Technische Universität München
Institut für Informatik

Diplomarbeit

Entwicklung und prototypische Implementierung eines Web-Frontends zur Visualisierung von Managementinformation

Roland Spatzenegger

Technische Universität München
Institut für Informatik

Diplomarbeit

Entwicklung und prototypische Implementierung eines Web-Frontends zur Visualisierung von Managementinformation

Bearbeiter:	Roland Spatzenegger
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Bernhard Maucher (Siemens), Norbert Wienold
Abgabedatum:	15. August 1997

Ehrenwörtliche Erklärung

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. August 1997

....
(Unterschrift des Kandidaten)

Abstract

Bisher benötigt der Administrator leistungsstarke Rechner um Managementplatformen zu benutzen. Mit dem Aufkommen des World Web Webs und Java entstand bei vielen der Wunsch Management-Funktionalitäten in einem Webbrowser verfügbar zu machen. In dieser Arbeit soll anhand einer gegebenen Managementplatform (INMS von Siemens) untersucht werden, wie sich Management per Web realisieren läßt. Es wird dabei eine Java Applikation entwickelt die bestimmte Funktionen (Visualisierung von Topologie-Informationen) bietet. Eine prototypische Implementierung soll zeigen, daß das entwickelte Client/Servermodell realisierbar ist.

Inhaltsverzeichnis

1. Motivation

1.1. Szenario

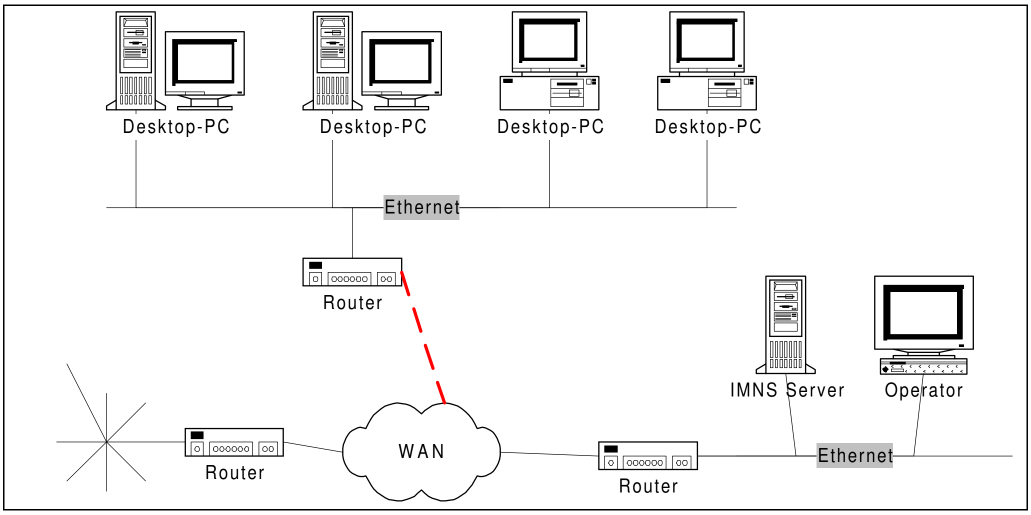


Abbildung 1 Szenario

Ein Netzoperator ist oft mit dem Problem konfrontiert, daß im Netz eine oder mehrere Komponenten ausfallen. Um das Problem zu beheben kann der Operator nun auf die einzelnen Komponenten remote zugreifen (z.B. über telnet, snmp) oder falls das nicht geht sich direkt vor Ort begeben. Vor Ort möchte er natürlich Zugriff auf seine Managementplattform um weitere Informationen über die einzelnen Komponenten und deren topologischen Anordnung erfahren.

Vor Ort fehlt nun meist der Zugriff auf die eingesetzte Managementplattform. Dies ist zum Beispiel der Fall, wenn die Verbindung zu dem Netz mit dem Server der Managementdatenbank nicht erreichbar ist, oder wenn auf den lokal vorhandenen Plattformen die Applikation nicht läuft. Die Gründe sind in einer heterogenen Rechnerumgebung zu suchen. Viele Managementplattformen basieren auf Serverbetriebssysteme wie Unix oder Windows NT. Arbeitsplatzrechner sind meist den Anforderungen an Speicher und Rechenleistung der Applikationen nicht gewachsen. Bei fehlender Netzverbindung kann die Verbindung über Modems kurzzeitig hergestellt werden, diese Verbindung hat aber nur sehr geringe Bandbreite, was eine Reduktion der Managementinformationen auf das Wesentliche voraussetzt.

Ein weiteres Szenario wäre die reine Statusüberwachung der Komponenten. Ein Management-applikation stellt aber viel mehr Funktionalität zur Verfügung, die dabei meist nicht gebraucht oder gar nicht erwünscht ist. Mit einer eingeschränkten Version könnte man den Benutzer des Netzwerks die Möglichkeit bieten selbst einfache Fehler zu erkennen und zu beheben (z.B. Papierstau beheben).

1.2. Mögliche Lösungsansätze

Grundsätzlich haben alle Lösungsansätze gemeinsam, daß sie auch über Verbindungen mit geringer Datenrate lauffähig sein sollen. Damit scheiden alle Ansätze aus, die versuchen nur die graphische Oberfläche von Server zu Client umzulenken, wie es zum Beispiel unter X/Window möglich ist. Die Erfahrung zeigt, daß ein sinnvolles Arbeiten mit X/Window über eine ISDN 64Kbit Leitung nicht möglich ist.

Der erste und wohl auch traditionelle Ansatz ist der Einsatz eines leistungsfähigen Laptops auf dem die Managementplattform läuft. Damit umgeht man das Problem mit den lokalen Rechnerplattformen und Installationen. Solche Laptops sind meist sehr teuer und müssen natürlich dem betreffenden Operator zur Verfügung stehen.

Ein anderer Lösungsansatz wäre die Bereitstellung eines Thin Clients durch die Management-applikation. Dieser Client stellt dann nur wenige Funktionen zur Verfügung und hat somit nur ein begrenztes Einsatzfeld. Die Einschränkung auf die Betriebsaspekte Informationsdienst und Überwachung, langt meist als Hilfestellung zur Problemlösung. Damit besteht der Client nur aus passive Komponenten, die ihre Informationen aus bestehenden Datenbanken auslesen. Ein aktives Eingreifen in die Systemkonfiguration und Fehlerbehandlung durch die Applikation erfordern Werkzeuge zum Ansprechen der einzelnen Komponenten, was wieder in einer größeren Applikation resultiert. Weiterhin sollte der Client auf möglichst vielen Plattformen laufen. Hier bietet sich zum Beispiel Java oder andere plattformunabhängige Sprachen an.

1.3. Aufgabe

Ziel der Diplomarbeit ist die Entwicklung und prototypische Implementierung eines Web-Frontends zur Visualisierung von Managementinformationen. Als Managementsystem steht INMS von Siemens zur Verfügung. Das Frontend soll es einem Operator ermöglichen, über einen Standardwebbrowser auf Managementinformationen, die INMS bereitstellt, zuzugreifen. Neben der graphischen Darstellung der Netztopologie, soll weiterhin der Status der Komponenten angezeigt werden. Dabei soll der Status einer Komponente jederzeit auf dem aktuellsten Stand sein. Neben der reinen Darstellung, soll es möglich sein Komponenten über Fremdapplikationen zu managen. Weiterhin soll betrachtet werden, wie Daten sicher über nicht vertrauenswürdige Verbindungen übertragen werden können.

2. Einführung

2.1. Methodik/OMT

Als Basis zur Entwicklung des Softwareprojekts wird OMT (Object Modeling Technique [OMT 94]) von J. Rumbaugh verwendet. OMT stellt eine strukturierte Methode zur objektorientierten Softwareentwicklung dar.

Die Methode gibt dem Entwickler nicht nur eine genaue Vorgehensweise vor, sondern unterstützt ihn auch mit diversen Modellen, die den Entwicklungsprozeß begleiten. Es werden dabei 3 Modelle verwendet:

1. Das *Objektmodell* repräsentiert die statischen, strukturellen, datenbezogenen Aspekte des System.
2. Das *dynamische Modell* repräsentiert die zeitlichen, verhaltensmäßigen, steuerungsbezogenen Aspekte des Systems.
3. Das *funktionale Modell* repräsentiert die Übergangs- und Funktionsaspekte eines Systems.

Diese Modelle sind keine statischen Gebilde, sondern entwickeln sich im Laufe des Entwicklungprozesses weiter. Der Entwicklungprozeß teilt sich in vier Phasen auf: *Analyse*, *Systementwurf*, *Objektentwurf* und *Implementierung*.

Die Analyse versucht das reale System verständlich zu modellieren. Die Analyse beginnt mit einer Problembeschreibung, die konkrete Anforderungen an das Projekt stellt. Aus dieser Problembeschreibung wird als erstes das Objektmodell entwickelt, dann das dynamische und danach das funktionale Modell.

Im Systementwurf wird nun die Architektur des Systems festgelegt. Das System wird dabei in Teilsysteme aufgeteilt. Jedes Teilsystem stellt dabei ein Paket von Funktionen und Objekten mit einem logischen Zusammenhang dar. Weiterhin werden die Schnittstellen zu externen Ressourcen festgelegt.

Der Objektentwurf baut nun auf den vorher gesammelten Wissen auf, und kombiniert diese um die endgültigen Klassen zu erhalten. Hier werden nun auch die Algorithmen für die Implementierung festgelegt. Die Klassenstruktur wird hier zum letzten Mal angepaßt, um weitere Vererbungen und Optimierungen von Assoziationen zu ermöglichen.

Die Implementierung ist die Übersetzung des Entwurfs in eine Programmiersprache.

2.2. Einführung in INMS

INMS (Integrated Network Management System) ist ein sogenanntes „Umbrella“-Netzmanagement-system. Es integriert verschiedene Netztechnologien, Typen von Netzelementmanager und Netzservices unter einer einheitlichen Benutzungsoberfläche und ermöglicht eine einheitliche Sicht auf heterogene Netze. INMS integriert Konfigurations-, Status- und Performanz-Information einzelner Elementmanagersysteme und faßt Sie über das Netzmanagementprotokoll SNMP zusammen. Dabei werden nicht SNMP-fähigen Komponenten Proxyagenten zur Seite gestellt. Dem Benutzer zeigt sich dies in einer einheitlichen Benutzungsoberfläche. Weiterhin bietet INMS Korrelation und Filterung von Status und Alarmen.

Das folgende Bild gibt einen Ausschnitt aus den momentan unterstützten Technologien und INMS Produkten wieder.

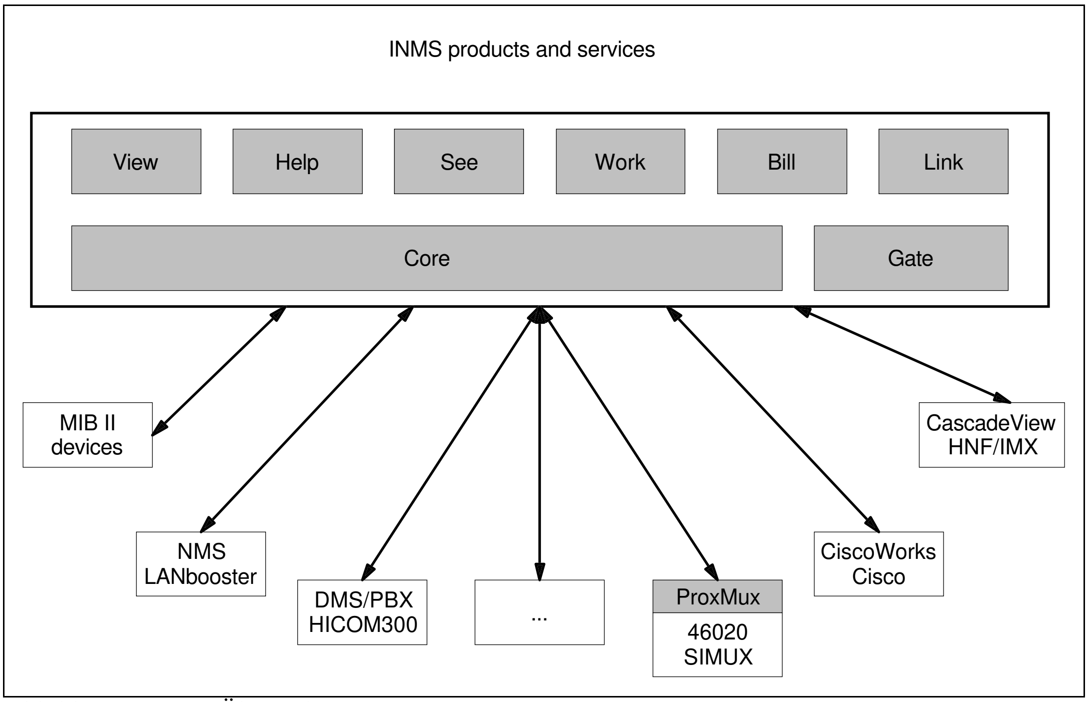


Abbildung 2 INMS Übersicht

Im weiteren werden nur die Komponenten View und Core näher besprochen. Weitere Informationen stehen in der INMS Systembeschreibung ([SIS 96]).

2.2.1. Core

Core stellt die Basisfunktionalität von INMS zur Verfügung, diese sind:

- Alarmkorrelation

  Hier werden netzelement-spezifische Alarme empfangen, korreliert, gefiltert und weitergeleitet, damit wird die Einfärbung der Netzelemente auf der graphischen Benutzungsoberfläche festgelegt.

- Zentrales Repository

  Alle Konfigurationsdaten des Netzes werden in einer zentrale Datenhaltung gespeichert. Es werden unter anderem die Topologiedaten, Informationen über Services und die Verwaltungsstruktur des mit INMS verwalteten Netzes gehalten. Zur Erfassung der Daten stehen dem Benutzer eine Autodiscoveryfunktion und ein Datenbankeditor zur Verfügung.

- Verwaltungsstruktur

  Zur Strukturierung des Netzes unterstützt INMS neben verschiedenen Benutzerrollen eine frei definierbare Domänenstruktur.

2.2.2. Domäne

Eine Domäne ist ein Teil eines Netzes, die einem oder mehreren Operatoren zugeordnet ist. Jede Domäne kann neben den Netzelementen weitere Domänen enthalten, womit sich eine Domänenhierarchie ergibt. INMS bietet verschiedene Domänentypen (organisatorisch, geographisch, technologisch, administrativ, generisch) an, so daß eine Einteilung des Netzes erleichtert wird. Es gibt keine Einschränkung welche Domänen in einer Hierarchie verwendet werden. Domänen können sich auch überlappen, so daß Netzelemente in mehreren Domänen enthalten sein können.

Beschreibung des Domänenbeispiels:

- Domäne A enthält 2 Domänen (B,C) und 2 Nodes, die über einen Link verbunden sind
- Domäne B enthält 4 Nodes und 2 Links
- Domäne C enthält 4 Nodes und 3 Links
- Node X ist in den Domänen B und C enthalten

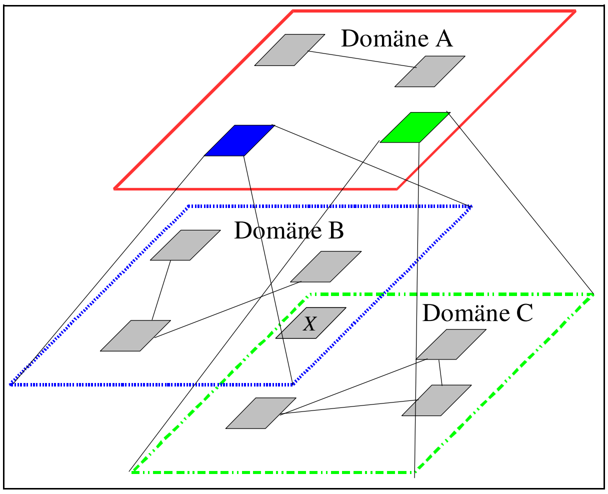


Abbildung 3 Domänenbeispiel

2.2.3. View

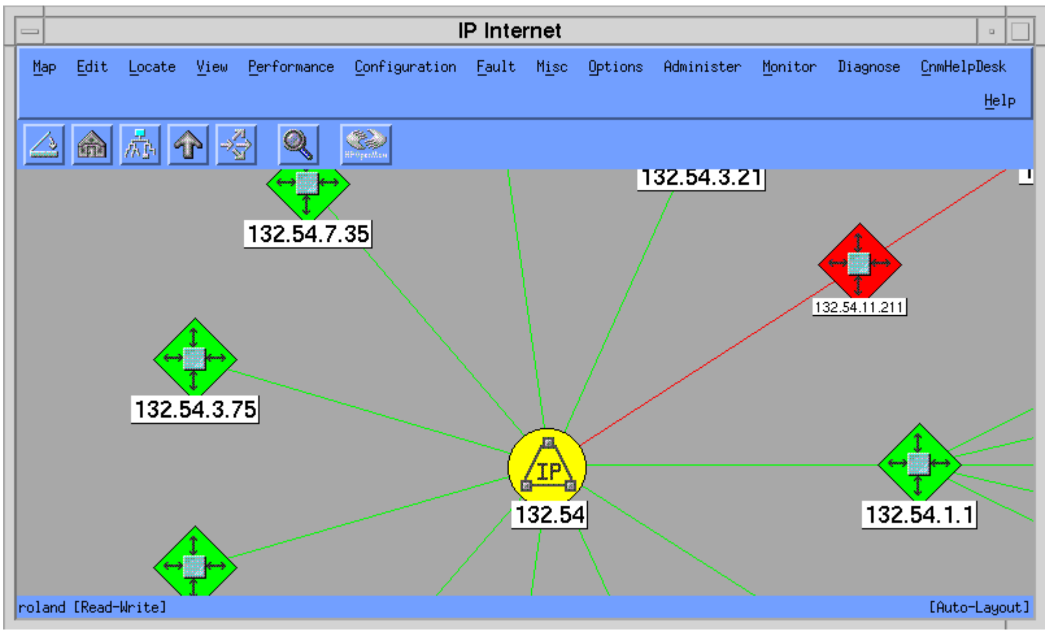


Abbildung 4 HP OpenView Screenshot

View ist für die graphischen Darstellung der Domänenhierarchie verantwortlich. View ist in die HP OpenView SNMP Managementplattform eingebunden. Neben der Darstellung der Netzknoten, Verbindungen zwischen den Knoten und sogenannter SUNI's (service user network interfaces), generiert View die operator-spezifischen Netzkarten aus der im Core abgespeicherten Domänenhierarchie. Mit Hilfe der graphischen Darstellung dieser Karten durch HP OpenView kann der Operator auf die Informationen der Element zugreifen, oder die entsprechenden Elementmanager aufrufen. Der Zustand jedes Elements wird durch verschiedene Einfärbungen des zugehörenden Symbols angezeigt.

2.2.4. Architektur

INMS besteht grundsätzlich aus 4 Softwarekomponenten:

- Datenbank-Server

  Hier werden alle Konfigurations- und Verwaltungsdaten des Netzes gehalten. Als Server wird die objektorientierte Datenbank GemStone eingesetzt.

- INMS-Server

  Der INMS-Server ist für die Alarm- und Statusverwaltung zuständig. Er empfängt und korreliert die Ereignismeldungen einzelner Elemente und sendet sie weiter an den Map-Server.

- Map-Server

  Der Map-Server generiert die Netzkarten aus der Datenbank und stellt die Statusinformationen, die vom INMS-Server geliefert werden, dar.

- CNM-Agent

  Ist nicht relevant für die Diplomarbeit.

2.2.5. Map Server

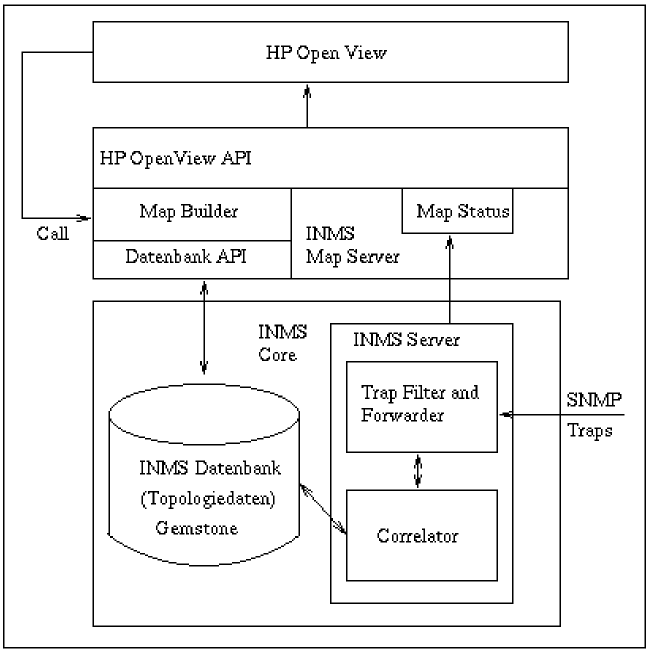


Abbildung 5 INMS Architektur

Wie Abbildung 5 zeigt ist der MapServer ein zentraler Bestandteil von INMS und sehr eng mit dem INMS Core und HP OpenView verbunden. Da er eine wichtige Rolle in dem Projekt spielt, wird er hier näher beschrieben.

Der Map Server hat grundsätzlich zwei Funktionalitäten, die Generierung der Netzkarten in HP OpenView und die Einfärbung der Symbole.

Die INMS Datenbank, in der alle Informationen über die verwalteten Netze liegen, ist eine objektorientierte Datenbank von GemStone. Die Abfrage der INMS Daten geht aber nicht direkt über eine GemStone API, sondern über die Condis API. In der Condis API ist zusätzliche Funktionalität modelliert, die GemStone nicht bietet, damit ist es nicht ohne Aufwand möglich auf die Daten direkt zuzugreifen.

Der MapBuilder hat nun die Aufgabe die INMS Daten mit Hilfe der Condis API aus der Datenbank auszulesen und daraus die Netze der einzelnen Domainen, sowie die Vernetzung aller Managed Object's in HP OpenView zu generieren. Dies geschieht über die HP OpenView API. Diese ist eine Sammelung von C-Funktionen, mit denen Objekte direkt in HP OpenView erzeugt werden. HP OpenView übernimmt diese Objekte und baut daraus die graphische Repräsentation der Netzkarten. HP OpenView bietet neben der reinen Darstellung noch mehr Funktionalität. Für jeden Operator hält es eine lokale Datenbank, in der neben der Netzkarte noch weitere Konfigurationsdaten gesichert sind. Dies sind zum Beispiel die Bildschirmkoordinaten der graphischen Symbole und weitere manuell hinzugefügte Objekte, die nicht in der INMS Datenbank aufgeführt sind, so daß die dargestellte Netzkarte nicht unbedingt mit der in INMS gespeicherten Domäne übereinstimmt. Die Layoutsteuerung wird komplett von HP OpenView übernommen und ist in INMS nicht gesichert.

Ein weiterer Teil des MapServers ist der MapStatus. Dieser ist für die Einfärbung der graphischen Symbole in HP OpenView zuständig. Jedes Managed Object hat einen Zustand. Dieser wird aus den Ereignissen (z.B. SNMP Traps), die die Objekt an den INMS Server schicken bestimmt. Der MapStatus pollt nun beim Aufbau einer Karte die Stati aus dem IMNS Server und teilt sie HP OpenView mit. HP OpenView färbt anhand dieser Stati die graphischen Symbole ein. Alle weiteren Statusänderungen werden via SNMP Traps vom INMS Server mitgeteilt, so daß die Darstellung der Managed Object's immer auf den aktuellsten Stand ist.

2.2.6. Objektmodell INMS

Das Objektmodell von INMS ist ziemlich komplex, da es einen Vielzahl von Technologien integrieren muß. Für die Diplomarbeit ist aber nur ein Teil des Objektmodells von Interesse, da die Modellierung verschiedene Operatorklassen oder SNMP Proxy's und Agenten bei der späteren Darstellung von Netzstrukturen nicht beachtet werden. Das komplette Modell kann in [DM 96] nachgelesen werden. Deshalb wird hier nur ein Teil des Objektmodells betrachtet.

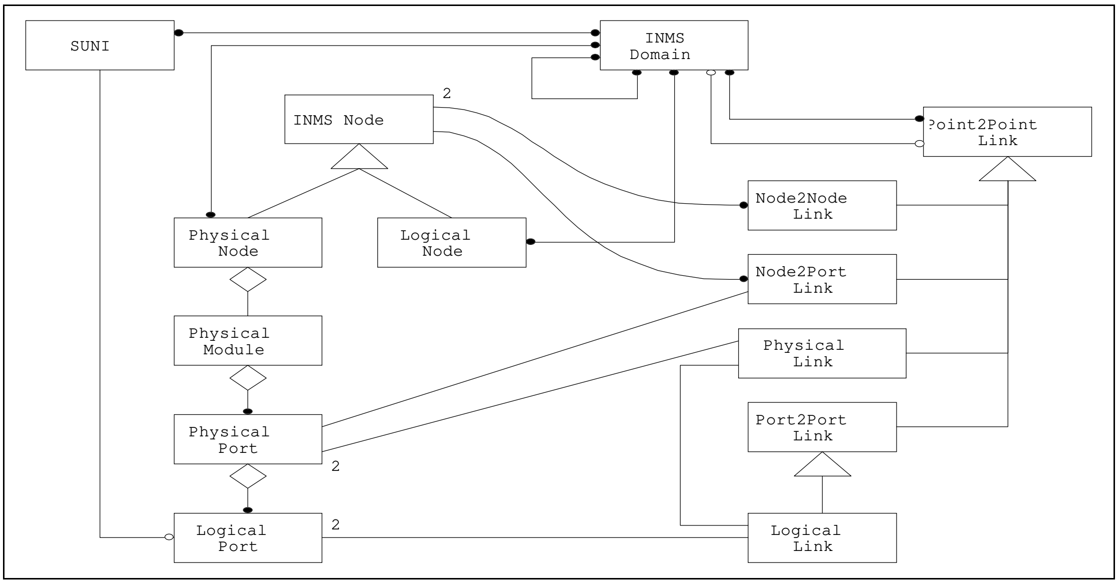


Abbildung 6 Auszug INMS V2.0 Datamodel

Zentraler Bestandteil ist die INMS Domäne. Jede Domäne kann Links, Nodes, Module, Ports, SUNI's oder weitere Domänen enthalten. Weiterhin kann ein Link mit einer Domäne verbunden werden, so daß dieser als Einstiegspunkt für eine weitere Domäne darstellt. Jeder Link ist mit zwei Nodes oder Ports verbunden, während ein Port nur einen Link bedienen kann, kann eine Node mit mehreren Links assoziiert sein. Zu jedem nicht-Node2NodeLink kann bestimmt werden, mit welcher Node er verbunden ist, da alle Modules und Ports Teil einer Node sind.

3. Analyse

In der Analyse wird ein kompaktes, präzises, verständliches und korrektes Modell der realen Welt entwickelt. Zuerst werden die Anforderungen an das System untersucht. Diese enstehen durch die Anforderungen des Auftraggebers und aus bestehenden Systemen (falls z.B. ein bestehendes System erweitert werden soll). Aus den Anforderungen werden die drei Modelle (Objekt-, funktionales, dynamisches Modell) entwickelt, wobei die Modelle am Ende der Analyse keine statischen Gebilde darstellen, sondern im Laufe des Entwicklungsprozeßes stetig erweitert werden.

Im folgenden wird als Bezeichnung für das Projekt VOMI (Visualization Of Management Information) verwendet.

3.1. Anforderungen

3.1.1. Siemens

Siemens als Betreuer der Diplomarbeit hat neben den allgemeinen Anforderungen, die man aus der Motivation lesen kann, konkrete Ansprüche. Diese sind aus dem bereits eingesetzten System (INMS) und aus Kundenwünschen (Management über Web) entstanden. Die Anforderungen sind:

- Aufbau auf INMS
  Die Applikation soll nicht eigenständig sein und INMS ersetzen, sondern soll in INMS soweit wie möglich integriert werden.
- Webgestützte Applikation
  Es soll möglich sein, die Applikation von jedem Webbrowser aufrufen zu können. Die Präferenz liegt dabei auf einer Java-Applikation. Es soll damit auch gezeigt werden, wo etwaige Grenzen eines solchen Ansatzes liegen.
- GUI Funktionalität wie HP OpenView
  Momentan wird als graphische Benutzungsoberfläche HP OpenView benutzt. Damit der Benutzer möglichst wenig Umlernzeit mit der neuen Applikation hat, soll die Funktionalität, die HP OpenView bietet, möglichst erhalten bleiben. Ein komplette Nachbildung ist nicht erwünscht.
- Reduktion auf Status- und Informationsdarstellung
  Die Applikation soll kein Komponentenmanagement realisieren, es soll sich auf Darstellung von Topologie-Informationen, Komponenteninformationen, sowie Zustand der einzelnen Komponenten beschränken.
- Aufruf externer Applikationen
  Die Zahl der Managementanwendungen die in Java lauffähig sind, steigt ständig, deshalb sollte es möglich sein aus der Applikation heraus Managementanwendungen für bestimmte physische Komponenten aufzurufen.
- Sicherer Austausch von Managementinformationen
  Es sollte möglich sein Managementinformationen zu verschlüsseln, die über unsichere Transportwege verschickt werden.

3.1.2. HP OpenView Benutzungsoberfläche

Dies ist eine Funktionsbeschreibung von HP OpenView wie sie sich dem Benutzer in Verbindung mit INMS darstellt. Dabei wurde die Beschreibung auf die Aspekte beschränkt, die später bei der Realisierung relevant sind. Diese Beschränkung ergibt sich aus den Anforderungen der Reduktion auf Status- und Informationsdarstellung. Anhand dieser Beschreibung werden weitere Aspekte zur funktionalen Beschreibung des Systems gewonnen, die in 3.1.1. nur grob beschrieben werden (GUI Funktionalität wie HP OpenView).

- Es kann immer nur eine Domänenhierarchie eines Operators dargestellt werden.
- OpenView speichert für jeden Operator die Map in einer eigenen Datenbank. Es wird dabei die Anordnung der Symbole einer Domäne nur in der OpenView Datenbank abgespeichert, und nicht in der INMS Datenbank.
- Über den Aufruf des Mapbuilders von INMS wird die komplette Map aufgebaut.
- Mit der linken Maustaste wir ein Symbol selektiert.
- Mit der mittlere Maustaste kann man eine Node verschieben.
- Mit einem Doppelklick auf ein Domänensymbol öffnet sich ein Fenster mit der neuen Domäne.
- Ein selektierter Link und „Show View“ im Menü zeigt die entsprechende SUNI oder Link View an.
- Mehrfache Links zwischen zwei Nodes werden als ein Link dargestellt und beim Doppelklick aufgeschlüsselt (auch „Meta Connection“ genannt).
- Selektierte Symbole werden in allen Submaps selektiert dargestellt
- In den SUNI/Link Views werden die Properties einzelner Managed Ojects dargestellt.
- Schließen eines SUNI Views, schließt alle von diesem aus geöffneten Fenster.
- Die graphischen Symbole für Nodes (und Domänen) sind in Klassen aufgeteilt, wobei es aus einer Grundklasse (Shape, Umriß), einer Subklasse (graphic, Bitmap) und einer Einfärbung besteht.
- Die Einfärbung ergibt sich aus dem Status einer physischen Komponente, die mit dem Symbol verbunden ist, oder aus den Stati mehrerer Komponente, die sich in einer darunterliegenden Submap befinden.

3.2. Objektmodell

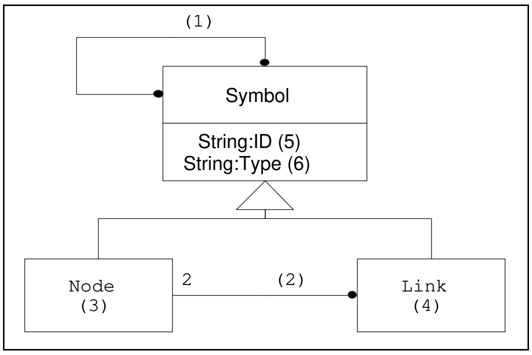


Abbildung 7 VOMI-Objektmodell

Das Objektmodell ergibt sich aus folgender Überlegung.

Das Frontend stellt Managed Objects dar. Diese Managed Objects stellen sich dem Benutzer entweder als graphische Symbole oder Linien, die die Symbole verbinden, dar. Die Managed Objects werden also in 2 Klassen aufgeteilt, die im weiteren Verlauf mit **Nodes (3)** und **Links (4)** bezeichnet werden. Nodes stellen Knoten im Netz dar, während Links die Nodes untereinander verbinden. (siehe auch 7.1.). Jeder Link muß dabei mit zwei Nodes assoziiert sein (2). Da sich die Managed Objects als graphische Objekte dem Benutzer präsentieren, werden sie im weiteren unter dem Begriff **Symbol** zusammengefaßt. Mehrere Symbole bilden eine Submap und jedes Symbol kann als Einstiegspunkt zu einer weiteren Submap dienen (1). Weiterhin ist jedes Symbol eindeutig durch eine Identifier (5,6) gekennzeichnet.

Wenn man das Objektmodell von INMS (siehe 2.2.6.) betrachtet, werden die Klassen auf der linken Seite als graphische Symbole dargestellt, während rechts die Links stehen, die als Linien dargestellt werden. Das obere Objektmodell stellt eine grobe Vereinfachung des INMS Objektmodells dar. Das INMS Objektmodell ist aus Netzwerksicht (Was wird gemanaged ?) entstanden, während das VOMI Objektmodell aus der Sicht des Benutzers entstanden ist (Was sieht der Benutzer auf der Benutzungsoberfläche ?). Es ist auch viel einfacher aufgebaut, damit bei späteren Änderungen von INMS nicht das VOMI Objektmodell betroffen ist. Dieser Unterschied zwischen den Modellen führt zu einer Umsetzung der INMS Objekten in die VOMI Objekten. Alle Domänen, Nodes, Ports, Module können als VOMI-Nodes, die verschiedenen Links als VOMI-Links modelliert werden. Die Unterscheidung der einzelnen INMS-Klassen geschieht über ein Attribut (6).

3.3. Benutzungsoberfläche

Die Funktionalität der Benutzungsoberfläche aus der Sicht des Benutzers, ergibt sich aus den Anforderungen von Siemens (3.1.1.) und der Analyse der HP OpenView Benutzungsoberfläche (3.1.2.).

3.3.1. Submap Display

Jedes Submap Display besteht aus einer Menüzeile und einem Fenster mit der Submap. Falls die Submap größer als das tatsächliche Fenster ist, werden Slider am Rand dargestellt, um den Inhalt zu verschieben.

Mausaktionen im Fenster	
Aktion	***Auswirkung***
Slider bewegen	Die Submap wird innerhalb des Fensters bewegt.
Einfachklick Links	Das unter dem Mauszeiger liegende Symbol wird selektiert. Falls Informationen über das Symbol vorliegen, wird der Informationsbutton selektierbar.
Doppelklick Links	Das unter dem Mauszeiger liegende Symbol wird geöffnet. Falls das Symbol eine Submap enthält, wird ein neue Submap Display mit der Submap geöffnet. Falls keine Submap unter dem Symbol liegt, wird das Information Display mit den Symbol Daten geöffnet. Falls keine Daten vorliegen, wird nichts weiter gemacht.
Drag	Herumziehen eines unter der Maus liegenden Nodes, durch gedrückthalten der Maustaste. Der Node folgt den Mausbewegungen in dem Fenster. Beim Loslassen der Maustaste, fällt der Node auf das Fenster zurück. Die Koordinaten werden gespeichert und das Fenster neugezeichnet.
Einfachklick rechts	Informationen über ein Symbol darstellen.

Menüaktionen	
Aktion	***Auswirkung***
Autolayout	Startet den Autolayout Vorgang erneut für eine Submap.
Autolayout Konfiguration	Es erscheint ein Dialog zur Auswahl und Konfiguration des Autolayout-Algorithmus.
Zoom In/Out	Vergrößern/Verkleinern der Submap, mit anschließendem Neuzeichnen der Submap.
Login	Es erscheint der Login Dialog, um sich neu in der Datenbank einzuloggen.
Mapstat	Es erscheint eine Übersicht über den Gesamtzustand der Map.
Submapbrowser	Es erscheint ein Fenster in dem die Hierarchie textuell angezeigt wird.
Eventbrowser	Alle bis jetzt empfangenen Ereignisse werden in einem neuen Fenster angezeigt.
Information	Informationen über ein Symbol darstellen.
Help	Zeigt ein Hilfefenster an
Root Submap	Springt in die oberste Hierarchie-Ebene.

Menüaktionen	
Elementmanger	Falls ein selektierter Node einen Elementmanager zur Verfügung stellt, wird dieser aufgerufen.

3.3.1.1. Login Display

Das Login Display besteht aus eine einfachen Eingabemaske. Hier logt sich der Operator in die INMS Datenbank ein. Falls der Verbindungsaufbau erfolgreich ist, werden alle vorhandenen Fenster, die einem anderem Operator zugeordnet sind, geschlossen. Der Login Display wird anschließend geschlossen und die View mit der Root Map angezeigt.

GUI Element	Beschreibung
Operatorname	Der Name mit dem sich der Operator in die INMS Datenbank einlogt.
Paßwort	Das zu dem Operatornamen gehörende Paßwort.
VOMI Server Adresse	FQDN oder IP Adresse des VOMI Servers. Ist vorbesetzt mit der Adresse von der das Applet geladen wurde.
Verschlüsselungstechnik	Liste mit Verschlüsselungsverfahren, die unterstützt werden.
Login Button	Startet die Anmeldung an den angegebenen Server mit dem Operatorname und Paßwort. Bei erfolgreichem Verbindungsaufbau wird ein neues SubmapDisplay mit der Root Submap geöffnet geöffnet.

3.3.1.2. AutoLayout Dialog

Mit dem Autorlayout Dialog kann der Operator den Algorithmus zur Anordnung der Symbole festlegen. Algorithmen können verschiedene Einstellungsfelder bereitstellen.

GUI Element	Beschreibung
Algorithmen Liste	Liste mit Autolayoutalgorithmen die zur Auswahl stehen.
Algorithmus Einstellungen	Bereich in dem Dialog wo algorithmusspezifische Einstellungen gemacht werden.

3.3.1.3. Symboldisplay

DasSymboldisplay zeigt Informationen über ein Symbol an.

GUI Element	Beschreibung
Liste von Informationen	Liste mit allen Management-Informationen die in einem Symbol enthalten sind.
Elementmanager	Falls ein selektierter Node einen Elementmanager zur Verfügung stellt, wird dieser aufgerufen.

3.3.1.4. Erweiterungen

- Submapbrowser

  Stellt die Map eines Operator textuell in einer Hierarchie, vergleichbar mit einer Dateiauswahlboxen, dar.

- Mapstat

  Der Mapstat gibt eine Übersicht über den Status aller Symbole einer Map an.

- Eventbrowser

  Es wird eine Liste aller Ereignisse dargestellt, die bis jetzt empfangen wurden.

3.4. Dynamisches Modell

Das dynamische Modell zeigt das zeitabhängige Verhalten der Objekte, wie sie auf externe Ereignisse reagieren und diese nach außen hin sichtbar sind. Der Zustand eines Objekts ergibt sich aus den Attributwerten und den Verknüpfungen mit anderen Objekten, und ändert sich je nach Art des Ereignisses.

In dem ganzen System gibt es nur zwei externe Ereignisquellen, das ist der Benutzer und das sind die Traps, die von den einzelnen Netzkomponenten zu dem INMS Server geschickt werden, und von dort weiter zu dem Frontend.

3.4.1. Benutzerereignisse

Das dynamische Objektmodell für die Benutzerereignisse ergibt sich direkt aus 3.3. .

3.4.1.1. Mausereignisse

Beschreibung aller Ereignisse die in der Darstellung der Submap von der Maus ausgelöst werden.

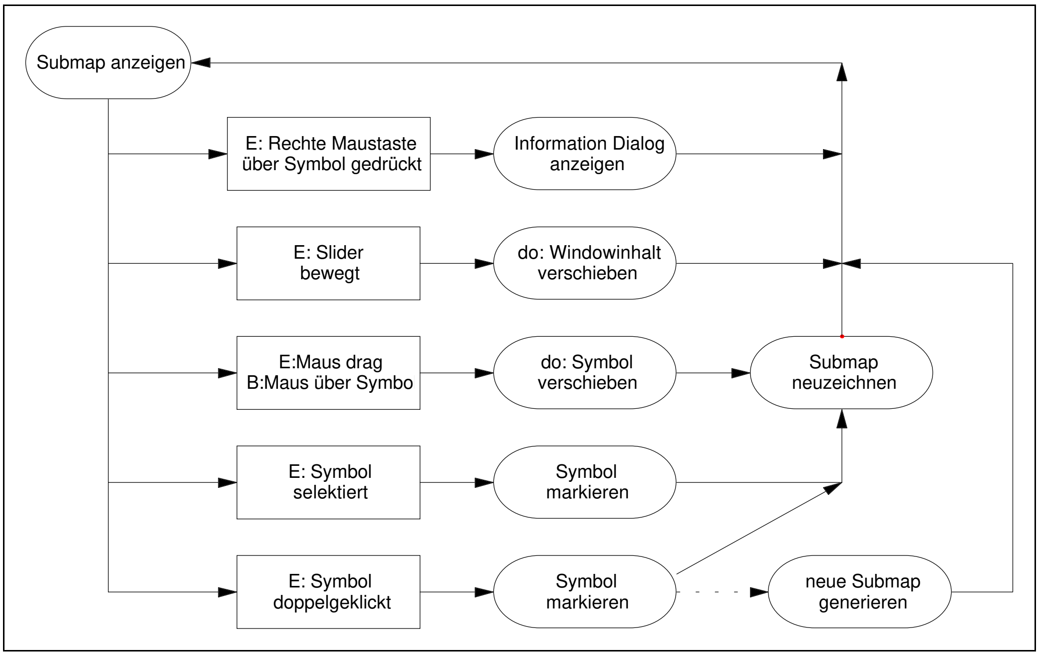


Abbildung 8 Mausereignisse

3.4.1.2. Menüereignisse

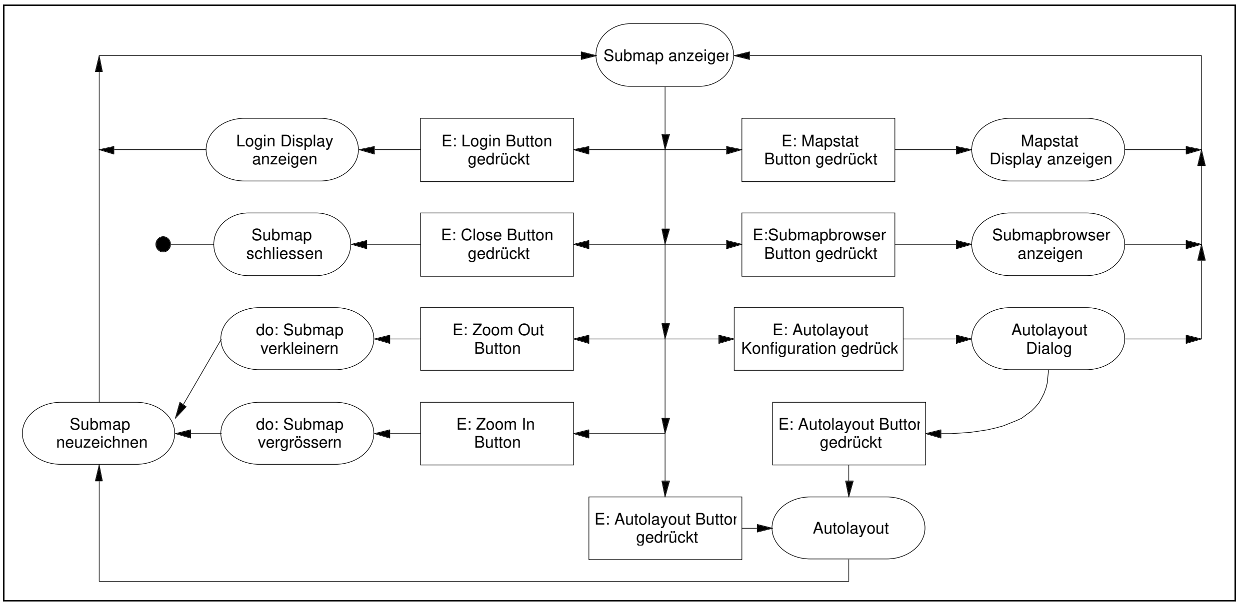


Abbildung 9 Menüereignisse

3.4.1.3. Zustandsänderung der Symbole

Jedes Symbol kann bestimmte Zustände annehmen. Der Zustand ergibt sich aus den Ereignissen (hier SNMP Traps), die der INMS Server an die Applikation weitersendet.

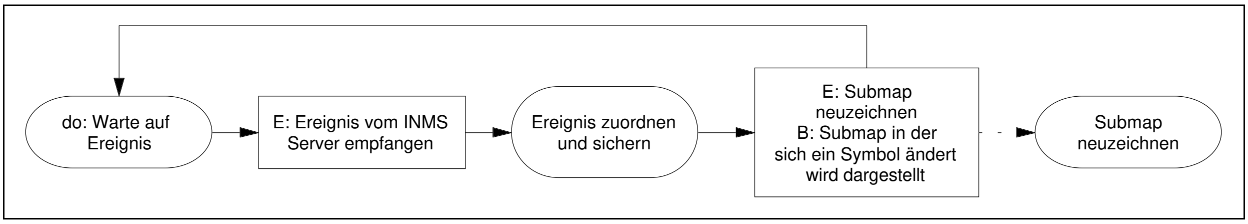


Abbildung 10 INMS Ereignisse

3.5. Funktionales Modell

Das funktionale Modell beschreibt die Berechnungen in dem System, es spezifiziert was geschieht. Es wird dabei gezeigt, wie die Ausgabewerte von den Eingabewerten abgeleitet werden. In diesem Projekt ist es trivial, da es hauptsächlich um Datenbankzugriffe geht und die Daten kaum verändert werden.

3.5.1. Datenflußmodell für Symbole

Beschreibung des Fluß von Symbolen, von der INMS Datenbank als Quelle bis zur Anzeige als Symbole auf der Benutzungsoberfläche.

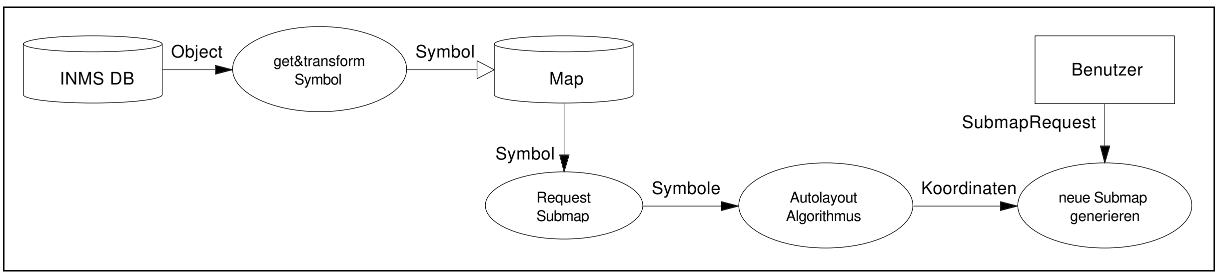


Abbildung 11 Symboldatenfluß

Will der Benutzer eine neue Submap anzeigen, generiert er damiteine Anfrage an das Programm zur

Erzeugung einer neue Submap. Zur Darstellung einer Submap werden die Koordinaten der einzelnen Symbole benötigt. Diese werden mit Hilfe des Autolayoutalgorithmus aus den Symboldaten erzeugt. Da in dem internen Datenspeicher **Map** nicht alle Objekte von Anfang an vorhanden sind, kann dies zu einer Abfrage in der INMS Datenbank führen. Diese liefert aber die Symbole in der INMS Datenbankform, so daß diese erst in die VOMI Objekte übergeführt werden müssen.

3.5.2. Datenflußmodell für Traps/Ereignisse

Der Fluß der SNMP Traps vom INMS Server bis zur Statusänderung eines Symbols.

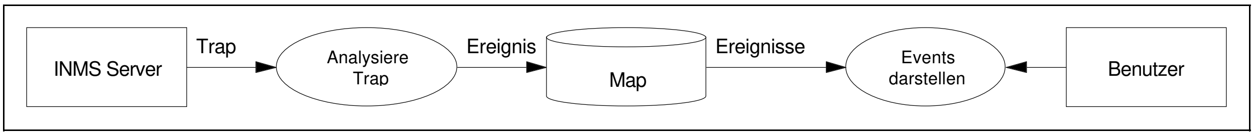


Abbildung 12 Trapdatenflußmodell

Traps werden asynchron von den einzelnen Komponenten im Netz verschickt. Der INMS Server sammelt und korreliert diese. Die gefilterten und korrelierten Traps werden dann an die Applikation weitergesendet. Diese muß feststellen, ob die gesendete Nachricht ein Symbol in der Applikation betrifft. Falls ja wird das Ereignis, dem entsprechenden Symbol hinzugefügt. Weiterhin kann sich der Benutzer alle empfangenen Ereignisse darstellen lassen.

4. Systementwurf

4.1. Systemdesign

Es wird nun versucht anhand der aus der Analyse gewonnen Daten die Systemarchitektur zu gestalten. Das System wird in Teilsysteme zerlegt, wobei jedes Teilsystem aus mehreren Klassen,, Assoziationen, Operationen und einer wohldefinierten Schnittstelle zu anderen Teilsystemen bestehen kann, die eine gemeinsame Eigenschaft besitzen. Ein Beispiel wäre das Teilsystem „Graphische Oberfläche", daß aus mehreren Dialogklassen besteht. Teilsysteme sollten möglichst unabhängig voneinander sein.

4.1.1. Systemarchitektur

Das in der Analyse vorgestellte Objektmodell ist für eine Implementierung noch nicht vollständig. Die Analyse hat gezeigt was das System zu leisten hat, jetzt muß betrachtet werden wie das System das leisten kann. Folgende Aufgaben muß das System erfüllen:

1. Objekte aus der INMS Datenbank auslesen und in Symbol-Objekte umwandeln.
2. Alle Symbole einer Map korrekt im Speicher aufbauen.
3. Die Map darstellen und die Symbole übersichtlich auf der Benutzungsoberfläche anordnen.
4. Ereignisse aus dem INMS Server auslesen und den Symbolen zuordnen.
5. Statusänderungen von Symbolen erkennen und am Bildschirm anzeigen.

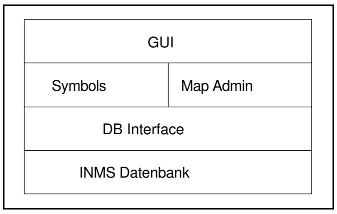


Abbildung 13 Gesamtarchitektur

Folgende Teilsysteme können anhand Ihrer Eigenschaften und Funktionen gebildet werden:

- GUI
  Die GUI stellt die Symbole und deren Informationen auf der graphischen Benutzungsoberfläche dar. Weiterhin nimmt sie Benutzereingaben zur Steuerung einzelner Teilsysteme entgegen und leitet diese an die entsprechenden Teilsysteme weiter.
- Symbole
  Die Symbole aus dem Analyse-Objektmodell bilden die Grundlage für die Submaps. Das sind die Objekte die auf der graphischen Oberfläche dargestellt werden.
- Mapverwaltung
  Hier werden die Symbole verwaltet, der MapAdmin stellt die Schnittstelle zwischen den Symbolen und der GUI dar. Er verwaltet die Map und die Zuordnung SubmapDisplay (Darstellung einer Submap) zu den entsprechenden Symbolen. Damit bekommt er eine zentrale Rolle, da er einerseits Kenntnisse über die unteren Schichten haben muß, um Symbole von der Datenbank in die Map einzutragen und über die darüberliegende Schichten. Statusänderungen von einzelnen Symbolen müssen der GUI mitgeteilt werden, dazu muß der Map Admin wissen welches GUI Objekt gerade dieses Symbol darstellt. Eine weitere Funktion ist die graphische Anordnung der Symbole durch einen Autolayoutalgorithmus.
- Datenbank Schnittstelle
  Dieses System stellt die Verbindung zu der INMS Datenbank her, und wandelt die Datenbankobjekte in Symbol Objekte um. Die Objekte werden an den Map Admin weitergereicht, der sie dann in die entsprechende Map einträgt. Neben der Schnittstelle zur INMS Datenbank muß noch eine Schnittstelle zu dem INMS Server bestehen, um den Status aller Komponenten zu erhalten. Über den INMS Server kann die Applikation auch Statusänderungen während der Laufzeit erhalten. Diese werden als SNMP Traps gesendet.

Aus dieser Gesamtarchitektur und der Aufgabenbeschreibung ergeben sich nun weitere Klassen zu dem Analyseobjektmodell.

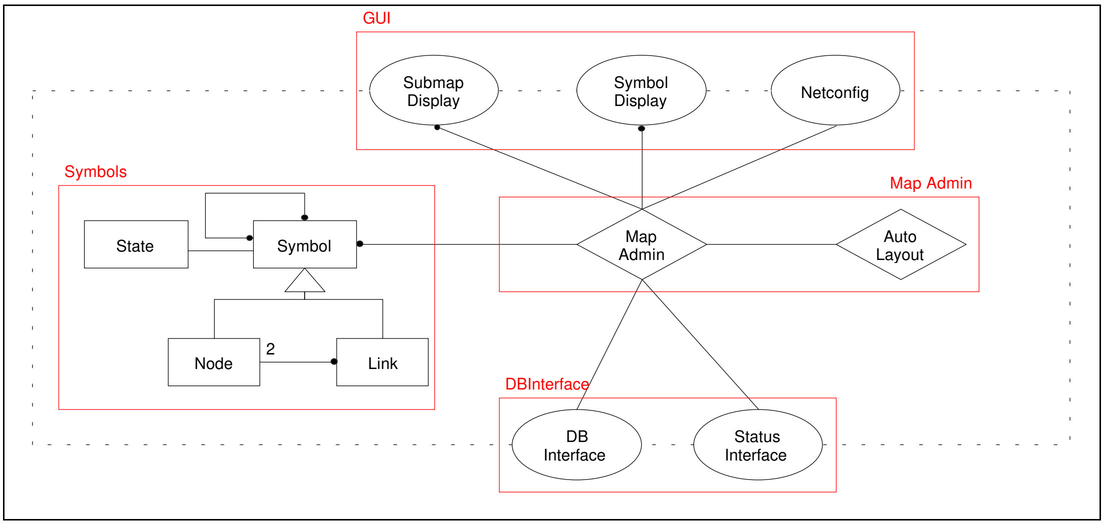


Abbildung 14 Objektmodell

Eine Beschreibung der Klassen (ohne den Klassen aus dem Analyseobjektmodel)

- DBInterface
  Das DBInterface ist die Schnittstelle zur INMS Datenbank. Es liest die Daten aus der Datenbank aus und erzeugt daraus die Symbolobjekte, die es dann an den Map Admin weitergibt. Das DBInterface wird in 4.2. näher beschrieben.
- StatusInterface
  Das StatusInterface bildet die Schnittstelle zu dem INMS Server. Es holt nach dem Aufbau der Submap den Status aller Symbole. Weiterhin soll es asynchrone Ereignisse (Traps) empfangen, die

während der Laufzeit der Applikation Statusänderungen signalisieren. Das StatusInterface wird in 4.3. näher beschrieben.

- Map Admin
  Der Map Admin ist der zentrale Teil der Applikation, er verwaltet alle Objekte. Die Funktionalität soll sich dabei auf die Verwaltung beschränken. Funktionen wie der Autolayout-Algorithmus werden als eigenständige Objekte verwaltet. Damit muß bei neuen Algorithmen nicht der Map Admin zusätzlich geändert werden.
- AutoLayout
  Der Algorithmus zur graphischen Anordnung der Symbole in einer Submap. Diese Klasse kann mehrere Algorithmen enthalten, die über eine einheitliche Schnittstelle angesprochen werden.
- SubmapDisplay
  Dies ist die erste Oberflächenklasse, die Symbol-Informationen darstellt. Hier wird der Aufbau einer Submap topologisch durch graphische Symbole (Bitmaps und Linien) dargestellt.
- SymbolDisplay
  In dieser Klassen werden die im Symbol enthaltenen Daten dargestellt. Vergleichbar mit dem SUNI View von INMS.
- NetConfig
  Diese Klasse dient zur Steuerung des Verbindungsaufbaus und dem Datenbanklogin.
- State
  Der Zustand eines Objekts.

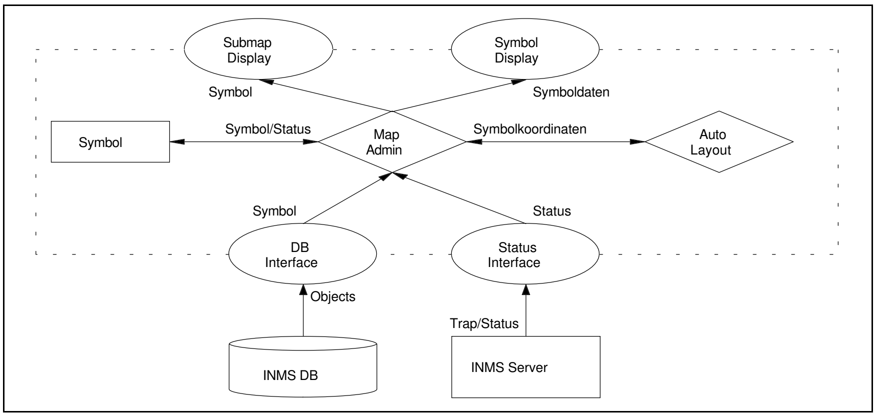


Abbildung 15 Datenfluß

Dies ergibt folgenden Datenfluß zwischen den Objekten.

4.1.2. Client Server Architektur

Wie in der Analyse schon angesprochen, soll die Applikation als Client-Server Applikation verwirklicht werden. Für die Aufteilung der Applikationen gibt es mehrere Ansatzpunkte, diese werden aber durch die gegebene Systemumgebung von INMS und Java eingeschränkt. Zuerst wird darauf eingegangen, wie der Server in die vorhandene INMS Umgebung integriert weden kann. Danach wird die Funktions- und Datenaufteilung zwischen Client/Server behandelt.

4.1.2.1. Ansatzpunkte am INMS System

Die Applikation muß auf die Datenbank im INMS System zugreifen können. Die Analyse von INMS zeigt hierzu nur zwei Möglichkeiten.

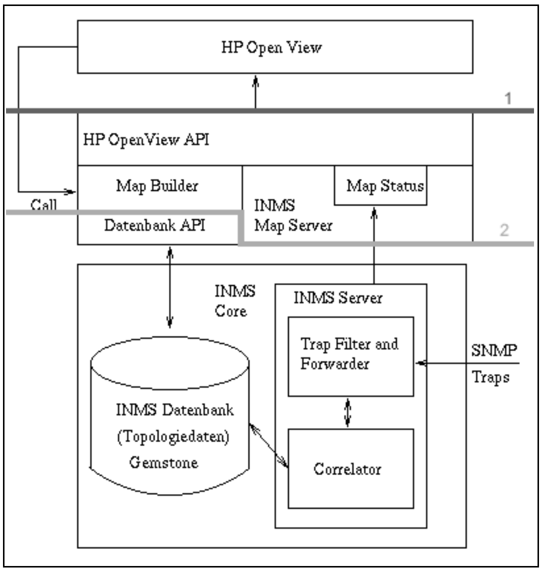


Abbildung 16 INMS

1. nach dem MapBuilder
2. vor dem MapBuilder

zu 1. nach dem MapBuilder:

Die Vorteile sind:

- Man greift auf bestehende Komponenten zurück, und es muß der MapBuilder nicht nachprogrammiert werden.
- Die interne Struktur der INMS Datenbank und die Zugriffe darauf, werden durch den MapBuilder verdeckt. Damit ist man unabhängig von Datenbankänderungen.
- Änderungen im MapBuilder und der Datenbankschnittstelle bleiben auf ein Paket beschränkt, was bei Nachprogrammierung des MapBuilders nicht der Fall ist. Die Schnittstelle zur Oberfläche (hier HP OpenView) darf als fest betrachtet werden.

Die Nachteile sind:

- Die HP OpenView API ist sehr mächtig und umfangreich, um eine möglichst breite Schnittstelle zu dem System zu bieten. Man muß die komplette Funktionalität von HP OpenView wie sie der MapBuilder benutzt implementieren, was bedeutet, daß auf jeden Fall mehr Funktionen und Datenstrukturen implementiert werden müssen, als später gebraucht werden.
- Der MapBuilder baut grundsätzlich die komplette Domainenhierarchie eines Operators auf. Dies führt zu Verzögerungen beim Start der Applikation und zu hohem Speicherverbrauch bei großen Hierarchien.
- Es gibt Probleme mit der Aktualität der Objekte, die dadurch entstehen, daß der MapBuilder die Mapobjekte in HP OpenView (oder hier in VOMI) aufbaut und bei Änderungen in der INMS Datenbank seine Objekt nicht aktualisiert. Dies geschieht erst, wenn der Operator die komplette Map neu erzeugen läßt. Dieses Problem tritt aber bei allen vorgestellten Architekturen auf, da immer ein Teil der Objekte lokal gehalten wird und es keine Alarmierung von Seiten der Datenbank bei Änderung ihres Inhaltes gibt.

Zu 2. vor dem MapBuilder:

Hier wird direkt auf die Condis-API (2.2.5.) aufgesetzt, so daß ein direkter Zugriff auf die INMS Datenbank möglich ist.

Die Vorteile sind:

- Zugriffe auf die Datenbank können optimiert werden, indem nur Daten geholt werden, die benötigt werden.
- Man kann die Datenbankschnittstelle von INMS (Condis API) durch eine von Java bereit gestellte Datenbankschnittstelle (wie JDBC oder von GemStone) ersetzen und somit einen direkten Zugriff von der Applikation auf die Datenbank erlauben.
- Es wird auf die HP OpenView API verzichtet, die einen großen Teil des MapBuilders ausmacht. Damit löst man sich von HP OpenView und ist nicht auf ein externes Produkt angewiesen.

Die Nachteile sind:

- Die Datenbankschnittstelle kann sehr aufwendig werden, da die Informationen für ein Managed Object in mehreren Datenbankobjekten verteilt sein können (siehe Unterschiede zum INMS Objektmodell).
- Wenn nur Teile der Maps geholt werden, sollte eine Verbindung zur INMS Datenbank immer offen sein, da erneute Einlogversuche relativ aufwendig sind. Dies liegt daran, daß die Condis API in Smalltalk geschrieben ist, und somit ein gewisser Overhead beim Öffnen der Datenbank vorhanden ist, der nicht optimiert werden kann. (Ein Einlogversuch bewegt sich so um die 10 Sekunden). Eine weitere Bremse können erweiterte Sicherheitsmechanismen wie Verschlüsselung sein, die beim Aufbau der Verbindung erst Schlüssel generieren und austauschen müssen.
- Änderungen an der Datenbank, ziehen Änderungen an zwei Stellen (MapBuilder und der Applikation) nach sich. Änderungen in der Objektstruktur ziehen meist Änderungen in den Abfragen nach sich, und somit müssen die Aufrufe in beiden Programmen angepaßt werden.

Schlußfolgerung:

Der Ansatz nach dem MapBuilder wird nicht weiter betrachtet, da dessen Nachteile schwerwiegender sind als die von dem direkten Datenbankansatz. Vor allem ist die Bereitstellung einer HP OpenView API vom Arbeits- und Zeitaufwand viel zu hoch. Dies würde auch zu einer weiteren Abhängigkeit von HP OpenView für INMS bedeuten. Auch die gezielte Abfrage der Datenbank nach Objekten die benötigt werden, fehlt vollkommen, was der Anforderung nach einem performanten Applikation widerspricht, da man dem MapBuilder nur anweisen kann, die komplette Domänenhierachie zu holen.

4.1.2.2. Client/Serveraufteilung der Applikation

Da die Applikation meist nicht auf dem Rechner ausgeführt wird, auf dem sich die Datenbank befindet, wird die Applikation als verteilte Anwendung realisiert. Man versucht die einzelnen Komponenten so auf die Systeme zu verteilen, daß ein möglichst gutes Zusammenspiel ermöglicht wird. Möglich wäre auch eine reine Serverapplikation, wo nur die graphischen Daten zu der Erzeugung der GUI übertragen werden (Beispiel: X-Protokoll). Da diese Protokolle zur Zeit eher plattformabhängige Lösungen darstellen und sehr ressourcenfressend sind, wird darauf nicht eingegangen.

Da die INMS Datenbank Schnittstelle in C geschrieben ist, aber der Client in Java laufen soll, muß dies bei den weiteren Betrachtungen berücksichtigt werden.

Die Gesamtarchitektur muß während der Verteilung immer erhalten bleiben, es dürfen nur weitere Zwischenschichten eingefügt werden.

1. GUI auf der Clientseite
2. Symbole auf der Client-Seite
3. Ersetzung des Servers durch eine Java-Datenbankschnittstelle

zu 1. GUI auf der Clientseite:

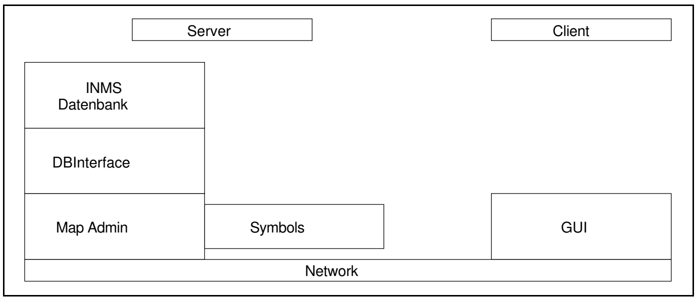


Abbildung 17 Architektur - GUI auf Clientseite

Die GUI befindet sich als einzige Komponente auf der Client Seite, alle Symbole, der MapAdmin und das Interface befindet sich auf dem Server. Der Server ist in den meisten Fällen der Leistungsstärkere und kann damit auch mehr Funktionen übernehmen.

Die Vorteile sind:

- Der Client ist extrem klein, da sich alle Funktionalität und Datenhaltung auf dem Server befindet. Damit können auch leistungsschwache Maschinen bedient werden
- Der Server kann systemabhängig implementiert werden, damit sind weitere Leistungssteigerungen möglich.
- Da mehr Komponenten sich zentral befinden, müssen bei Änderungen nicht auch alle Clients aktualisiert werden.
- Es werden nur die momentan benötigten Daten übers Netz geholt.
- Durch Einsatz von RMI (vergleichbar mit Sun-RPC) auf der Client Seite, können die Zugriffe auf den Server völlig transparent geschehen.

Die Nachteil sind:

- Bei jeder Benutzerinteraktion, die Daten vom Server anfordert, müssen diese übertragen werden. Abhilfe würde ein lokaler Caching Algorithmus bringen.
- Bei Einsatz von RMI würde ein nur schwer zu wartender Java/C Server enstehen.
- Bei nicht Einsatz von RMI muß eine Art RPC implementiert werden, was die Entwicklungszeit verlängert.
- Der Server muß extrem leistungsfähig sein, um viele Clientverbindungen verwalten zu können.

Zu 2. Symbole auf der Client-Seite:

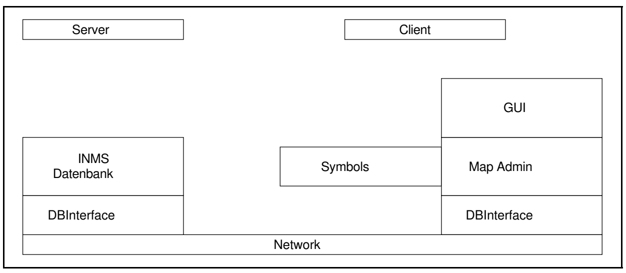


Abbildung 18 Architektur - Symbole auf Clientseite

Hier befinden sich nun alle Managed Objects die zur Darstellung einer Map notwendig sind auf der

Client-Seite. Die Verwaltung der Symbole durch den MapAdmin ist sehr eng an die Symbole gebunden, deswegen sollten beide nicht getrennt werden, um nicht unnötigen Verkehr über das Netzwerk zu erzeugen. Eine Möglichkeit wäre die teilweise Auslagerung des MapAdmins auf den Server (im speziellen die Autolayout-Funktionalität), von dem wird aber abgesehen, da die meisten Funktionen weniger rechenintensiv sind, sondern viel mehr zugriffsintensiv. Das DBInterface wird zwischen Client und Server aufgesplittet, wobei der Serverteil sich mit der Umsetzung von Clientanfragen in INMS spezifische Datenbankbefehlen beschäftigt und die Umsetzung von Datenbankdaten in Javaobjekte vorbereitet. Der Client baut dann aus den vorbereiteten Javaobjekten die tatsächlichen Objekte (das genaue Vorgehen wird in 4.2. beschrieben).

Die Vorteile sind:

- Die Objekte für die angezeigten Submaps sind lokal vorhanden. Zugriffe übers Netz müssen nur noch gemacht werden, falls das Objekt nicht vorhanden ist.
- Zwischen dem Server und dem Client müssen nur Daten ausgetauscht werden und keine Funktionsaufrufe, so daß auf RPC Mechanismen verzichtet werden kann.
- Es kann eine Verschlüsselungsschicht implementiert zwischen DBInterface und Network werden, um sichere Datenaustausch zu gewähren. Dabei muß nicht Rücksicht auf von Java bereitgestellte Algorithmen genommen werden. (siehe auch 4.2.2.)

Die Nachteile sind:

- Es ist ein Server vorhanden mit sehr geringer Funktionalität, der neben dem Client auch gewartet werden muß.
- Die Rechenlast ist fast komplett auf der Clientseite.

Zu 3. Ersetzung des Servers durch eine Java-Datenbankschnittstelle:

Es wird hier nun komplett auf einen Serveranteil verzichtet. Der Client greift unter Verwendung von Systembibliotheken direkt auf die INMS Datenbank zu. Dabei muß die Funktionalität der CondisAPI von dem DBInterface bereitgestellt werden.

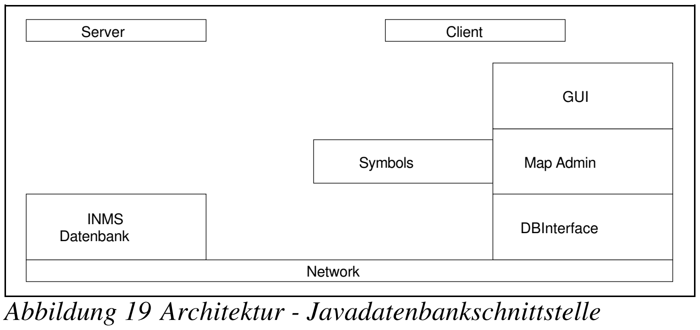


Abbildung 19 Architektur - Javadatenbankschnittstelle

Die Vorteile sind:

- Die Applikation ist komplett in einer Sprache geschrieben und nicht mehr abhängig von anderen Programmen.
- Zugriffe auf die Datenbank können noch gezielter und performanter geschehen, da eine weiter Zwischenschicht fehlt.
- Es müssen weniger Software-Packete anderer Hersteller benutzt werden als bei den anderen Verfahren (Verzicht auf Condis API)

Die Nachteile sind:

- Der Client ist ein völlig eigenständiges Produkt, was bei Erweiterungen in INMS zu erhöhten Aufwand bei Wartung und Programmierung führt.
- Es ist noch nicht möglich dies auf einfache Weise zu implementieren. Die verwendete Datenbank Gemstone hat noch keine Schnittstelle zu den von JAVA bereitgestellt Mechanismus JDBC.
- Die erweiterte Funktionalität der Condis API muß programmiert werden.

Schlußfolgerung:

Der erste Ansatz hat den großen Nachteil, daß zu viele Daten übertragen werden. Dies kann bei Verbindungen mit niedriger Bandbreite zu erheblichen Verzögerungen des Systems auf Benutzer-reaktionen führen, was die Akzeptanz dieses Produkts erheblich mindert. Weiterhin ist der Ressourcenbedarf auf der Serverseite nicht zu unterschätzen, da mehrere Clients sich an den Server anbinden sollen.

Der dritte Ansatz wäre der Idealste vom Implementierungs- und Wartungsaufwand. Er entlastet erheblich das Netzwerk und den Server. Er läßt sich aber noch nicht realisieren, da die notwendigen Komponenten erst in Entwicklung sind. Dadurch bleibt nur der zweite Ansatz, der dem dritten Ansatz sehr ähnlich ist.

4.2. DBInterface

Die DBInterface hat eine zentrale Rolle. Es muß die Objekte über das Netz transportieren und die INMS Objekte in die Symbolobjekte umsetzen.

Objekte sind hier nur reine Datenobjekte, d.h. die Objekte enthalten nur Attribute und keine Methoden.

Da Daten zwischen dem Server und dem Client ausgetauscht werden, muß ein geeignetes Protokoll gefunden werden. Als weitere Schwierigkeit kommt hinzu, daß der Server in C, der Client in Java geschrieben und beide auf verschiedenen Plattformen laufen, so daß ein direkter Austausch nicht möglich ist. Es ergeben sich damit 2 Problemfelder, der Austausch von Mitteilungen und der transparente Objektaustausch.

4.2.1. Objektaustausch

Es gibt mehrere Möglichkeiten die Objekte über das Netz zu transportieren, dabei ergibt sich das Problem der Datenkonvertierung. Jede Plattform und Programmiersprache hat seine eigene Darstellung von Datentypen, so daß ein direkter Austausch nicht möglich ist (Beispiel: big/little endian Codierung von Integer, Zeichen in Unicode oder Byte, usw.). Deshalb muß entweder ein Zwischenformat gefunden werden, daß von allen Seiten verstanden wird oder man setzt die Daten am Server um, und versendet gleich Javaobjekte.

Neben der Frage wie die Daten ausgetauscht werden, muß auch betrachtet werden, was ausgetauscht wird. Es gibt verschiedene Daten, die verschiedene Komplexitäten von dem Protokoll verlangen. Diese wären in ansteigender Komplexität:

1. primitive Datentypen (Integer, Real, Char)
2. zusammengesetzte Objekte (Objekte die aus primitive Datentypen bestehen)
3. komplexe Datenstrukturen (Liste von Objekten, Arrays, Hashtable)

Aus dem Projekt ergeben sich noch folgende Randbedingungen:

- Server in C++, Plattform Sun oder HP

- Client in Java, Plattform Java Virtual Machine
- effektiver Austausch von Daten über Slow-Links (Modemverbindungen)

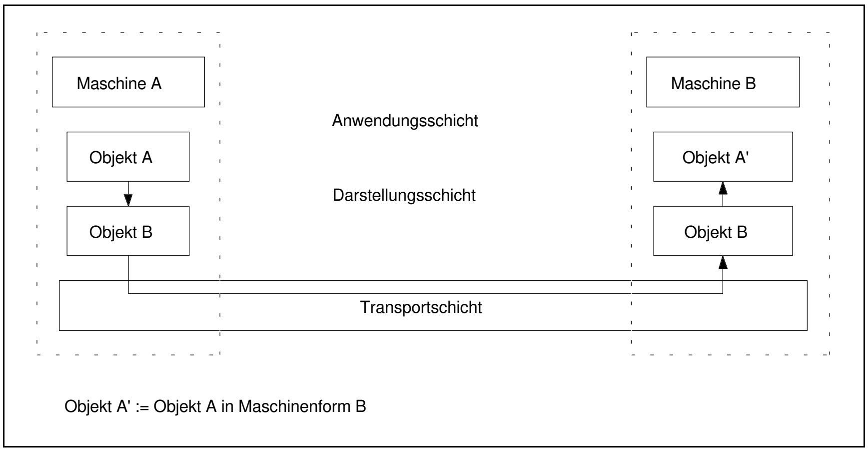


Abbildung 20 Objektübermittlung

4.2.1.1. ICE-T

ICE-T steht für Interprise Computing Environment Toolkit ([ICE 96]).

ICE-T ist ein Entwicklungsumgebung von Sun um bestehende Client/Server Programme nach Java portieren zu können. ICE-T wurde von Siemens als Entwicklungsumgebung vorgeschlagen, da schon ein ähnliches Projekt damit realisiert wurde, und somit auf die Erfahrungen zurückgegriffen werden kann.

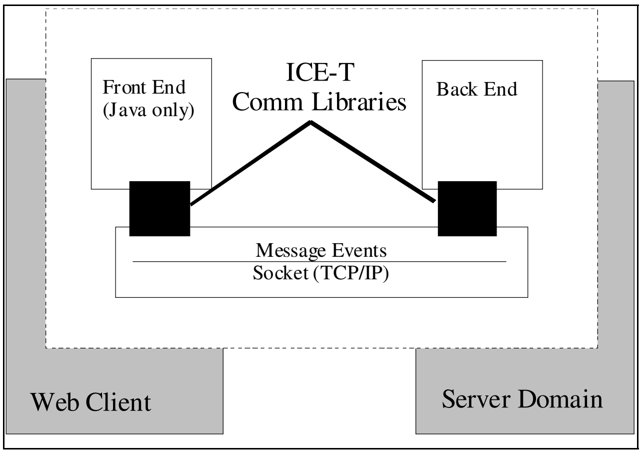


Abbildung 21 ICE-T

ICE-T geht von einem bestehenden C/C++ Server aus, und einem Client der nach Java portiert werden soll. Es stellt Funktionen zum transparenten Austausch von primitiven Datentypen und zusammengesetzten Objekten zur Verfügung. Die C++ Objekte werden dabei gleichzeitig in Java Objekte umgewandelt. Weiterhin wird die Kommunikation zwischen Client/Server komplett von ICE-T übernommen, so daß man sich ganz auf die Implementierung der Anwendungsschicht beschränken kann. Dafür stellt ICE-T weitere unterstützende Komponenten zur Verfügung, um z.B. asynchrone Nachrichten zu empfangen.

Nachteil an ICE-T ist, daß die Entwicklung komplett eingestellt wurde und nur eine Betaversion existiert. Diese ist zwar voll funktionsfähig, es fehlt aber die Unterstützung mehrere Clients zu einem Server (ein Server kann nur einen Client behandeln) und die Unterstützung von komplexen Datenstrukturen. Sun will die mit ICE-T gewonnen Erkenntnisse in ihre Produktpalette integrieren, wo und wann das geschehen soll, ist aber unklar.

4.2.1.2. Remote Methode Invocation / Object Serializer

Der Object Serializer (OS) von Java ist zwar für das Projekt vorerst uninteressant, da er Client und Server in Java voraussetzt, aber davon abgesehen gut einsetzbar wäre.

Der OS ermöglicht das Verschicken von Java-Objekten über das Netz. Es gibt keine Einschränkung in der Komplexität der Objekte. Es werden dabei vollwertige Objekte mit Methoden und Attributen übertragen. Jede Klasse die versendet werden soll, muß dabei den Object Serializer implementieren. Dies beschränkt sich auf die Implementierung zweier Methoden zum Senden und Empfangen der Klasse. Alle in JDK bereitgestellten Datenstrukturen haben dieses Interface implementiert.

Zusammen mit Remote Methode Invokation (RMI, Aufruf von Methoden in verteilten Objekten), stellt er eine gute Basis zur Programmierung von verteilten Anwendungen. OS und RMI sind voll im Java-Development-Kit 1.1.1 integriert, somit ist die Weiterentwicklung gesichert.

RMI ist vergleichbar mit RPC (Remote Procedure Call), nur daß es komplett auf Java ausgerichtet ist, und nicht interoperabel mit anderen Plattformen ist. Es bietet die Möglichkeit von einem Client Methoden in Objekten auf einem Server aufzurufen. Auf der Clientseite müssen sogenannte Stubs implementiert sein. Diese Stubs bieten eine transparente Sicht auf die verteilten Objekte. Für den Client sehen die Aufrufe so aus als wären sie lokal. Weiter besteht RMI aus einem Server, der ankommende Anfragen verwaltet und an die entsprechenden Objekte weiterleitet. Auf dem Server sind nun die Objekte tatsächlich implementiert. Aus diesen Objekten werden zur Compilierungszeit die Stubs für den Client erzeugt.

RMI/OS basieren auf der Socketimplementierung (TCP/IP Sockets) von Java, können aber auch auf HTTP als Transportprotokoll zurückgreifen, was es ermöglicht RMI/OS auch (eingeschränkt) über Firewalls/Proxies zu benutzen.

Weiterhin wurde der Garbage Collector für verteilte Objekte erweitert, so daß dieser auch feststellen kann ob Objekte auf dem Server noch vom Client referenziert werden und umgekehrt.

4.2.1.3. Eigenes Protokoll

Eine weitere Möglichkeit wäre die Implementierung eines eigenen Protokollstacks für die Anwendung. Dies hätte den Vorteil, daß man auf externe Pakete nicht angewiesen ist und man das Protokoll gut auf die Anwendung anpassen kann.

Zuerst muß man sich Gedanken machen was übertragen wird. Grundsätzlich gibt es 3 Nachrichten-typen:

1. Anfragen nach bestimmten Objekten (wie Submaps) und Aktionen
2. Antworten mit den angefragten Objekten
3. Mitteilungen über Statusänderungen

Anfragen und Antworten sind immer gekoppelt, während Mitteilungen asynchrone Nachrichten sind, die zum Beispiel durch Traps vom INMS Server erzeugt werden.

Ein typischer Ereignispfad wäre:

Neben Statusmeldungen (wie Verbindungsaufbau erfolgreich), werden auch Objekte übertragen. Da es einen Unterschied zwischen den Server- und Clientobjekten gibt, müssen diese in ein unabhängiges Format umgewandelt werden. Grundsätzlich werden alle Daten in ASCII übertragen und nicht in Binärform. Da die Objektstruktur Client und Server bekannt sind, können die Datentypen der einzelnen Objekte eindeutig durch die Attributnamen bestimmt werden. Dies führt zu einer weiteren Vereinfachung des Protokolls. Weiterhin müssen die Objekte von dem INMS-Datenbankformat in das Applikationsobjektmodell umgewandelt werden. Das Objektmodell ist unabhängig von INMS, deshalb ist die beste Stelle zur Umsetzung der Server, da bei Datenbankänderungen nur der Server geändert werden muß.

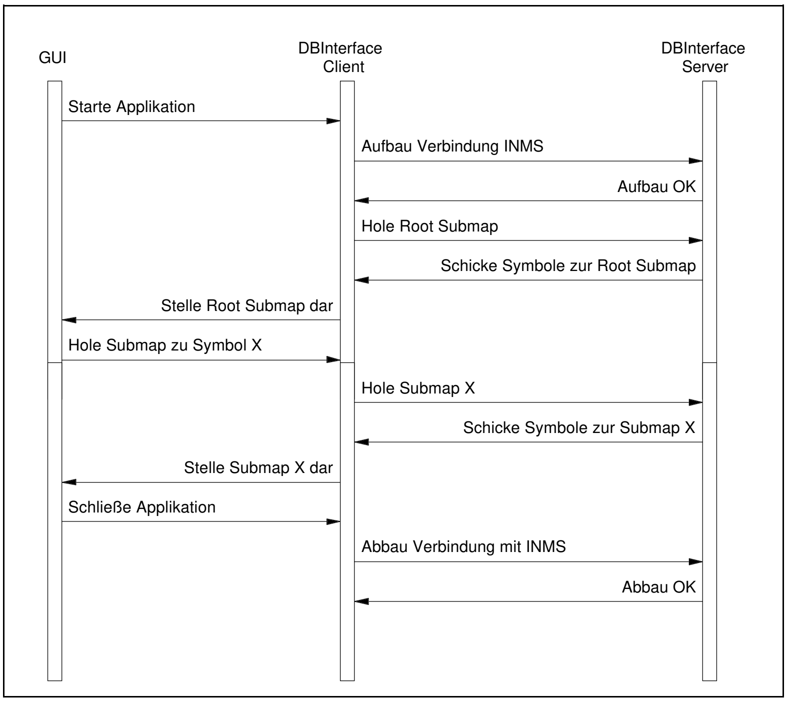


Abbildung 22 Obejktaustauch

4.2.1.3.1. Protokollbeschreibung

Die allgemeine Beschreibung in BNF Notation:

<Nachricht>::=	<Start Tag>\n[<Body>]<Ende Tag>\n
<Kurznachricht>::=	<Start Tag><Ende Tag>\n
<StartTag>::=	Start <Typ>
<Ende Tag>::=	End
<Typ>::=	<Anfrage>|<Antwort>|<Mitteilung>
<Body>::=	<Nachricht>|<Objekt>
<Objekt>::=	{<Zeile>}
<Zeile>::=	<Attributname>=<Attributwert>\n
<Attributname>::=	<Attribut>{.<Attribut>}
<String>::=	<Zeichen>[<Zeichen>]

<Attribut>|<Attributwert>|<Parameter>|<username>|<password>|<submapid>|<symbolid>|

<Fehlermeldung>::= <String>

<Zeichen> ist ein Zeichen aus dem ASCII Alphabet, mit folgenden Umsetzungen:

\n→\\n .→\.
\→\\ =→\=

Es gibt folgende Arten von Nachrichten (in `Courier` ist die <[Kurz]Nachricht> gesetzt):

<Anfragen>	***<Antworten>***

Aufbau einer Verbindung zu dem INMS Server
`start open`
`username=<username>`
`password=<password>`
`end`
	Positive Quittung
	`start ok end`
	Fehlermeldung
	`start error`
	`error=<Fehlermeldung>`
	`end`

Abbau einer Verbindung zu dem INMS Server `start close end`	

<table>
<tr><th><Anfragen></th><th><Antworten></th></tr>
<tr><td>Holen einer Submap

start getsubmap
submapid=<submapid>
end</td><td>Liste mit den Symbolobjekten

<Symbolliste>::=
start symbollist
submapid=<submapid>
[<Symbolobjekt>|<Symbolliste>]
end</td></tr>
<tr><td>Holen von Statusinformationen zu einer Submap

start getstatusofsubmap
submapid=<submapid>
end</td><td>Liste mit den Statusinformationen

start statuslist
[<Statusobjekt>]
end</td></tr>
<tr><td>Holen eines Symbols

start getsymbol
symbolid=<symbolid>
end</td><td>Symbolobjekt

<Symbolobjekt>::=
start symbol
<Object>
end</td></tr>
<tr><td>Holen des Status eines Symbols

start getstatusofsymbol
symbolid=<symbolid>
end</td><td>Statusinformation eines Symbols

<Statusobjekt>::=
start status
<Object>
end</td></tr>
</table>

<table>
<tr><th><Mitteilung></th></tr>
<tr><td>Event

start event
event=<Event>
end</td></tr>
</table>

4.2.2. Sicherheit

Da es sich um eine Client/Server Applikation handelt, müssen Daten zwischen Client und Server ausgetauscht werden. Es gibt nun 3 Punkte an denen die Sicherheit berücksichtigt werden kann:

1. Server

   Auf dem Server sind meist sensible Daten abgelegt, die vor fremden Zugriff geschützt werden sollen. Deshalb sollten nur authentifizierte Benutzer zugreifen können. Die Authentifizierung geschieht bei INMS über ein einfaches Login/Paßwort Verfahren. Jeder Operator bekommt einen Namen und Paßwort zugewiesen, mit dem er sich vor der INMS Datenbank authentifiziert.

2. Medium

   Der Datenaustausch kann über ein unsicheres Medium (wie z.B. dem Internet) geschehen, deshalb ist es nötig die Daten, die übertragen werden, zu schützen. Es gibt verschiedene Verfahren, die einen sicheren Datenaustausch gewähren. Diese Verfahren setzen auf verschiedenen Ebenen an. Diese Verfahren beinhalten teilweise schon Client- und Serverseitige Authentifizierung. Beispiele für Protokolle: IPv6, SSL, PGP

3. Client

   Der Client muß auch von fremden Servern geschützt werden, die ihm gefälschte Daten liefern um zum Beispiel Paßwörter für den Server zu erhalten. Dies ist vor allem in Bereichen von autonom handelnden Systemen interessant, wie bei mobilen Agenten. Dieser Punkt wird in dem Projekt nicht betrachten.

Die sichere Übertragung von Daten sollte möglichst leicht in das System integrierbar sein, dies ist aber abhängig von dem verwendet Protokoll zum Austausch von Objekten.

1. ICE-T

   Verschlüsselung war geplant, wurde aber nicht implementiert.

2. RMI/OS

   Verschlüsselung ist in RMI/OS nicht implementiert. Sicherheit wird nur innerhalb des Java-Sicherheitsmodells betrachtet. Das bedeutet hauptsächlich den Schutz des Clients vor „bösen“ Javaklassen. (siehe Java Secruity Manager [VSS 95]). Trotzdem ist es einfach möglich Verschlüsselungsmethoden einzuführen, indem eigene Streamklassen implementiert werden, die die Orginalmethoden überschreiben (siehe RMI Doc). Die Streamklassen sind Filterklassen, die einen formatierten Datenfluß über beliebige I/O Routinen liefern. Damit ist man frei in der Wahl eines kryptographischen Algorithmusses.

3. Eigenes Protokoll

   Wie bei RMI müssen die entsprechenden Verschlüsselungsklassen selbst geschrieben werden, da Java in den Standardklassen keine bietet. Die Implementierung erfolgt analog zu RMI/OS, nur mit doppelte soviel Aufwand, da die Klassen auch in C++ auf der Serverseite geschrieben werden müssen.

4.2.3. Schlußfolgerung

ICE-T wurde als Implementierung der Schnittstelle im Prototypen nicht verwendet. Es hätte zwar eine schnelle Entwicklung der Serverseite gebracht, aber die bereitgestellten Klassen auf der Clientseite sind so aufgebaut, daß man eine Applikation auf ICE-T aufbaut und nicht ICE-T in eine Applikation integriert. Dies hätte neben einer Verzögerung bei der Programmierung des Client auch bei einem späterer Wechsel auf ein anderes Transportprotokoll zu Schwierigkeiten geführt.

Da der Object Serializer und RMI für C++ oder eine Java-Schnittstelle zur Datenbank noch nicht verfügbar sind, wird vorerst auf das eigene Protokoll gesetzt. Ziel sollte es aber sein, die Serverseite durch einen Javaserver zu ersetzen und RMI/OS einzusetzen. Durch die Möglichkeit Native Calls (Aufrufe von Methoden außerhalb von Java) in Java einzubinden, wäre sogar ein Mischserver aus Java und C++ möglich. Die Vorteile gegenüber anderen Lösungen wären:

1. Leichtere Wartbarkeit des Systems, da größte Teile in einer Programmiersprache sind.
2. Eine gesicherte Weiterentwicklung der Entwicklungsumgebung.
3. Transparenter Austausch von komplexen Datenstrukturen.
4. Leichte Integrierbarkeit von Verschlüsselungstechniken

4.3. StatusInterface

Das StatusInterface dient zum Empfangen von Traps und Aktualisieren des Status eines Symbols. Die Traps werden von dem INMS Server als SNMP Traps geliefert, so daß auf jeden Fall ein SNMP Proxy und Interface implementiert werden muß. Auch hier gilt analog zum DBInterface, daß die Daten in das VOMI Objektmodell angepaßt werden müssen.

Eine Möglichkeit ist die Verwendung des obigen Protokolls zu der Übermittlung der Daten. Davon wird aber abgesehen, da es schon gute SNMP-Interface Implementierungen (Java Management API [JMAPI 96], Advent SNMP Package [ADV 97]) gibt. Somit muß auf der Serverseite ein SNMP Proxy installiert werden, der SNMP Traps vom Server empfängt und an die VOMI Clients weiterleitet.

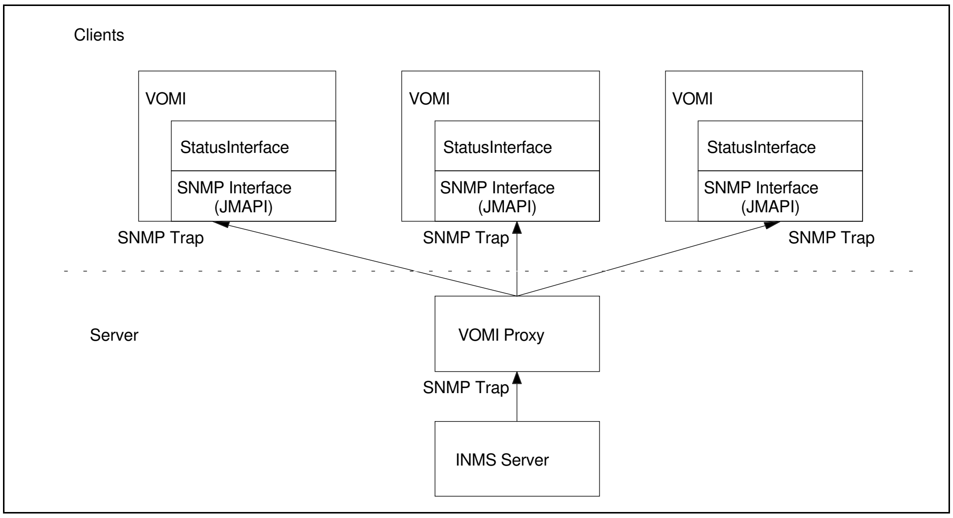


Abbildung 23 StatusServer

5. Objektentwurf

Im Objektentwurf werden nun alle Modelle zusammengeführt und eine detaillierte Beschreibung für die Implementierung erstellt. Hier werden die letzten Entscheidungen zur Realisierung des Systems festgelegt. Hier werden als letztes Änderungen an den Klassen, Methoden und Attributen durchgeführt oder neue interne Klassen hinzugefügt.

5.1. Java

Java ([Java 95]) ist eine relativ junge Programmiersprache, die bei Sun zur Steuerung intelligenter Haushaltsgeräte entwickelt wurde. Die Designphilosophie war eine plattformunabhängige, einfache, objektorientierte und robuste Sprache zu entwickeln. Als Basis diente C++. Java läuft auf einer virtuellen Maschine und stellt somit ein System in einem System dar. Durch diese Kapselung erreicht man ein hohes Sicherheitsniveau, da es den Programmen nicht möglich ist direkt auf das unter der virtuellen Maschine liegende Betriebssystem direkt zugreifen zu können. Mit dem Aufkommen des World Wide Webs (WWW) erkannte man, daß sich Java, bedingt durch die Heterogenität des Internets, hervorragend zur Verteilung von Programmen eignen würde. Damit begann eine rasante Entwicklung. Der Sprung von JDK (Java Development Kit) Version 1.0.2 auf 1.1.1 brachte eine Vielzahl Neuerungen, wie RMI, Native Calls, neues AWT (Abstract Windowing Toolkit, Schnittstelle zur Benutzungsoberfläche des Systems). Zum Zeitpunkt der Diplomarbeit unterstützte noch kein Webbrowser JDK 1.1.1 voll, deshalb wurde entschieden den Prototypen erst in JDK 1.0.2 zu entwickeln. JDK liegt nun in der Version 1.1.3 vor.

5.1.1. JMAPI

JMAPI (Java Management API, [JMAPI 96]) stellt eine Klassensammlung zur Entwicklung von Netz-management-Applikationen dar. Es bietet neben Klassen zur Verwaltung von Managed Objects und neuen GUI-Elementen auch eine SNMP Schnittstelle. JMAPI liegt zum Zeitpunkt der Diplomarbeit nur für JDK 1.0.2 vor, die Version für JDK 1.1.3 ist noch in der Entwicklungsphase. Für den Prototypen werden nur die folgenden erweiterten GUI Klassen des JMAPI benutzt:

1. ImageScroller

   Der ImageScroller stellt ein Image in einem Fenster dar, und übernimmt neben der GUI-Eventverwaltung, die Verwaltung der Scrollbars, sobald das darzustellende Image größer als das Fenster ist. Wird in dem SubmapDisplay (5.2.4.1.) verwendet.

2. ImageButton

Ermöglicht die Verwendung von Images als Button. Wird in dem SubmapDisplay (5.2.4.1.) und SymbolDisplay (5.2.4.3.) verwendet.

3. Browser

   Stellt eine hierarchische Liste linear dar, vergleichbar mit Dateiauswahlboxen. Wird für den SubmapBrowser (5.2.8.) benötigt.

5.1.2. Hashtable

Die Hashtable ist eine Listendatenstruktur in Java, die Indizierung von Objekten in der Liste über beliebige andere Objekte erlaubt. Weiterhin stellt sie eine optimierte Suchmethode (Hashing Verfahren) zur Verfügung um schnell auf den Inhalt zugreifen zu können.

Die Hashtable speichert Objekte als Schlüssel/Wert Paare. Der Wert ist dabei das Objekt das gespeichert wird und der Schlüssel ist eine eindeutige Identifikation dieses Objekts. Als Standardschlüsselobjekt wird ein String angenommen. Aus diesem wird der Hashwert jedes Objektes berechnet. Der Hashwert dient der Hashtable zur effizienten Suche in der Liste der Objekte. Formal ist die Hashfunktion als h:K→A definiert. K = Menge der Schlüssel, A = Menge der Adressen, an denen die Zielobjekte liegen. Ziel ist es für h eine möglichst einfache Funktion zu finden.

Beispiel:

K = { traktor, fahrrad, auto, kfz } mit folgender Zuweisung:

traktor → (Traktor Objekt)
auto → (Auto Objekt)
fahrrad → (Fahrrad Objekt)
kfz → (Kfz Objekt)

A = {0,1,2,3}

f:{a,b,c,...,z}→{1,2,3,...,26}

$h(k)$ = f(1.Buchstabe von k) mod 4

k (Schüssel)	traktor	fahrrad	auto	kfz
h(k) (Hashwert und Position in der Liste)	0	2	1	3

Daraus ergibt sich folgende geordnete Liste:

Adresse	0	1	2	3
Inhalt(Zielobjekte)	Traktor Objekt	Auto Objekt	Fahrrad Objekt	Kfz Objekt

Der hauptsächliche Verwendungszweck in VOMI ist aber die einfache Realisierung von assoziativen Felder durch die Hashtable. Die Vaterklasse Dictionary von der die Hashtable abgeleitet ist, wäre zwar besser geeignet, aber sie ist eine abstrakte Klasse und somit nicht implementiert.

In der Klassenbeschreibung wird hinter einem Hashtable Attribut noch angegeben welchen Typ der Schlüssel und das gespeicherte Objekt haben. Beispiel:

```
public Hashtable <name>;     //<key type>,<value type>
```

5.2. Klassenbeschreibung

5.2.1. Konventionen

- Klassennamen werden groß geschrieben.

  ```
  (class Symbol,class Node,class Link)
  ```

- Attribute werden klein geschrieben.

  `(int zähler, float koordinate, String name)`

- Methoden werden klein geschrieben.

  `(int erhöhe(int variable), void zeichne(Symbol objekt))`

- Klassennamen oder Objektnamen die in Attributen oder Methoden vorkommen, werden groß geschrieben

  `(void zeichneSymbol(Symbol objekt), Liste listeSymbol)`

- Jede Klasse hat einen Konstruktor, der das Objekt initialisiert
- Der Zugriff von anderen Objekten auf Attribute geht über `get<Attributname>()` und `set<Attributname>()`. Diese Methoden werden nur bei Bedarf implementiert und sind hier nur beschrieben, falls sie nicht trivial sind.
- Die Notation erfolgt in Java Syntax und entspricht der Notation aus „Java in a Nutshell" ([FL 96])

5.2.2. Klassen

Es folgt eine Klassenbeschreibung, aller Klassen die notwendig zur Implementierung der Applikation sind. Die Methoden und Attribute ergeben sich aus den vorangegangen Kapiteln, hauptsächlich aber aus der Analyse.

5.2.2.1. Symbol

Das Symbol ist die Vaterklasse von Node und Link. Es enthält Informationen über die Struktur der Submap und die Attribute, die Node und Link gemeinsam haben. Ein Symbol kann zugleich eine Submap sein.

Einige Funktionen werden hier schon definiert, aber erst in den Subklassen implementiert. Dies soll ein einheitliche Schnittstelle zu allen Symbolen gewähren. Weiterhin bedeutet es, daß von der Klasse `Symbol` kein Objekt existieren kann.

Es ist vorgesehen, daß jedes Symbol selbst verantwortlich für die Repräsentation auf der Benutzungsoberfläche ist. Da Nodes aber Bitmaps sind (die aus GIF Dateien gelesen werden) muß ein ImageObserver vorhanden sein. Der ImageObserver überwacht den Transfer von Images in andere graphische Komponenten (wie Fenster). Dieser muß aber von einem beteiligten Objekt bereitgestellt werden. Durch die Vererbungshierarchie von Java, kann dies aber nur das SubmapDisplay.

```
public class Symbol extends Object {
// Attribute
        public String id;
        public Type type;
        public Hashtable information;         //String,String
        public State state;
        public Hashtable children;            //String,Symbol
        public Hashtable parents;             //String,Symbol
        public boolean selected;
        public String layoutid;
        public boolean hasSubmap;
// Methoden
        public Symbol();
        public void addChild(Symbol child);
        public void removeChild(Symbol child);
        public void addParent(Symbol parent);
        public Symbol findSymbolat(Point p,float zoom);
        public Symbol searchSymbol(String id);
        public abstract float isinside(Point p,float zoom);
        public abstract void paint(Component watcher,Graphics graphics,
                float zoom);
        public int hashCode();
        public boolean equals(Object obj);
}
```

5.2.2.1.1. Attribute

- `public String id`

  Jedes Symbol ist eindeutig identifizierbar durch seine `id`.

- `public Type type`

  Gibt den Typ des Objekts an, analog zu den von INMS vergebenen Typen.

- `public Hashtable information`

  Hier werden zusätzliche Informationen zu einem Symbol gespeichert, wie z.B. Hersteller einer physikalischen Node. Diese Informationen werden in dem `SymbolDisplay` dargestellt. Als Key wird die Bezeichnung der Information benutzt, und die Information ist der Value.

- `public State state`

  Jedes Symbol kann auf der Benutzungsoberfläche einen von mehreren Zuständen annehmen. Der Grundzustand ist *unknown,* da die INMS Datenbank die Objekte zustandslos abspeichert. Erst nach dem Datenaustausch mit dem INMS Server kann der tatsächliche Zustand bestimmt werden. Der Zustand kann sich während der Laufzeit der Applikation ändern.

- `public Hashtable children`

  Falls es sich bei dem Symbol um eine Submap handelt, werden die Kindobjekte der unter dem Symbol liegenden Submap hier abgespeichert.
  Keys sind die Id's der Objekte, Values sind die Objekte an sich.

- `public Hashtable parents`

  Jedes Symbol ist Teil einer Submap. Jede Submap ist einem Symbol der darüberliegenden Submap zugeordnet. Somit hat jedes Symbol mindestens einen Vater. Eine Ausnahme bildet das Rootsymbol, daß keine weiteren Väter hat. Die Einschränkung auf ein Vaterobjekt wurde bewußt nicht gewählt, da ein Objekt auch Teil mehrere Submaps sein kann (z.B. Router die zwei Netze verbinden und in zwei Submaps vorhanden sind).
  Keys sind die Id's der Objekte, Values sind die Objekte an sich.

- `public boolean selected`

  Variable die kennzeichnet ob ein Symbol auf der GUI markiert wurde.

- `public String layoutid`

  Identifier des Algorithmus, der für die Anordnung der Symbole zuständig ist.

- `public boolean hasSubmap`

  Attribut zum Feststellen, ob unter dem Symbol eine Submap liegt. Aus der Anzahl der Kinder kann nicht bestimmt werden, ob das Symbol eine Submap besitzt, da bei dem Aufbau einer Submap nicht die weiter unten liegenden Submaps vom Server geholt werden.

5.2.2.1.2. Methoden

- `public int hashCode()`

  Liefert den Hashcode des Objekts zurück, dies ist notwendig für die Suchfunktion der Hashtable.

- `public boolean equals(Object obj)`

  Liefert TRUE falls das übergebene Objekt das Eigene ist. Dies ist notwendig für die Suchfunktion der Hashtable.

- `public Symbol findSymbolat(Point p,float zoom)`

  Findet das Symbol das sich unter dem Mauszeiger befindet. Die Funktion greift auf `isinside()` seiner Kinder zurück. Weiterhin wird der Vergrößerungsfaktor übergeben, da Nodes je nach Vergrößerungsfaktor des SubmapDisplay's verschiedene Größen haben können.

- `public Symbol searchSymbol(String id)`

  Sucht rekursiv nach einem Symbol in einer Map.

- `public abstract float isinside(Point p, float zoom)`

  Liefert den Abstand des Punktes `p` zu dem Symbol.

- `public abstract void paint(Component dest, Graphics graphics, float zoom)`

  Zeichnet das Symbol in das graphische Objekt `graphics` mit dem Vergrößerungsfaktor `zoom`. Dabei wird die übergebende Componente zur Überwachung des Vorgangs eingesetzt. (siehe Java ImageObserver Funktionsweise).

5.2.2.2. Node

```
public class Node extends Symbol {
// Attribute
        public Hashtable links;            //String,Link
        public Point coor;
        public ImageCenter imagecenter;
        public Dimension symbolsize;
// Methoden
        public Node();
        public float isinside(Point p, float zoom);
        public void paint(Component watcher, Graphics graphics,
                float zoom);
        public void insertLink(Link link);
}
```

5.2.2.2.1. Attribute

- `public Hashtable links`

Jede Node kann über mehrere Links verfügen.
Keys sind die Id's der Objekte, Values sind die Linkobjekte.

- `public Point coor`

  Die Position der Node am Bildschirm, dabei wird als Referenzpunkt die Mitte des graphischen Symbols genommen.

- `public ImageCenter imagecenter`

  Der ImageCenter verwaltet alle Bitmaps die dargestellt werden. Dies soll eine Beschleunigung der Bildgenerierung bieten.

- `public Dimension symbolsize`

  Die Größe des graphischen Symbols. Dieses Attribut wird für die Funktion `isinside()` benutzt, um festzustellen, ob ein Punkt innerhalb des Objekts liegt. Die Größe ist zwar auch im Image gespeichert, auf diese kann aber nicht direkt zugegriffen werden, da dieser Klasse der ImageObserver fehlt.

5.2.2.2.2. Methoden

- `public float isinside(Point p, float zoom)`
  `public void paint(Component watcher, Graphics graphics, float zoom)`

  Gleiche Funktionalität wie in Symbol besprochen, die Methoden werden hier implementiert.

5.2.2.3. Link

```
public class Link extends Symbol {
// Attribute
         public Hashtable nodes;               //String,Node
// Methoden
         public Link();
         public float isinside(Point p, float zoom);
         public void paint(Component watcher, Graphics graphics,
                   float zoom);
         public Enumeration getnodes();
         public Enumeration getnodeobjs();
         public void insertNode(Node node);
         public void inserttmpNode(String nodeid);
}
```

5.2.2.3.1. Attribute

- `public Hashtable nodes`

  Jeder Link ist mit 2 Nodes assoziiert.

5.2.2.3.2. Methoden

- `public float isinside(Point p, float zoom)`
  `public void paint(Component watcher, Graphics graphics, float zoom)`

  Gleiche Funktionalität wie in Symbol besprochen, die Methoden werden hier implementiert. Es muß vor dem Zeichnen sichergestellt werden, daß der Link auch zwei Nodes in der gleichen Submap verbindet (dies ist in der INMS Datenbank nicht immer gewährleistet).

- `public Enumeration getnodes()`

Liefert eine Liste mit den Id's der Nodes zurück, die mit einem Link verbunden sind.

- `public Enumeration getnodeobjs()`

  Liefert eine Liste mit den Node-Objekten zurück, die mit einem Link verbunden sind.

- `public void insertNode(Node node)`

  Fügt einen Node in die Nodeliste des Links ein.

- `public void inserttmpNode(String nodeid)`

  Fügt den Identifierstring eines Nodes und ein Dummy-Objekt in die Nodeliste ein. Diese Funktion wird für den Aufbau einer Submap benötigt, falls die Nodeobjekte noch nicht vorhanden sind.

5.2.2.4. State

Jedes Symbolobjekt hat ein Zustandsobjekt, in dem der Zustand des Symbols und die Ereignisse, die zu diesem Zustand geführt haben, enthalten sind. Unterschieden werden muß zwischen internen Zustand und dem Zustand des Symbols. Der interne Zustand ist der Zustand, wie er von dem INMS Server geliefert wird. Der Zustand des Symbols ist die Summe der Zustände der Symbole die unter diesem Symbol liegen plus der interne Zustand.

```
public class State extends Object {

// Attribute

          public Symbol symbol;
          public Integer state;
          public Integer internstate;
          public Hashtable events;              //String,String
          private Hashtable statenames;         //Integer,String

// Methoden

          public State(Symbol symbol);
          public Enumeration getevents();
          public void insertevent(String event);
          public void setinternstate(Integer nr);
          public Integer getstate();
          public Hashtable getstatenames();
          public void calc_state();
          public Color getColor();
}
```

5.2.2.4.1. Attribute

- `public Symbol symbol`

  Das Symbol, das dem Zustand zugeordnet ist.

- `public Integer state`
  `public Integer internstate`

  (siehe oben)

- `public Hashtable events`

  Die Eventsliste speichert alle SNMP-Traps des entsprechenden Symbols, die von dem INMS Server an die Applikation geschickt werden. Um die gespeicherte Datenmenge zu reduzieren, ist ein geeigneter Expirealgorithmus notwendig, der bestimmte Ereignisse entfernt.

- `private Hashtable statenames`

  Die Namen der einzelnen Zustände als Strings.

5.2.2.4.2. Methoden

- `public Enumeration getevents()`

  Liefert eine Liste mit allen bisher aufgetretenen und gespeicherten Ereignissen.

- `public void insertevent(String event)`

  Fügt ein weiteres Ereignis der Liste hinzu. Dabei wird das Ereignis asgewertet, um zu überprüfen, ob sich der Zustand geändert hat. Wenn ja, wird `internstate` entspechend geändert

- `public void setinternstate(int nr)`

  Setzt den internen Zustand. Dies kann eine Welle von Zustandsänderungen in übergeordneten Submaps zur Folge haben.

- `public int getstate()`

  Liefert den Zustand des Symbols.

- `public Hashtable getstatenames()`

  Liefert die Namen aller Zustände zurück.

- `public void calc_state()`

  Diese Funktion wird von Kindern einer Submap aufgerufen, falls sich deren Zustand geändert hat.

- `public Color getColor()`

  Liefert die Farbe zurück, mit dem ein Symbol eingefärbt werden soll.

5.2.3. Interne Klassen

5.2.3.1. MapAdmin

```
public class MapAdmin extends Object {
// Attribute
         public Node root;
         public Hashtable submapsDisplays;     //String,SubmapDisplay
         public Hashtable symbolDisplays;      //String,SymbolDisplay
         public Autolayout autolayout;
         public DBInterface inmsdb;
         public StatusInterface inmsserver;
// Methoden
         public MapAdmin(String address, int port);
         public void openSubmapDisplay(Symbol submap);
         public void removeSubmapDisplay(Symbol submap);
         public void openSymbolDisplay(Symbol obj);
         public void removeSymbolDisplay(Symbol obj);
         public void openNetconfig();
         public void startAutolayout(Symbol submap);
         public void openAutolayoutConfig(Symbol submap);
         public void requestSubmap(Symbol submap);
         public void requestStateOfSubmap(Symbol submap);
         public void insertsymbol(Symbol obj, String parent);
}
```

5.2.3.1.1. Attribute

- `public Node root`

  Die Rootmap dient als Einstieg in die Submaphierarchie, die einem Operator zugeordnet ist. Die Applikation kann nur ein Rootobjekt besitzen. Das Rootobjekt ist nicht das gleiche wie das von

INMS, da in INMS mehrere Rootdomainen pro Operator existieren können. Das Rootmap-Symbol kann nie angezeigt werden.

- `public Hashtable submapsDisplays`

  Die Liste mit allen SubmapDisplays die sich momentan auf der Benutzungsoberfläche befinden.

- `public Hashtable symbolDisplays`

  Die Liste mit allen SymbolDisplays die sich momentan auf der Benutzungsoberfläche befinden.

- `public AutoLayout autolayout`

  Verweis auf das Objekt zur Verwaltung der Autolayout-Algorithmen.

- `public DBInterface inmsdb`

  Verweis auf das Objekt zur Verwaltung der INMS Datenbankverbindung.

- `public StatusInterface inmsserver`

  Verweis auf das Objekt zur Verwaltung der INMS Datenbankverbindung.

5.2.3.1.2. Methoden

- `public MapAdmin(String dbaddress, int dbport, String saddress, int sport)`

  Erzeugt einen neuen MapAdmin. Das Paar `dbaddress` und `dbport` bezeichnen die Adresse des VOMI Servers der für die Datenbankzugriffe und `saddress, sport` die Adresse des Servers, der für den StatusServer zuständig ist.

- `public void openSubmapDisplay(Symbol submap)`

  Die Funktion öffnet für die Submap des übergebenen Symbols ein neues Display. Falls das entsprechende Display schon existiert, wird dieses in den Vordergrund gebracht.

- `public void removeSubmapDisplay(Symbol submap)`

  Es wird ein SubmapDisplay aus der Liste `submapsDisplays` entfernt, weil es auf der graphischen Benutzungsoberfläche geschlossen wurden.

- `public void openSymbolDisplay(Symbol obj)`

  Die Funktion öffnet für das übergebene Symbol ein neues Display. Falls das entsprechende Display schon existiert, wird dieses in den Vordergrund gebracht.

- `public void removeSymbolDisplay(Symbol obj)`

  Entfernt das zum dem Symbol gehörende SymbolDisplay aus der Liste `symbolDisplays`.

- `public void openNetconfig()`

  Hier wird die Netconfig-Dialogbox geöffnet. Sie steuert die Verbindung zu der INMS Datenbank und ist für die Eingabe der Operatordaten (Name, Paßwort, Server). Von dieser Funktion kann eine neue Verbindung unter einem neuen Operator aufgebaut werden. Da aber nur ein Operator existieren kann, wird hier bei einem neuen Einlogversuch unter einen anderem Operator alle Objekte der alten Rootmap entfernt.

- `public void startAutolayout(Symbol submap)`

  Es wird der Autolayoutalgorithmus für die übergebene Submap aufgerufen.

- `public void openAutolayoutConfig(Symbol submap)`

  Statt des Autolayoutalgorithmus wird die Verwaltungsdialogbox für die Algorithmen aufgerufen. Neben der Konfiguration wird eine Submap übergeben, auf die der Algorithmus gleich angewendet werden kann. Die Abfolge ist bei einer neuen Submap, daß zuerst `startAutolayout()` angewandt wird und diese Funktion erst auf Benutzeranfrage gestartet wird.

- `public void requestSubmap(Symbol submap)`

  Holt eine Submap, falls diese noch nicht vorhanden ist, oder nochmal geholt werden soll. Diese Methode wird von `openSubmapDisplay()` aufgerufen, das einen Anfrage für eine Submap bekommen, aber diese von dem DBInterface noch nicht erhalten hat.

- `public void requestStateOfSubmap(Symbol submap)`

  Holt den Status einer kompletten Submap.

- `public void insertsymbol(Symbol symbol, String parent)`

  Fügt ein weiteres Symbol in die durch `parent` gekennzeichnete Submap ein.

5.2.3.2. Autolayout

Die Autolayoutklasse besteht tatsächlich aus mehreren Klassen, die aber nach außen hin sich wie eine Klasse sich darstellen. Jeder Algorithmus zu der Anordnung von Symbolen auf der Benutzungsoberfläche, wird in einer eigenen Klasse modelliert. Angesprochen werden die einzelnen Klassen über eine Wrapperklasse die neben einer einheitlichen Programmierschnittstelle auch die Oberflächenelemente liefert.

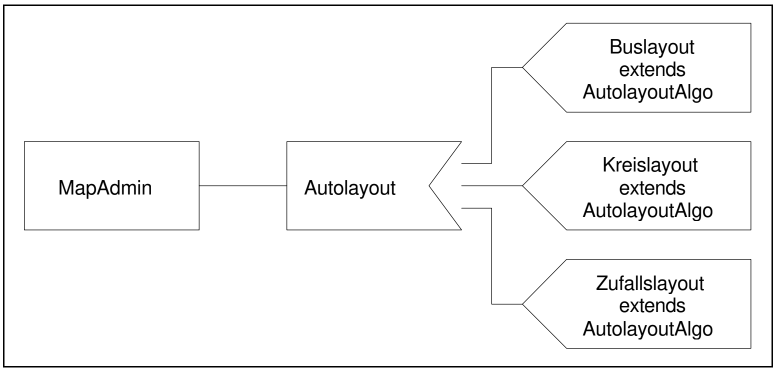


Abbildung 24 Autolayoutklassen

```
public class Autolayout extends Frame {
// Attribute
         public Hashtable algorithmen;         //String,AutolayoutAlgo
// Methoden
         public Autolayout();
         public void startAutolayout(Symbol submap);
         public void openAutolayoutConfig(Symbol submap);
}
```

5.2.3.2.1. Attribute

- `public Hashtable algorithmen`

  Liste mit allen vorhanden Algorithmen-Objekte.

5.2.3.2.2. Methoden

- `public Autolayout()`

Generiert ein neues Wrapperobjekt für die Layoutalgorithmen. Hier wird auf dem Server nach einer Datei mit einer Liste von zusätzlichen Layoutalgorithmen gesucht und geladen. In dieser Liste stehen die Klassennamen von den Algorithmen. Diese Klasse lädt dann die Klassen nach und erzeugt die entsprechenden Objekte.

- `public void startAutolayout(Symbol submap)`

  Startet den Autolayoutvorgang für eine bestimmte Submap. Der für diese Submap zuständige Algorithmus wird aus dem Submapobjekt ausgelesen und aufgerufen.

- `public void openAutolayoutConfig(Symbol submap)`

  Stellt die Dialogbox mit der Auswahl von Algorithmen dar, und ruft nach Beendigung durch den Benutzer den entsprechend ausgewählten Algorithmus auf.

5.2.3.3. AutolayoutAlgo

Alle Algorithmen sind von dieser Klasse abgeleitet, und müssen alle Methoden implementieren.

```
public abstract class AutolayoutAlgo extends Object {
// Attribute
        public String Id;
// Methoden
        public AutolayoutAlgo();
        public abstract void startAutolayout(Symbol submap);
        public abstract void openAutolayoutConfig(Symbol submap);
        public abstract Panel configure();
}
```

5.2.3.3.1. Methoden

- `Public abstract Panel configure()`

  Liefert eine graphische Umgebung für den Konfigurationsdialog von `Autolayout` zurück.

5.2.4. GUI Interfaces

5.2.4.1. SubmapDisplay

```
public class SubmapDisplay extends Frame {
// Attribute
         public Symbol submap;
         public ImageScroller submapScroller;
         public Image submapImage;
         public Graphics submapGraphics;
         public float zoomfactor;
         public Symbol selected;
         public MapAdim mapadmin;
         public Font nodefont;
// Methoden
         public SubmapDisplay(MapAdmin mapadmin, Symbol submap);
         public void show();
         public void paintSubmap();
         public synchronized void repaintSymbol(Symbol obj);
         public void topit();
         public void addNotify();
         public boolean handleEvent(Event evt);
         public Point klickPoint(Event evt);
// Menüereignisse
         public void close();
         private void zoomin();
         private void zoomout();
         private void rootSubmap();
         private void autolayoutConfig();
         private void autolayout();
         private void enterSubmap();
         private void showSymbolDisplay();
// Mausereignisse
         private void selectSymbol(Symbol obj);
}
```

5.2.4.1.1. Attribute

- `public Symbol submap`

  Zeiger auf das Symbol dessen Submap dargestellt wird.

- `public ImageScroller submapScroller`
  `public Image submapImage`
  `public Graphics submapGraphics`

  Die 3 Attribute regeln die Darstellung einer Submap in dem Fenster. Als Hilfsklasse wird ImageScroller von JMAPI verwendet. Diese Klasse vereinfacht die Darstellung eines Images auf dem Bildschirm, indem Sie sich komplett um die GUI Verwaltung (z.B. Positionierung und Größe der Scrollbars, falls das Image größer ist als der dargestellte Bereich) kümmert. `SubmapImage` ist das Image das dargestellt wird. Die Benutzung eines Offscreen-Images ([MSS 96]) hat den Vorteil, daß bei GUI Aktionen, wie Scrollen des Inhalts, der dargestellte Bereich nicht neu aufgebaut wird. Dies verhindert z.B. das Flackern von Objekten, was die Akzeptanz der Applikation erhöht. Nachteil ist der Speicherbedarf, durch die immer vorhandene komplette Submap und das etwas schwierig zu handhabende Neuzeichnen einzelner Objekte, falls sich die Objekte überlappen.

- `public float zoomfactor`

  Gibt den Vergrößerungsfaktor einer Submap an. Die Standardgröße ist 1.0 .

- `public Symbol selected`

Das momentan selektierte Symbol.

- `public MapAdim mapadmin`

  Verweis auf den MapAdmin, damit weitere Fenster geöffnet werden können.

5.2.4.1.2. Methoden

- `public SubmapDisplay(MapAdmin mapadmin, Symbol submap)`

  Der Konstruktor der Klasse erzeugt für die Submap `submap` eine neue graphische Repräsentation. Weiterhin wird der MapAdmin übergeben, der alle Fenster verwaltet. Beim Öffnen weitere Fenster geschieht die Erzeugung über den MapAdmin.

- `public void show()`

  Initialisiert das Objekt und öffnet das Fenster mit der Submap.

- `public void paintSubmap()`

  Initialisiert das Image mit der Submap und zeichnet alle Objekte.

- `public synchronized void repaintSymbol(Symbol obj)`

  Zeichnet ein Objekt neu in der Submap. Die Funktion ist als Monitor implementiert, damit nur immer eine redraw-Routine auf das Submapimage zugreifen kann. Dies kann passieren beim Neuzeichnen eines Links, da hier auch die angrenzenden Nodes neugezeichnet werden. Von anderen Quellen können aber auch `repaintSymbol` Anfragen kommen (z.B. bei Statusänderungen), deshalb sollte nur eine Anfrage zu einem Zeitpunkt erledigt werden, um graphische Unkorrektheiten zu unterdrücken.

- `public void topit()`

  Bringt das Fenster nach vorne. Der MapAdmin hat eine Anfrage zum Öffnen einer neuen Submap erhalten und festgestellt, daß diese schon existiert, deshalb wird nur das entsprechende SubmapDisplay nach vorne gebracht.

- `public void addNotify()`

  Interne Java GUI Funktion.

- `public boolean handleEvent(Event evt)`

  Die Verwaltung von Benutzerinteraktionen mit dem Programm wird hier ausgewertet, und in die entsprechenden Methoden verzweigt.

- `public Point klickPoint(Event evt)`

  Berechnet aus den Klickkoordinaten in dem Submapfenster die absoluten Koordinaten in dem Submapimage.

- `public void close()`

  Schließt das SubmapDisplay ordnungsgemäß.

- `private void zoomin()`
  `private void zoomout()`

  Routinen zum Vergrößern und Verkleinern einer Submap. Dabei muß die komplette Submap neu gezeichnet werden, und ein neues `submapImage` mit entsprechender Größe angelegt werden. Die Größe der graphischen Symbole wird entsprechend angepaßt.

- `private void rootSubmap()`

  Öffnet die Rootmap eines Operators.

- `private void autolayoutConfig()`

Ruft die Dialogbox zur Konfiguration des Autolayoutalgorithmusses auf.

- `private void autolayout()`

  Startet den Autolayoutalgorithmus erneut.

- `private void enterSubmap()`

  Öffnet das SubmapDisplay des momentan selektierten Symbols. Falls das Symbol kein Kinder hat, wird das SymbolDisplay geöffnet.

- `private void showSymbolDisplay()`

  Öffnet das SymbolDisplay des momentan selektierten Symbols.

- `private void selectSymbol(Symbol obj)`

  Selektiert ein ein Symbol. Falls ein anderes Symbol selektiert war, wird diese unselektiert dargestellt und dessen `selected` Attribut zurückgesetzt. Das Attribut `selected` dieser Klasse wird auf das neu selektierte Objekt gesetzt, und dieses in der Submap selektiert dargestellt. Anschließend wird die Submap auf der Benutzungsoberfläche neu gezeichnet.

5.2.4.2. NetconfigDisplay

Das NetconfigDisplay ist die Eingabemaske für die Operatordaten, die zum Aufbau einer Verbindung notwendig sind. Das NetconfigDisplay hat insofern eine Sonderstellung, da es eine neue Verbindung mit dem Server initiieren kann und den aufrufenden MapAdmin beenden.

```
public class NetconfigDisplay extends Frame {
// Attribute
         public MapAdmin mapadmin;
// Methoden
         public NetconfigDisplay(MapAdmin mapadmin);
         public boolean show();
         public void close();
}
```

5.2.4.3. SymbolDisplay

Das SymbolDisplay stellte die Informationen die in jedem Symbol vorhanden sind (`id`, `type`, `state`, `information`), in einem Dialog dar.

```
public class SymbolDisplay extends Frame {
// Attribute
         public Symbol symbol;
// Methoden
         public SymbolDisplay(Symbol symbol);
         public boolean show();
         public void close();
}
```

5.2.5. Interfaces

5.2.5.1. DBInterface

Das DBInterface ist die Schnittstelle zwischen INMS Server und VOMI. Hier wird die komplette Kommunikation mit der Datenbank behandelt.

```
public class DBInterface extends Object {
// Attribute
         public Socket socket;
         public DataInputStream in;
         public DataOutputStream out;
// Methoden
         public DBInterface();
         public boolean openConnection(String adresse, String name,
                  String password);
         public boolean closeConnection();
         public Hashtable requestSubmap(String id);
         public Symbol requestSymbol(String id);
         public Hashtable requestallStatus(String id);
         public Status requestStatus(String id);
//Hilfsmethoden
         public String readline();
         public void writeline(String line);
//Nachrichten empfangen
         public Queue decode(Queue nachricht, String firstline);
         public Symbol decodeSymbol(Queue nachricht);
         public boolean decodeState(Queue nachricht, Hashtable states);
// Nachricht senden
         public void sendopen(String name, String password);
         public void sendgetSubmap(String id);
         public void sendgetSymbol(String id);
         public void sendgetstatusofSubmap(String id);
         public void sendgetstatusofSymbol(String id);
}
```

5.2.5.1.1. Attribute

- `public Socket socket`

  Der Socket für die Verbindung mit dem Serverteil der Applikation.

- `public DataInputStream in`
  `public DataOutputStream out`

  Die Datenstreams zur Ein/Ausgabe von Daten über den Socket.

5.2.5.1.2. Methoden

- `public boolean openConnection(String adresse, String name,String password)`

  Öffnet die Verbindung zu dem INMS Server über den Serverteil von VOMI. Die Name des Servers wird in `adresse` übergeben. Weiterhin wird der Operatorname und das Paßwort mitübergeben. Falls die Operation erfolgreich war wird `true` zurückgeliefert.

- `public boolean closeConnection()`

  Schließt die Verbindung.

- `public Hashtable requestSubmap(Symbol submap)`

  Holt alle Symbole einer Submap. Dabei können auch Symbole mitgeliefert werden, die in Submaps dieser Submap liegen. Dies ist zum Beispiel der Fall, wenn nur die komplette Map übertragen werden kann.

- `public Symbol requestSymbol(String id)`

Holt das Objekt zu einem bestimmten `id`.

- `public Hashtable requestallStatus(Symbol submap)`

  Holt den Status aller Objekte in einer Submap.

- `public Status requestStatus(String id)`

  Holt den Status zu einem Objekt mit dem Identifikationsstring `id`.

- `public String readline()`
  `public void writeline(String line)`

  Routinen zum Lesen und Schreiben auf den Sockets.

- `public Queue decode(Queue nachricht, String firstline)`

  Liest eine Nachricht vom Server komplett ein, falls die Nachricht weitere Nachrichten enthält, wird diese Routine rekursiv aufgerufen. Die komplette Nachricht wird in einer Queue (Warteschlange nach dem First In First Out Prinzip) zurückgeliefert. Dabei werden die Attributname/werte-Paare gleich in die Datenstruktur DBValue umgesetzt.

- `public Symbol decodeSymbol(Queue nachricht)`

  Die Methode extrahiert ein Symbol aus der übergebenen Queue.

- `public boolean decodeState(Queue nachricht, Hashtable states)`

  Die Methode extrahiert einen Status aus der Queue und trägt diesen in die übergebenen Hashtable ein.

- `public void sendopen(String name, String password)`
  `public void sendgetSubmap(String id)`
  `public void sendgetSymbol(String id)`
  `public void sendgetstatusofSubmap(String id)`
  `public void sendgetstatusofSymbol(String id)`

  Diese Methoden senden die entsprechenden Nachrichten an den Server.

5.2.5.2. StatusInterface

Weitere Informationen siehe 4.3.

```
public class StatusInterface extends Thread {
// Attribute
         public SNMPSocket socket
         public MapAdmin mapadmin;
// Methoden
         public StatusInterface(String adresse, String name,
                  String password);
         public void run();
         public void waitforTraps();
}
```

5.2.5.2.1. Attribute

- `public SNMPSocket socket`

  Die Socketverbindung zu dem SNMP Proxy.

- `public MapAdmin mapadmin`

  Der Verweis auf den MapAdmin wir benötigt um die ankommenden Traps den einzelnen Symbolen zuordnen zu können. Dabei wird aus dem Trap erst die betroffenen Komponente extrahiert, danach das entsprechende Symbol über den MapAdmin gesucht und in das Symbol die Nachricht eingetragen.

5.2.5.2.2. Methoden

- `public StatusInterface(String adresse, String name,String password)`

  Generiert den Status-Clientteil von VOMI. Übergeben wird neben der Adresse des Proxies, auch der Operatorname und das Paßwort, damit nur berechtigte Personen SNMP Traps empfangen können.

- `public void run()`

  Da die Klasse asynchron Nachrichten empfängt, ist sie als eigenständiger Thread implementiert. Die Methode `run()` wird bei der Generierung des Thread's aufgerufen und stellt danach den Threadprozess dar.

- `public void waitforTraps()`

  Methode die auf ankommende Traps wartet, um Sie dann über den MapAdmin in den Symbolen einzutragen.

5.2.6. Server

Der Server ergänzt die DBInterface um die entsprechenden Methoden mit denen die Verbindung zu der Datenbank aufgebaut wird. Weiterhin geschieht hier die Umsetzung von INMS Objekten in VOMI Objekten. Die syntaktische Beschreibung des Servers ist zwar in Java gehalten, aber die tatsächliche Implementierung in C++.

```
public class Server extends Object {
// Attribute
         public Socket socket;
         public DataInputStream in;
         public DataOutputStream out;
// Methoden
         public Server();
         public void main();
         public void handleConnections();
         public boolean openDB(String name,String password);
         public boolean closeDB();
         public Hashtable requestSubmap(String id);
         public Symbol requestSymbol(String id);
         public Hashtable requestallStatus(String id);
         public Status requestStatus(String id);
         public String convertNode2Symbol(DBSymbol symbol);
         public String convertLink2Symbol(DBSymbol symbol);
         public String convertDomain2Symbol(DBSymbol symbol);
         public String convertSUNI2Symbol(DBSymbol symbol);
}
```

5.2.6.0.1. Attribute

Siehe 5.2.5.1.

5.2.6.0.2. Methoden

- `void handleConnections()`

  Verwaltet die Verbindung zu dem Client, und wartet auf neue Nachrichten.

- `public boolean openDB(String name,String password)`
  `public boolean closeDB()`
  `public Hashtable requestSubmap(String id)`
  `public Symbol requestSymbol(String id)`
  `public Hashtable requestallStatus(String id)`
  `public Status requestStatus(String id)`

  Die entsprechenden Gegenstücke zu den Clientfunktionen.

- `public String convertNode2Symbol(DBSymbol symbol)`
  `public String convertLink2Symbol(DBSymbol symbol)`
  `public String convertDomain2Symbol(DBSymbol symbol)`
  `public String convertSUNI2Symbol(DBSymbol symbol);`

  Konvertiert ein INMS Objekt in das entsprechende VOMI Objekt, dabei wird kein Javaobjekt zurückgegeben, sondern der String der zu dem Client gesendet wird.

5.2.7. Hilfsklassen

Hier sind nun einige Klassen beschrieben, die weitere Datenstrukturen oder zusätzliche Funktionalität implementieren, die von Java nicht bereitgestellt wird.

5.2.7.1. StateFilter

Die Klasse ändert die Farbe eines Pixels in einem Image von der Farbe `from` zu der Farbe `to`.

```
public class StateFilter extends RGBImageFilter {
// Attribute
         public int from;
         public int to;
// Methoden
         public StateFilter(Color from, Color to);
         public int filterRGB(int x ,int y, int rgb);
}
```

5.2.7.2. DBValue

Die Klasse liefert einen Datenstruktur zum Speichern von Key/Value Paare.

```
public class DBValue extends Object {
// Attribute
         public String key;
         public String value;
// Methoden
         public DBValue(String line);
}
```

5.2.7.2.1. Methoden

- `public DBValue(String line)`

  Generiert aus einem String der Form “<key>=<value>“ ein DBValue Objekt.

5.2.7.3. ImageCenter

Erste Tests mit dem Prototypen haben ergeben, daß das Einfärben der Images für Nodes mit Hilfe des StateFilters sehr zeitaufwendig ist. Weiterhin ist eine Stelle zur Zuordnung von Nodetypen zu graphischen Bitmaps erforderlich, da in INMS die graphische Darstellung von Nodes durch HP OpenView über die Klassen- und Instanzennamen erfolgt.

Es handelt sich um eine „statische“ Klasse, d.h. die Klasse wird einmal am Programmstart instanziert, und über statische Methoden aufgerufen. Die Klasse ist von Panel abgeleitet, damit Sie einen ImageObserver für die weitere Bildverarbeitung zur Verfügung hat.

```
public class ImageCenter extends Panel {
// Attribute
         public static Hashtable images;        //Type,Image
         public static Vector related;          //TypeExp,Image
// Methoden
         public ImageCenter();
         public Image getImage(Type type, Color color);
}
```

5.2.7.3.1. Attribute

- `public static Hashtable images`

  Liste mit den eingefärbten Images, als Index wird der Typ der Node und die Farbe mit dem das Symbol eingefärbt werden soll. (Beispiel: `Type.toString()+"::"+Color.toString()`)

- `public static Vector related`

  Diese Liste wird aus einer Datenbank (oder Datei) generiert, die die Zuordnung von Nodes zu Images beschreibt. Es sind Wildcard-Ausdrücke erlaubt, um den Verwaltungsaufwand zu minimieren. (siehe auch `Type` Beschreibung 5.2.7.4.) Beispieldatei:

  domain,geographic,geodom.gif
  domain,*,genericdomain.gif
  ,,other.gif

  Formal nach INMS-Attributen: `<Class>,<Type>,<Image>`

  Die Liste ist keine Hashtable, da ein Symboltyp auf mehrere Einträge passen kann. Der Eintrag der mit dem Symboltyp am Besten übereinstimmt, wird verwendet.

5.2.7.3.2. Methoden

- `public Image getImage(Type type, Color color)`

  Liefert ein fertiges Image mit der entsprechenden Farbe zurück. Dabei wird zuerst in `images` gesucht, ob dieses Image schon einmal erzeugt wurde, falls nicht, wird das Image anhand von `related` bestimmt, erzeugt und in `images` gesichert.

5.2.7.4. Type

Diese Klasse ist analog zur Klassifizierung von INMS Objekten. Jedes Symbol kann in bestimmte Klassen und Instanzen der Klassen unterteilt werden. (Beispiel: Klasse: Cisco-Routerproduktpalette, Instanz: Cisco 2501 Router)

```
public class Type extends Object {
// Attribute
         public String Class;
         public String Instance;
// Methoden
         public Type(String Class, String Instance);
         public int hashCode();
         public boolean equals(Object obj);
}
```

5.2.8. Erweiterungen

Hier werden einigen Klassen besprochen, die in der Analyse angeschnitten wurden, aber hauptsächlich als Erweiterung gedacht sind, um die Arbeit zu erleichtern oder neue Funktionalität zu bringen, die über die passive Darstellung hinausgehen.

5.2.8.1. Submapbrowser

Der Treebrowser ist die textuelle Version des SubmapDisplays. Statt die Submaphierarchie und alle Symbole mittels Fenster und graphischen Objekten darzustellen, wird eine Liste erzeugt, in der man die Struktur der Hierarchie schneller erkennen kann. Der Nachteil ist, daß man in einer Submap nicht mehr die Verkettung der einzelnen Symbole und Links erkennt, da Sie in einer linearen Liste dargestellt werden.

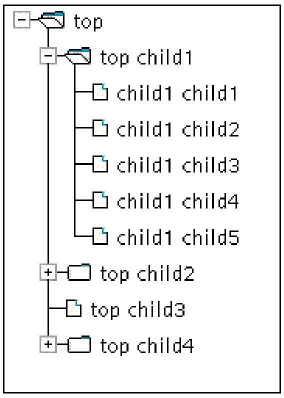


Abbildung 25 Treebrowser

```
public class SubmapBrowser extends Panel {
// Attribute
        public Symbol root;
// Methoden
        public SubmapBrowser(Symbol root);
        public void show();
        private void openSubmapDisplay(Symbol selected);
        private void openSymbolDisplay(Symbol selected);
        private void getSubmap(Symbol obj);
        public void close();
        public boolean handleEvent(Event evt);
}
```

5.2.8.1.1. Methoden

- `void getSubmap(Symbol obj)`

  Hier wird eine neue Submap geholt und in den Browser eingefügt.

Mapstat zeigt den Gesamtzustand des Netzes an. Es wird eine statistische Auswertung über alle Symbole und deren Zustand gemacht. Dieser wird periodisch auf den neuesten Stand gebracht.

5.2.8.2. Mapstat

```
public class Mapstat extends Frame {
// Attribute
         public Symbol root;
// Methoden
         public Mapstat(Symbol root);
         public void show();
         public void makeupdate();
         public void close();
         public boolean handleEvent(Event evt);
}
```

5.2.8.2.1. Attribute

- `public Symbol root`

  Verweis auf das Root-Symbol.

5.2.8.3. ApplicationCallCenter

Das ApplicationCallCenter ermöglicht es externe Elementmanager in Java aufzurufen. Der Aufbau ist ähnlich dem Autolayouter. Dies ist eine Wrapperklasse, die für jedes Symbol bestimmen kann, ob ein Elementmanager für dieses vorhanden ist. Diese Klasse ruft die Elementmanager noch nicht auf, sondern leitet die Aufrufe weiter an Klassen, die speziell auf die Manager zugeschnitten sind. Damit können weitere Manager leicht eingebunden werden, indem man weitere Verwaltungsklassen zu den bestehenden hinzufügt und sie dem ApplicationCallCenter mitteilt (z.B. über Konfigurationsdateien).

```
public class ApplicationCallCenter extends Object {
// Attribute
         public Hashtable manager;             //String,Manager
// Methoden
         public ApplicationCallCenter();
         public boolean openManager(Symbol obj);
}
```

5.2.8.3.1. Attribute

- `public Hashtable manager`

  Eine Liste mit den Objekten, die Elementmanager bedienen können.

5.2.8.3.2. Methoden

- `public boolean openManager(Symbol obj)`

  Ruft den Elementmanager für ein bestimmtes Objekt auf. Dabei werden erst alle Managerobjekte aus `manager` durchgegangen, ob Sie dieses Objekt behandeln können, falls ja wird `startManager()` in dem Objekt aufgerufen. Die Funktion liefert `false`, falls kein Manager gefunden wurde.

5.2.8.4. Manager

Eine generische Klasse zur Implementierung von Aufrufklassen für Elementmanager.

```
public abstract class Manager extends Object {
// Attribute
        public String Id;
// Methoden
        public Manager();
        public abstract boolean canhandle(Symbol obj);
        public abstract boolean openManager(Symbol obj);
}
```

5.2.8.4.1. Methoden

- `public abstract boolean canhandle(Symbol obj)`

  Liefert `true` zurück, falls der Elementmanager das Symbol `obj` managen kann.
- `public abstract boolean openManager(Symbol obj)`

  Startet den Elementmanager für das Objekt `obj`, dabei wird ein neuer Thread generiert, so daß der aufrufende Prozeß nicht auf die Beendigung des Managers warten muß.

5.3. Abläufe

Die Darstellung der Abläufe soll einen Einblick in die Funktionsweise des Programmes bieten.

5.3.1. Objektgenerierung

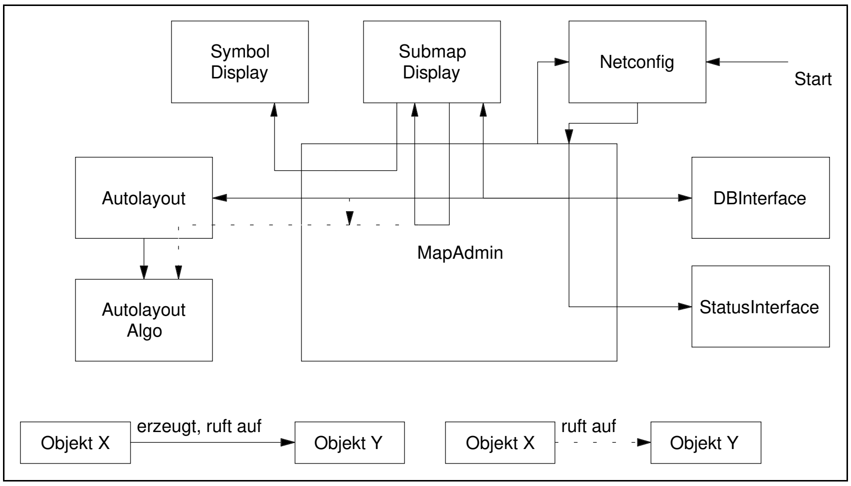


Abbildung 26 Objekterzeugung

Abbildung 26 beschreibt den Erzeugungsablauf der Applikation. Der Startablauf des Progammes kann folgendermaßen zusammengefaßt werden (<X>:= Instanz von Klasse X):

1. <VOMI> Starte Java Applet/Application
2. <VOMI> Erzeuge Netconfig
3. <NetConfig> Frage die Daten zur Verbindungsaufnahme mit dem INMS Server ab
4. <NetConfig> Erzeuge MapAdmin
5. <MapAdmin> Erzeuge DBInterface
6. <DBInterface> Baue Verbindung mit INMS Server über VOMI Server auf
7. <MapAdmin> Erzeuge StatusInterface
8. <StatusInterface> Baue Verbindung mit VOMI SNMP Proxy auf

9. <MapAdmin> Hole Rootsubmap eines Operators und Status

10. <DBInterface> Hole Submap und Stati der Objekte

11. <MapAdmin> Erzeuge SubmapDisplay

12. <SubmapDisplay> Stelle Submap dar

5.3.2. Virtuelle Objekte

Ein INMS Objekt kann Teil mehrerer Domänen sein. Obwohl es nur einmal in der Datenbank vorkommt, wird es vom DBInterface für jede Submap in der es sich befindet als neue Instanz zurückgeliefert. Probleme ergeben sich nun, wenn neue Informationen wie der Status hinzukommen, die nun dem Objekt zugeordnet werden müssen. In diesem Fall müßten allen Instanzen dieses neue Information zugeordnet werden.

Da in Java alle Datentypen (außer den primitiven Datentypen) Objekte sind, bietet es sich an stattdessen nur ein Orginalobjekt anzulegen. In allen weiteren Instanzierungen des gleichen Objekts werden die Attribute auf die Attribute des Orginalobjekts referenziert. Ausnahmen sind die Attribute, die die Instanz in einer Submap oder in VOMI auszeichnen (z.B. Submap-Koordinaten). Wenn nun Änderungen in den referenzierten Attributen gemacht werden, ändern sich automatisch auch alle Attribute in den anderen Instanzen. Wenn nun Objekte zerstört werden, bleiben die Attribute, da sie auch Objekte sind, solange erhalten bis keine Referenz mehr auf sie besteht. Es dürfen die Attribute aber nicht ersetzt werden, sondern nur geändert, da sonst bei allen anderen Objekte Referenzen auf alte Attribute bestehen.

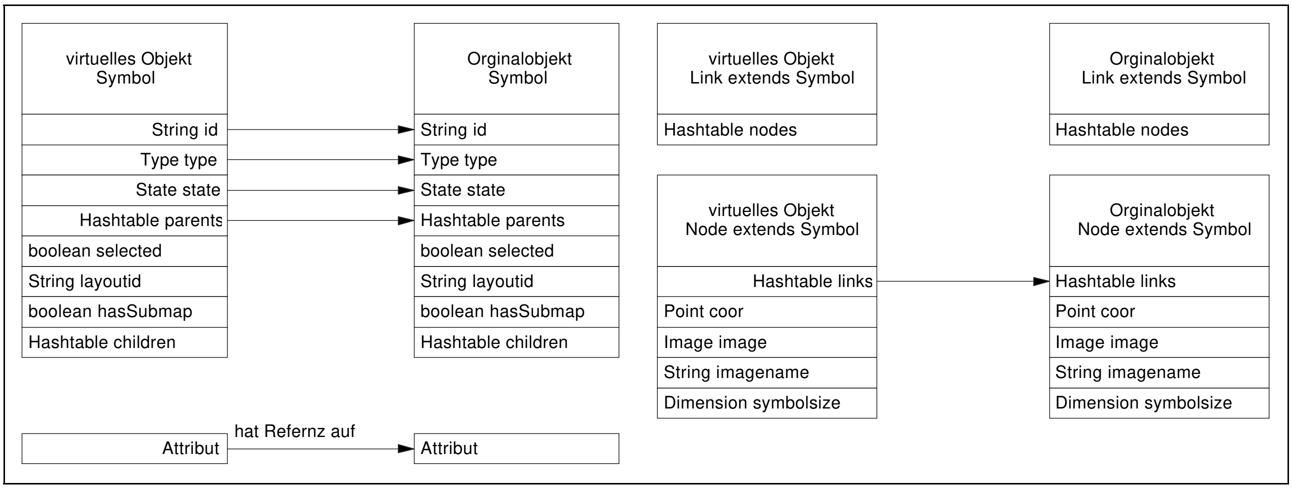


Abbildung 27 virtuelle Objekte

Wie man an dem Objektmodell sehen kann, ist das VOMI Objektmodell nicht gleich dem von INMS, deshalb müssen die INMS Objekte erst konvertiert werden bevor, sie von VOMI verwendet werden können. Die Umsetzung geschieht auf Serverseite, damit das INMS Objektmodell vor dem Client verschattet bleibt. Folgende Umsetzungen finden statt (<X(Y=Z)>:= Instanz von Klasse X mit Attribut Y auf Z gesetzt):

5.3.3. Objektkonvertierung

1. <INMS Domäne>→ <Node(hasSubmap=true)>

2. <INMS Node|Module|Port>→ <Node(hasSubmap=false)>

3. <INMS NodeToNodeLink>→ <Link(hasSubmap=false)>

5.3.3.1. LinkView

Alle anderen Links als der NodeToNodeLink erzeugen eine Submap, da sie nicht direkt Nodes miteinander verbinden. Aus Übersichtlichkeit werden aber in einer Submap nur Nodes dargestellt. Ähnlich wie bei HP OpenView gibt es einen LinkView (siehe Abb. 28), der in VOMI als normales SubmapDisplay dargestellt wird. Dazu erzeugt der Server ein Objekt <Link(hasSubmap=true)> und die entsprechende Submap. Die Submap enthält bei einem PortToPortLink neben dem eigentlichen Link, noch die entsprechenden Ports, Module und Nodes. Somit ergeben sich folgende Symbole in der Submap:

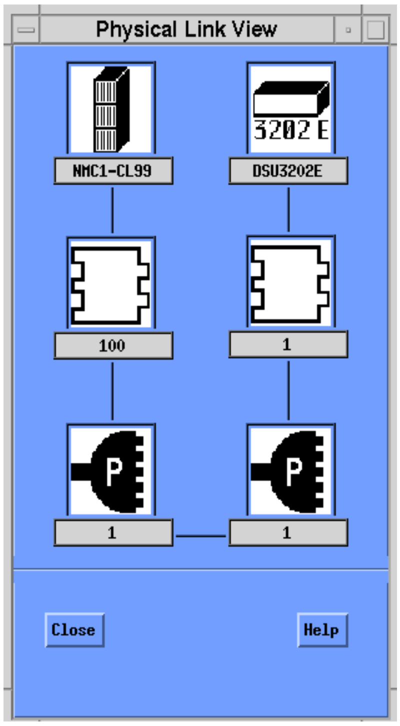


Abbildung 28 LinkView

1. <PortToPortLink>→ <Link(nodes=sport,eport)>
2. <INMS Start-Node>→ <Node(id=snode)>
3. <INMS End-Node>→ <Node(id=enode)>
4. <INMS Start-Module>→ <Node(id=smod)>
5. <INMS End-Module>→ <Node(id=smod)>
6. <INMS Start-Port>→ <Node(id=sport)>
7. <INMS End-Port>→ <Node(id=eport)>

Sowie vier Links, die zur besseren Darstellung dienen:

<Link(nodes=snode,smod)>, <Link(nodes=enode,emod)>,

<Link(nodes=smod,sport)>, <Link(nodes=emod,eport)>

Weiterhin muß dem Autolayouter mitgeteilt werden, daß dieses SubmapDisplay speziell gelayoutet wird (siehe Abbildung). Dies geschieht über das `Type`-Attribut, daß auf `LinkView` gesetzt wird (siehe 5.3.4.).

Für Links mit gemischten Endpunkten (NodeToPortLink) werden die entsprechenden Ports und Module weggelassen.

5.3.3.2. MetaConnection

Die MetaConnection ist aus der Tatsache entstanden, daß mehrere Links zwischen zwei Nodes existieren können, dies aber bei vielen Links nicht übersichtlich darstellbar ist. Als Lösung wird ein neuer Link generiert, der eine Submap enthält, indem alle Links übersichtlich angeordnet sind (siehe Abbildung 29). Dazu werden für *n* Links jeweils für die Start- und End-Node *n* virtuelle Nodes angelegt.

Es ergibt sich folgender Ablauf (Startnode ist *snode* und Endnode ist *enode*):

1. Ersetze alle *n* Links zwischen *snode* und *enode* durch einen neuen Link <Link (hasSubmap=true)>
2. Füge *n* virtuelle *snode* und *n* virtuelle *enode* in den neuen Link ein.
3. Füge die ersetzten *n* Links ein und verbinde sie mit den neuen virtuellen Nodes, so daß jede Node nur mit einen Link assoziiert ist.

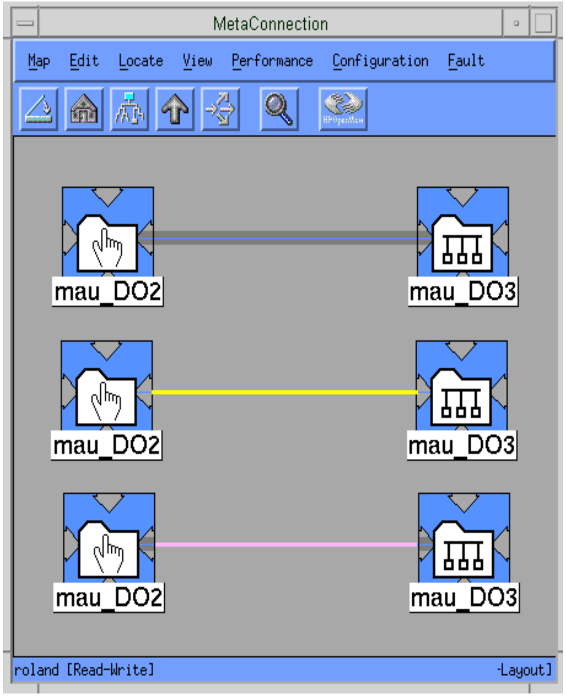


Abbildung 29 MetaConnection

5.3.4. Attributkonvertierung

Alle von der Datenbank gelieferten Attribute werden in der Hashtable `information` (siehe 5.2.2.1.) abgelegt. Als Schlüssel für die Hashtable wird der Attributname und als Wert der Attributwert von der Datenbank genommen. Ausnahmen stellen die Attribute Class, Type und Instance dar. Folgende Zuordnung findet statt:

- <Instance> → <Symbol(id=<Instance>)>
- <Class> → <Symbol(type.Class=<Class>)>
- <Type> → <Symbol(type.Instance=<Type>)>

Neben den INMS Typen gibt es noch spezielle Typen, die von VOMI angelegt werden. Diese werden im type.Class Attribut mit "VOMI" bezeichnet. Folgende zwei Instanzen ergeben sich aus 5.3.3. :

- type.Instance="LinkView"
- type.Instance="MetaConnection"

Diese dürfen nicht von der INMS Datenbank geholt werden, da Sie dort unbekannt sind.

5.4. Algorithmen

Einer der wenigen Algorithmen die in der GUI vorhanden ist, ist der Algorithmus zur Bestimmung des graphischen Symbols, das mit der Maus angeklickt wurde. Als einzige Prämisse ist der Vorrang von Nodes vor Links und das die Symbole von Nodes rechteckige Ausmaße haben. Der Algorithmus ist nun folgendermaßen aufgebaut:

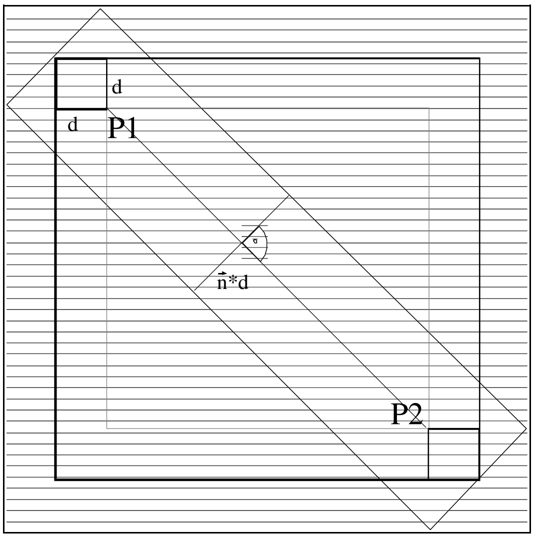


Abbildung 30 Abstandsberechnung

p:=Klickpunkt

n:=Normalvektor auf der Strecke P1P2

d:=maximaler Klickabstand

D:=Vektor (d,d)

P1,P2:=Koordinaten der Endpunkte des Links

1. Gehe alle Nodes durch und überprüfe ob der Klickpunkt innerhalb des graphischen Symbols liegt.
2. Gehe alle Links durch
    1. Falls p nicht innerhalb des Rechtecks, daß P1-D, P2+D aufspannt liegt, nehme nächsten Link
    2. Bestimme den Abstand vom Klickpunkt p zu der Linie P1P2, falls der kleiner als d ist, wurde dieser Link angeklickt.

Der Normalvektor wird weiterhin benötigt um selektierte Linien zu zeichnen, ähnlich der schraffierten Fläche in Abb. 30.

5.5. Prototyp

Ein Ziel der Diplomarbeit ist die prototypische Implementierung des Programmes. Die Klassenbeschreibung ist so ausgelegt, daß das Projekt komplett verwirklicht werden kann. Aus Zeitgründen wurde auf die Implementierung einiger Klassen und Methoden verzichtet, trotzdem werden die wichtigsten Funktionalitäten unterstützt (graphische Darstellung einer Domäne, Anzeige des Status einzelner Symbole).

Folgende Klassen wurden komplett implementiert:

Symbol (5.2.2.1.),Node(5.2.2.2.),Link(5.2.2.3.),MapAdmin(5.2.3.1.),AutolayoutAlgo(5.2.3.3.), DBValue(5.2.7.2.), StateFilter(5.2.7.1.)

Teilweise wurde implementiert:

DBInterface(5.2.5.1.),State(5.2.2.4.)

Nicht implementiert wurde:

Autolayout(5.2.3.2.), SymbolDisplay(5.2.4.3.), NetconfigDisplay(5.2.4.2.),ImageCenter(5.2.7.3.), alle Erweiterungsklassen(5.2.8.)

Der Server setzt zu großen Teilen auf vorhandenen Sourcecode auf. Anfang 1997 implementierte

Michael Moeckel bei Siemens eine textuelle Darstellung der Managed Objects von INMS, vergleichbar mit dem SubmapBrowser. Die Serverseite ist in C++, die Clientseite in Java implementiert und als Übertragungsprotokoll wurde ICE-T verwendet. Aus diesem Projekt wurden die Sourcecodes von der Serverseite weiterverwendet und an VOMI angepaßt.

Mit dem Prototyp ist es möglich sich in der INMS Datenbank einzuloggen und sich durch die Domänenhierarchie zu klicken. Es wird dabei die ganze Hierarchie domänenweise vom Server abgefragt, wobei für jede Domäne eine Submap erzeugt wird, indem die Domänen, Nodes und Links graphisch dargestellt werden. Vor der Darstellung wird noch der INMS Server nach dem Status aller dargestellten Objekte abgefragt, dementsprechend werden die Symbole eingefärbt. Abbildung 31 zeigt ein typisches SubmapDisplay, wie es sich dem Benutzer präsentiert.

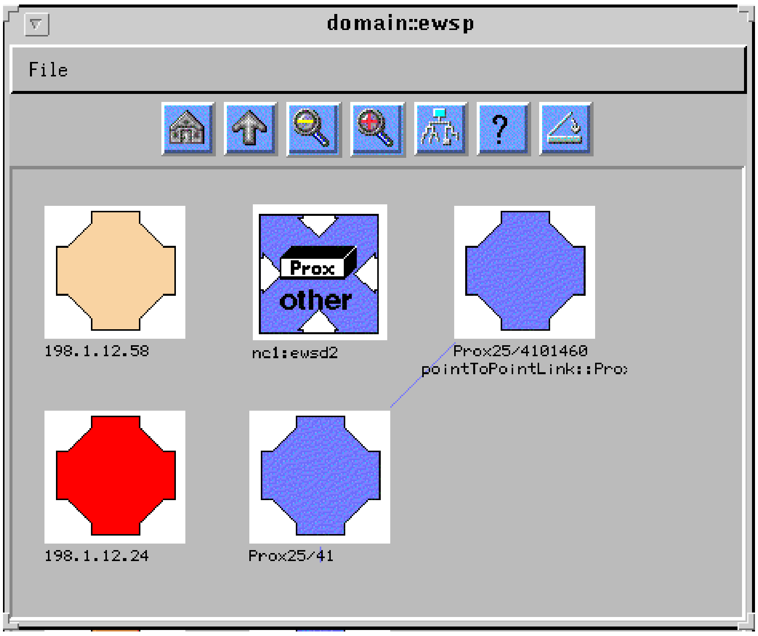


Abbildung 31 SubmapDisplay

Die Buttons in der obersten Zeile bedeuten (von links nach rechts):

Springe zur Root Map, Gehe eine Submapebene hoch, Zoom Out, Zoom In, Unbenutzt, Hole Status aller dargestellten Objekte, Schließe Fenster

6. Ausblick

Der Prototyp zeigt, daß es möglich ist Teile von Managementfunktionen auf ein Javaprogramm zu übertragen. Als größtes Hindernis stellte sich die die Mischung von C++/Java und die Versionsunterschiede in den Javabibliotheken heraus. Da der Server unter INMS vorerst immer auf eine C++ API zurückgreifen muß, ist ein reiner Javaserver nicht ohne größeren Aufwand zu erledigen. Weiterhin wird bei Sun versucht die Erweiterungen wie JMAPI unter JDK 1.1.1 lauffähig zu machen, so wie weitere Hersteller versuchen JDK 1.1.1 in ihre Produkte zu integrieren. Als Erweiterungen und Verbesserungen stehen an:

1. Vollständige Implementierung aller Klassen in JDK 1.1.1

   Der Prototyp implementiert nur einen Teil der beschriebenen Klassen. Neben der vollständigen Implementierung des Clients, sollte gleichzeitig auf JDK 1.1.1 umgestiegen werden, da die Entwicklungen für JDK 1.0.2 eingestellt sind. Beispiel dafür ist JMAPI, wo Fehler nur noch in Versionen für JDK 1.1.1 verbessert werden. Dies bedeutet aber auch ein Umschreiben der GUI-Ereignisverwaltung, da diese sich in JDK 1.1.1 komplett geändert hat. Von einem Ausbau des Protokolls im DBInterface sollte abgesehen werden, falls in absehbarer Zeit der Server in Java umgesetzt wird. Der Zeitaufwand liegt unter 2 Monaten und es werden gute Javakenntnisse vorausgesetzt.

2. Implementierung eines Servers in Java mit Native Calls

   Ein Server in Java ist die Voraussetzung zur Verwendung von RMI. Damit ist es möglich Objekte über das Netz zu versenden und auf ein eigenes Objektaustauschprotokoll zu verzichten. Das DBInterface wird dabei komplett vom Client gelöst und auf dem Server implementiert und via RMI aufgerufen. Da die Schnittstelle zur Datenbank weiterhin in C++ vorhanden ist, muß ein Konzept überlegt werden, wie die bestehenden C-Sourcecodes von INMS in Java eingebunden werden. Es sollte dabei soviel wie möglich auf vorhandenen Sourcecodes zurückgegriffen werden. Im Prototypen ist der Sourcecode nicht identisch mit dem von INMS, so daß bei Änderungen in INMS, auch Änderungen im Prototypsourcecode nach sich ziehen. Der Javaserver sollte dabei so weit wie möglich unabhängig von INMS Server sein, damit ein Einsatz in anderen Systemen einfach zu realisieren ist. Der C++ Teil hat dann eine fest definierte Schnittstelle für die Native Calls aus Java und ist direkt von der verwendeten Datenbank abhängig.

   Voraussetzungen sind sehr gute Kenntnisse in C++ und Java, der Zeitaufwand liegt deutlich über 3 Monate.

3. Implementierung der SNMP Trap Verwaltung

   Neben dem StatusInterface muß noch ein SNMP Proxy implementiert werden. Dieser Proxy kann in Java implementiert werden, was vor allem die Verwaltung von mehreren Socketverbindungen durch das Threadkonzept stark vereinfacht. Damit ist es dann möglich das Netz aktiv zu monitoren.

   Weiterhin ist ein Algorithmus zu suchen, wie Statusänderungen in der Hierarchie propagiert werden, so daß schon in höheren Submapschichten erkannt wird, ob in in tieferliegenden Submaps Komponenten ausgefallen sind.

   Voraussetzungen sind gute Kenntnisse in SNMP und Java, der Zeitaufwand dürfte sich auf 3 Monate beschränken.

4. Verschlüsselung

   Bereitstellen von Klassen die kryptographische Verfahren bieten. Die Implementierung beschränkt sich weitgehend in der Überschreibung der Lese/Scheibroutinen der Streamklassen in Java. Das Vorgehen wird in vielen Dokumenten über Java genau beschrieben (siehe http://java.sun.com/). Der Zeitaufwand liegt bei eine Monat, Voraussetzung ist aber ein in Java implementierter Server.

5. Autolayoutalgorithmen

   Wünschenswert sind Algorithmen die Netze kreuzungsfrei, so daß sich keine Links und Nodes überschneiden, darstellen. Dazu gibt es eine Vielzahl von formalen Methoden, die Lösungen für dieses Problem bieten.

7. Anhang

7.1. Data Dictionary

- Autolayout

  Autolayout bezeichnet den Vorgang graphische Objekte auf dem Bildschirm so anzuordnen, daß Strukturen für den Benutzer leicht erkennbar sind.

- Condis API

  Datenbankschnittstelle zur GemStone Datenbank, die von INMS verwendet wird

- GUI

  Graphical **U**ser **I**nterface (Benutzungsoberfläche)

- INMS

  Integrated **N**etwork **M**anagement **S**ervices, von Siemens entwickeltes Umbrella-Netzmanagement-System.

- INMS-Server

  Der INMS-Server ist für die Alarm- und Statusverwaltung zuständig. Er empfängt und korreliert die Ereignismeldungen einzelner Elemente und sendet sie weiter an den Map-Server.

- JDK

  Das **J**ava **D**evelopment **K**it wird von Sun herausgegeben und dient als Richtlinie für alle Java-Implementierungen anderer Hersteller. Alle im JDK beschriebenen Klassen müssen in Java-Implementierungen von Fremdherstellern die gleiche Funktionalität bieten.

- JMAPI

  Java **M**anagement **API** ist eine Erweiterung von Sun zu Java, die Klassen zur Implementierung von Management-Anwendungen bietet.

- Link

  Eine Link verbindet 2 Nodes. Es gibt physische und logische Links. Logische Links können Supmaps enthalten. Links werden als Linien zwischen Nodes dargestellt.

- Managed Object

  Ein Managed Object stellt eine zu managende Ressource/Information im Netz dar. Node, Link, Traps und Submaps stellen Managed Objects dar.

- Map-Server

  Der Map-Server generiert die Netzkarten aus der Datenbank und stellt die Statusinformationen, die vom INMS-Server geliefert werden, dar.

- Native Calls

  Java ist in einer virtuellem Maschine gekapselt, somit ist es nicht möglich externe Programme oder Bibliotheken aufzurufen. Native Calls bieten nun die Möglichkeit Funktionen in nicht-Javabibliotheken aufzurufen. Dies hat den Nachteil, daß das Sicherheitskonzept von Java ausser Kraft gesetzt wird.

- Node

  Der Node kann sowohl eine physische Ressource (Router, Rechner, ...) oder ein logisches Objekt (Subnetz) sein. Ein Node wird durch eine Bitmap dargestellt.

- Submap

  Submap ist eine Ansammlung von Managed Objects, die im logischen und/oder physichen Zusammenhang miteinander stehen (z.B. Subnetze). Eine Map ist die Summe aller Submaps.

- Status

  Der Status ist ein Teil der Information die ein Managed Object enthält. Der Status gibt Auskunft über den Zustand eines Managed Object's und kann sich laufend ändern.

- Symbol

  Ein graphisches Objekt (Bild, Bitmap), das ein Managed Object auf der Benutzungsoberfläche darstellt.

- Trap

  Der Trap ist ein Ereignis, daß in einem Managed Object ausgelöst wurde und als Nachricht z.B. an eine Managementplatform geschickt wird. (z.B. SNMP Traps)

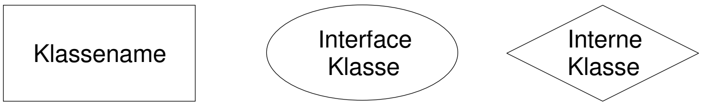
- VOMI

  (Visualization Of Management Information) Kurzbezeichnung für das erstellte Webfrontend.

7.2. OMT Notationen

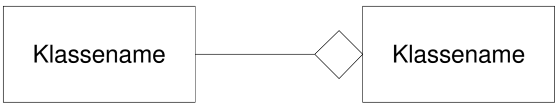
7.2.1. Objektmodell

- Klasse

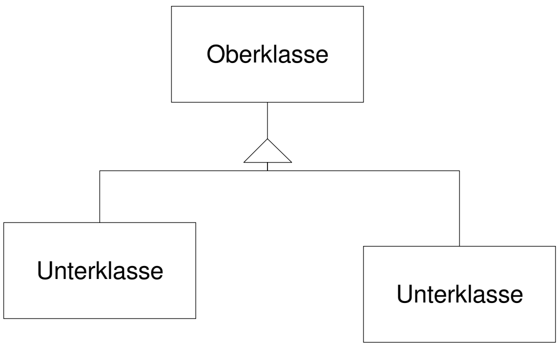


- Assoziation

- Aggregation



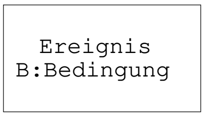
- Generalisierung (Vererbung)



7.2.2. Dynamsiches Modell

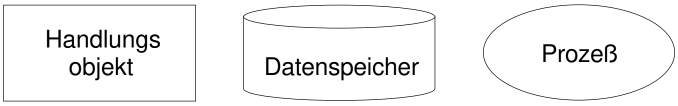
- Zustand

- Ereignis

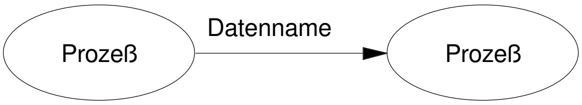


7.2.3. Funktionales Modell

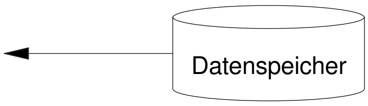
- Objekte



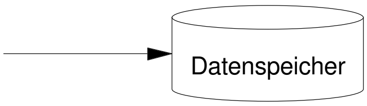
- Datenfluß



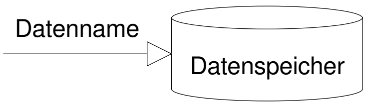
- Zugriff auf einen Datenspeicherwert



- Aktualisierung eines Datenspeicherwertes



- Generierung eines neuen Datenspeicherwerts



7.3. Literaturverzeichnis

[Fl 96] D. Flanagen, Java in a Nutshell, O'Reilly & Associates, Inc, 1996

[MSS 96] S. Middendorf, R Singer, S. Strobel, Java Programmierhandbuch und Referenz, dpunkt, 1996

[OMT 94] J. Rumbaugh, Objektorientiertes Modellieren und Entwerfen, Carl Hanser Verlag, 1994

[HeAb 94] H.-G. Hegering and S. Abeck, Integrated Network and System Management Addison-Wesley, 1994

[Java 95] Sun Microsystems, The Java Language Enviroment: A White Paper, 1995

[JMAPI 96] Sun Microsystems, Java Management Application Programming Interface, 1996 http://java.sun.com/products/JavaManagement/

[ADV 97] Advent SNMP Package Version 1.1, http://www.adventnet.com/snmp_api.html

[RFC 1157] J. D. Case, M. Feder, M. L. Schoffstall and C. Davin, „Simple Network Management Protocol (SNMP)", RFC 1157, IAB, May 1990.

[SIS 96] INMS Systembeschreibung, Siemens, 1996

[IMB 96] INMS, Implementation of Component Map Builder, Siemens,1996

[Fl 96] D. Flanagen, Java in a Nutshell, O'Reilly & Associates, Inc, 1996
[CAPI 96] INMS, Testapi, The Database Interface, Siemens, 1996
[DM 96] INMS V2.0 Datamodel, Siemens, 1996
[VER 95] Frank Yellin, Low-level Security in Java, WWW3 Conference, December 1995
[ICE 96] ICE-T User's Guide, Sun Microsystems, 1996

7.4. Sourcecode

7.4.1. Symbol

```
// Klassenname: Symbol
// Author     : Roland Spatzenegger
// Beginn     : 15.5.97
// Last Change: 19.7.97
// Version    : 0.3
// Bemerkung  :

// ***************************************************************************
// Imports
// ***************************************************************************

import Node;
import Link;
import State;
import java.lang.*;
import java.util.*;
import java.awt.*;

public abstract class Symbol {

  // ***************************************************************************
  // Public Attribute
  // ***************************************************************************

  public String id;
  //     public Type type;
  public Hashtable information;
  public State state;
  public Hashtable children;
  public Hashtable parents;
  public boolean selected;
  public boolean hasSubmap;

  // ***************************************************************************
  // Constructor
  // ***************************************************************************

  public Symbol() {
    children = new Hashtable();
    parents = new Hashtable();
    information = new Hashtable();
    selected = false;
    id = "NULL";
    state = new State(this);
  }

  // ***************************************************************************
  // Methoden
  // ***************************************************************************

  public void addChild(Symbol child) {
    if(children == null) {
      System.out.println("WUAH");
```

```
    }
    System.out.println("Kinder zu "+this.id+":"+child.getid()+"/"+(String)
child.information.get("type")+"/"+child.information.get("information.type"));

    if( child instanceof Link ) {
      Enumeration nodelist;
      Node endpoint;
      String endid;

      nodelist = ((Link) child).getnodes();
      endid = (String) nodelist.nextElement();
      endpoint = (Node) searchSymbol(endid);
      if( endpoint == null ) {
          System.out.println("Endpunkt "+endid+" nicht gefunden, Objekt
verworfen !");
          return;
      }
      ((Link) child).insertNode(endpoint);

      endid = (String) nodelist.nextElement();
      endpoint = (Node) searchSymbol(endid);
      if( endpoint == null ) {
          System.out.println("Enpunkt "+endid+" nicht gefunden,Objekt verworfen
!");
          return;
      }
      ((Link) child).insertNode(endpoint);
    }
    children.put(child.getid(),child);
    child.addParent(this);
  }

  // ****************************************************************************

  public void addParent(Symbol parent) {
    parents.put(parent.getid(),parent);
  }

  // ****************************************************************************

  public Symbol findSymbolat(Point p,float zoom) {
    Enumeration findlist;
    Symbol fsymbol;
    Node fnode;
    Link flink;
    float dist,mindist;

    mindist = 10000;
    fnode=null;
    flink=null;

    findlist = children.elements();
          //alle Kinder holen
    while( findlist.hasMoreElements() ) {
      fsymbol = (Symbol) findlist.nextElement();
      dist = fsymbol.isinside(p,zoom);
      if( dist < mindist ) {                   //ist p innerhalb eines Kindes ?
          if( fsymbol instanceof Node ) {
            fnode = (Node) fsymbol;
          } else {
            flink = (Link) fsymbol;
          }
          mindist = dist;
      }
                }
    if( fnode != null ) {
```

```
      return fnode;
    } else {
      return flink;
    }
  }

  // ***********************************************************************

  public Symbol searchSymbol(String id) {
    Enumeration findlist;
    Symbol fsymbol,rsymbol;

    if( id.equals(this.id) ) {
      return this ;
                  //gefunden
    }

    rsymbol = null;
                           // Rueckgabewert der Kinder
    findlist = children.elements();             //alle Kinder holen
    while( findlist.hasMoreElements() ) {
      fsymbol = (Symbol) findlist.nextElement();
      rsymbol = fsymbol.searchSymbol(id);
      if( rsymbol != null) {
         break;
      }
    }
    return rsymbol;
  }

  // ***********************************************************************

  public abstract float isinside(Point p, float zoom);
  public abstract void paint(Component watcher, Graphics graphics, float zoom);

  // ***********************************************************************

  public int hashCode() {
    return this.id.hashCode();
  }

  // ***********************************************************************

  public boolean equals(Object obj) {
    if( obj instanceof Symbol ) {
      if( this.id.equals(((Symbol) obj).id) ) {
         return true;
      }
    }
    return false;
  }
  // ***********************************************************************
  // Attribut Leseoperationen
  // ***********************************************************************

  public String getid() { return this.id; }

  public Enumeration getparents() { return parents.keys(); }
  public Enumeration getparentobjs() { return parents.elements(); }

}
```

7.4.2. Node

```
// Klassenname: Node
// Author     : Roland Spatzenegger
```

```
// Beginn     : 19.5.97
// Last Change: 19.7.97
// Version    : 0.3
// Bemerkung  : 0.3 Node zeichnet sich selber

// ***************************************************************************
// Imports
// ***************************************************************************

import Symbol;
import Link;
import StateFilter;
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.image.*;


public class Node extends Symbol {

  // ***************************************************************************
  // Public Attribute
  // ***************************************************************************

  public Hashtable links;
  public Point coor;
  public Image image;
  public String imagename;
  public Dimension symbolsize;

  // ***************************************************************************
  // Constructor
  // ***************************************************************************

  public Node() {
    super();
    links = new Hashtable();
    coor = new Point(0,0);
    symbolsize = new Dimension(1,1);
  }

  // ***************************************************************************
  // Methoden
  // ***************************************************************************

  public float isinside(Point p,float zoom) {
    float h,w;
    h = symbolsize.height * Math.min(zoom,1) / 2;
    w = symbolsize.width * Math.min(zoom,1) / 2;
    if( p.x > coor.x * zoom - w && p.x < coor.x * zoom + w &&
         p.y > coor.y * zoom - h && p.y < coor.y * zoom + h ) {
      return (float) -1 ;
    } else {
      return (float) 100000;
    }
  }

  // ***************************************************************************

  public void paint(Component watcher, Graphics graphics, float zoom) {

    Thread t = Thread.currentThread();
    int width,height;
    int xp,yp;
    Image paintImage;
```

```
    if( image == null ) {
      image = watcher.getToolkit().getImage(imagename);

      if( watcher.prepareImage(image, watcher) == false) {
         while ((watcher.checkImage(image,watcher) & watcher.ALLBITS) !=
watcher.ALLBITS) {
           try { t.sleep(50); } catch(InterruptedException e) {
             System.out.println("Error: "+e);
           }
         }
      }
      symbolsize.height = image.getHeight(watcher);
      symbolsize.width = image.getWidth(watcher);
      System.out.println(imagename+":"+symbolsize.width+","+symbolsize.height);
    }
    width = (int) (symbolsize.width * Math.min(zoom,1));
    height = (int) (symbolsize.height * Math.min(zoom,1));

    paintImage = watcher.getToolkit().createImage(new FilteredImageSource(
           image.getSource(),new StateFilter(new Color(114,126,255),
state.getColor())));

    if( watcher.prepareImage(paintImage, width, height, watcher) == false) {
      while ((watcher.checkImage(paintImage, width, height,watcher) &
watcher.ALLBITS) != watcher.ALLBITS) {
         try { t.sleep(50); } catch(InterruptedException e) {
           System.out.println("Error: "+e);
         }
      }
    }

    xp = (int) (coor.x * zoom);
    yp = (int) (coor.y * zoom);

    if( selected == true) {
      graphics.setColor(Color.gray);
      graphics.fillRect( xp - width/2 -2,
                         yp - height/2 -2, width+4, height+4);
    } else {
      graphics.clearRect( xp - width/2 -2,
                          yp - height/2 -2, width+4, height+4);

    }
    graphics.drawImage(paintImage, xp - width/2, yp - height/2, width, height,
watcher);

    graphics.setColor(Color.black);
    graphics.setFont(watcher.getFont());
    graphics.drawString(id, xp - width/2, yp + height/2+12);

  }

  // ***************************************************************************
  // Attribut Schreiboperationen
  // ***************************************************************************

  public void insertlink(Link link) { links.put(link.id,link); }
}
```

7.4.3. Link

```
// Klassenname: Link
// Author     : Roland Spatzenegger
// Beginn     : 19.5.97
// Last Change: 19.7.97
// Version    : 0.3
```

```
// Bemerkung  :

// ***********************************************************************
// Imports
// ***********************************************************************

import Symbol;
import Node;
import java.lang.*;
import java.util.*;
import java.awt.*;

public class Link extends Symbol {

  // ***********************************************************************
  // Public Attribute
  // ***********************************************************************

         public Hashtable nodes;

  // ***********************************************************************
  // Constructor
  // ***********************************************************************

  public Link() {
    super();
    nodes = new Hashtable();
  }

  // ***********************************************************************
  // Methoden
  // ***********************************************************************

  public float isinside(Point p,float zoomfactor) {

    Enumeration nodelist;
    Node n1,n2;
    float xs,xe,ys,ye;
    float dist;
    float nd,nx,ny;

    nodelist = getnodeobjs();
    n1 = (Node) nodelist.nextElement();
    n2 = (Node) nodelist.nextElement();

    dist = 8;

    xs = Math.min(n1.coor.x,n2.coor.x) * zoomfactor;
    xe = Math.max(n1.coor.x,n2.coor.x) * zoomfactor;
    ys = Math.min(n1.coor.y,n2.coor.y) * zoomfactor;
    ye = Math.max(n1.coor.y,n2.coor.y) * zoomfactor;

    if( (p.x >= xs - dist) && (p.x <= xe + dist) &&
         (p.y >= ys - dist) && (p.y <= ye + dist) ) {
      // Normalvektor berechnen
      xs = n1.coor.x * zoomfactor;
      xe = n2.coor.x * zoomfactor;
      ys = n1.coor.y * zoomfactor;
      ye = n2.coor.y * zoomfactor;

      nd = (float) Math.sqrt(Math.pow(xs-xe,2)+Math.pow(ys-ye,2));
      nx = ( ys - ye ) / nd;
      ny = ( xe - xs ) / nd;

      return (float) Math.abs(nx*(p.x-xs)+ny*(p.y-ys));
    }
```

```
    return (float) 1000000;
  }

  // ***************************************************************************
  public void paint(Component watcher, Graphics graphics, float zoom) {
    Enumeration nodelist;
    Node n1,n2;
    float xs,ys,xe,ye;
    Polygon shade;
    float nd,nx,ny;

    nodelist = nodes.elements();
    n1 = (Node) nodelist.nextElement();
    n2 = (Node) nodelist.nextElement();

    xs = n1.coor.x * zoom;
    ys = n1.coor.y * zoom;
    xe = n2.coor.x * zoom;
    ye = n2.coor.y * zoom;

    if(selected == true) {
      graphics.setColor(watcher.getBackground());
      graphics.setXORMode(Color.gray);
      nd = (float) Math.sqrt(Math.pow(xs-xe,2)+Math.pow(ys-ye,2));
      nx = ( ys - ye ) / nd * 8;
      ny = ( xe - xs ) / nd * 8;
      shade = new Polygon();
      shade.addPoint((int)(xs+nx),(int)(ys+ny));
      shade.addPoint((int)(xe+nx),(int)(ye+ny));
      shade.addPoint((int)(xe-nx),(int)(ye-ny));
      shade.addPoint((int)(xs-nx),(int)(ys-ny));
      graphics.fillPolygon(shade);
    }
    graphics.setPaintMode();
    graphics.setColor(state.getColor());
    graphics.drawLine((int) xs,(int) ys,(int) xe,(int) ye);

    graphics.setColor(Color.black);
    graphics.setFont(watcher.getFont());
    graphics.drawString(id,(int) ((n1.coor.x+n2.coor.x)/2 * zoom),
                          (int) ((n1.coor.y+n2.coor.y)/2 * zoom ));
  }

  // ***************************************************************************
  // Attribut Leseoperationen
  // ***************************************************************************

  public Enumeration getnodes() {
    return nodes.keys();
  }
  public Enumeration getnodeobjs() {
    return nodes.elements();
  }

  // ***************************************************************************
  // Attribut Schreiboperationen
  // ***************************************************************************

  public void insertNode(Node node) {
    nodes.put(node.getid(),node);              //Fehlerabfrage > 2 Nodes
  }
  public void inserttmpNode(String nodeid) {
    Node dummy;
    dummy = new Node();
    nodes.put(nodeid,dummy);                              //Fehlerabfrage > 2
Nodes
```

```
  }

}
```

7.4.4. State

```
// Klassenname: State
// Autor      : Roland Spatzenegger
// Beginn     : 29.7.97
// Last Change: 29.7.97
// Version    : 0.1
// Bemerkung  :

// *****************************************************************************
// Imports
// *****************************************************************************

import Symbol;
import java.lang.*;
import java.awt.*;
import java.util.*;

public class State extends Object {

  // *****************************************************************************
  // Public Attribute
  // *****************************************************************************

  public Symbol symbol;
  public Integer state;
  public Integer internstate;
  public Hashtable events;

  // *****************************************************************************
  // Privat Attribute
  // *****************************************************************************

  private Hashtable names;
  private Hashtable colors;

  // *****************************************************************************
  // Constructor
  // *****************************************************************************

  public State(Symbol symbol) {
    names = new Hashtable();
    colors = new Hashtable();
    this.symbol = symbol;
    colors.put(new Integer(1),new Color(00,230,38));
    names.put( new Integer(1),"normal");
    colors.put(new Integer(2),new Color(65,255,255));
    names.put( new Integer(2),"warning");
    colors.put(new Integer(3),new Color(255,255,41));
    names.put( new Integer(3),"minor");
    colors.put(new Integer(4),new Color(255,133,0));
    names.put( new Integer(4),"major");
    colors.put(new Integer(5),new Color(255,0,0));
    names.put( new Integer(5),"critical");
    colors.put(new Integer(6),new Color(114,124,255));
    names.put( new Integer(6),"unknown");
    colors.put(new Integer(7),new Color(249,211,162));
    names.put( new Integer(7),"testing");
    colors.put(new Integer(999),new Color(114,124,255));
    names.put( new Integer(999),"invalid");
    internstate = new Integer(999);
    state = new Integer(999);
```

```
  }

  // ***********************************************************************
  // Methoden
  // ***********************************************************************

  public void setinternstate(int nr) {
    if (nr < 1 || nr > 7) {
      nr = 999;
    }
    internstate = new Integer(nr);
    state = new Integer(nr);
  }

  public Color getColor() {
    Color ret;
    ret = (Color) colors.get(state);
    return(ret);
  }
}
```

7.4.5. SubmapDisplay

```
// Klassenname: SubmapDisplay
// Autor      : Roland Spatzenegger
// Beginn     : 3.6.97
// Last Change: 19.7.97
// Version    : 0.7
// Bemerkung  : Darstellung einer Submap in einem Fenster
//            : 0.3: Eventhandling
//            : 0.4: Neugeschrieben und an Klassenbeschreibung V5 angepasst
//            : 0.5: GUI Funktionen

// ***********************************************************************
// Imports
// ***********************************************************************

import Node;
import Link;
import MapAdmin;
import java.lang.*;
import java.awt.*;
import java.util.*;
import sunw.admin.avm.base.*;
import java.awt.image.*;

public class SubmapDisplay extends Frame {

  // ***********************************************************************
  // Public Attribute
  // ***********************************************************************

  public Symbol submap;
  public ImageScroller submapScroller;
  public Image submapImage;
  public Graphics submapGraphics;
  public float zoomfactor;
  public Symbol selected;
  public MapAdmin mapadmin;
  public Font nodefont;

// ***********************************************************************
// Privat Attribute
// ***********************************************************************

  private MenuBar myMenu;
```

```
  private Menu fileMenu;
  private Panel buttonPanel;
  private ImageButton zoomOutButton,zoomInButton,closeButton,rootButton,
    upButton,layoutButton,helpButton;

// ********************************************************************
// Constructor
// ********************************************************************

  public SubmapDisplay(MapAdmin mapadmin,Symbol submap) {
    super(submap.id);                                                  //
Fensternamen setzen
    this.mapadmin = mapadmin;
    this.submap = submap;

    // GUI Aufbau
    resize(400,400);                                                   // 400x400 als
Fenstergröße
    addNotify();                                                       // damit wir
Images verwenden können
    setLayout(new BorderLayout());              // Wir wollen Borderlayout
    nodefont = new Font("Times Roman",Font.PLAIN,12);
    zoomfactor = 1;

    // Menue (macht zwar noch nicht viel Sinn)
    fileMenu = new Menu("File");
    fileMenu.add("open");
    fileMenu.add("close");
    fileMenu.add("exit");
    myMenu = new MenuBar();
    myMenu.add(fileMenu);
    this.setMenuBar(myMenu);

    //Buttonleiste
    buttonPanel = new Panel();                               // Buttonliste ist ein
Panel

    buttonPanel.add(rootButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/rootmap_ico.gif")));
    buttonPanel.add(upButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/up_ico.gif")));
    buttonPanel.add(zoomOutButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/minus_ico.gif")));
    buttonPanel.add(zoomInButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/plus_ico.gif")));
    buttonPanel.add(layoutButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/tree_ico.gif")));
    buttonPanel.add(helpButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/help_ico.gif")));
    buttonPanel.add(closeButton =
                       new
ImageButton(this.getToolkit().getImage("buttons/close_ico.gif")));

    add("North",buttonPanel);

    submapImage = createImage(40,40);           // Dummy Map
    add("Center", submapScroller = new ImageScroller(submapImage));
    submapScroller.setFont(nodefont);

  }
```

```
  // ***********************************************************************
  // Methoden
  // ***********************************************************************

  public void show()
    {
      System.out.println("show()");
      paintSubmap();
      super.show();
    }

  // ***********************************************************************

  public void paintSubmap() {

    //    mapadmin.startAutolayout(submap);

    // Als erstes wird bestimmt wie gross das Image sein muss.
    Enumeration symbols;
    Symbol work;
    int maxx,maxy;
    maxx=maxy=0;
    symbols = submap.children.elements();                //ihh kein direkter
Zugriff

    while(symbols.hasMoreElements()) {
      work = (Symbol) symbols.nextElement();
      if( work instanceof Node) {
          maxx=(int) Math.max(maxx,((Node) work).coor.x * zoomfactor );
          maxy=(int) Math.max(maxy,((Node) work).coor.y * zoomfactor );
      }
    }
    maxx+=64;
    maxy+=64;
    System.out.println("Size:"+maxx+","+maxy);

    submapImage = createImage(maxx, maxy);      // Neue Map
    submapScroller.setImage(submapImage);
    submapGraphics = submapImage.getGraphics();

    // Zeichnen der Links
    symbols = submap.children.elements();                //ihh kein direkter
Zugriff
    while(symbols.hasMoreElements()) {
      work = (Symbol) symbols.nextElement();
      if( work instanceof Link ) {
          work.paint(submapScroller, submapGraphics, zoomfactor);
      }
    }
    // Zeichnen der Nodes
    symbols = submap.children.elements();                //ihh kein direkter
Zugriff
    while(symbols.hasMoreElements()) {
      work = (Symbol) symbols.nextElement();
      if( work instanceof Node ) {
          work.paint(submapScroller, submapGraphics, zoomfactor);
      }
    }

  }

  // ***********************************************************************

  public synchronized void repaintSymbol(Symbol obj) {
    obj.paint(submapScroller, submapGraphics, zoomfactor);
```

```
    if( obj instanceof Link ) {
      Enumeration nodes;
      nodes = ((Link) obj).getnodeobjs();
      ((Symbol) nodes.nextElement()).paint(submapScroller, submapGraphics,
zoomfactor);
      ((Symbol) nodes.nextElement()).paint(submapScroller, submapGraphics,
zoomfactor);
    } else {
      System.out.println("Unbekanntes Symbol");
    }
    submapScroller.setImage(submapImage);
    super.show();
  }

  // ***************************************************************************

  public void topit() {
    this.toFront();
  }

  // ***************************************************************************
  // Menüereignisse
  // ***************************************************************************
  public void nothelp() {
    mapadmin.requestStateOfSubmap(submap);
    paintSubmap();
  }
  public void closeSubmap() {
    System.out.println("Close");
    mapadmin.removeSubmapDisplay(submap);
    this.dispose();
  }

  // ***************************************************************************

  private void zoomin() {
    System.out.println("zoomIn");
    zoomfactor*=1.5;
    show();
  }

  // ***************************************************************************

  private void zoomout() {
    System.out.println("zoomout");
    zoomfactor/=1.5;
    show();
  }

  // ***************************************************************************

  private void rootSubmap() {
    System.out.println("root");
    if( mapadmin.openSubmapDisplay(mapadmin.root) == false ) {
      System.out.println("WUAH: Root Fehler");
    }
  }

  // ***************************************************************************

  private void autolayoutConfig() {
    System.out.println("al config");
  }

  // ***************************************************************************
```

```
private void autolayout() {
  System.out.println("al");
}

// ***********************************************************************

private void enterSubmap() {
  System.out.println("enter submap");
  if( mapadmin.openSubmapDisplay(selected) == false ) {
    mapadmin.openSymbolDisplay(selected);
  }
}

// ***********************************************************************

private void goup() {
  System.out.println("go to parent submap");
  Enumeration parents = submap.getparentobjs();
  if( parents.hasMoreElements()) {
    Symbol firstparent = (Symbol) parents.nextElement();
    if( mapadmin.openSubmapDisplay(firstparent) == false ) {
        System.out.println("WUAH: Parent Fehler");
    }
  }
}

// ***********************************************************************

private void showSymbolDisplay() {
  System.out.println("symbol");
}

// ***********************************************************************
// Mausereignisse
// ***********************************************************************

private void selectSymbol(Symbol obj) {
  if(selected != null) {
    if( selected instanceof Node) {
        selected.selected = false;
    }
    repaintSymbol(selected);
    if( selected instanceof Link) {
        selected.selected = false;
    }

    if(selected.id != obj.id ) {
        obj.selected = true;
        repaintSymbol(obj);
        selected = obj;
    } else {
        selected = null;
    }
  } else {
    obj.selected = true;
    repaintSymbol(obj);
    selected = obj;
  }

}

// ***********************************************************************
// GUI/AWT Methoden
// ***********************************************************************

public void addNotify() {
```

```
    super.addNotify(); // DON'T OMIT THIS!
  }

  // ***************************************************************************
  // Event Handling, WO ist ein GUI Event aufgetreten.
  // ***************************************************************************

  public boolean handleEvent(Event evt)
    {
      /*System.out.println("Event in:"+evt.target.getClass().getName());
          System.out.println("Event:"+ evt.toString()); */
      if(evt.id == Event.MOUSE_UP) {
          if(evt.target == zoomOutButton) {
            zoomout();
          } else if(evt.target == zoomInButton) {
            zoomin();
          } else if(evt.target == closeButton) {
            closeSubmap();
          } else if(evt.target == rootButton) {
            rootSubmap();
          } else if(evt.target == upButton) {
            goup();
          } else if(evt.target == layoutButton) {
            autolayoutConfig();
          } else if(evt.target == helpButton) {
            nothelp();
          }
      }
      if (evt.target instanceof ViewPanel) {    /*sigh*/
          switch(evt.id) {
          case Event.MOUSE_DOWN:
            Symbol klickSymbol;
            Point klick;
            klick = klickPoint(evt);
            klickSymbol = submap.findSymbolat(klick,zoomfactor);
            if( klickSymbol != null ) {
              if( evt.clickCount > 1 || evt.modifiers == Event.META_MASK ) {
                selected = klickSymbol;
                enterSubmap();
              } else {
                selectSymbol(klickSymbol);
                System.out.println("Shading:"+klickSymbol.getid());
              }
            }
          }
      }
      return true;
    }

  // ***************************************************************************

  public Point klickPoint(Event evt) {
    Point p, klick;
    klick=submapScroller.location();
    p = new Point(evt.x-
klick.x+submapScroller.getHorizontalScrollbar().getValue()
                    ,evt.y-
klick.y+submapScroller.getVerticalScrollbar().getValue());
    return p;
  }

}
```

7.4.6. StateFilter

```
// Klassenname: StateFilter
```

```
// Autor      : Roland Spatzenegger
// Beginn     : 29.7.97
// Last Change: 29.7.97
// Version    : 0.1
// Bemerkung  : Aendert die Farbe eines Bildes in eine andere

// ***************************************************************************
// Imports
// ***************************************************************************

import java.lang.*;
import java.awt.image.*;
import java.awt.*;

public class StateFilter extends RGBImageFilter{

  // ***************************************************************************
  // Public Attribute
  // ***************************************************************************

  public int from;
  public int to;

  // ***************************************************************************
  // Constructor
  // ***************************************************************************

  public StateFilter(Color from,  Color to) {
    this.from = from.getRGB();
    this.to = to.getRGB();
  }

  // ***************************************************************************
  // Methoden
  // ***************************************************************************

  public int filterRGB (int x, int y, int rgb) {
    if( rgb == from) {
      return(to);
    } else {
      return(rgb);
    }
  }
}
```

7.4.7. MapAdmin

```
// Klassenname: MapAdmin
// Autor      : Roland Spatzenegger
// Beginn     : 2.7.97
// Last Change: 29.7.97
// Version    : 0.2
// Bemerkung  :

// ***************************************************************************
// Imports
// ***************************************************************************

import Symbol;
import Gitterlayout;
import Link;
import DBInterface;
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.io.*;
```

```
import java.net.*;

public class MapAdmin {

  // ***********************************************************************
  // Public Attribute
  // ***********************************************************************

  public Node root;
  public Hashtable symbols;
  public Hashtable submapDisplays;
  public Hashtable symbolDisplays;
  //      public AutoLayout autolayout;
  public DBInterface inmsdb;
  //      public StatusInterface inmsserver;

  // ***********************************************************************
  // Constructor
  // ***********************************************************************

  public MapAdmin(int port) {
    root = new Node();
    root.id = "The Root";
    root.coor = new Point (40,40);
    submapDisplays = new Hashtable();
    symbolDisplays = new Hashtable();
    inmsdb = new DBInterface();
    inmsdb.openConnection("127.0.0.1", port, "nmsadm", "nmsadm");
    requestSubmap(root);
  }

  // ***********************************************************************
  // Methoden
  // ***********************************************************************

  public boolean openSubmapDisplay(Symbol submap) {
    SubmapDisplay newdisplay,olddisplay;
    olddisplay = (SubmapDisplay) submapDisplays.get(submap.id);
    if( olddisplay != null ) {
      olddisplay.topit();
    } else {
      if( submap.hasSubmap && submap.children.size() == 0 ) {
          requestSubmap(submap);
      }
      if( submap.children.size() > 0 ) {
          Gitterlayout layout;
          layout = new Gitterlayout();
          layout.startAutolayout(submap);
          newdisplay = new SubmapDisplay(this,submap);
          newdisplay.show();
          submapDisplays.put(submap.id,newdisplay);
      } else {
          return false;
      }
    }
    return true;
  }

  public void removeSubmapDisplay(Symbol submap) {
    SubmapDisplay olddisplay;
    olddisplay = (SubmapDisplay) submapDisplays.get(submap.id);
    if( olddisplay != null ) {
      submapDisplays.remove(submap.id);
    } else {
      System.out.println("WUAH:Fehler in der SubmapDisplayliste");
    }
```

```
}
// ***************************************************************************

public void openSymbolDisplay(Symbol obj) {
  System.out.println("Symbol Display:"+obj.getid());
}

// ***************************************************************************

public void openNetconfig() {
  System.out.println("Netconfig\n");
}

// ***************************************************************************

public void startAutolayout(Symbol submap) {
  System.out.println("Autolayout\n");
}

// ***************************************************************************

public void openAutolayoutConfig(Symbol submap) {
  System.out.println("AutolayoutConfig\n");
}

// ***************************************************************************

public void requestSubmap(Symbol submap) {
  Hashtable children;
  Enumeration symbols;
  Symbol work;

  children = inmsdb.requestSubmap(submap.id);
  if(children == null) {
    return;
  }
  symbols = children.elements();
  while(symbols.hasMoreElements()) {
    work = (Symbol) symbols.nextElement();
    if( work instanceof Node ) {
        submap.addChild(work);
    }
  }
  symbols = children.elements();
  while(symbols.hasMoreElements()) {
    work = (Symbol) symbols.nextElement();
    if( work instanceof Link ) {
        submap.addChild(work);
    }
  }
  requestStateOfSubmap(submap);
}

// ***************************************************************************

public void requestStateOfSubmap(Symbol submap) {
  Hashtable states;
  Enumeration work;
  Symbol changed;
  String symid;

  states = inmsdb.requestStateOfSubmap(submap.getid());
  if(states == null) {
    return;
  }
  work = states.keys();
```

```
    while(work.hasMoreElements()) {
      symid = (String) work.nextElement();
      changed = (Symbol) submap.searchSymbol(symid);
      if(changed != null) {
         changed.state.setinternstate(Integer.valueOf(
           (String) states.get(symid)).intValue());
      } else {
         System.out.println("Not found (state):"+symid);
      }
    }
  }

  // **************************************************************************
  // Attribut Schreiboperationen
  // **************************************************************************
  public void insertsymbol(Symbol obj,String parent) {
  }

}
```

7.4.8. AutolayoutAlgo

```
// Klassenname: AutolayoutAlgo
// Autor      : Roland Spatzenegger
// Beginn     : 25.7.96
// Last Change: 25.7.96
// Version    : 0.1
// Bemerkung  :

// **************************************************************************
// Imports
// **************************************************************************

import java.lang.*;
import java.awt.*;

public abstract class AutolayoutAlgo extends Object {

  // **************************************************************************
  // Public Attribute
  // **************************************************************************

  public String id;

  // **************************************************************************
  // Constructor
  // **************************************************************************

  public  AutolayoutAlgo() {
  }

  // **************************************************************************
  // Methoden
  // **************************************************************************

  public abstract void startAutolayout(Symbol submap);
  //  public abstract Panel configure();

}
```

7.4.9. Gitterlayout

```
// Klassenname: Gitterlayout
// Autor      : Roland Spatzenegger
// Beginn     : 25.7.96
```

```
// Last Change: 25.7.96
// Version    : 0.1
// Bemerkung  :

// ***************************************************************************
// Imports
// ***************************************************************************

import java.lang.*;
import java.util.*;
import java.awt.*;

public class Gitterlayout extends AutolayoutAlgo {

// ***************************************************************************
// Public Attribute
// ***************************************************************************

  public String id;

// ***************************************************************************
// Constructor
// ***************************************************************************

  public Gitterlayout() {
    id = "Gitterle";
  }

// ***************************************************************************
// Methoden
// ***************************************************************************

  public void startAutolayout(Symbol submap) {
    Hashtable nodelist;
    Enumeration symbols;
    Symbol work;
    Node node;
    int x,y;
    int linecount;

    nodelist = new Hashtable();

    // erst alle Nodes herausholen
    symbols = submap.children.elements();
    while(symbols.hasMoreElements()) {
      work = (Symbol) symbols.nextElement();
      if( work instanceof Node ) {
          nodelist.put( work.getid(), work );
      }
    }
    linecount = (int) Math.sqrt(nodelist.size())+1;
    symbols = nodelist.elements();
    for(x = 0; x < linecount && symbols.hasMoreElements(); x++) {
      for(y = 0; y < linecount && symbols.hasMoreElements(); y++) {
          node = (Node) symbols.nextElement();
          node.coor.x = x*128+64;
          node.coor.y = y*128+64;
      }
    }
  }

  //  public Panel configure(Symbol submap) {
  // }
}
```

7.4.10. DBInterface

```
// Klassenname: DBInterface
// Autor      : Roland Spatzenegger
// Beginn     : 13.7.97
// Last Change: 22.7.97
// Version    : 0.2
// Bemerkung  :

// ***********************************************************************
// Imports
// ***********************************************************************

import Symbol;
import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;
import DBValue;

public class DBInterface extends Object {

  // ***********************************************************************
  // Privat Attribute
  // ***********************************************************************

  private Socket socket;
  private DataInputStream in;
  private DataOutputStream out;

  // ***********************************************************************
  // Constructor
  // ***********************************************************************

  public DBInterface() {
  }

  // ***********************************************************************
  // Methoden
  // ***********************************************************************

  public String readline() {
    String line;
    line = " ";
    do {
      try {
          line = in.readLine();
      } catch(Exception e) {
          e.printStackTrace();
      }
    } while(line.length()<=2);
    if(line != null ) {
      System.out.println("IN:"+line);
    }
    return(line);
  }

  public void writeline(String line) {
    try {
      out.writeBytes(line);
    } catch(Exception e) {
      e.printStackTrace();
    }
    System.out.print("OUT:"+line);
  }
```

```
  // **************************************************************************

  public Stack decode(Stack nachricht, String firstline) {
    String tag,command,param,line;
    StringTokenizer linetoken;

    linetoken = new StringTokenizer(firstline," ");
    tag = linetoken.nextToken();
    while(!tag.equals("start")) {
      line = readline();
      linetoken = new StringTokenizer(line," ");
      tag = linetoken.nextToken();
    }

    // Start Tag zerlegen

    command = linetoken.nextToken();
    nachricht.addElement(command);

    // Kurznachricht ?

    if( linetoken.hasMoreTokens() ) {
      param = linetoken.nextToken();
      if(param.equals("end")) {
          nachricht.addElement(param);
          return(nachricht);
      } else {
          System.out.println("ERROR: Falscher END Token");
          return(nachricht);
      }
    } else {
      // Body zerlegen
      do {
          line = readline();
          // Zeile zerlegen
          linetoken = new StringTokenizer(line," ");
          tag = linetoken.nextToken();
          // Nachricht in Nachricht ?
          if(tag.equals("start")) {
            nachricht = decode(nachricht,line);
          } else if(!tag.equals("end")) {
            nachricht.addElement(new DBValue(line));
          }
      }while(!line.equals("end"));
      nachricht.addElement("end");
    }
    return(nachricht);
  }
  // **************************************************************************

  public boolean openConnection(String adresse, int port, String name, String
password) {
    // Neuen Socket+Streams generieren
    Stack nachricht;
    try {
      socket = new Socket(adresse,port,true);
      in = new DataInputStream(socket.getInputStream());
      out = new DataOutputStream(socket.getOutputStream());
    } catch(Exception e) {
      e.printStackTrace();
      return false;
    }
    send_open(name,password);
    nachricht = decode(new Stack(),"dummy");
    if( ((String) nachricht.pop()).equals("ok") ) {
      return true;
```

```
    } else {
      System.out.println(((String) nachricht.pop()));
      return(false);
    }
  }

  // *********************************************************************

  public boolean closeConnection() {
    send_close();
    return(true);
  }

  // *********************************************************************
  public Stack swapstack(Stack stack) {
    Stack newstack;
    newstack = new Stack();
    while(!stack.empty()) {
      newstack.push(stack.pop());
    }
    return(newstack);
  }
  public Hashtable requestSubmap(String id) {
    Stack nachricht;
    send_getsubmap(id);
    nachricht = decode(new Stack(),"dummy");
    nachricht = swapstack(nachricht);

    if( ((String) nachricht.pop()).equals("symbollist") ) {
      // Symbolliste im Anflug
      Hashtable symbollist;
      symbollist = new Hashtable();
      Symbol symbol;
      while((symbol = decodeSymbol(nachricht)) != null) {
          symbollist.put(symbol.getid(),symbol);
      }
      return symbollist;
    } else {
      return(null);
    }
  }
  public Hashtable requestStateOfSubmap(String id) {
    Stack nachricht;
    send_getstatusofsubmap(id);
    nachricht = decode(new Stack(),"dummy");
    nachricht = swapstack(nachricht);
    if( ((String) nachricht.pop()).equals("statelist") ) {
      // Stateliste im Anflug
      Hashtable statelist;
      statelist = new Hashtable();
      while(decodeState(nachricht,statelist)) {
      }
      return statelist;
    } else {
      return(null);
    }
  }
  // *********************************************************************
  public void getImagename(Node node) {
    String type;
    type = (String) node.information.get("type");
    if( type.equals("administrative") ) {
      node.imagename = "symbols/admindom.gif";
    } else if( type.equals("geographical") ) {
      node.imagename = "symbols/geodom.gif";
    } else if( type.equals("organisation") ) {
```

```
      node.imagename = "symbols/orgdom.gif";
    } else if( type.equals("generic") ) {
      node.imagename = "symbols/gendom.gif";
    } else if( type.equals("HICOM") ) {
      node.imagename = "symbols/hicomdom.gif";
    } else if( type.equals("LAN") ) {
      node.imagename = "symbols/landom.gif";
    } else if( type.equals("technological") ) {
      node.imagename = "symbols/techdom.gif";
      //           } else if( type.equals("") ) {
      //                    node.imagename = "symbols/.gif";
    } else if( type.equals("HicomNode") ) {
      node.imagename = "symbols/hicom.gif";
    } else if( type.equals("OtherNode") ) {
      node.imagename = "symbols/other.gif";
    } else if( type.equals("OtherIPNode") ) {
      node.imagename = "symbols/otherip.gif";
    } else if( type.equals("OtherProxiedNode") ) {
      node.imagename = "symbols/othprox.gif";
    } else {
      node.imagename = "symbols/genewsm.gif";
    }
  }
  // ***********************************************************************
  public boolean decodeState(Stack msg,Hashtable statelist) {
    String end;
    DBValue node,level;

    if( ((String) msg.peek()).equals("state") ) {
      end = (String) msg.pop();
      node = (DBValue) msg.pop();
      level = (DBValue) msg.pop();
      statelist.put(node.value,level.value);
      end = (String) msg.pop();
      return true;
    } else {
      return false;
    }
  }
  // ***********************************************************************
  // Holt aus dem Nachrichtenstack ein Objekt raus
  public Symbol decodeSymbol(Stack msg) {
    String end;
    if( ((String) msg.peek()).equals("node") ) {
      Node sym;
      sym = new Node();
      DBValue read;
      end = (String) msg.pop();
      read = (DBValue) msg.pop();
      sym.id = new String(read.value);
      read = (DBValue) msg.pop();
      sym.information.put(read.key, read.value);
      read = (DBValue) msg.pop();
      sym.information.put(read.key, read.value);
      read = (DBValue) msg.pop();
      if( read.value.equals("true") ) {
          sym.hasSubmap = true;
      } else {
          sym.hasSubmap = false;
      }
      end = (String) msg.pop();
      getImagename(sym);
      System.out.println("Symbol:"+sym.id+" "+read.value);
      return(sym);
    } else if( ((String) msg.peek()).equals("link") ) {
      Link sym;
```

```
      sym = new Link();
      DBValue read;
      end = (String) msg.pop();
      read = (DBValue) msg.pop();
      sym.id = new String(read.value);
      read = (DBValue) msg.pop();
      sym.information.put(read.key, read.value);
      read = (DBValue) msg.pop();
      sym.information.put(read.key, read.value);
      read = (DBValue) msg.pop();
      sym.inserttmpNode(read.value);
      read = (DBValue) msg.pop();
      read = (DBValue) msg.pop();
      sym.inserttmpNode(read.value);
      read = (DBValue) msg.pop();
      read = (DBValue) msg.pop();
      if( read.value.equals("true") ) {
         sym.hasSubmap = true;
      } else {
         sym.hasSubmap = false;
      }
      end = (String) msg.pop();
      return(sym);
    } else  {
    }
    return(null);
  }

  // ***********************************************************************

  /*      public Symbol requestSymbol(String id) {
          } */

// ***********************************************************************

  /*      public Hashtable requestallStatus(String id) {
          }
  */
  // ***********************************************************************
  /*
    public Status requestStatus(String id) {
    }
  */
  // ***********************************************************************
  // Senderoutinen
  // ***********************************************************************

  public void send_open(String username,String password) {
    writeline("start open\n");
    writeline("username="+username+"\n");
    writeline("password="+password+"\n");
    writeline("end\n");
  }

  public void send_close() {
    writeline("start close end\n");
  }

  public void send_getsubmap(String id) {
    writeline("start getsubmap\n");
    writeline("submapid="+id+"\n");
    writeline("end\n");
  }

  public void send_getsymbol(String id) {
    writeline("start getsymbol\n");
```

```
    writeline("symbolid="+id+"\n");
    writeline("end\n");
  }

  public void send_getstatusofsubmap(String id) {
    writeline("start getstatusofsubmap\n");
    writeline("submapid="+id+"\n");
    writeline("end\n");
  }

  public void send_getstatusofsymbol(String id) {
    writeline("start getstatusofsymbol\n");
    writeline("symbolid="+id+"\n");
    writeline("end\n");
  }

}
```

7.4.11. DBValue

```
// Klassenname: DBValue
// Autor      : Roland Spatzenegger
// Beginn     : 22.7.97
// Last Change: 22.7.97
// Version    : 0.1
// Bemerkung  :

// ***********************************************************************
// Imports
// ***********************************************************************

import java.lang.*;
import java.util.*;

public class DBValue extends Object {

  // ***********************************************************************
  // Public Attribute
  // ***********************************************************************

  public String key;
  public String value;

  // ***********************************************************************
  // Constructor
  // ***********************************************************************

  // kv::=<attr key>=<attr value>

  public DBValue(String kv) {
    StringTokenizer token;
    token = new StringTokenizer(kv,"=");
    key = token.nextToken();
    if( token.hasMoreTokens() ) {
      value = token.nextToken();
    } else {
      value = "Unknown";
    }
  }
}
```

7.4.12. VOMI

```
// Klassenname: VOMI
// Author     : Roland Spatzenegger
// Beginn     : 2.6.97
```

```
// Last Change: 8.7.97
// Version    : 0.3
// Bemerkung  : main() von Visualisation of Management Information

// ***************************************************************************
// Imports
// ***************************************************************************

import java.lang.*;
import java.awt.*;
import SubmapDisplay;

public class VOMI {

  // ***************************************************************************
  // Constructor
  // ***************************************************************************

  public VOMI() {
  }
  public static void main(String args[]) {
    SubmapDisplay Submap1;
    MapAdmin mapadmin;
    Symbol root;
    mapadmin = new MapAdmin(Integer.valueOf(args[0]).intValue());
    boolean back=mapadmin.openSubmapDisplay(mapadmin.root);
  }
}
```

7.4.13. Server

Der Serversourcecode besteht zum größten Teil aus vorhandenen INMS Sourcecode.