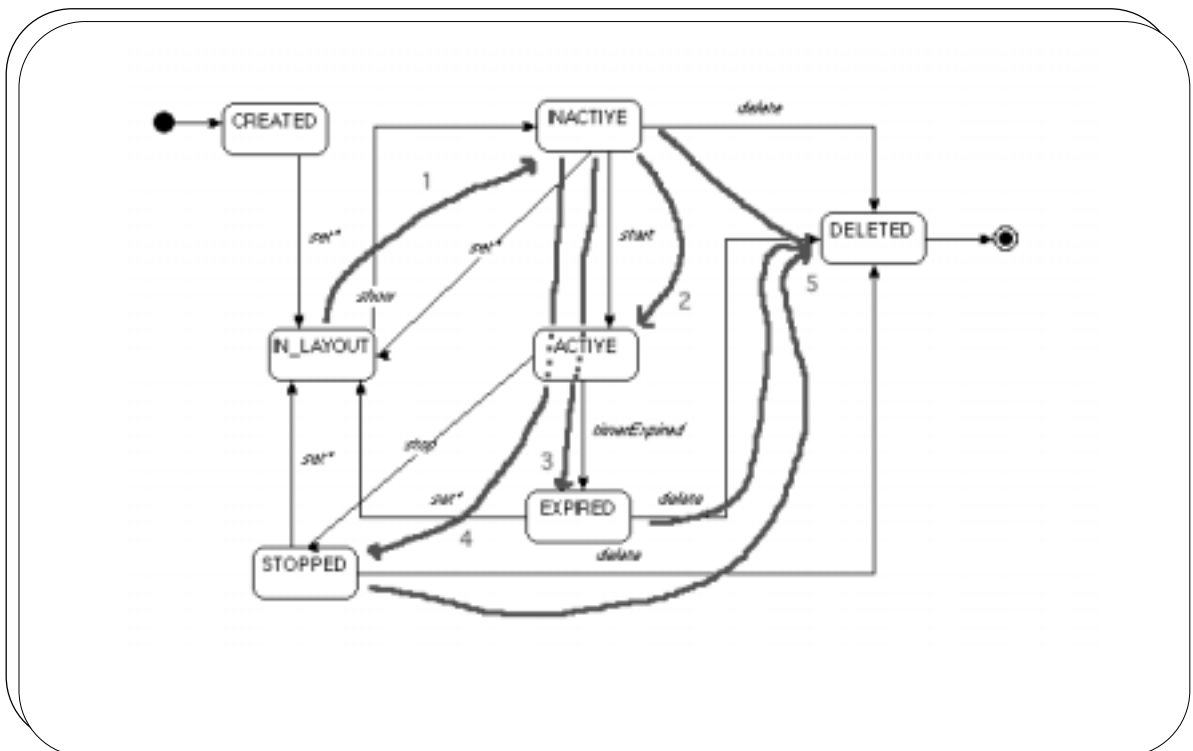


Abbildung 36 : Visualisierungsschritte SimpleDigitalTimer

Der Timer wird angezeigt (1). Auslösende Methodenaufrufe: `simpleDigitalTimer.set*` - `simpleDigitalTimer.show()`  
 Informationen auf dem Stack:

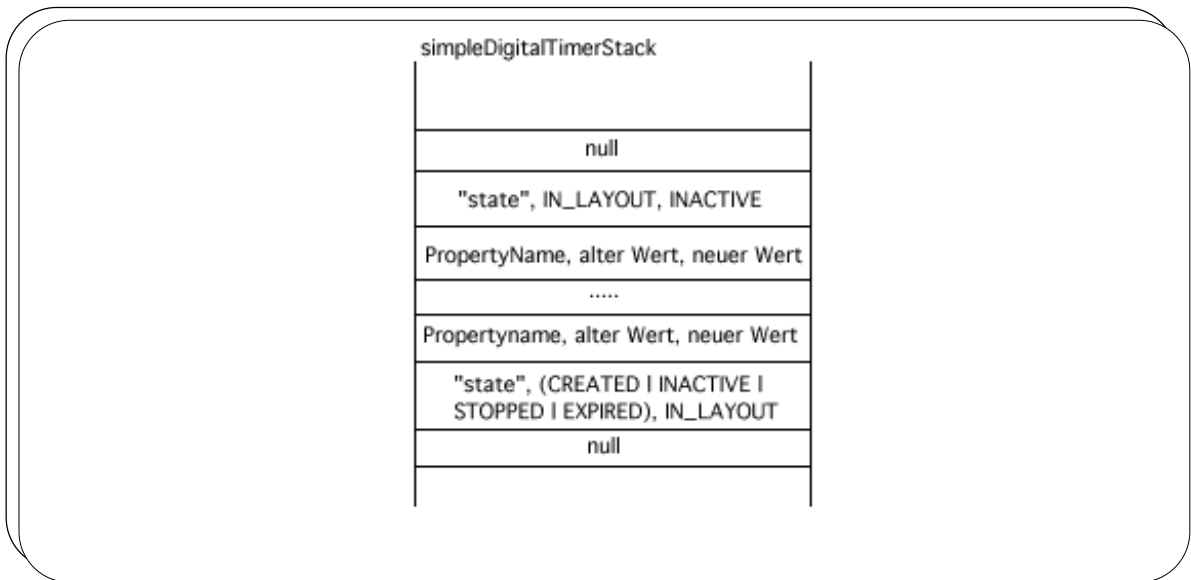


Abbildung 37 : Stackinhalt - SimpleDigitalTimer angezeigt

Der Timer wurde gestartet und läuft noch (2). Auslösende Methodenaufrufe:  
`simpleDigitalTimer.start()`  
 Informationen auf dem Stack:

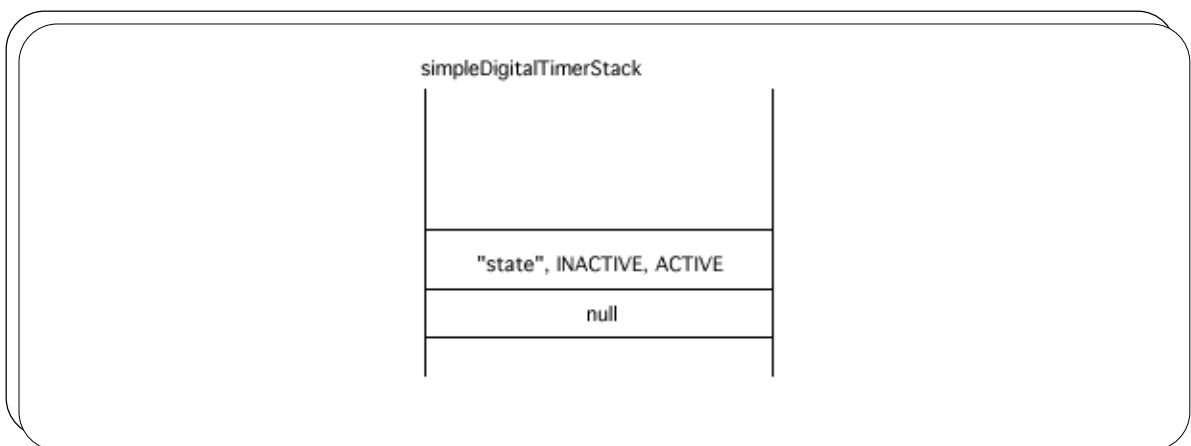


Abbildung 38 : Stackinhalt - SimpleDigitalTimer gestartet und läuft noch

Der Timer wurde gestartet und ist abgelaufen (3). Auslösende Methodenaufrufe:  
`simpleDigitalTimer.start()` - `simpleDigitalTimer.timerExpired()`  
 Informationen auf dem Stack:

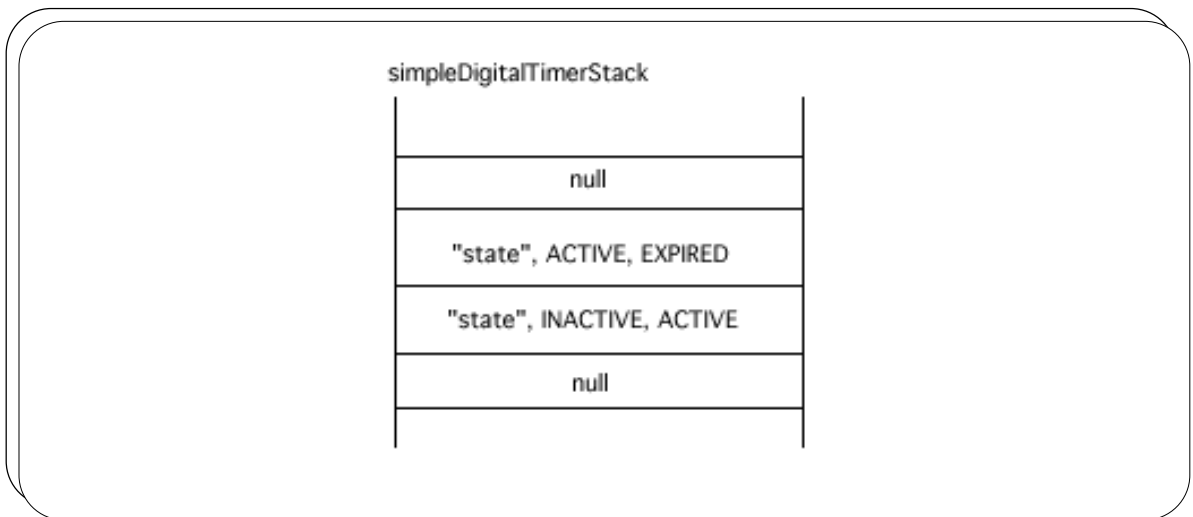


Abbildung 39 : Stackinhalt - SimpleDigitalTimer gestartet und abgelaufen

Der Timer wurde gestartet und gestoppt (4). Auslösende Methodenaufrufe:  
 simpleDigitalTimer.start() - simpleDigitalTimer.stop()  
 Informationen auf dem Stack:

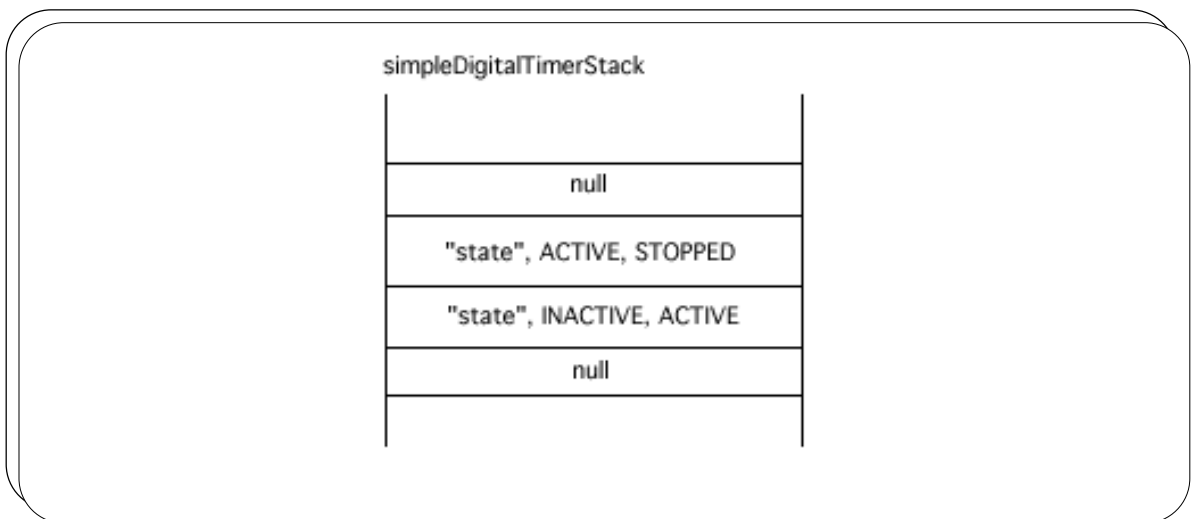


Abbildung 40 : Stackinhalt - SimpleDigitalTimer gestartet und gestoppt

Der Timer wurde gelöscht (5). Auslösende Methodenaufrufe: simpleDigitalTimer.delete()  
 Informationen auf dem Stack:

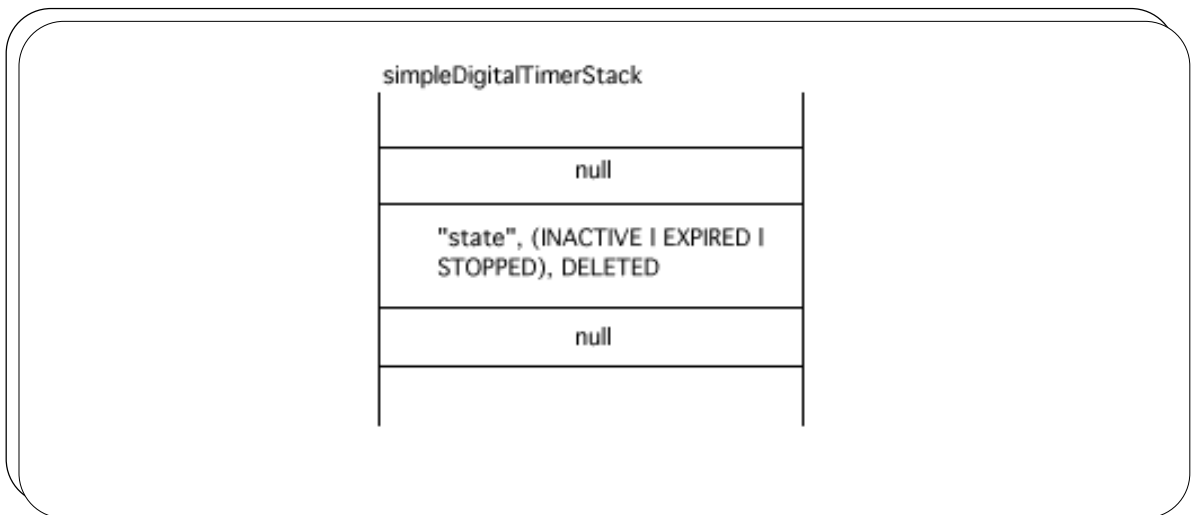


Abbildung 41 : Stackinhalt SimpleDigitalTimer gelöscht

### **5.1.5 Undo-Informationen der Klasse AnimatedDigitalTimer**

Auf der Klasse `AnimatedDigitalTimer` sind die gleichen Visualisierungsschritte wie auf ihrer Oberklasse `SimpleDigitalTimer` definiert.

### **5.1.6 Undo - Informationen der Klasse SimpleSDLEdge**

Auf der Klasse `SimpleSDLEdge` sind 4 Visualisierungsschritte definiert:

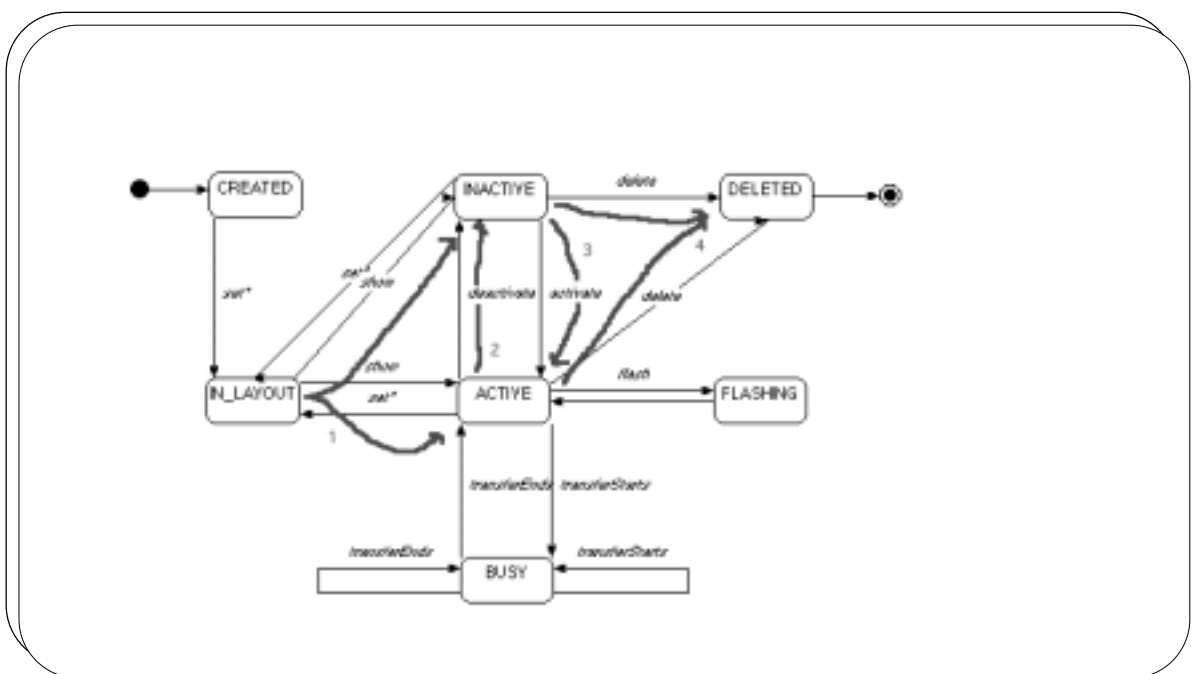


Abbildung 42 : Visualisierungsschritte der Klasse SimpleSDLEdge

Die Kante wird angezeigt (1). Auslösende Methodenaufrufe: simpleSDLEdge.set\* - simpleSDLEdge.show()  
 Informationen auf dem Stack:

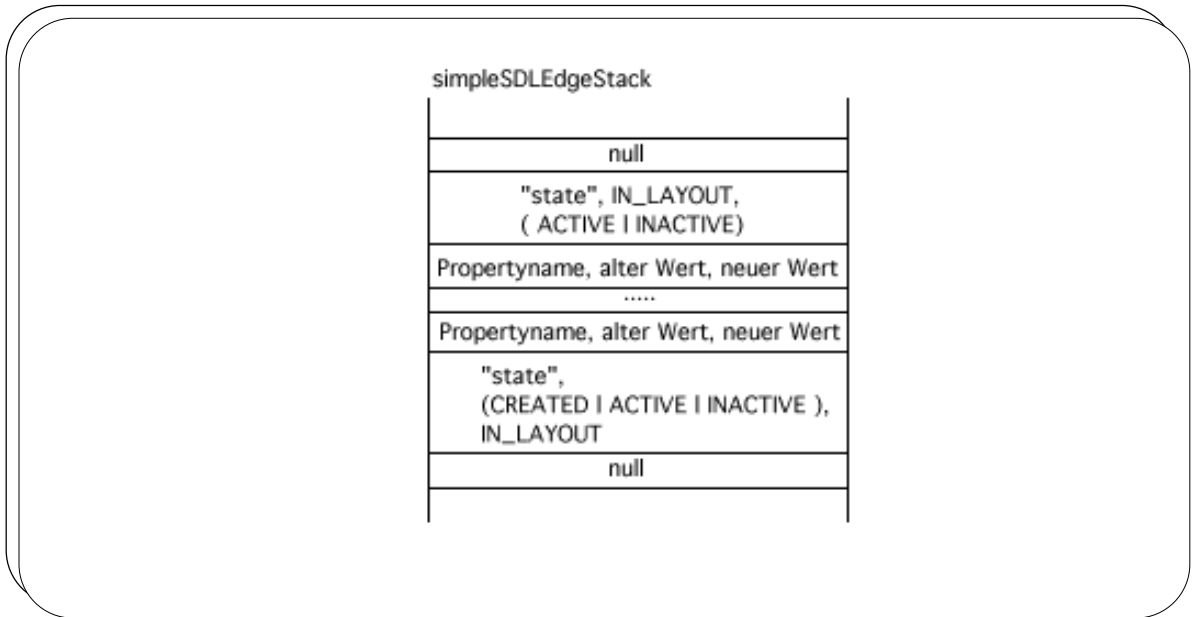


Abbildung 43 : Stackinhalt - SimpleSDLEdge angezeigt

Die Kante wird deaktiviert (2). Auslösende Methodenaufrufe: simpleSDLEdge.deactivate()  
 Informationen auf dem Stack:

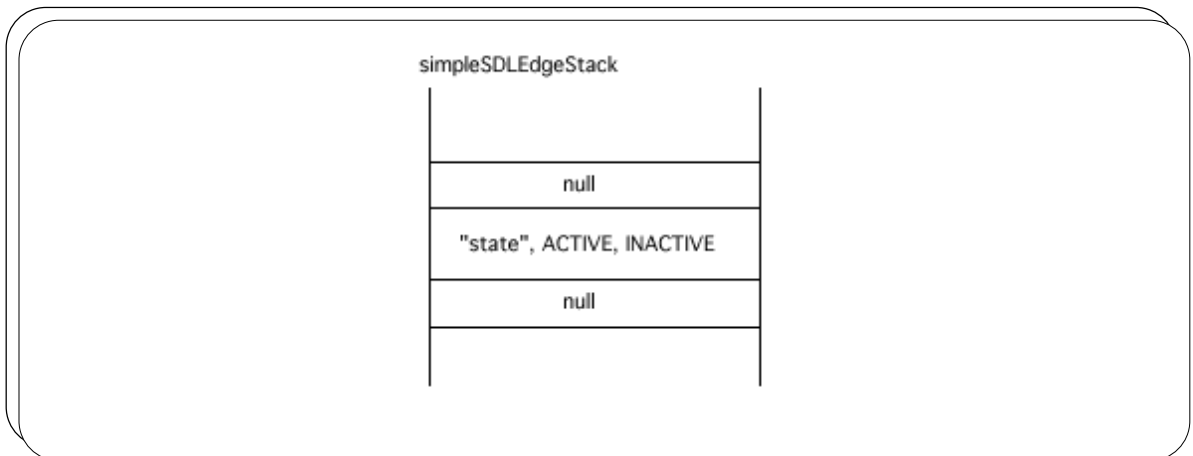


Abbildung 44 : Stackinhalt - SimpleSDLEdge deaktiviert

Die Kante wird aktiviert (3) Auslösende Methodenaufrufe: simpleSDLEdge.activate()  
 Informationen auf dem Stack:

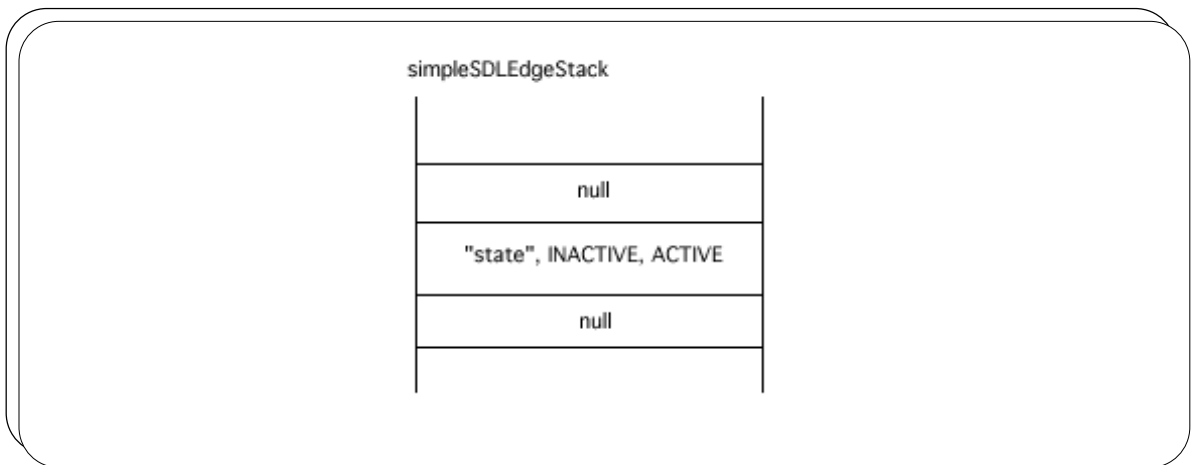


Abbildung 45 : Stackinhalt - SimpleSDLEdge aktiviert

Die Kante wird gelöscht (4). Auslösende Methodenaufrufe: simpleSDLEdge.delete()  
 Informationen auf dem Stack:

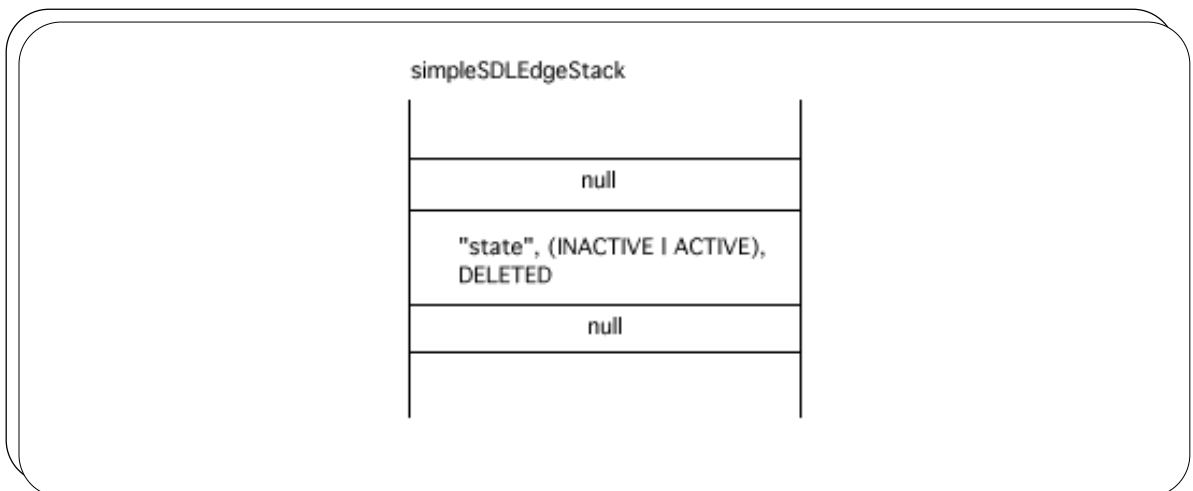


Abbildung 46 : Stackinhalt - SimpleSDLEdge gelöscht

### **5.1.7 Undo-Informationen der Klasse FlashingSignal**

Auf der Klasse FlashingSignal ist 1 Visualisierungsschritt definiert:



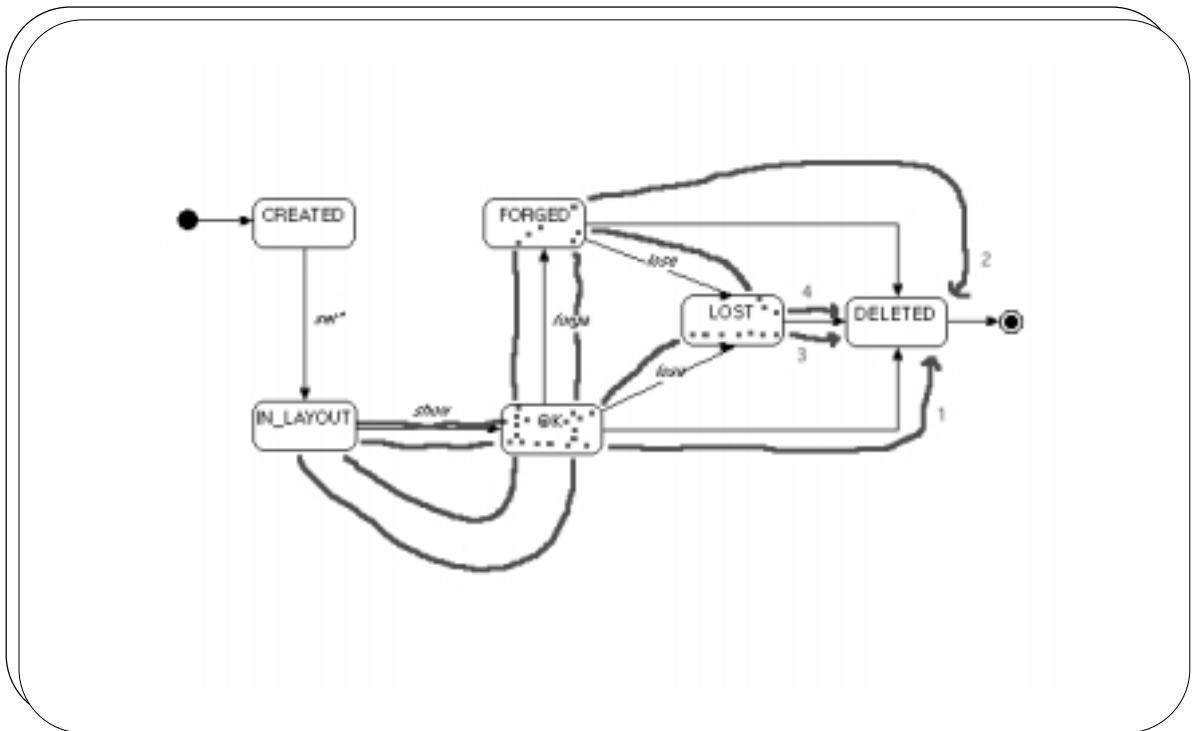


Abbildung 49 : Visualisierungsschritte der Klasse SimpleTravellingSignal

Das Signal wird angezeigt (1). Auslösende Methodenaufrufe: simpleTravellingSignal.show()  
 Informationen auf dem Stack:  
 a.) Das Signal ist noch unterwegs

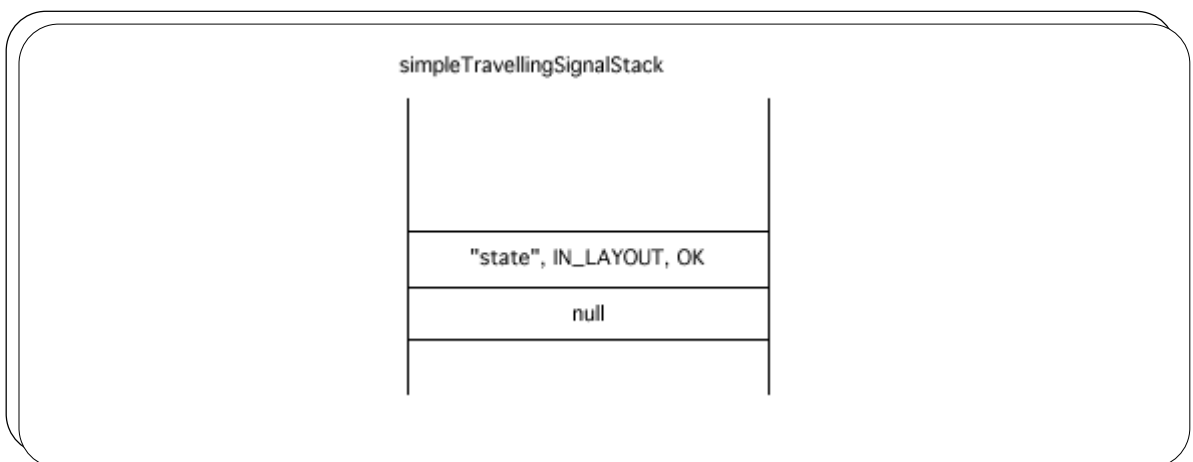


Abbildung 50 : Stackinhalt - SimpleTravellingSignal angezeigt und noch unterwegs

b.) Das Signal ist schon angekommen

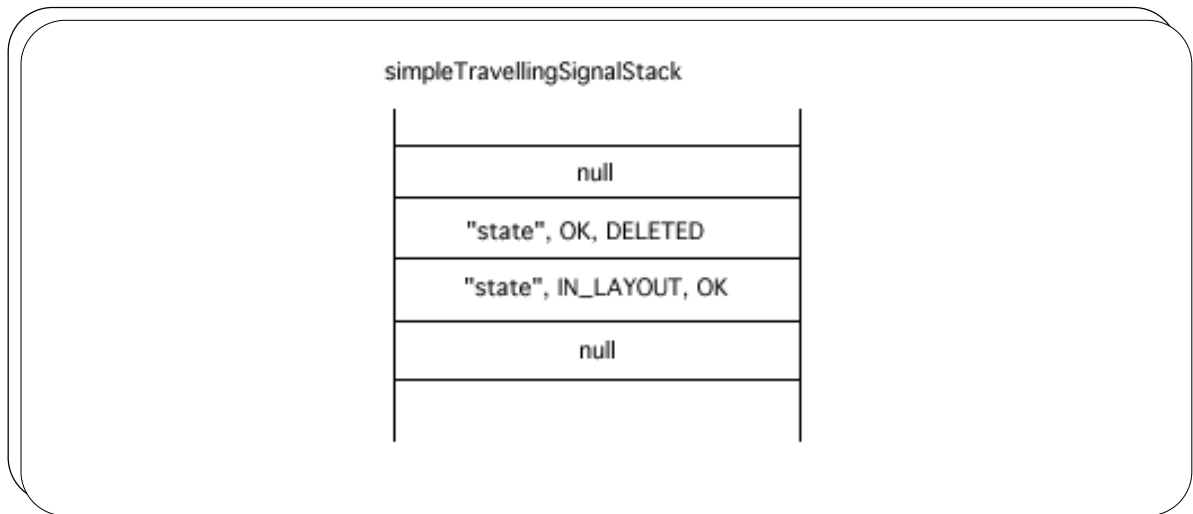


Abbildung 51 : Stackinhalt - SimpleTravellingSignal angezeigt und schon angekommen

Das Signal wird angezeigt und verfälscht (2). Auslösende Methodenaufrufe:

simpleTravellingSignal.show() - simpleTravellingSignal.forge()

Informationen auf dem Stack:

a.) Das Signal ist noch unterwegs

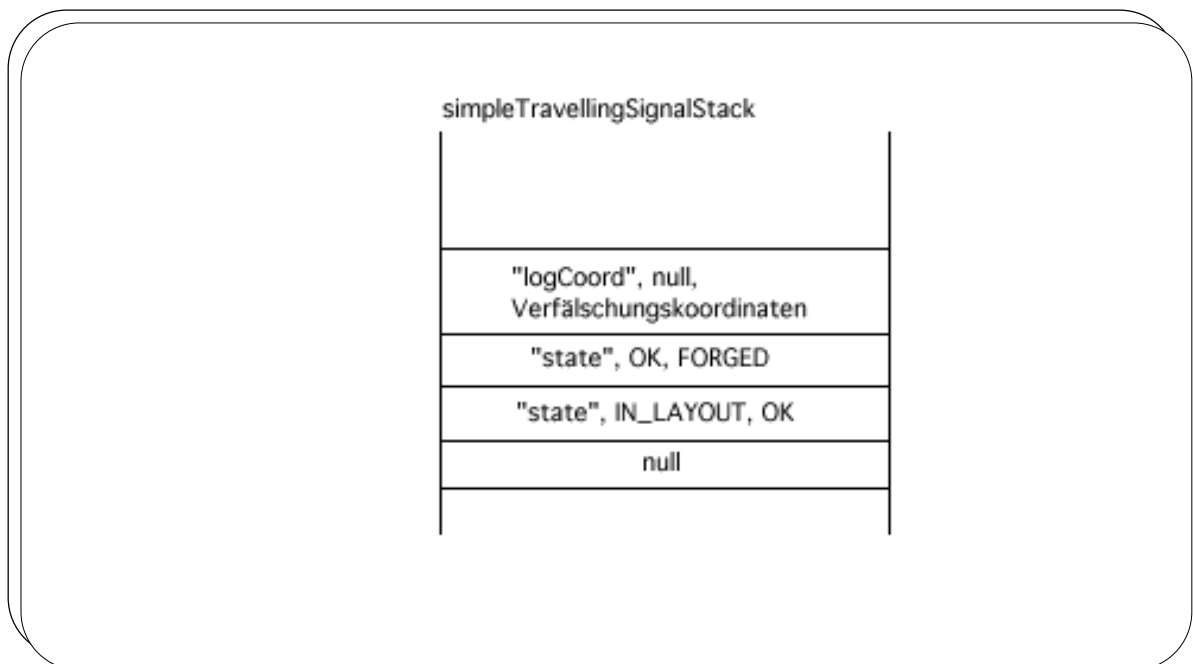


Abbildung 52 : Stackinhalt - SimpleTravellingSignal angezeigt, verfälscht und noch unterwegs

b.) Das Signal ist schon angekommen

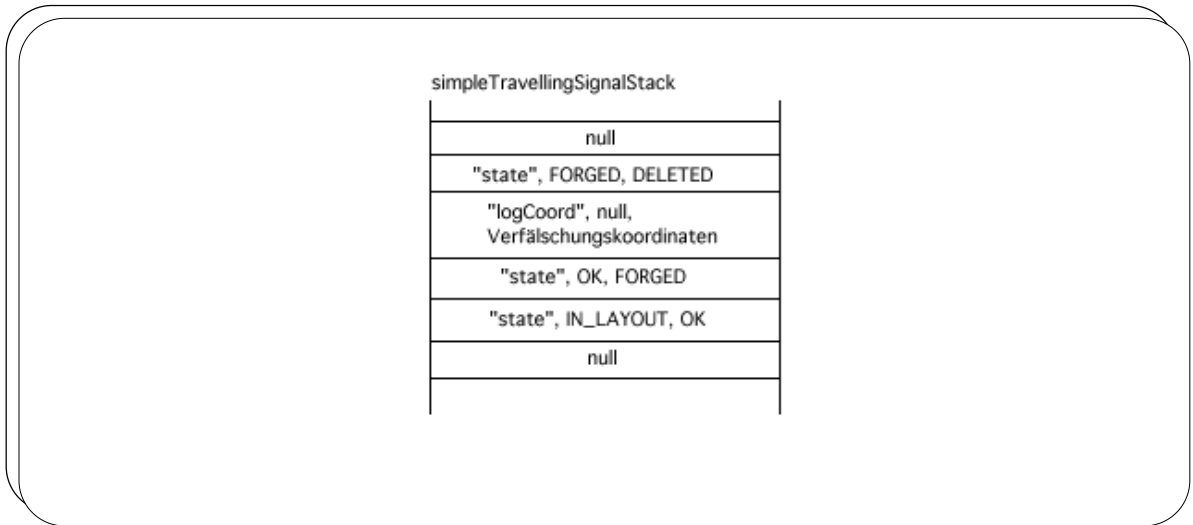


Abbildung 53 : Stackinhalt - SimpleTravellingSignal angezeigt, verfälscht und schon angekommen

Das Signal geht verloren (3). Auslösende Methodenaufrufe: simpleTravellingSignal.show() - simpleTravellingSignal.lose()  
Informationen auf dem Stack:

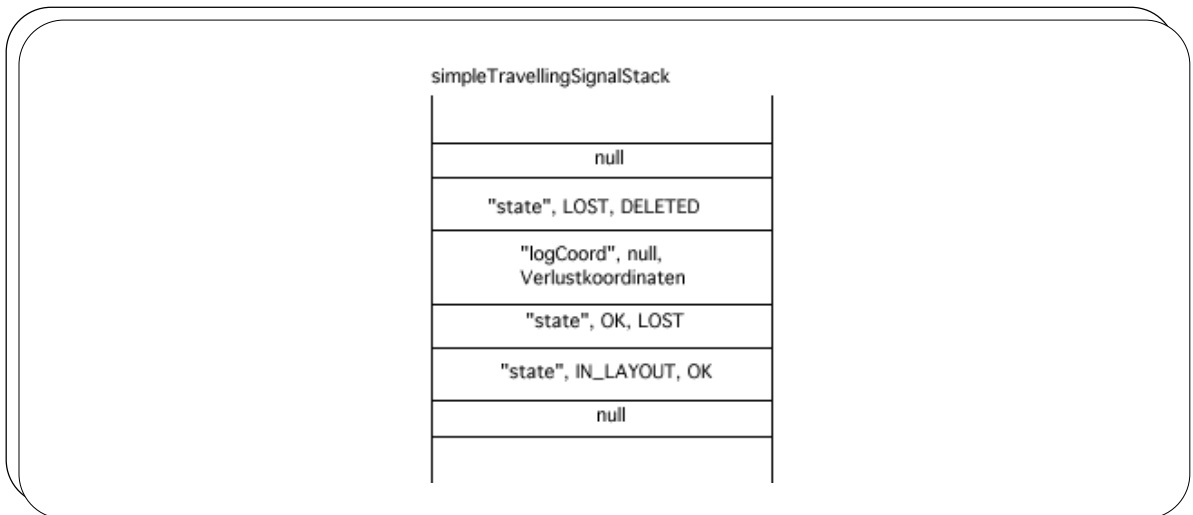


Abbildung 54 : Stackinhalt - SimpleTravellingSignal angezeigt und verloren

Das Signal wird angezeigt, verfälscht und geht verloren (4). Auslösende Methodenaufrufe: simpleTravellingSignal.show() - simpleTravellingSignal.forge() - simpleTravellingSignal.lose()  
Informationen auf dem Stack:

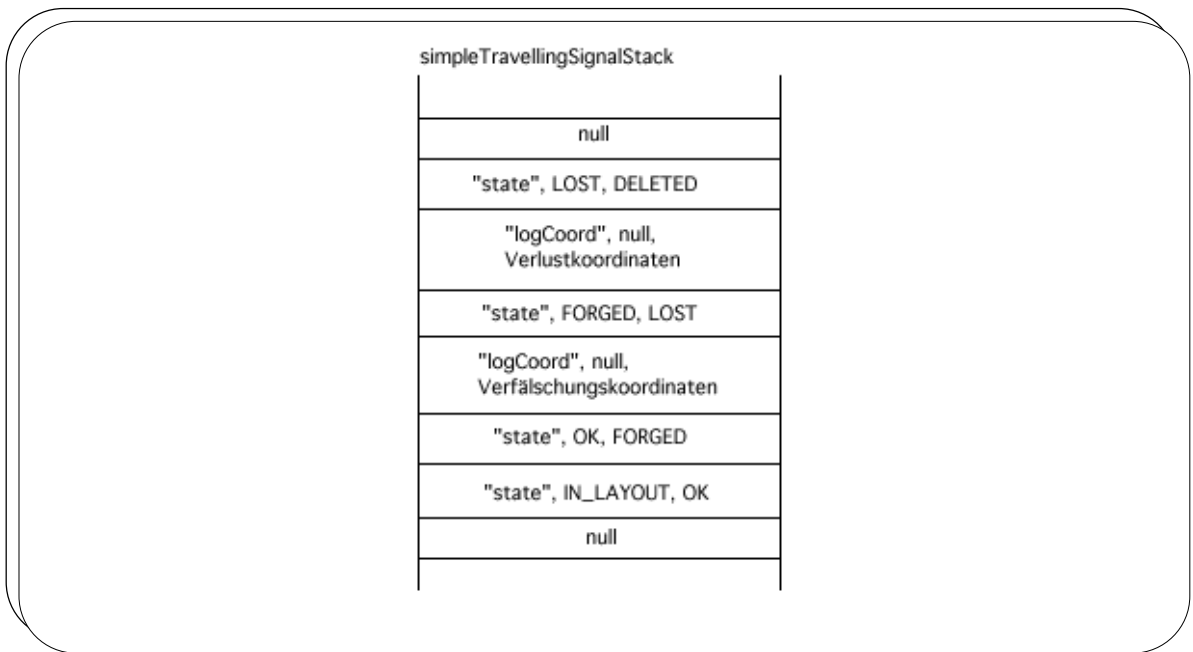


Abbildung 55 : Stackinhalt - SimpleTravellingSignal angezeigt, verfälscht und verloren

### **5.1.9 Undo-Informationen der Klasse AnimatedTravellingSignal**

Auf der Klasse `AnimatedTravellingSignal` sind die gleichen Visualisierungsschritte wie auf ihrer Oberklasse `SimpleTravellingSignal` definiert.

## 5.2 Entwurfsänderung des " ImageString " - Panel

Bei der Implementierung eines Prototyps für den " ImageString " Panel wurde der Ansatz für eine dynamische Anpassung des " ImageString " - Panels bei Eintragungen verworfen. Das Layout des Panels wurde zerstört bei der Anpassung von mehreren Einträgen (Icons, Labels) was nicht erwünscht wurde, da die anderen Panel kleiner wurden und die Größe der Buttons bei mehreren Einträgen immer kleiner wurden bis zur Unleserlichkeit.

Ein Eintrag besteht aus einem Icon und einer Beschriftung (Label).

Es wurde daher die Anzahl der Einträge auf maximal 10 begrenzt. Für das " ImageString " - Panel wurde ein GridLayoutManager Manager ausgewählt der bis zu maximal 10 Einträgen (5 Zeilen, 4 Spalten) erlaubt. Diese 10 Einträge sollten ausreichen, da im Container sinnvollerweise maximal 6 bis 7 Elemente gleichzeitig auftreten können. Wenn die Anwendung mehr wie 10 Einträge darstellen will kann ein weiteres " ImageString " - Panel erzeugt werden.

## 5.3 Lösung des Anpassungsproblems Image - Canvas

Bei der Implementierung des " ImageString " - Panels bestand das Problem ein Image von beliebiger Breite und Höhe in ein Canvas von ebenfalls beliebiger Breite und Höhe einzufügen. Da die Größe des Panels vom Anwendungsprogramm abhängt, sind die Breite und Höhe des Canvas Anfangs unbekannt. Nachdem das Panel erzeugt wird, besteht die Möglichkeit die Breite und Höhe des Canvas als auch später bei der Übergabe des Image vom Anwendungsprogramm dessen Größe abzufragen. Die Anpassung des Images soll dabei an einem Beispiel demonstriert werden.

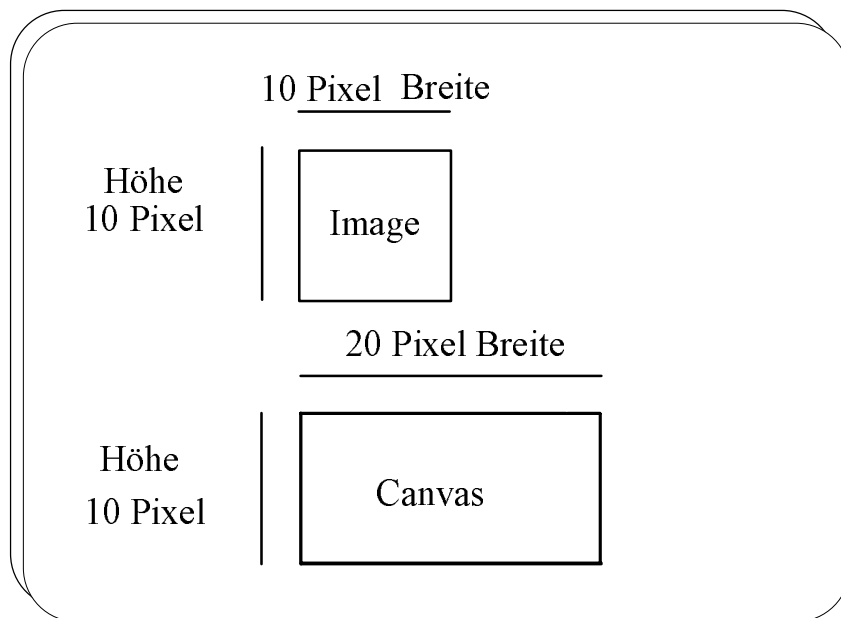


Abbildung 56 : Image - Canvas Anpassung

Als erstes wird das Verhältnis von Canvas und Image bestimmt. (Breite von Canvas / Breite von Image) = 2 d.h die Breite des Canvas ist doppelt so groß wie die Breite des Image.

Anschließend wird (Höhe des Canvas / Höhe des Image = 1) ausgerechnet.

Die Frage die sich stellt, ist mit welchem Faktor soll multipliziert (Skalierungsfaktor) werden.

Wenn die Breite des Image mit 2 multipliziert und die Höhe mit 1 ergibt sich eine Verzerrung des Bildes. Dies sollte aber nicht eintreten. Um keine Verzerrung des Bildes zu erreichen, muß die Breite und Höhe des Image mit dem selbem Faktor multipliziert werden damit das Verhältnis zwischen der Höhe und Breite konstant bleibt. Daraus folgt, daß das Image mit dem kleinerem Faktor in diesem Fall 1 multipliziert wird und so keine Verzerrung erfolgt.

Bei einer Vergrößerung eines Image wird mit einer Zahl  $> 1$  multipliziert und entsprechend bei einer Verkleinerung mit einer Zahl  $< 1$  multipliziert. Das Ergebnis ist, das sich das Image dem Canvas angepaßt hat. Die Anpassung erfolgt in einer Hilfsklasse von der Klasse "ImageString" die "ImageCanvas" genannt wird und vom Anwendungsprogramm indirekt benutzt wird.

## 6. Zusammenfassung und Ausblick

### 6.1 Zusammenfassung

Im Fokus der Arbeit lagen dabei mehrere Aspekte: Die Forderung den Visualisierungsbaukasten auf die aktuelle Java-Version (1.1.2) zu portieren. Da die Programmiersprache "Java" plattformunabhängig ist, können Java-Programme mühelos auf jedem beliebigen Computersystem ausgeführt werden auf dem eine virtuelle Java-Maschine (Bytecode-Interpreter) installiert ist, ohne die Notwendigkeit den Quellcode neu kompilieren zu müssen.

Mit Hilfe dieser neuen Version sollte die Benutzerschnittstelle derart verbessert werden, daß sowohl "Look - and - Feel" als auch die Funktionalität der Benutzerschnittstelle an Qualität gewinnen sollten.

Eine grundlegende Funktionalität zur Rücknahme bestimmter Schritte die als "Undo" bezeichnet wurde, sollte ebenfalls in dieser Studienarbeit spezifiziert und implementiert werden. Ein höherer Grad von Interaktivität und Verständlichkeit ist hierbei das Ziel gewesen.

### 6.2 Ausblick

In der beschriebenen Undo - Struktur ist es auch noch möglich, andere Container außer SDL und FSM (Fine State Machine) zu integrieren.

Weiterhin wird durch eine weitere Verbreitung der Programmiersprache Java mit ihrer Plattformunabhängigkeit und sowohl dem größeren Einsatz von Rechnern zur Verbesserung der Lehre als auch der Ansatz des neuen Visualisierungsbaukastens auf einer graphenbasierten Lösung, das Ziel der **ProtoVis Projektgruppe** ein generisches System zur Visualisierung von Protokollen, einen Schritt weiter bringen.

Also, weiter so PROTOVIS - PROJEKTGRUPPE.

## 7. Literatur

Peter W. Schurr, (Studienarbeit Nr. 1608, Universität Stuttgart 1997)  
" Visualisierung von Lehrinhalten mittels Java Applets (Java-Baukasten) "  
<http://www.informatik.uni-stuttgart.de/ipvr/vs/personen/burger/lehre/visualisierung/>

Edgar Kogel, (Studienarbeit Nr. 1522, Universität Stuttgart 1997)  
" Visualisierung von Kommunikationsprotokollen mit Java "

Ralph Gorth, (Studienarbeit Nr. 1671, Universität Stuttgart 1998)  
" Erweiterung des Java-Baukastens zur Visualisierung von Prokollen:  
Erweiterung des Baukastens zur Visualisierung von SDL-Konstrukten "

Homepage von Javasoft  
<http://www.javasoft.com>

Homepage von Sun - Microsystems  
<http://www.sun.com/>

" ProtoVis " (Protokoll-Visualisierung) Homepage des Instituts für Parallele und Verteilte  
Höchstleistungsrechner der Universität Stuttgart,  
<http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/ProtoVis/protovis.html>

ESTELLE, LOTOS, und SDL, Geleitwort von F. Vogt,  
" Standard-Spezifikationsprachen für verteilte Systeme ", Springer-Verlag, Berlin

J. Rumbaugh, M. Blaha, W. Premerlani, F.Eddy, W.Lorensen,  
" Objektorientiertes Modellieren und Entwerfen ", Carl Hanser - Verlag und  
Prentice-Hall International, Wien - London 1993

Ralf Melchisedech, Folien zur Vorlesung:  
" Konzeption und Aufbau objektorientierter Software ", WS 1996  
Universität Stuttgart, Institut für Informatik, Abteilung Software Engineering

Gary Cornell, Cay S.Horstmann,  
" Core Java ", The SunSoft Press, Mountain View, California 1996

Ralf Reißing,  
Folien zum " Java Kompaktkurs ", 14.07.1997 - 18.07.1997  
Universität Stuttgart, Institut für Informatik, Abteilung Software Engineering

Laura Lemay, Charles L.Perkins,  
" Java 1.1 in 21 Tagen ", Markt & Technik - Verlag, München 1997

David Flanagan,

" Java in a Nutshell ", Verlag O' Reilly & Associates, 2. Auflage, Köln 1997

Antje Binas-Holz, Michael Schumann

" Das Programmierhandbuch Java 1.1 ", Sybex - Verlag, 2. Auflage, Düsseldorf 1997

Wolfgang H.-D. Kierdorf

" Internet-Programmierung mit Java ", ECON Taschenbuch Verlag, Düsseldorf 1996

James Gosling, Frank Yellin & das Java-Team,

" Das Java API, Band 1: Die Basispakete ", Addison-Wesley-Longman, Bonn 1997

James Gosling, Frank Yellin & das Java-Team,

" Das Java API, Band 2: Das Windows Toolkit und Applets ", Addison-Wesley-Longman,  
Bonn 1997

## **Anhang**

Aufgeführt werden im folgenden mit Hilfe des Java - Dokumentationsgenerators "Javadoc" die Klassen mit den wichtigsten Methoden und entsprechenden Kommentaren, sowie ihren Parametern und eventuellen Rückgabewerte.

---

## Class `protovis.visapi.ButtonPanel`

```
java.lang.Object
|
+----java.awt.Component
      |
      +----java.awt.Container
            |
            +----java.awt.Panel
                  |
                  +----protovis.visapi.ButtonPanel
```

---

```
public class ButtonPanel
extends Panel
implements ActionListener, AdjustmentListener
```

This class represents an " Panel " with several Buttons

---

`ButtonPanelMethod()`

This method is used to visualize the " Button " - Panel from the programmer

`ButtonPanelMethod1(Color, Font)`

This method is used to visualize the " Button " - Panel from the programmer

---

```
ButtonPanelMethod
public void ButtonPanelMethod()
```

```
ButtonPanelMethod1
public void ButtonPanelMethod1(Color color,Font writing)
```

Parameters:

color - The programmer adds the Color  
writing - The programmer adds the Font

---

## Class `protovis.visapi.Area1Panel`

```
java.lang.Object
|
+----java.awt.Component
      |
      +----java.awt.Container
            |
            +----java.awt.Panel
                  |
                  +----protovis.visapi.Area1Panel
```

---

```
public class Area1Panel
extends Panel
```

This class represents the " TextArea " - Panel

---

```
Area1PanelMethod()
```

This method visualize the " TextArea " - Panel

```
clear()
```

This method clears all the contents of the " TextArea " - Panel

```
display(String)
```

This method adds a String to the " TextArea " - Panel

---

```
display
```

```
public void display(String arg)
```

```
Parameters:
```

```
arg - The String to add to the " TextArea " - Panel
```

```
clear  
public void clear()
```

```
Area1PanelMethod  
public void Area1PanelMethod()
```

-----

---

## Class `protovis.visapi.Area2Panel`

```
java.lang.Object
|
+----java.awt.Component
      |
      +----java.awt.Container
            |
            +----java.awt.Panel
                  |
                  +----protovis.visapi.Area2Panel
```

---

```
public class Area2Panel
extends Panel
```

This class represents the " Statuszeile " - Panel

---

```
Area2PanelMethod()
```

This method visualize the " Statuszeile " - Panel

```
status(String)
```

This method adds a String to the " Statuszeile " - Panel

---

```
status
public void status(String str)
```

Parameters:

arg - The String that is to add to the " Statuszeile " - Panel

```
Area2PanelMethod
public void Area2PanelMethod()
```

---

---

## Class `protovis.visapi.Undo`

```
java.lang.Object
|
+----protovis.visapi.Undo
```

---

```
public class Undo
extends Object
```

This class is the center for all undo activity. The several containers register the visualization steps occurred in them and get called by it when the programmer requests a global undo.

---

```
undo()
```

This method performs a global undo. The Undo class calls the appropriate container where the last visualization step occurred.

---

```
undo
public static void undo()
```

---

## **Erklärung**

Ich erkläre hiermit, die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel benutzt zu haben.

Sindelfingen, den

\_\_\_\_\_ (Papoulidis Konstantinos)