INSTITUT FÜR INFORMATIK
der Ludwig−Maximilians−Universität München

Fortgeschrittenen−Praktikum

Entwicklung und Implementierung von Topologie−Visualisierungselementen in Java

Aufgabensteller: Prof. Dr. Heinz−Gerd Hegering
Betreuer: Christian Ensel
Bearbeiter: Alessandro Macrì

20.8.2001

Inhaltsverzeichnis

1. Einführung

Um in der modernen EDV ein Maximum an Flexibilität und Leistung zu erzielen, müssen die verwendeten Systeme, Hardware wie Software, immer vielfältigere Aufgaben wahrnehmen. Entsprechend steigt ihre Zahl stetig an, was zu immer neuen Anforderungen an das Management der gesamten IT−Infrastruktur führt. Inbesondere die Verbindungen und Schnittstellen zwischen einzelnen Komponenten lassen sich immer schwerer überblicken und werden deshalb oft mittels Software−Werkzeugen (Managementplattformen) verwaltet.
Auch diese Werkzeuge nehmen jedoch immer komplexere Formen an und liefern dem Verwalter mehr Daten, als dieser über eine gewöhnliche graphische Oberfläche sinnvoll aufnehmen kann. Diesem Problem kann durch die Einführung einer Zwischenschicht, eines sog. „surface tools“ begegnet werden. Das surface tool ersetzt dabei einen Teil der eigentlichen GUI, nimmt von der zugrundeliegenden Managementanwendung (MA) Daten entgegen und transformiert sie in eine Darstellung, die dem menschlichen Betrachter entgegenkommt. Dabei können die Rahmenmedingungen der Darstellung durch die MA beliebig gehandhabt werden. Dieses Fortgeschrittenen−Praktikum beinhaltet die Entwicklung einer Visualisierungkomponente, welche einen Großteil der Aufgaben eines solchen ST zur Darstellung von vernetzten Umgebungen übernimmt

2. Problemstellung und Anforderungen

2.1. Problemstellung

Aufgabe dieses Praktikums war es, eine Visualisierungskomponente (VK, Topology Viewer) zu entwickeln, die einem menschlichen Betrachter eine darzustellende Topologie in geeigneter Form darbietet. Der Komponente werden dazu von einer beliebigen MA Informationen über die Topologie zur Verfügung gestellt, die sie für das menschliche Auge aufbereitet.
Als Anhaltspunkte für die Strukturierung wurden gängige Netztopologien betrachtet, wobei für funktional oder topologisch zusammenhängende Komponenten eine gruppierte, für funktional getrennte Gruppen aber eine getrennte Darstellung sinnvoll ist.
Weiter muß der Betrachter anhand dieser Darstellung auch mit der zugrundeliegenden MA kommunizieren können, d.h. es muß ihm möglich sein, die angebotenen Funktionen der MA bzgl. des visualisierten Systems auszuschöpfen. Der Einsatzschwerpunkt der VK sind die Visualisierung von Rechnernetzen, sowie Systemen, die über Rechnernetzen ausgeführt werden, zu sehen.
Diese Überlegungen führten zu folgenden Kriterien:

(a) die VK muß <u>Strukturen, wie sie in der EDV typischerweise verwendet werden</u>, sinnvoll anordnen und darstellen können; derartige Strukturen werden in Kap. 4 untersucht und beschrieben
(b) das <u>Datenmodell</u> wird von einer MA gestellt, welche sich der VK bedient
(c) die <u>Kommunikation des Anwenders mit der MA</u> findet ausschließlich über das Datenmodell statt; die Visualisierung sorgt für die Transparenz der Kommunikation zwischen Mensch und Managementanwendung. Die Objektrepräsentationen entsprechen also den zugrundeliegenden managed objects und kann quasi als Ein−/Ausgabe−schnittstelle zur Managementanwendung betrachtet werden.

2.2. Anforderungen an die Art der Implementierung

Da die VK, wie oben beschrieben, als Schnittstelle zu einer Managementanwendung dient, muß auch ein Implementierungsrahmen definiert werden, der diese Abhängigkeit programmtechnisch definiert:

- die MA muß den zu visualisierenden Objekten Aktionen zuordnen können, welche dem Anwender zur Ausführung angeboten werden
- die VK muß systemunabhängig einsetzbar sein (unabdingbar für den Einsatz in heterogenen Umgebungen). Speziell sollte die VK im Rahmen des Agentensystems MASA eingesetzt werden können

Diese Anforderungen flossen direkt in einen Lösungsansatz ein.

3. Lösungsansatz

3.1. Implementierungsmerkmale

Um die unter 2.2. genannten Anforderungen zu erfüllen bietet sich die Implementierung in Java an. Da Java–Programme durch eine virtuelle Maschine interpretiert werden, sind sie unabhängig von der vorhandenen Hardware–Architektur und somit auch in heterogenen Umgebungen einsetzbar. Durch die Einbindung der Swing–Bibliotheken, die ab Java 1.2 obligatorisch sind (Einbindung in die Standard–API), muß die Darstellung nicht mehr spezifisch auf die verwendete Oberfläche zugeschnitten werden, was (a) entgegenkommt, da dem Anwender immer die gleiche Ansicht geboten werden kann.

Weiter ist das Swing–typische Model–View–Control–Konzept (MVC–Konzept) geeignet um (b) und (c) nahezu vollständig abzudecken: die MA stellt ein Modell bereit, bei dem sie die VK „anmeldet". Dieses Modell ist im MVC–Konzept die alleinige Grundlage für die darzustellenden Informationen und somit getrennt von der Art der Darstellung. Benutzeraktionen auf der Darstellung werden zur Bearbeitung an das Datenmodell weitergeleitet, sodaß die Transparenz der Schnittstelle gewährleistet ist.

3.2. Der CORBA Topology Service

Zur Speicherung und Vermittlung der Daten bietet sich der CORBA Topology Service (TS) an, wie er in einer Spezifikation der Hewlett–Packard Company (16.10.1997) vorgeschlagen wurde. Die Verwendung des Service würde zudem eine einheitliche Schnittstelle für jegliche gängige Plattform bereitstellen, ohne implementierungsspezifische Merkmale zu beinhalten. Zudem existiert bereits eine Implementierung in Java, die im Rahmen eines früheren Fortgeschrittenen–Praktikums entstand [Rö99].

Allerdings sprechen drei wesentliche Gründe gegen die Verwendung des TS:

1. Das Datenmodell des TS ist inhärent flach. Objekte können in Aggregationen zusammengefaßt („aggregate entity"), diese Aggregationen durch Relationen verbunden werden (Abb. 1).
   Dabei werden die aggregierten Einheiten über die enthaltenen „topologically managed entities" identifiziert. Diese Untereinheiten können zwar umfangreich typisiert sein, jedoch nur ein einzelnes Attribut enthalten. Die aggregierte Einheit wird dann durch sämtliche Untereinheiten charakterisiert.
   Eine echte Strukturierung, z.B. im Sinne einer hierarchischen Containment–Relation, kann nur durch eine aufgeprägte Typisierung erzielt werden, der man eine Interpretations–semantik auferlegt.

2. Die im TS abgelegten Topologie−Daten werden durch Anfragen in einer speziellen Sprache („Topology Query Language“) ausgelesen. Diese Sprache agiert über den aggregierten Einheiten. Anfragen beziehen sich dabei stets auf Assoziationen zw. aggregierten Einheiten und deren gesammelte Typisierung. Ein strukturiertes topologisches Datenmodell müßte erst durch mehrere solche Anfragen ermittelt und die Semantik der gefundenen Objekte interpretiert werden.

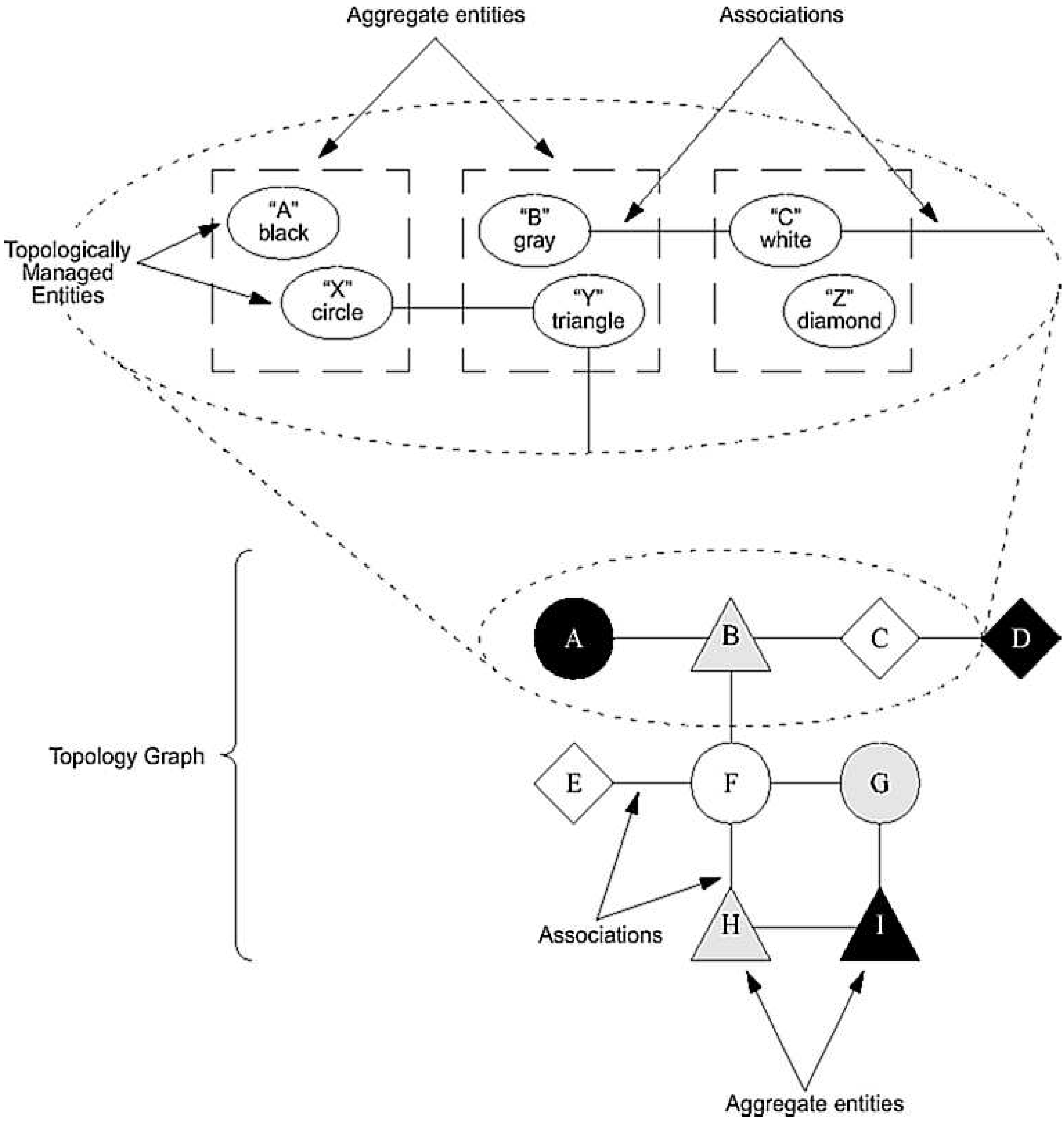


Abb. 1 : Datenmodell des CORBA Topology Service (entnommen aus [HP97])

3. Der TS wurde nur als Konzept von der Entwicklergemeinschaft begrüßt, stieß jedoch auf wenig Gegenliebe, sobald die exakten Spezifikationen vorlagen. Zum einen vermischte er verschiedene Schichten aus der Modellierung der CORBA Services, zum anderen wurde teilweise durch andere CORBA−Dienste aufgehoben. Dies bewog Hewlett−Packard kurz nach der Fertigstellung ihres Beitrags zum RFP der Telecom DTF der OMG, diesen wieder zurückzuziehen, da man daran glaubte, daß die neueren Entwicklungen in der OMG diesen unnötig gemacht hatten. Ersatz soll vor allem der allgemeinere Entity−Relationship−Service darstellen, insbesondere, da die weitere Entwicklung von CORBA strukturiertere Datentypen anbietet [MN97]. Allerdings wird auch diese Variante in der Praxis kaum verwendet.

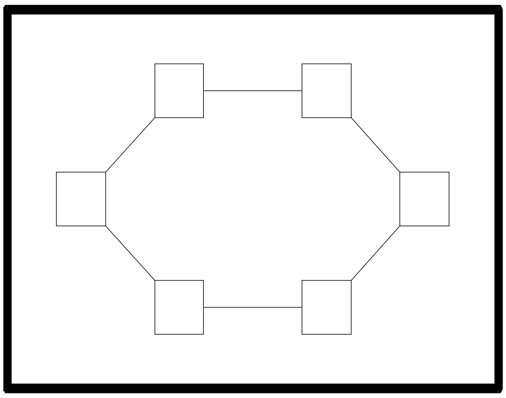
Als Schlußfolgerung wurde die Verwendung des Corba Topology Service verworfen und stattdessen ein Paket von Java−Klassen zur Beschreibung einer strukturierten Topologie entworfen und implementiert.

3.3. Grundtypen des Datenmodells

Punkt (a) der Kriterien stellt die weitaus wichtigste Frage: welche Strukturen, die in der EDV, bzw. in der Strukturierung und Organisation von Rechnersystemen, üblicherweise verwendet werden, sollen dargestellt werden?
Dabei können sich die visualisierten Strukuren durchaus von den realen unterscheiden. Wichtig für die Wahrnehmung durch den Menschen ist vielmehr die Umsetzung der logischen und funktionalen Bezüge.
Die Wahl fiel auf eine Modellierung, die verschiedene Strukturen hierarchisch schachteln kann. Als Basisstrukturen dienen folgende drei aufgeführten Graphentypen.

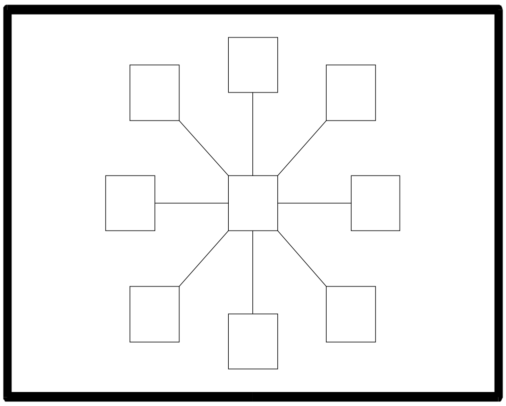
I. Ring

Unter einem Ring versteht man eine Menge von Komponenten, welche jeweils mit genau zwei anderen Elementen der Menge verbunden sind. Der Graph muß jedoch zusammenhängend sein.
In der Praxis kommen solche Topologien bei Token–Ring–Netzen vor (IEEE 802.5), z.B. FDDI und CDDI (auch wenn die Topologie der physikalischen Kabelverbindung nicht mit der logischen Netz–topologie übereinstimmt.



II. Stern

Unter einem Stern versteht man eine Menge von Komponenten, die ein ausgezeichnetes Element enthält (Zentrum, Nabe, Hub). Das ausgezeichnete Element ist mit jedem anderen Element der Menge verbunden, jedes andere Element jedoch bloß mit dem ausgezeichneten Element. Physikalische Sterntopologien werden oft zur Abbildung anderer logischer Topologien verwendet und mit verschiedenen Zugangsverfahren, wie TDMA, Tokenzugangsverfahren oder auch CSMA/CD, betrieben werden (Beispiele: IEEE 802.3j –1993, StarLAN).



III. Bus

Ein Bus ist i.W. ein Spezialfall des Sterns. Das ausgezeichnete Element, welches alle anderen Elemente der Menge verbindet, besitzt jedoch keine weiteren Attribute und wird deshalb nicht als Einzelkomponente, sondern eben als inhärente Verbindung gesehen, z.B. als physikalisches Übertragungsmedium. In der Netzwerkarchitektur sind Ethernet (IEEE 802.3) oder Token–Bus (IEEE 802.4) typische Vorkommen.

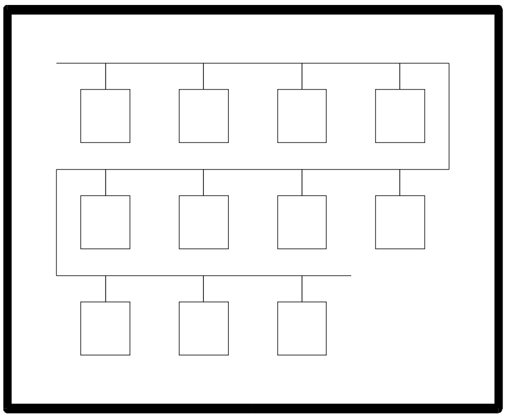


Abb. 2 : typische Strukturen

Natürlich gibt es zu diesen Topologien auch Mischformen, wie z.B. Stern–Ring–Topologien, in denen ein zentrales Element das Kommunikationsmanagement der einzelnen Komponenten übernimmt, um Ausfälle besser kompensieren zu können. Diese Topologien lassen sich jedoch in ihrer logischen Struktur in der Regel auf die oben erwähnten abbilden.
Weiter tritt z.B. in WANs oft die Situation auf, daß die einzelnen Kommunikationssysteme und –bausteine frei vermascht werden. Die Visualisierung eines solchen Graphen würden jedoch weit über diese Arbeit hinausgehen.

3.4. Hierarchie und Typisierung des Datenmodells

Die unter 3.3. beschriebenen Strukturtypen können nun auch hierarchisch geschachtelt werden, d.h. jede Komponente eines Grundtyps kann wiederum aus einer Menge von Elementen bestehen, die ebenfalls als einer der drei Grundtypen organisiert ist. Dadurch kann eine hierarchisch strukturierte Containment–Relation auf dem Modell aufgebracht werden. Z.B. könnten die äußeren Komponenten eines Stern jeweils Gruppen von anderen Komponenten sein, die als Bus organisiert sind. (Anmerkung: eine Ausnahme hierzu ist der Hub des Sternmodells. Hier ist von einer Einzelkomponente auszugehen.)
Das Datenmodell muß jedoch auch Verbindungen zulassen, welche über die durch die Strukturhierarchie vorgegebenen Verbindungen („implizite Verbindungen“) hinausgehen („globale Verbindungen“), welche auch zwischen Elementen existieren dürfen, die nicht direkte Untersysteme desselben Systems sind.
Das endgültige Datenmodell, beinhaltet dann

- die drei Grundgraphen,
- die Möglichkeit diese ineinander zu schachteln, um baumähnliche Strukturen zu erzeugen,
- Verbindungen zwischen Elementen, die nach den obigen zwei Punkten nicht direkt verbunden sind.

Diese Punkte sind in Abb. 3 illustriert.

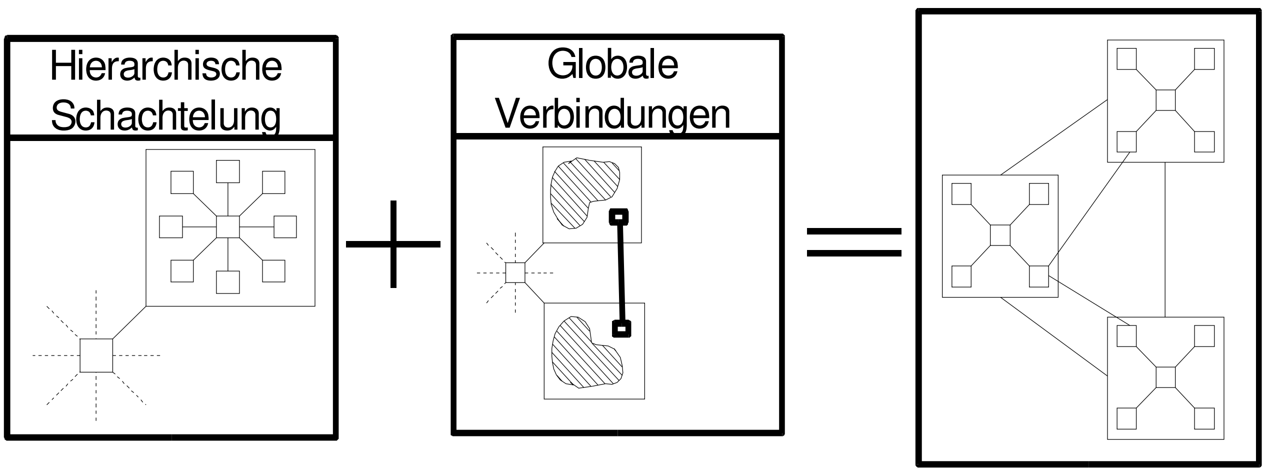


Abb. 3 : tiefere Strukturierung des Datenmodells

Weiter muß das Datenmodell klar festlegen, ob eine Komponente aus einer Menge von anderen Komponenten besteht („Polystruktur“), oder ob es sich um eine einzelne, nicht weiter aufteilbare Komponente handelt („Monostruktur“).

3.5. Sonstige Attribute

Komponenten verfügen zum besseren Handling für den Benutzer, die VK und die MA über verschiedene weitere Attribute, je nach Typ:

1. Allgemein:
   - Name: ein Identifikator, der in der Darstellung mit dem entsprechenden Symbol verbunden ist
   - Kommentar: beliebige Texte, die der Benutzer zu einem Objekt abrufen kann
   - zuzuordnendes Icon: die eigentliche Repräsentation des Objekts
   - Verweis auf das Mutterobjekt: die Aufwärtsrichtung der Containment−Relation
2. Monostruktur:
   - Typ: festzulegen durch die MA, z.B. Datenendeinheit, Datenübertragungseinheit, etc.
3. Polystruktur:
   - Liste der enthaltenen Subsysteme: Abwärtsrichtung der Containment−Relation
   - Liste der in dieser Struktur enthaltenen globalen Verbindungen: alle globalen Verbindungen, die nicht über diese Struktur hinausgehen, aber auch nicht komplett in einem der Subsysteme untergebracht werden könnnen

(Anm. : aus Effizienzgründen wird die Containment−Relation in beiden Richtungen gespeichert. Dies erleichtert auch evtl. Prüfungen auf Inkonsistenzen durch die MA)
Desweiteren ist es sinnvoll einige Informationen über die gewünschte Art der Darstellung bei der Komponente selbst abzulegen.

4. Architektur

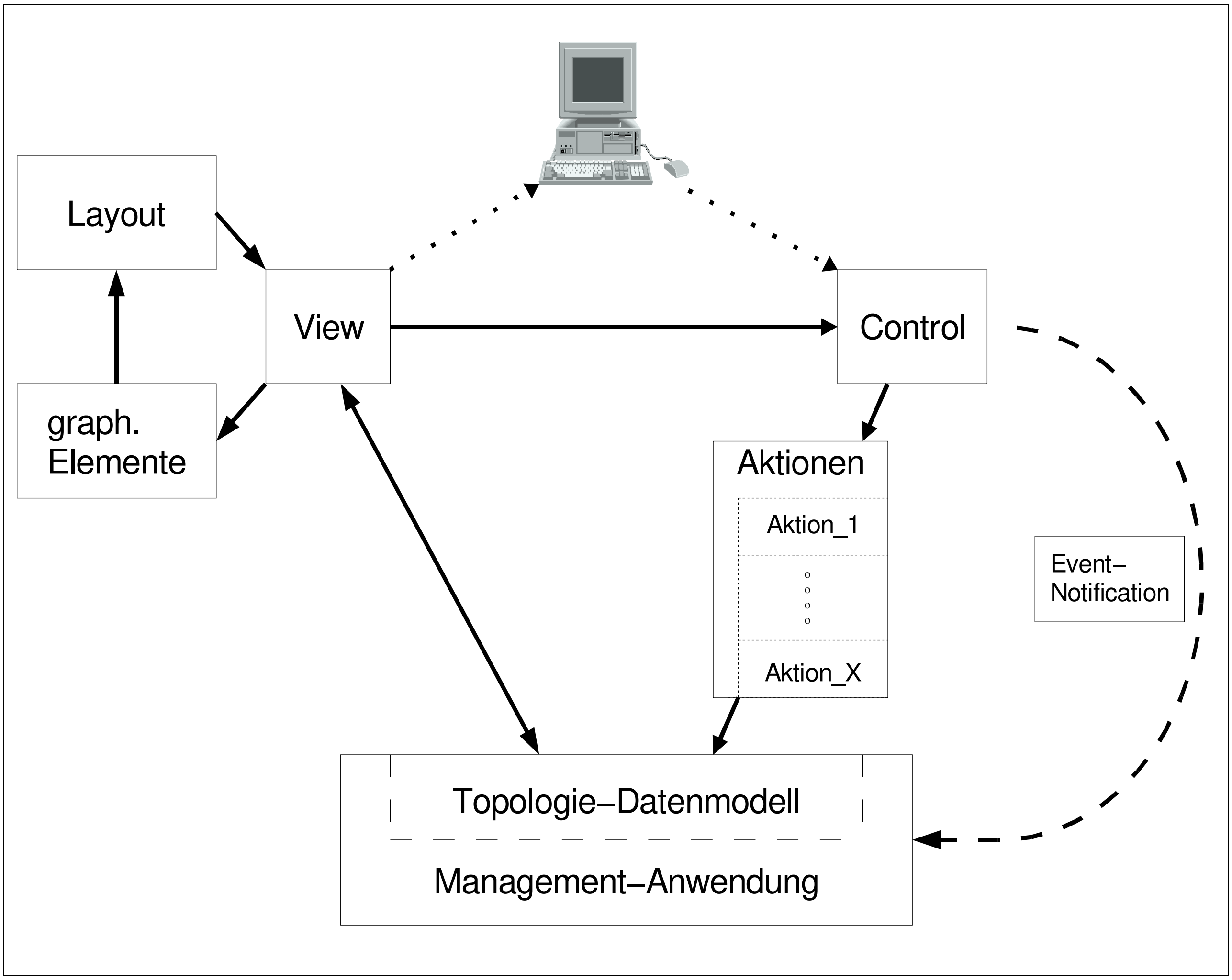


Abb. 4: Architektur der VK

In Abb. 4 wird die Architektur der VK schematisch dargestellt. Der wesentliche Arbeitsablauf bewegt sich um das Dreigespann Modell–View–Control.
Dabei enthält das Modell („*Topologie–Datenmodell*“) sämtliche Daten, die visualisiert werden sollen, sowie evtl. Darstellungspräferenzen, die durch die MA auferlegt werden. Weiter kann die MA jedem zu visualiserenden Objekt ein oder mehrere Aktionsobjekte zuordnen, die auf diesem ausgeführt werden können.
Die Viewkomponente („*View*“) erledigt die Darstellung der Daten auf dem Ausgabemedium (im Allgemeinen der Bildschirm). Dazu bedient sie sich einer ausgelagerten Layoutkomponente (*Layout*“*)*, die sich um Plazierung und Rendering der graphischen Anzeigeelemente kümmert. Die fertig ausgerichtete Darstellung wird durch die Viewkomponente angezeigt.
Alle Operationen des Anwenders auf der Darstellung werden durch die Kontrollkomponente („Control“) koordiniert und schlagen sich im Datenmodell nieder. Möchte der Anwender eine der von der MA zur Verfügung gestellten Aktionen ausführen, so ermittelt die Kontrolle aus der Darstellung das zugrundeliegende Objekt des Datenmodells, führt eine Relokation des aufgetretenen Ereignisses auf dieses Objekt durch und übergibt das Ereignis an die entsprechende Aktion. Dies ist aus Sicherheits– und Flexibilitätsgründen die einzige Möglichkeit der Kontrollkomponente, Änderungen am Datenbestand durchzuführen, die sich nicht ausschließlich auf Darstellungsangelegenheiten beziehen. Natürlich sollte die MA nur Aktionen ausführen, die

auf ihre Legitimität überprüft wurden. Die Aktionen des Benutzers werden dabei durch die VK vollkommen transparent an die MA übergeben; aus ihrer Sicht agiert der Benutzer direkt über dem Datenmodell.

5. Beschreibung der Klassen

Natürlich ist die Beschreibung der implementierten Klassen nicht vollständig, sie bietet lediglich einen Überblick über die Verwendung der Klassen im TopologyViewer. Für genauere Informationen und implementierungstechnische Aspekte, sei auf die Programmdokumentattion (javadoc) verwiesen, die wesentlich umfassender ist.

5.1. Datenmodell

Das Datenmodell enthält alle notwendigen Informationen über die darzustellende Topologie, sowie evtl. auch über die Art der gewünschten Darstellung.

Klasse	Aufgabe
TopModel	*TopModel* konzentriert den Zugriff auf die Topologie. Es verweist auf die Struktur und bietet Methoden, die sich auf die Gesamttopologie beziehen, z.B. die Sammlung aller Verbindungen im Modell, die von der Hierarchie abweichen.
TopComponent	*TopComponent* ist die (abstrakte) Basisklasse aller Topologieklassen. Sie führt verschiedene Attribute der Komponenten ein, z.B. Name und Kommentar, sowie eine Liste von Aktionen, die auf der Komponente ausführbar sein sollen.
Link	*Link* symbolisiert eine Verbindung zwischen zwei Strukturen, auf die es verweist. Es enthält ansonsten nur Methoden von interner Bedeutung (clone(), equals(), getSize(), etc.)
Struktur	Von *Struktur* leiten sich alle restlichen Klassen ab. Es implementiert eine Richtung der Containment−Relation (Verweis auf das Mutterobjekt) und einige Angaben, wie die Komponente dargestellt werden möchte, z.B. Darstellungstiefe, etc.
Mono	Diese Klasse beschreibt eine Einzelkomponente, die keine weiteren Subsysteme enthalten kann.
Poly	*Poly* enthält die Verwaltung der Subsysteme einer Polystruktur. Von dieser Klasse leiten sich die drei nachfolgenden speziellen Topologieklassen ab.
Bus	Dies ist die Implementierung der Objektrepräsentation für busähnliche Strukturen, wie in Kap. 3.3. beschrieben.
Ring	Dies ist die Implementierung der Objektrepräsentation für ringähnliche Strukturen, wie in Kap. 3.3. beschrieben.
Star	Dies ist die Implementierung der Objektrepräsentation für sternähnliche Strukturen, wie in Kap. 3.3. beschrieben.

5.2. View

5.2.1. Klassen

Der *View* enthält alle Klassen, welche die direkte graphische Darstellung am Bildschirm erledigen. Er greift dabei stark auf GUI–Klassen aus Swing zurück.

Klasse	Aufgabe
TopView	Die Viewkomponente ist eine Erweiterung von *javax.swing.JPanel*. Sie integriert die gelayoutete Darstellung der Topologie (siehe *5.2.2. TopPanel)* zur Darstellung in einer Fremdanwendung, wie in Abb. 5 dargestellt. Dem View kann dazu eine beliebige Layoutkomponente zugeordnet werden.
TopPanel	*TopPanel* (ebenfalls eine Eweiterung von *javax.swing.JPanel)* ist die primäre Anzeigefläche für das Layout der Topologie.
TopLayout	Die Layoutkomponente enthält sämtliche Methoden, die nötig sind, um die Topologiedaten zu visualisieren. Sie ist quasi die Implementierung einer Interpretation über den Datenstrukturen des Topologiemodells. Durch einen Austausch dieser Komponente können unterschiedliche Darstellungsformen bewerkstelligt werden.
TopGraphObject	*TopGraphObject* ist lediglich ein Tag–Interface für die untenstehenden Graphikobjekte. Diese werden vom Layout verwendet, um die spezifische Darstellung der Topologie zu erzeugen.
TopBox	Diese Klasse wird vom Topology Viewer als Repräsentation von Kästen für das graphische Layout benutzt.
TopBus	Diese Klasse repräsentiert die graphische Darstellung einer Busstruktur. Sie führt die Busstruktur einfach auf einen durchgängigen Linienzug zurück, der alle Subsysteme verbindet und ist von der Klasse *java.awt.Polygon* abgeleitet.
TopCircle	Diese Klasse repräsentiert jegliche Art von Kreisscheiben. Farbige Kreisscheiben werden im Moment zur darstellung der Containment–Relation verwendet.
TopPolygon	*TopPolygon* ist (dem Namen entsprechend) für die Zeichnung eines beliebigen Polygonzuges zu gebrauchen. Auch diese Klasse ist von der Klasse *java.awt.Polygon* abgeleitet.
TopStar	*TopStar* ist das graphische Äquivalent zu einer Sternstruktur. Strahlen werden in gleichem Winkelabstand vom Zentralelement ausgehend gezeichnet.
GraphLink	*GraphLink* dient der Darstellung von globalen Verbindungen. Jedes Objekt dieser Klasse verweist auf das entsprechende Modellobjekt der Klasse *Link*.
WrongStructureException	Diese Ausnahme wird im Layout geworfen, wenn eine Struktur auftritt, die der Layouter nicht interpretieren und damit nicht darstellen kann.

5.2.2. Screenshot

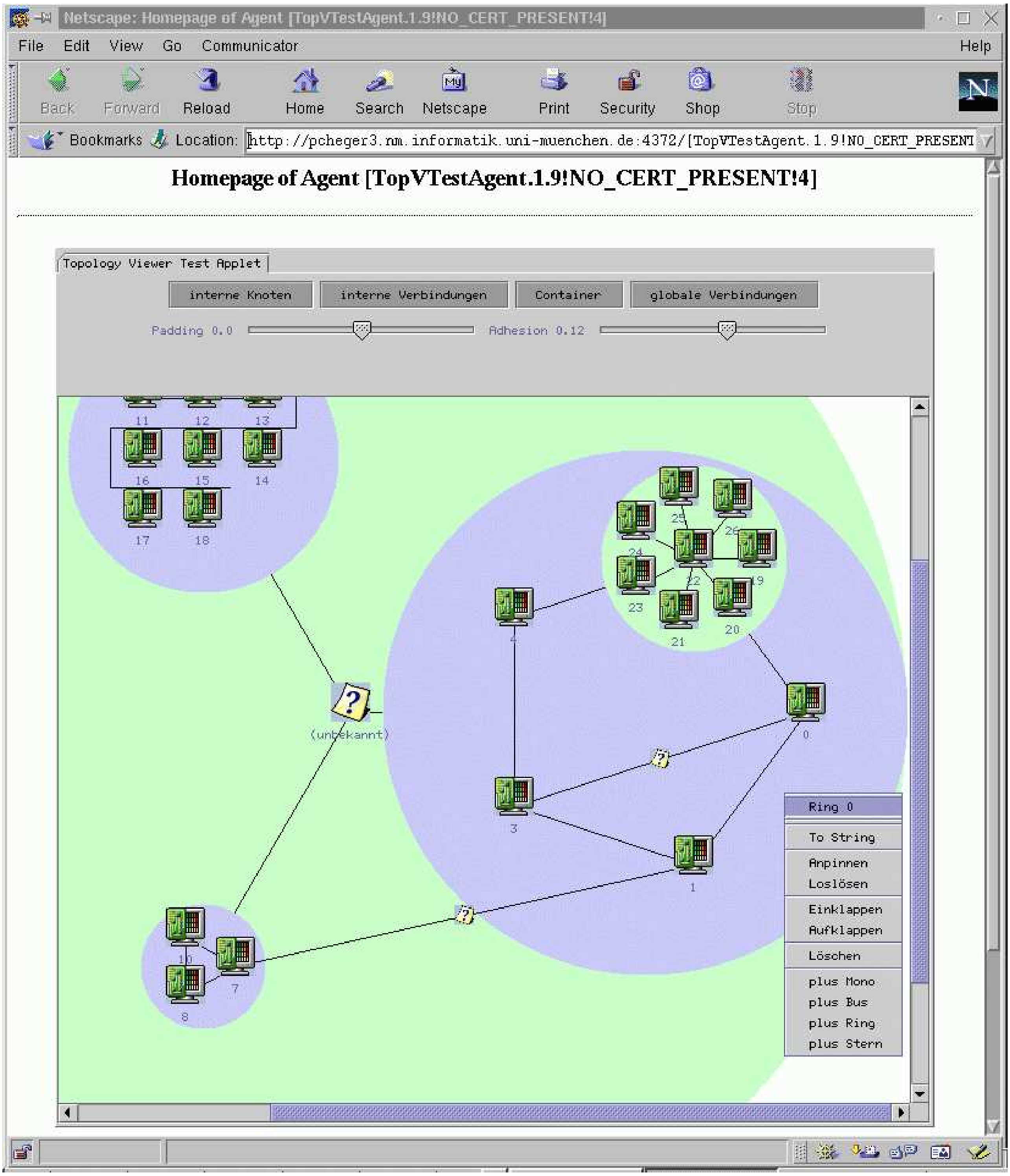


Abb. 5 : Screenshot mit allen integrierten Komponenten

5.3. Control

Die *Control* besteht aus den Klassen zur Bearbeitung und Weiterleitung von Ereignissen zwischen Anwender und MA.

Klasse	Aufgabe
TopControl	*TopControl* ist der zentrale Dreh– und Angelpunkt der Eventverarbeitung. Es nimmt jegliche benutzererzeugten Ereignisse (i.e. Mausklicks) entgegen, ermittelt aus der graphischen Quelle das entsprechende Objekt des Datenmodells und übergibt es gegebenenfalls einem Aktionsobjekt in Form eines neu generierten Events. Damit stellt *TopControl* die Transparenz gegenüber der Managementanwendung her.
TopAction	Diese Ableitung von *javax.swing.AbstractAction* ist die Mutterklasse für alle Aktionen, die die Managementanwendung dem Benutzer zur Verfügung stellen möchte.
TopEvent	Über diese Eventklasse benachrichtigt *TopControl* die Managementanwendung von außerordentlichen Ereignissen. Da mangels einer ernsthaften MA jedoch noch keine solchen Ereignisse auftreten können, ist dieser Mechanismus derzeit noch ohne Funktion.
TopPopup	Diese Ableitung von *javax.swing.PopupMenu* kann zu einem gegebenen graph. Anzeigeobjekt ein Popupmenü erstellen, indem es das repräsentierte Objekt des Datenmodells ausfindig macht und dessen zugeordnete Aktionen dem Benutzer zur Auswahl stellt.

5.4. Schnittstellen zur Management–Anwendung

5.4.1. Klassen

Klasse	Aufgabe
TopInterface	*TopInterface* zentralisiert für die Managementanwendung den Zugriff auf die Visualisierungs–komponente und stellt ihr einfache Möglichkeiten zur Verfügung, z.B. einen Anzeigerefresh selbst auszulösen.
TopListener	Dieses Tag–Interface muß die Managementanwendung implementieren, um von der Visualisierungskomponente durch *TopEvents* (siehe 5.3.3.) benachrichtigt zu werden.

5.4.2. Exkurs: Verwendung der VK durch eine Managementanwendung

Für die Verwendung der VK sind im Wesentlichen nur folgende Schritte notwendig:

1. **Konstruktion des Modells**
   Aus den Datenmodellklassen (s. 5.1.) wird die darzustellende Struktur aufgebaut. Da die Polystruktur–Klassen als Container angelegt sind (nicht im Sinne des AWT !) können mittels der Methoden *addSubsystem()* und *removeSubsystem()* einfach Elemente in die Containment–Relation integriert, bzw. wieder entfernt werden. Inwiefern den Komponenten bereits weitere Informationen zugeordnet werden (Name, Kommentar, Icon, etc.) bleibt der MA überlassen. Eine Darstellung von Komponenten, denen werden ein Icon, noch ein Name zugeordnet wurde, ist jedoch nicht sinnvoll.

Sämtliche Objektvariablen können mit den entsprechenden set– und get–Methoden angesprochen werden.
Bsp. : ein Netz besteht aus einem Switch, der acht Hubs vebindet, an welchen wiederum je sechs Unix–Rechner mit EtherNet–Interface hängen. Der Switch wird als *TopStar* modelliert, jeder einzelne Hub als Bus, welcher sechs Monostrukturen enthält: die Rechner.

2. **Zuordnung der Aktionen**
Bei der Generierung des Modells oder zu einem beliebigen späteren Zeitpunkt können jedem Objekt des Datenmodells Aktionen zugeordnet werden. Aktionsobjekte müssen das Interface *TopAction* implentieren, ihre Kompetenzen werden durch die MA kontrolliert. Weiter kann jedem Objekt eine Standardaktion zugeordnet werden (*setStandardAction(),* siehe auch **6. Benutzung**)
Bsp. : eine Aktion S*tartTelnet* implentiert *TopAction,* und dient der Öffnung einer Telnet–Verbindung zu einem Rechner. Sie wird jeder Monostruktur des Modells zugeordnet. Die Aktion übergibt das Objekt des Datenmodells, auf dem sie angewendet wurde, der MA, welche dann alles Nötige zur Öffnung der Telnet–Verbindung in die Wege leitet.
3. **Generierung des Interface**
Mit dem Datenmodell aus 1. als Parameter wird nun das *TopInterface* erzeugt, welches sich selbständig um die Generierung von View und Control kümmert.

4. **Einbindung des *TopView***
Das *JPanel*, welches *TopInterface.getView()* liefert, kann nun einfach in die Oberfläche der MA eingebunden werden.

5. **Setzen des ControlListeners**
Wenn gewünscht, kann die MA sich selbst oder ein anderes Objekt als Listener bei der Control–Instanz anmelden. Ein solcher Listener muß jedoch das Interface *TopListener* implementieren. Er erhält von der Control–Instanz Events der Klasse *TopEvent*, wenn sie Meldungen hat, die sie nicht selbst verarbeiten kann.
Bsp. : da die Demo–Inplementation des Topology Viewer noch keine derartigen Ereignisse wirft oder unterstützt verzichten wird darauf, einen Listener einzutragen.

6. **Darstellung und Benutzung**
Sobald die VK auf dem Bildschirm dargestellt ist (*setVisible()),* überwacht sie alle Operationen des Benutzers auf der Darstellung. Ein Linksklick auf ein Objekt führt die Standardaktion aus (siehe auch **2. Zuordnung der Aktionen**), ein Rechtsklick erzeugt ein Popup–Menü mit den derzeitig verfügbaren Aktionen, welche auf dem Objekt ausgeführt werden können.
Bsp. : ein Linksklick auf ein Endsystem bleibt wirkungslos, da diesem keine Standard–aktion zugeordnet wurde. Ein Rechtsklick öffnet jedoch ein Popup–Menü, das die gesetzte *StartTelnet*–Aktion enthält.

5.5. Demo−Anwendung

Um die Funktionalität des Topologyviewer überprüfen und demonstrieren zu können, wurde eine Demo−Anwendung in Form eines MASA−Agenten implementiert. Der Agent bleibt dabei leer, nur sein Applet stellt die Managementanwendung dar, die sich des Topologyviewers bedient. Mit dieser Demo−Anwendung ist es möglich einfache Strukturen nach Wunsch selbst zu konstruieren. Negativ fallen dabei oft Zeichen− und Refresh−Fehler auf, die auf die Verwendung von Swing zurückzuführen sind. Ein Screenshot der Demoanwendung ist weiter oben als Abb. 5 zu sehen.

5.5.1. Basisklassen

Da die Demo−Anwendung nur eine leere MA repräsentiert, umfaßt sie auch nur die absolut notwendigen Klassen.

Klasse	Aufgabe
TopVTestAgentApplet	Das Agentenapplet bleibt nahezu leer, es fügt lediglich ein *TopVTestPanel* in seine Anzeige ein.
TopVTestAgentAppletConstants	Hier werden die Ausmaße des Applets verzeichnet.
TopVTestPanel	Dies ist die eigentliche Managementanwendung. Sie konstruiert ein einfaches Modell, versieht es mit dern untenstehenden Aktionen und überläßt es der Visualisierungskomponente. Aktionen

5.5.2. Aktionsklassen

Sämtliche Aktionen sind Teil der Managementanwendung. Dementsprechend fällt es auch ihr zu, gegebenenfalls Authorisierung und Legitimität zu überprüfen. Dies ist insbesondere von großer Wichtigkeit, wenn die Visualisierungskomponente nicht nur zur Anzeige, sondern als Schnittstelle zu einem Verwalter dient, dessen Aktionen Auswirkungen auf reale Systeme haben können.

Klasse	Aufgabe
AddBusAction	Fügt dem Objekt einen aus drei Monostrukturen bestehenden Bus hinzu.
AddRingAction	Fügt dem Objekt einen aus drei Monostrukturen bestehenden Ring hinzu.
AddStarAction	Fügt dem Objekt einen aus drei Monostrukturen bestehenden Stern hinzu.
AddMonoAction	Fügt dem Objekt eine Monostruktur hinzu.
FixAction	Nimmt das Objekt aus dem automatischen Layout heraus. Seine Position wird nicht mehr verändert.
FreeAction	Führt das Objekt wieder dem automatischen Layout zu.
FoldUpAction	Sorgt dafür, daß eine Polystruktur „eingeklappt“ wird, d.h. sie wird durch ein einzelnes Symbol repräsentiert, nicht mehr durch die Darstellung ihrer Subsysteme.
UnfoldAction	Sorgt dafür, daß eine Polystruktur „aufgeklappt“ wird, d.h. alle Subsysteme werden gelayoutet und dargestellt.

Klasse	Aufgabe
RemoveAction	Entfernt das Objekt aus dem Modell.
ToStringAction	Gibt eine Stringrepräsentation des Objekts auf der Standardausgabe aus.

6. Literatur

[HP97] Hewlett–Packard Company
„Response to Topology Service RFP / Revised Summission“
OMG TC Document Telecom/97–10–02
(ohne Ort), 1997

(Anm. des Autors: Der Status dieser Veröffentlichung ist unbekannt, da sie von Hewlett–Packard kurz nach Erscheinen einen vorläufigen Version wieder zurückgezogen wurde {s. Kap.3.2}, die veröffentlichung mithin also nie stattgefunden hat. Sie wurde hier jedoch erwähnt, da Abb. 5 aus dieser Arbeitsversion entnommen ist.)

[Li99] Lipinski, Klaus
„Lexikon der Datenkommunikation.“
6. Auflage
MITP Bonn, 1999

[MN97] (unbekannt)
„Meeting Notes of the Telecommunications DTF 1–5 December 1997“
Bericht zum Treffen der Telecom DTF
East Brunswick, 1997

[Ri98] Richardson, Tony
„Review of Current Architecure Integration Status and Trends“
in „Difference 7a“
(ohne Ort), 1998

[Rö99] Rölle, Harald
„Implementierung eines CORBA TopologyService in Java“
Fortgeschrittenen–Praktikum, TU München
München, 1999

[Su97] Sun Microsystems
„JavaTM Platform 1.1.4 Core API“
Sofware Dokumentation
Palo Alto, 1997

[Su97] Sun Microsystems
„JavaTM Platform 1.2 API Specification“
Sofware Dokumentation
Palo Alto, 1998

[To96] Tollis, Yoannis G.
„Graph Drawing and Information Visualization“
Working Group on Computational Geometry
ACM Computing Surveys, 1996