INSTITUT FÜR INFORMATIK
DER LUDWIG - MAXIMILIANS - UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Integration von Protokoll- und Interaktionsmechanismen in das Benutzerinterface einer 3D-Netzmanagementplattform

Bearbeiter: David Le
Tobias Jungclaus

Aufgabensteller: Prof. Dr. Claudia Linnhoff-Popien
Prof. Dr. H.-G. Hegering

Betreuer: Dipl. Inf. Timo Baur
Prof. Dr. Gabi Dreo-Rodosek

Abgabetermin: 27.05.05

Hiermit versichern wir, dass wir die vorliegende Ausarbeitung selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

München, den 27.05.05

. . . .

(Unterschriften der Kandidaten)

Zusammenfassung

Das erste in dieser Ausarbeitung behandelte Thema ist die Umstrukturierung einer statischen Benutzeroberfläche in eine dynamische Variante, die über ein XML-Interface generiert wird, und zudem eine vom Benutzer frei konfigurierbare Seitenleiste enthält. Die Herausforderung bei einer dynamischen Benutzeroberfläche ist hierbei, ein ansprechendes Layout zu generieren, das dem Eindruck eines statischen Designs entspricht. Es folgt eine Beschreibung, wie in Java3D Interaktionsmöglichkeiten in Form von MouseEvents auf 3D-Objekten implementiert werden.
Als drittes wird die Implementierung eines XML Interfaces für eine JMX-Anwendung behandelt, und betrachtet, wie diese in ein vorhandenes Projekt eingebettet werden kann. Zuletzt folgt die Verbesserung des Netzwerkprotokolls durch Erhöhung der Übertragungsgranulatät und Einführung neuer Datenstrukturen. Insbesondere werden Lösungen für asynchrone Nachrichtenbearbeitung und Indizierung von Objekten (in linearer Zeit) in sich ständig verändernden Graphen vorgestellt.

Inhaltsverzeichnis

1 Einführung

1.1 NEVE kurz vorgestellt

NEVE (Network Visualization Environment) entstand im Jahre 2002 als Tragfähigkeitsnachweis der Diplomarbeit von Timo Baur [Baur 02] und dient zur Darstellung und Verwaltung von Netzwerken. NEVE wurde durch eine Client-Server Anwendung realisiert und basiert auf einer Komponentenarchitektur, wobei die wichtigsten verwendeten Technologien JAVA 3D sowie die Java Management Extensions JMX [JMX] (in der Implementierung mx4j [mx4j]) sind. mx4j stellt einen "Application Server" bereit, der die funktionalen Einheiten des Programms sowie das Netz beschreibende Managementobjekte hält. Der Server sammelt Netzdaten, bereitet diese auf und legt ihre Darstellung fest. Die Darstellung aller gespeicherten Managementobjekte erfolgt durch eine Visualisierungspipeline, deren Ausgabe Java3D-Objekte definiert, die entsprechend der Relationen zwischen den Managementobjekten gelayoutet werden und so im Client später zu einer Java3d-Szene zusammengefügt werden können. Aufgabe des Clients ist die dreidimensionale Repräsentation der vom Server gelieferten Daten (Abb. 1). Die Interaktion mit NEVE (also unter Anderem die Netzwerkverwaltung) wurde durch ein formularbasiertes Browserinterface realisiert, welches von mx4j [mx4j] gestellt wird (Abb. 2). Die Interaktion am Client beschränkt sich auf die Navigation durch die 3D-Szene, dem Speichern und Laden von Szenen, sowie Einstellungen der Darstellungsart (Beleuchtung, Flächen/Drahtgitter etc.).

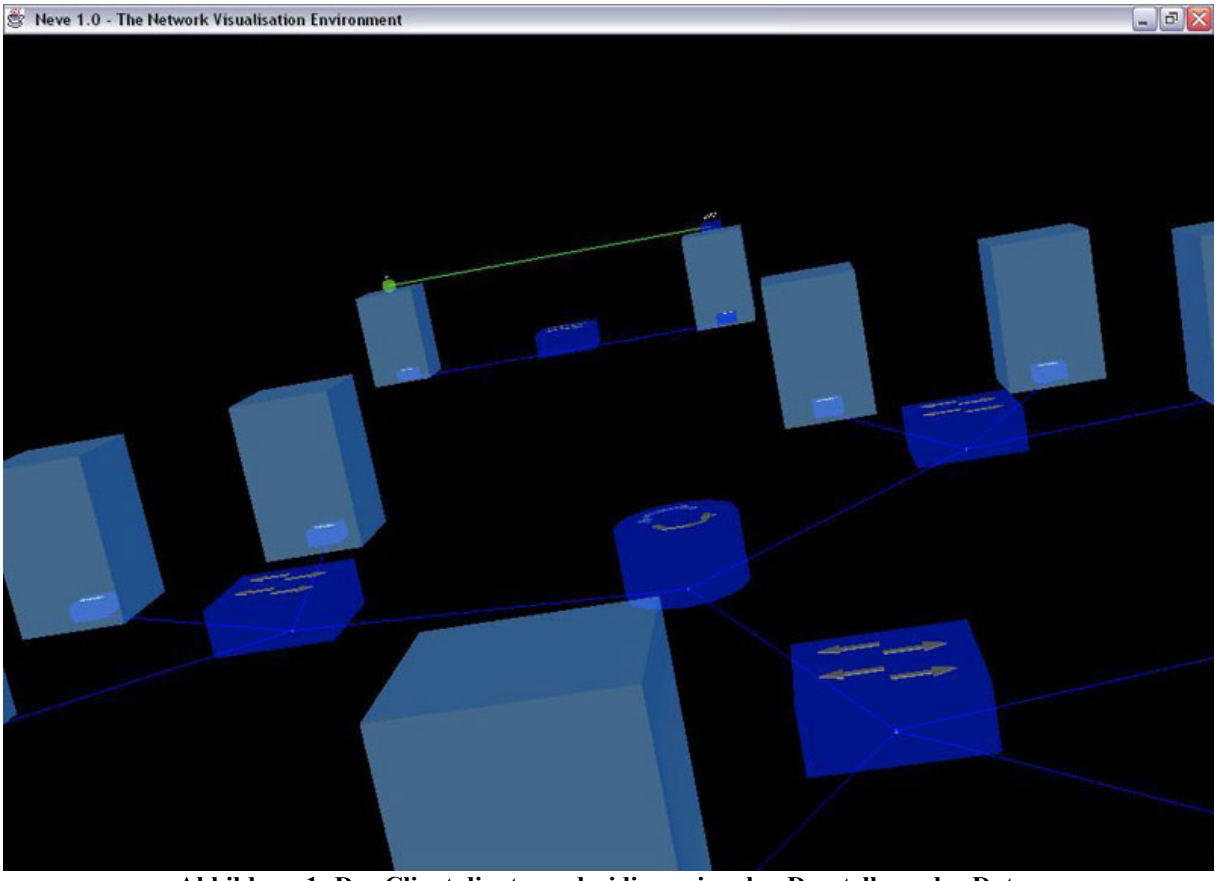

Abbildung 1: Der Client dient zur dreidimensionalen Darstellung der Daten

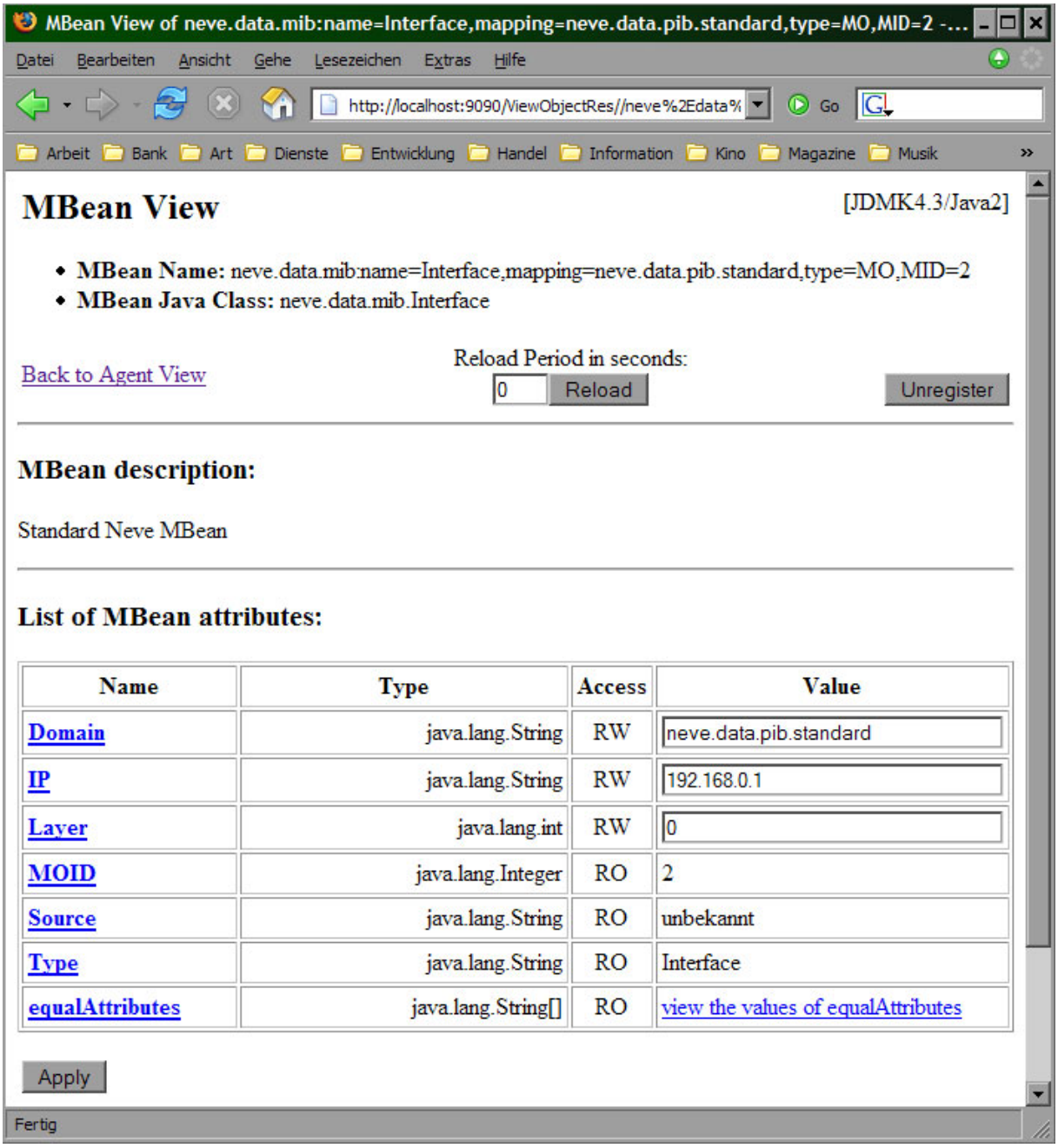

Abbildung 2: Formularbasiertes Browserinterface zur Verwaltung der Daten

1.2 Motivation

Wie in 1.1 geschildert, zwang die vorhandene Implementierung den Benutzer noch zur Interaktion mit zwei Programmen, nämlich dem NEVE Client sowie einem Webbrowser. Deutlich intuitiver wäre es, direkt im Client auch Einstellungen an den dargestellten Netzkomponenten vornehmen zu können. Vielleicht sogar ganz auf das Browserinterface zu verzichten, und das gesamte Management mit Hilfe des Clients zu erledigen. Folglich war eine Integration der Webinterface-Funktionalität in den Client notwendig.

Weitere Probleme der bisherigen Implementierung waren die verbesserungsfähige Performance, sowie die Starrheit des Netzwerkcodes. Beide genügen einem Prototypen völlig, jedoch bekam man recht schnell Probleme, wenn man größere Änderungen am Programm vornehmen wollte. (Und das hatten wir schließlich vor.)

2 Aufgabenstellung

Überarbeitung des Benutzerinterfaces:

Am Benutzerinterface sollten ergonomische und funktionale Änderungen vorgenommen werden, das heißt, der Client sollte bedienbarer sowie mächtiger werden. Wie in der Motivation bereits vorgeschlagen, sollten die Funktionen des Webinterfaces möglichst vollständig in den Client übernommen werden. Da diese jedoch dynamisch von mx4j [mx4j] generiert werden, musste auch die Clientoberfläche dynamisch aufgebaut werden.

Clientseitige Ansteuerung des XML Interfaces:

Neben der HTML Schnittstelle (=Browserinterface) bietet JMX [JMX] auch eine XML Schnittstelle, welche im Grunde das Gleiche leistet, jedoch maschinenlesbarer ist. Der überarbeitete Client sollte diese XML Schnittstelle ansteuern.

Überarbeitung des Netzprotokolls:

Es stellte sich recht bald heraus, dass viele Aspekte der Aufgabenstellung mit dem implementierten Netzprotokoll nicht zu realisieren waren (alles dazu in 3.4). Somit mussten grundlegende Änderungen am Übertragungsprotokoll vorgenommen werden, die uns erlaubten, kleinere Einheiten und detailiertere Informationen zu übertragen, als dies bislang der Fall war.

3 Umsetzung

3.1 Ergonomische Anpassungen des Benutzerinterfaces

Zum Vergleich wird hier nochmal das alte Benutzerinterface vorgestellt. Wie in der Abbildung 3 zu sehen ist, fehlen jegliche Funktionsleisten. Nur auf der linken Seite ist das Controlpanel zu erkennen, welche eine unsortierte Vielzahl von Funktionen bereithält.
In dem neuen überarbeiteten Benutzerinterface (siehe Abbildung 4) finden sich nun die für Windowsfenster typischen Funktionsleisten (Menubar, Toolbar, Statusbar, Controlbar) wieder. Die Menubar enthält die Dropdown-Menüs „Datei“, „Viewpoints“, „Services“, „Management“, „Extras“ und „?“(Hilfe). Da das alte Controlpanel zu unübersichtlich war, wurden die Elemente umstrukturiert, verschoben und durch andere Elemente ersetzt. Die Anmeldeprozedur im alten Interface erfolgte durch das Anklicken der Checkbox „Connection Enabled“. Sie wurde durch ein Connection-Fenster abgelöst und die Checkbox durch einen Button ersetzt. Mit Einführung der Statusbar hat der Benutzer immer den Verbindungsstatus im Auge und kann mit einem Klick das Connection-Fenster aufrufen.
Die Funktionen „Load“, „Save“, „Clear“ und „Exit“ wurden übersetzt und ins DropDown-Menü „Datei“ verschoben. Die Auswahlboxen der „Rendering Attributes“ wurden durch bebilderte Buttons in der Toolbar ersetzt. Ob das „Lighting“ aktiviert ist oder nicht, erkennt man nun an der Glühbirne in der Toolbar. Desweiteren wurden noch die häufig benutzten Funktionen wie „Show All“ und „View Behaviour“ in die Toolbar übernommen. Die Funktion „Warp to“ findet man in dem DropDown-Menü „Viewpoints“ wieder. Die „Szenenstatistik“ befindet sich nun im Menü „Extras“. Der neue „Refresh“-Button in der Toolbar ist lediglich ein Workaround, um das Interface bei einem der sporadisch auftretenden „Grafik-Hänger“ zu reseten.
Im DropDown-Menü „?“ stehen weitere Informationen zum Neve-Projekt wie Links zur Diplomarbeit, Fopra-Ausarbeitung usw. zur Verfügung, welche im default Internet Browser angezeigt werden.
Außerdem wurde der Ladescreen durch ein moderneres Bild ersetzt, und der hinzugefügte Ladebalken informiert dem Benutzer zudem über die Ladezeit mit der zugehörigen Aktivität.

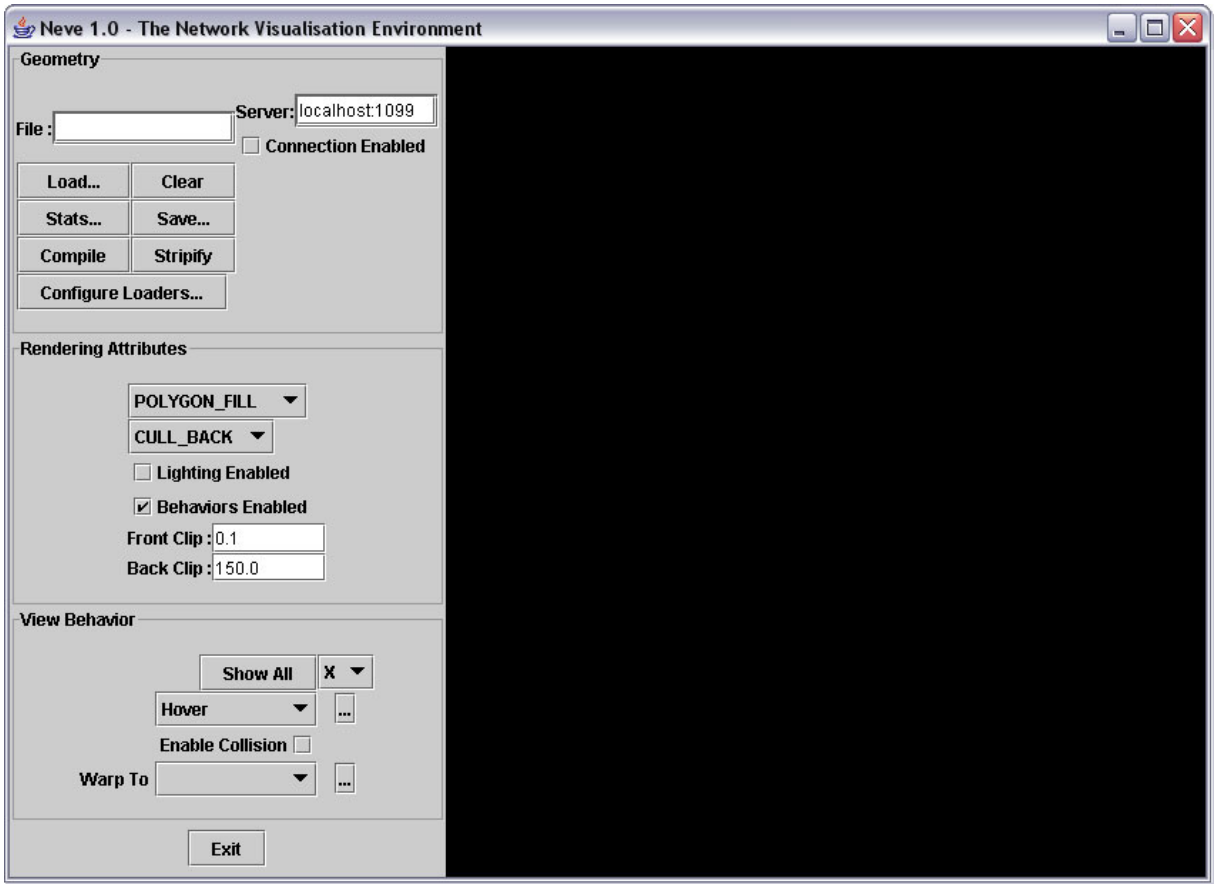

Abbildung 3: Altes Design

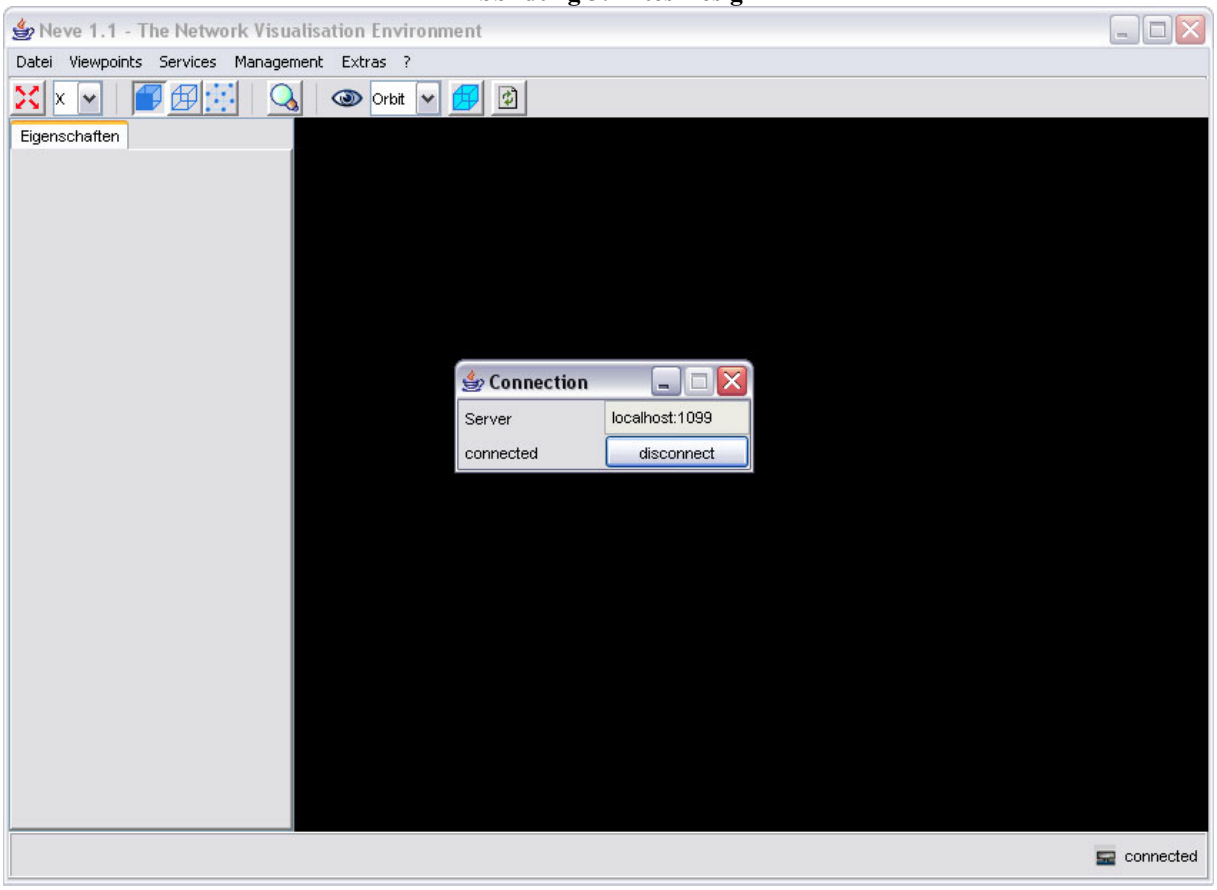

Abbildung 4: Design der neuen Version

3.2 Funktionale Anpassungen des Benutzerinterfaces

Beim Einloggen werden die Menüeinträge der DropDown-Menüs „services“ (Neve.Base) und „managment“ (Neve.Managementanwendung) dynamisch generiert. Dazu werden die Bezeichner aus den Komponentendomänen Neve.Base und Neve.Management über das XML-Interface geholt, daraus JMenuItems erzeugt und mit entsprechender Funktionalität versehen. Diese besteht aus einer wiederum dynamischen Erzeugung eines JFrames, wobei die Buttons und eventuellen Eingabefelder den MBean Operationen mit eventuellen Parametern entsprechen. Wird ein Button ausgelöst, erfolgt ein Invoke Kommando über das XML-Interface.
Ein solches Kommando hat zum Beispiel die Form:

```
/InvokeAction//Neve.Managementanwendung%3Aname%3DExampleHTTPCreator/
action=createExample?action=createExample
```

Das alles läuft für den Benutzer natürlich völlig transparent ab.
Im Menü „Extras“ gibt es die neuen Funktionalitäten zur Anpassung der Seitenleiste. Diese wären „Seitenleiste anzeigen“ (blendet Seitenleiste ein oder aus) und „Seitenleiste konfigurieren“. Letztere öffnet eine Baumdarstellung, deren Knoten MBeans und deren Blätter MBean Operationen sind. Jedes Blatt ist mit einer JCheckbox versehen, wobei eine Aktivierung die Darstellung der MBean Operation in der Seitenleiste bewirkt.

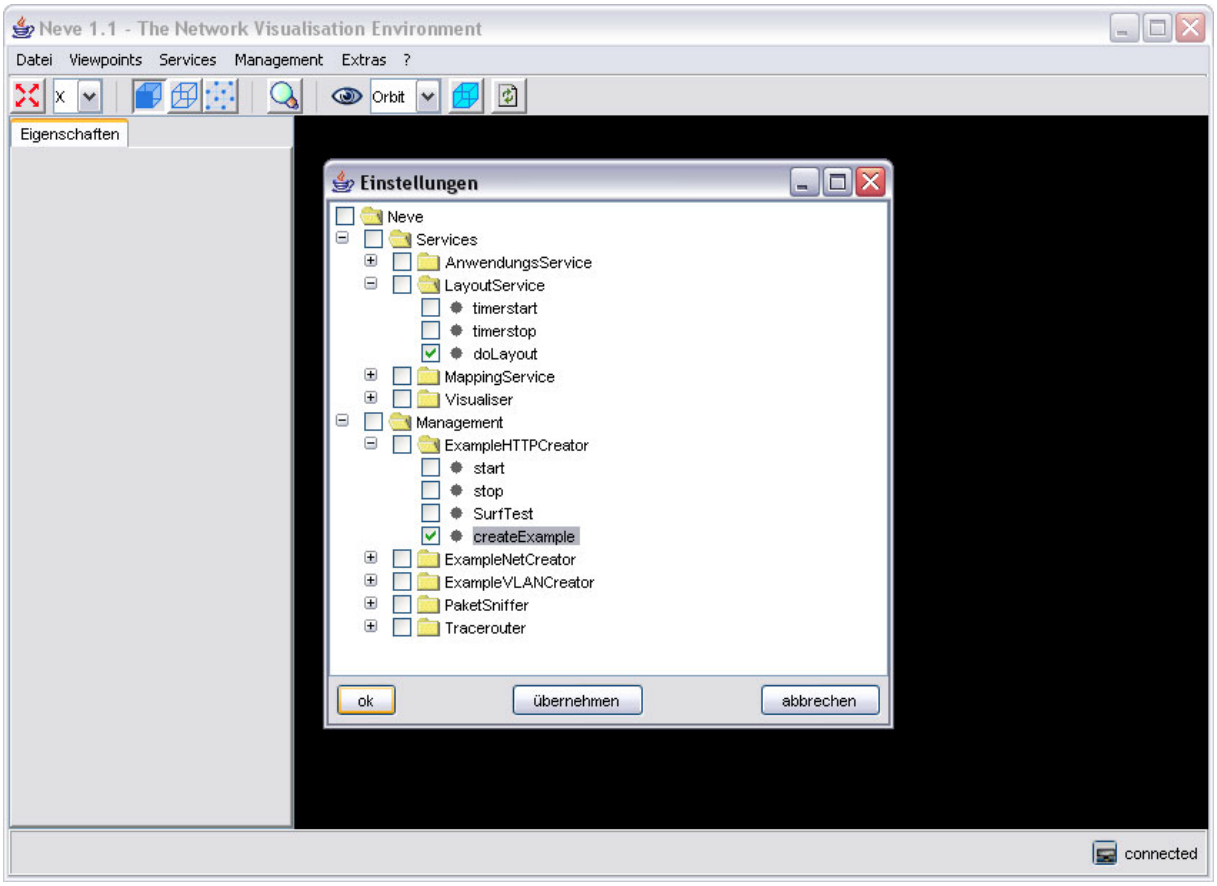

Abbildung 5: Konfiguration der Seitenleiste

In der Abbildung 5 erkennt man, dass die MBean Operationen „doLayout“ und „createExample“ ausgewählt worden sind. Sobald man die Auswahl bestätigt hat, wird die Baumdarstellung geschlossen und die Seitenleiste aktualisiert, wobei MBean Operationen nach ihren MBeans gruppiert sind, wie auf Abbildung 6 zu erkennen.

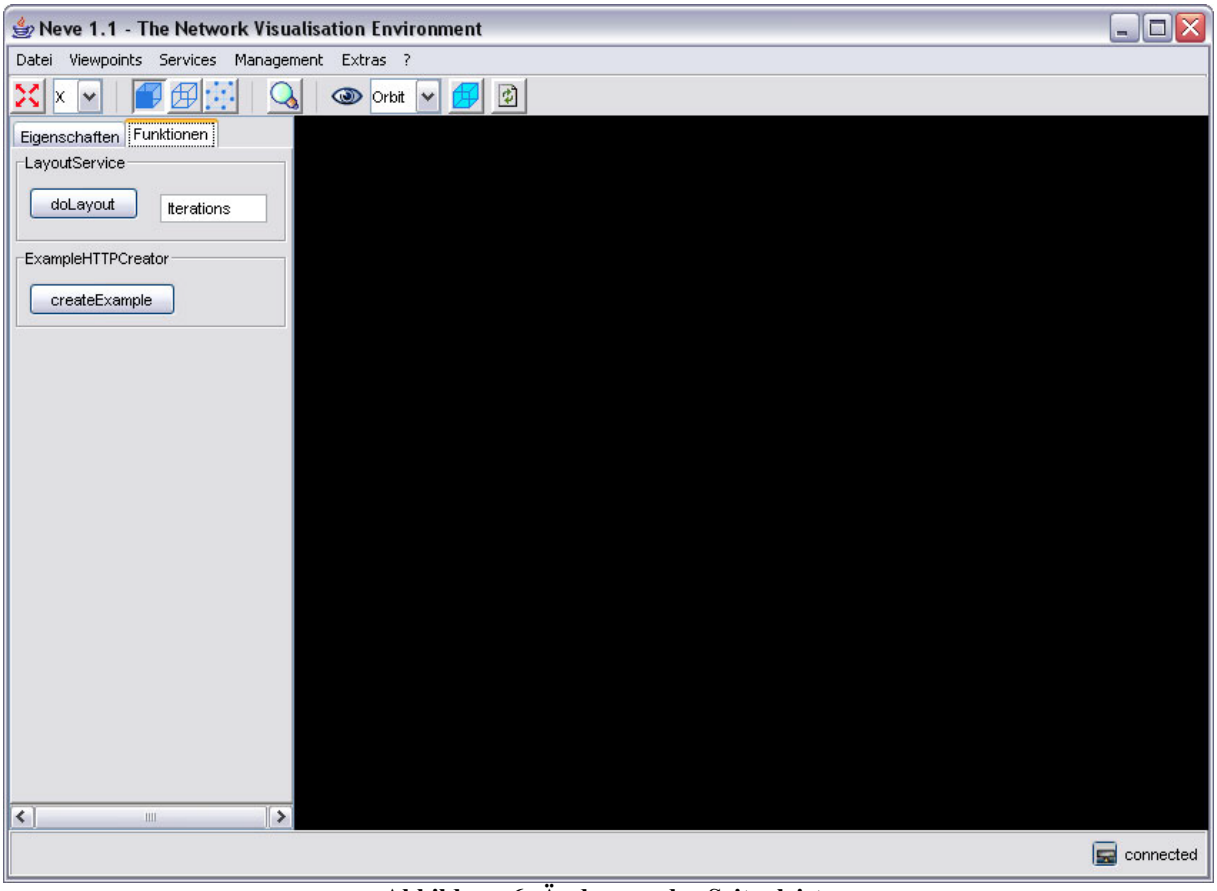

Abbildung 6: Änderung der Seitenleiste

Die vom Client angezeigte, zuvor nicht interaktive 3D-Welt wurde durch die Einführung von Clickevents erweitert. Es werden Linksklicks und Rechtsklicks unterschieden, sowie Klicks auf Objekte und Klicks ins Leere. Der Linksklick selektiert ein Objekt und zeigt vom Benutzer gewählte Eigenschaftsinformationen in der Statusleiste an. Der Rechtsklick auf ein Objekt erzeugt die Darstellung der MBean Attribute als Eigenschaften des Objekts in der Seitenleiste (ausgewählte Eigenschaften werden beim Linksklick in der Statusleiste angezeigt).
Jedem 3D-Objekt wurde nun ein IntersectInfoBehavior gegeben, welches die Anklickbarkeit ermöglicht. Die Klasse IntersectInfoBehavior verarbeitet die Klicks der Benutzer und ermittelt den Pfad des angeklickten Objekts im Szenegraphen (Details zur Pfaddarstellung folgen im Kapitel 3.4). Anhand des Pfades wird das eigentliche Objekt, insbesondere die MIOD, ermittelt. Mit der MIOD können die Eigenschaften des Objekts über das XML-Interface abgerufen werden.

3.3 Ansteuerung des XML Interfaces

Zunächst musste der Aufbau des von Neve gelieferten XMLs analysiert werden. Es ist defaultmäßig an Port 9091 zu finden und in folgender Weise aufgebaut.
Jeder Service (z.B. Layoutservice zum Anordnen der Objekte im Raum), jede Applikation (z.B. Examplecreator), jedes Objekt (z.B. Host XY) und überhaupt alles wird durch ein MBean-Tag repräsentiert. Der Aufruf `server:9091/` liefert uns alle vorhandenen MBeans. Wollen wir den Host XY näher betrachten, geschieht das durch einen Aufruf der Form `server:9091/mbean?objectname=HostXY.` Das zurückgelieferte XML beinhaltet nun nur noch das angefragte MBean, sowie dessen Untertags, in diesem Fall Attribute wie MacAdresse und Typ. Untertags können auch Operationen sein, was direkt zum nächsten Punkt führt. Um einen Service zu starten (o-

der vielmehr eine Operation eines Services), genügt ein Aufruf der Form `server:9091/invoke?objectname=layout&operation=doLayout.` Das Ergebnis-XML teilt uns Erfolg oder Fehler mit, sowie eventuelle Rückgabewerte der Operation.
Nun, da wir die XML Schnittstelle kannten, benötigten wir einen geeigneten XML Parser. Eine kurze Google-Recherche führte uns direkt zu SAXON [sax], einem Opensource XML-Parser, der recht weit verbreitet und dazu frei verfügbar ist. SAXON unterstützt mehrere Parsingmethoden; wir entschieden uns für XPath.
Um nun zum Beispiel den Wert eines bestimmten Attributes auszulesen, holt man sich zunächst den XML-Stream wie zuvor beschrieben. Diesen übergibt man dem SAXON XPathEvaluator und führt dann den XPath-Ausdruck `string(/MBean/Attribute[@name=$attrib]/@value)` aus.
Die gesamte geschilderte Funktionalität ist in der Klasse XML-Interface zusammengefasst, die Methoden für alle Anwendungsfälle enthält, auf die wir während der Entwicklung gestoßen sind (z.B. Setzen und Lesen von Attributen, Auslesen aller Attribute, Ausführen von Operationen mit und ohne Parametern usw.).

3.4 Verbesserung des Netzprotokolls

Zunächst sollten wir betrachten, wie NEVE 1.0 auf der Netzwerkebene funktionierte, und was daran problematisch für die Weiterentwicklung war.
Da alle Netzwerkkomponenten auf dem NEVE Server erstellt werden, und dort auch die 3D Darstellung generiert wird, genügt ein relativ „dummer“ Client. Der Server schickt typischerweise eine Nachricht an alle Clients „ADD 1“, woraufhin sich alle Clients, die die das Objekt mit der ID 1 noch nicht haben, selbiges vom Server (mittels RMI) holen. In NEVE 1.0 ist die Einheit, die dabei übertragen wird, eine sogenannte Komponente. Sie besteht beispielsweise aus einem Host inklusive Browser und Netzwerkkarte, die beide dem Host untergeordnet sind. Der Client 1.0 weiß jedoch nichts von Hosts, Netzwerkkarten usw. – für ihn gibt es lediglich ein 3D-Objekt, dass es darzustellen gilt, das genau einen Identifikator hat: „Komponente 1“. Da es jedoch für NEVE 1.1 vorgesehen war, dass alle sichtbaren Teile des Netzwerks in der 3D Darstellung einzeln anwählbar sein sollten, konnte das Prinzip der Gesamtkomponenten nicht bestehen bleiben.
Offensichtlich musste die Granularität der Übertragungseinheiten erhöht werden. „Komponente 1“ bestehend aus Host, Browser und Netzwerkkarte ist in NEVE 1.1 aufgeteilt in die drei Komponenten „Host 1“, „Browser 1,1“ und „Netzwerkkarte 1,2“ (vereinfacht), wodurch der Client zum einen weiß, welche Komponenten welchen untergeordnet sind, und zum anderen, welchen Typs diese Komponenten sind. Eine Servernachricht (ein sogenannter „Notify“) an die Clients, die das gleiche leistet, wie zuvor „ADD 1“ besteht nun also aus drei Notifies: „ADD 1“, „ADD 1,1“ und „ADD 1,2“. Um diese vielen einzelnen Notifies loszuschicken, gibt es serverseitig eine neue Methode namens „notifyAllSubPaths“, der man eine komplette Komponente übergibt, und die für die Komponente selbst, sowie für alle untergeordneten Komponenten Einzelnnachrichten an die Clients schickt.
Man könnte sich an dieser Stelle denken, dass so viele Nachrichten doch den Netzverkehr erhöhen, zumal ja auch die tatsächlichen Unterkomponenten nun einzeln übertragen werden und somit insgesamt sicherlich mehr Netzverkehr verursachen müssten. Das stimmt.
Allerdings trifft das nur auf den initialen Datenfluss zu. Sobald ein Client erstmal alle Daten hat, sind alle weiteren Übertragungen (im Wesentlichen Updates und Hinzukommen von Unterobjekten) deutlich sparsamer. Wenn nun auf einem der Hosts ein Webserver installiert wird, muss nicht mehr der gesamte Host inklusive aller Unter-

komponenten übertragen werden. Es reicht ein „ADD 1,3“ – der Client holt sich vom Server die Komponente 1,3 und bekommt den reinen Webserver geliefert. Meist ist bereits nach einer einzelnen solchen Übertragung der gesamte zuvor zusätzlich verursachte Datenfluss wieder „reingeholt“.
Diese Änderungen genügten bereits, um clientseitig Clickevents auf Unterkomponenten richtig auswerten zu können. Bei einem Klick wird über getParent und getChild Methoden der Pfad im Szenebaum ermittelt. Es wird eine Anfrage mit dem Pfad an den Server geschickt, der daraufhin den eindeutigen Identifier (die sogenannte MIOID) in Form eines TransferBeans liefert. Das TransferBean nimmt die MIOD und Typ eines MBeans auf und ist dabei deutlich schlanker als ein Objekt der Klasse core.object.mpo. Mit dieser ID kann der Client (über das neu implementierte XML-Interface) alle Informationen abrufen, sowie sämtliche Änderungen durch- und Operationen ausführen, die das Objekt zu bieten hat.
Doch es gab noch eine weitere Übertragungsredundanz, mit der wir uns beschäftigt haben. Sie tritt sehr häufig auf und sollte daher insbesondere vermieden werden: Immer wenn ein Objekt geändert wird, schickt der Server einen „UPD“ Notify, woraufhin sich alle Clients das entsprechende Objekt vom Server holen, und ihr eigenes überschreiben. So eine Änderung ist jedoch in 90% der Fälle lediglich eine Verschiebung innerhalb der 3D-Szene – das Objekt müsste also eigentlich gar nicht komplett übertragen werden; die Transformationsangaben würden völlig reichen. In Java3D werden diese in der sogenannten TransformGroup gespeichert, die dem 3D Objekt (der sogenannten BranchGroup) hierarchisch übergeordnet ist. Unser nächster Schritt war also eine weitere Aufsplittung. Der Server implementiert nun nicht mehr nur die Methode getKomponent, sondern auch getBranchGroup und getTransformGroup. Zu diesem Zweck wurde die neue Klasse TransformBranchGroup erstellt. Sie ist nötig, da in Java3D kaum Methoden auf TransformGroups alleine anwendbar sind. Die TransformBranchgroup ist eine Branchgroup, die eine TransformGroup enthält, und einfach alle Methoden (wie z.B. addChild) an diese „weiterleitet“. Der Client kann nun (bzw. könnte, wir haben diesen Aspekt aus zeitlichen Gründen leider nicht weiterverfolgen können) auch einzelne Transformationsangaben einer Objekten abrufen. Ausgelöst würde dies am besten durch einen neuen Notify-Typ „UPDT“ (Update Transformation).
Eine weitere Sache, in die wir viel Zeit und Mühe gesteckt haben, die aber letztendlich nicht mehr innerhalb des zeitlichen Rahmen realisiert werden konnte, ist die Umstellung des Netzprotokolls auf MIOIDs anstatt von Pfaden. Dieser Schritt wurde nötig, als wir versuchten, auf die neueste Version 3.0.1 von mx4j [mx4j] umzusteigen, die deutlich gebugfixt sowie performanter als die von uns verwendete Version 2.01 ist. Den größten Perfomanzschub erfährt sie durch die asynchrone Abarbeitung von Nachrichten. In der Praxis läuft es so ab, dass der Server immer noch die gleichen Notifies verschickt wie früher (auch in der gleichen Reihenfolge), jedoch wartet er nicht darauf, dass sich die Clients die Daten holen. Tatsächlich kommen dadurch äußerst unschöne Situationen zustande: Der Server schickt bspw. die Nachricht „ADD 3,1“ und arbeitet sofort weiter. Wenn die Clients schließlich soweit sind und sich vom Server das Objekt mit dem Pfad 3,1 holen wollen, hat der Server mittlerweile im Szenebaum soviel verändert und verschoben, dass an dem durch den Pfad 3,1 bezeichneten Knoten möglicherweise ein völlig anderes Objekt hängt (wenn überhaupt).
Sendet man innerhalb des Notifies jedoch statt dem etwas unsicheren Pfad eine eindeutige MIOID, kann diesbezüglich nichts mehr passieren: Auch wenn das Objekt mittlerweile woanders im Szenebaum hängt, hat es noch immer die gleiche ID. Zu Lösen bleibt nur noch das Problem der korrekten Adressierung. Wie finde ich im

Szenebaum das Objekt mit der MIOID xy, ohne ihn komplett durchsuchen zu müssen?
Es bietet sich eine Indizierungstabelle der Form „xy“--- „3,1“ an. Doch was passiert, wenn nun das Objekt mit dem Pfad „3“ verschoben wird und den neuen Pfad „2,5“ bekommt? Objekt xy müsste die Zuordnung „2,5,1“ erhalten, und gleiches gilt für alle Kinder von Objekt xy. Der Aufwand für Verschiebungen wäre exponentiell. Kurz gesagt, es musste eine verbesserte Darstellung gefunden werden. Möglicherweise erfüllt eine Darstellung der Form („xy“ --- „vaterID,1“) , („vaterID“ --- „wurzel,3“) den Zweck. Wird nun das Objekt 3 verschoben, muss lediglich der Eintrag „vaterID“ --- „wurzel,3“ geändert werden. „wurzel“ wird durch die ID des neuen Vaters ersetzt.
Mit wenigen zu implementierenden Methoden kann auf diese Weise eine Hashtabelle erstellt werden, die das Finden, Hinzufügen, Verschieben und Löschen von Objekten im Szenebaum in linearer Zeit ermöglicht.
Um weitere unnötige Übertragungen einzusparen, sollte der Client außerdem überflüssige Notifies aussortieren. Hat er in seinem Nachrichtenstack zweimal einen UPD des gleichen Objektes, sollte eine einzige Ausführung reichen.

4 Zusammenfassung und Ausblick

4.1 Was ist geschafft?

Es ist uns gelungen, die gestellten Aufgaben weitestgehend zu erfüllen. Die Interaktionsebene wurde vereinheitlicht, alle Einstellungen können nun über das Clientinterface vorgenommen werden. Die Clientoberfläche ist dynamisch, das heißt, auch neu hinzukommende Dienste des Servers werden in den entsprechenden Untermenüs des Clients angezeigt und können mittels der Interfaceanpassung in die Seitenleiste übernommen werden. Um die clientseitige Verarbeitung von Unterobjekten zu ermöglichen, haben wir die Übertragungsgranularität erhöht – mit dem nützlichen Nebeneffekt, dass weniger Overhead bei der Übertragung entsteht und dadurch die Netzlast im laufenden Betrieb gesenkt werden konnte.

4.2 Was bleibt zu tun?

Es gibt mittlerweile eine neuere mx4j Version (v3.01) [mx4j], als sie im NEVE Projekt verwendet wird (v2.01). Beim Versuch, diese zu übernehmen, stießen wir jedoch auf Probleme mit dem Netzwerkcode, die den Zeitrahmen des Fortgeschrittenenpraktikums endgültig gesprengt hätten. (Zur Erinnerung: eigentlich ging es nur um Verlagerung der Interaktionsebene, sowie Interfaceoptimierung.) Das neue mx4j [mx4j] arbeitet Nachrichten asynchron ab, was bedeutet, dass der Server nicht sicher sein kann, wie weit der Client bei der Abarbeitung ist. Dadurch können Situationen entstehen, die beim momentanen Übertragungsprotokoll zu Inkonsistenzen führen. Zu beheben wäre dies durch eine Umstellung der Objektadressierung von Pfaden auf Identifier. Die Zuordnung der Identifier zu der Position im Baum könnte mittels Hashtabellen geschehen, wie im Abschnitt 3.4 beschrieben.
Das Netzprotokoll sollte außerdem doch dahingehend verbessert werden, dass Updates von Objektpositionen (also ohne das Objekt selbst) möglich sind. Dadurch kann die Netzlast weiter verringert werden, wenn sich ein Objekt lediglich im Raum verschiebt, ohne dabei seine übrigen Eigenschaften zu verändern.
Die Clientoberflache ist noch immer verbesserungswürdig. Als klar wurde, dass umfangreiche zusätzliche Arbeiten am Netzprotokoll anstanden, legten wir unser Hauptaugenmerk auf die Funktionalität und weniger auf das Design. Auch sollte noch eine Speicherung von Einstellungen eingebaut werden, sodass sich NEVE merkt, welche Funktionen man stets links im Controlpanel sehen möchte. Es gibt auch nach wie vor einige „Hänger“ im 3D Fenster, die noch als NEVE 1.0 stammen und die wir nicht ausfindig machen konnten.

Literaturverzeichnis

[Baur 02] BAUR, TIMO: Entwurf einer Architektur zur Integration von Netzplanungs- und -managementwerkzeugen in eine VR-Umgebung, 2002, http://www.nm.informatik.uni-muenchen.de/php-bin/pub/show_pub.php?key=baur02

[Java] SUN, Java API, http://java.sun.com/j2se/1.3/docs/api/

[J3D] SUN, Java3D API, http://java.sun.com/products/java-media/3D/

[Sax] Saxon Documentation, http://saxon.sourceforge.net/

[JMX] JMX Documentation, http://java.sun.com/products/JavaManagement/reference/docs/

[mx4j] http://mx4j.sourceforge.net/