Patrick Kosiol

Praxissemesterbericht

Hochschule Harz, Wernigerode
Hochschule für angewandte Wissenschaften (FH)

1. Praxissemester

Studiengang Kommunikationsinformatik, Fachrichtung Distributed Computing

5. Fachsemester

Projekt Bfpl

Elektronisches **B**ild**f**ahr**pl**an Informationssystem

Vorgelegt von: Patrick Kosiol

Friedrichstr. 54 Wohnheim 4, Zimmer 301

Matrikelnummer: 6607

patrick.kosiol@gmx.de

Betreuer der Hochschule: Prof. Dr. B. Zimmermann

Praxisstelle: IBS – Ingenieurbüro für Bahnbetriebssysteme GmbH

Roscherstr. 7

30161 Hannover

Betreuer der Praxisstelle: Prof. Dr. H. Dannenberg

Zeitraum des Praktikums: 02.09.2002 – 31.01.2003

Eingereicht am: 28. Februar 2003

Inhaltsverzeichnis

0. Einleitung / Allgemeines

Die Ziele des ersten Praxissemesters sind das Kennenlernen betrieblicher Abläufe sowie das produktive und wirtschaftliche Arbeiten an einem Projekt. Die daraus entstandene Arbeitsleistung soll dem Unternehmen dienen und es in die Lage versetzen, daraus einen wirtschaftlichen Nutzen zu ziehen.

1. Firmenprofil [1)]

IBS (Ingenieurbüro für Bahnbetriebssysteme) ist seit der Gründung im Jahr 1994 vor allem als Consulting- und Softwaredienstleister für Eisenbahn- und Nahverkehrsunternehmen tätig.
Das Unternehmen mit Sitz in Hannover unterstützt die Bahnen vor allem mit leistungsfähiger Simulationssoftware bei der Lösung ihrer komplexen betrieblichen Aufgaben.
Durch Simulation von Zugfahrten kann ein Fahrplan entwickelt und analysiert werden. Dies umfasst die Disposition von Triebfahrzeugen, die Optimierung der Infrastruktur, die Instandhaltung sowie die Umlaufplanung.
Durch den Einsatz der Planungsinstrumente können die Bahngesellschaften ihre eigenen Produkte weiter optimieren und einen hochwertigen, zuverlässigen, sicheren und wirtschaftlichen Schienenverkehr gewährleisten.

Die Arbeitsschwerpunkte von IBS sind Planungs- und Beratungsleistungen sowie Softwareentwicklungen in den folgenden Bereichen:

- Eisenbahnbetriebssimulationen
- Entwicklung und Vergleich von Fahrplankonzepten, Bemessung von Gleisanlagen und Fahrzeugen
- Baubetriebsplanung
- Planung von Instandhaltungsarbeiten
- Fahrzeugeinsatzplanung und Umlaufoptimierung
- Fahrdynamische Untersuchungen

In diesen Bereichen hat IBS bereits zahlreiche Projekte für nationale und internationale Bahnen durchgeführt.

Die Mitarbeiter von IBS bilden interdisziplinäre Teams aus Ingenieuren und Informatikern, die über umfassende Erfahrungen im Eisenbahnbetrieb, bei Consultingaufgaben und bei der Softwareentwicklung verfügen. Die Infrastruktur- und Betriebsplanungen erfolgen mit speziell für den Bahnbetrieb erstellten Softwarelösungen. Sie werden in Zusammenarbeit mit unseren Kunden weiterentwickelt. Die Kooperation mit Forschungsinstitutionen (z.B. HS Harz) sichert, dass jederzeit die technologisch und wissenschaftlich neuesten Verfahren für die Lösung von Schienenverkehrsproblemen verfügbar sind.

Die von IBS entwickelten Programme sind hinsichtlich der verwendeten Daten zueinander und zu externen Standardprogrammen kompatibel und ermöglichen auch dadurch einen problemlosen Einsatz zur Bearbeitung unterschiedlichster Fragestellungen.

Das Ingenieurbüro für Bahnbetriebsysteme führt Planungen, Studien und Beratung rund um den Eisenbahnbetrieb durch. Zu den Kunden zählen Bahnunternehmen wie die Deutsche Bahn (DB AG) oder Österreichischen Bundesbahnen (ÖBB), Nahverkehrsbahnen, Öffentliche Auftraggeber und Gebietskörperschaften. Über den deutschsprachigen Raum hinaus hat IBS auch zahlreiche Arbeiten für internationale Bahnen durchgeführt. Schwerpunkt war die Entwicklung und Prüfung neuer Verfahren und Konzepte, um den Eisenbahnbetrieb kostengünstiger und damit wirtschaftlicher zu gestalten.

Neben der Beratung entwickelt IBS Programmsysteme, die die Mitarbeiter von Bahnen bei der Planung des laufenden Betriebes unterstützen. Das Ziel ist, die bei Ihnen vorhandenen Daten intelligent zu verknüpfen und effizient zu nutzen. Beispiele sind das Simulationsmodell SIMU und das Baustellenplanungssystem BAUPLAN. Diese Programmsysteme werden direkt bei Bahnen (z.B. ÖBB, DB AG) eingesetzt, um laufende Planungsaufgaben durchzuführen. Aktuelle Infrastruktur- und Fahrplandaten werden dabei importiert und stehen dem Anwender ohne zusätzlichen Bearbeitungsaufwand zur Verfügung. Diese Verfahren werden jeweils an die speziellen Bedürfnisse und Randbedingungen unserer Kunden angepasst.

Geschäftsführer sind Prof. Dr.-Ing. Hartmut Dannenberg, Dr.-Ing. Volker Klahn und Dr.-Ing. Jürgen Hörstel.

Zur Sicherung und stetigen Verbesserung der Qualität der Dienstleistungen und Produkte wurde bei IBS ein Qualitätsmanagementsystem eingeführt, das gemäß DIN EN ISO 9001 zertifiziert ist.

2. Beschreibung des Projekts bzw. der Aufgabenstellung

2.1 Aufgabenstellung vor Antritt des Praxissemesters

Zahlreiche Eisenbahnunternehmen setzen Planungssoftware zur Erstellung ihrer Fahrpläne ein. Sie werden i. a. in Datenbanken hinterlegt und können für andere Anwendungen exportiert und ausgedruckt werden. Im Rahmen dieses Praktikums soll ein bestehendes Programm zur Bereitstellung von Bildfahrplänen im Intranet eines Eisenbahnunternehmens erweitert werden.
Ziel ist es, die Handhabung zu verbessern und neue Funktionen zur Layoutgestaltung bereitzustellen.
Aufbauend auf dem vorhandenen JAVA – Programm soll die Anzeige so verbessert werden, dass Beschriftungen (Zugnummern, Minutenzahlen, …) möglichst überschneidungsfrei ausgegeben werden. Weiterhin soll der Bearbeiter die Möglichkeit erhalten, den Plan ganz oder in Ausschnitten auszudrucken (Auswahlmaske mit Layoutfunktionen, wie Größe, Farben, auszugebende Daten, Legende, …). Weiterhin soll das Programm Fahrpläne übernehmen können, die mit dem von IBS entwickelten Betriebssimulationsprogramm SIMU erstellt wurden. Die genaue Gestaltung der erweiterten Programmfunktionen ist während des Praktikums mit dem Betreuer bei IBS abzustimmen.

2.2 Änderungen der Aufgabenstellung

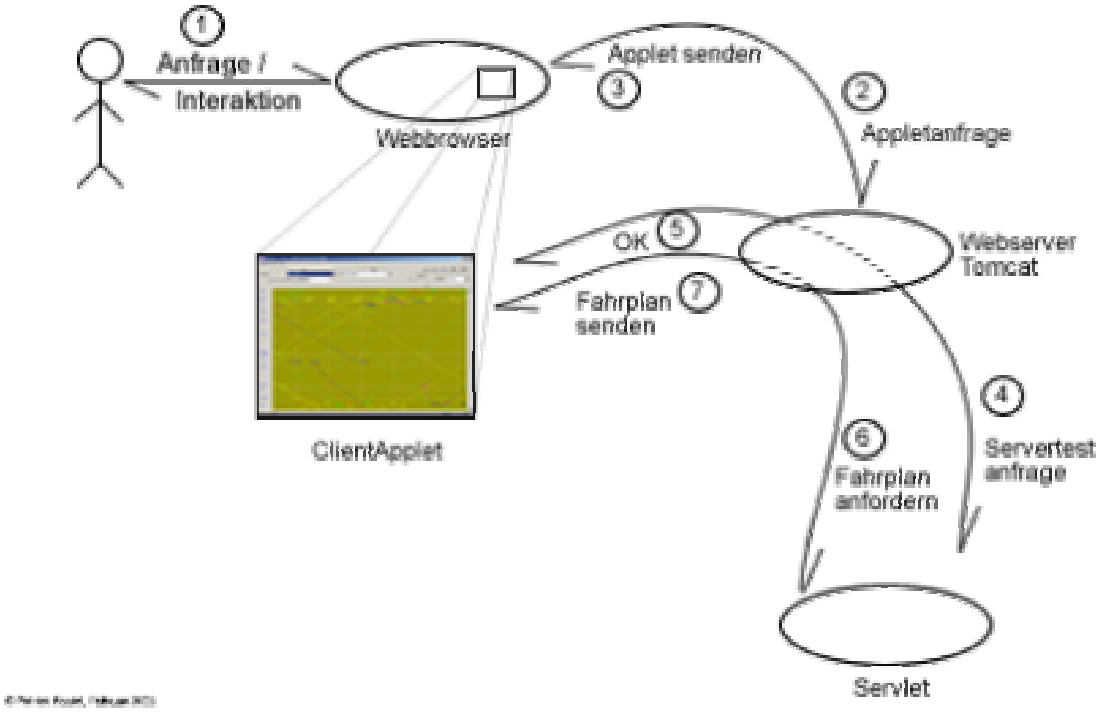
Während des Praxissemesters ist dann aufgefallen, dass eine Erweiterung der Aufgabenstellung nötig war. Diese angesprochenen Änderungen werden im Laufe des Berichts erläutert, da sie sich aus dem Kontext her ergaben und somit an dieser Stelle aus der Luft gegriffen wären.

3. Bisherige Realisierungen

Aufgrund des vorgefundenen Entwicklungsordners des Vorgängers wurde versucht, die schon vorhandene Realisierung funktionsfähig zu installieren. Es existierten auch Installationsanweisungen für den Server sowie für den Client, allerdings erwiesen sie sich als nicht sehr sinnvoll.

3.1 Die Grundsätzliche Idee des Systems

Ein Webserver stellt eine Applikation und Fahrpläne für die Clients eines Intranets zur Verfügung. Hierbei sollte darauf geachtet werden, dass der Installationsaufwand bei den Clients möglichst gering gehalten wird. Die einzig notwendige Installation bei dem Client ist das JAVA Runtime Environment von SUN Microsystems. Ein Benutzer an einem Client-Rechner soll per Browser auf eine HTML – Seite zugreifen, welche auf einem Webserver liegt. Diese HTML – Seite enthält die Anweisungen ein Applet von genau diesem Server herunter zuladen und zu starten. Dieser Server antwortet auf die Anfrage indem er ein Applet zu dem Client-Browser zurück schickt. Dieses Applet nimmt wiederum Kontakt zu dem Server (denkbar ist aber auch ein anderer entfernter Server) auf und stellt eine Verbindung mit einem Servlet (auf dem Webserver) her, welches die angeforderten Streckendaten zur Verfügung stellt und an das Applet sendet. Dieses Applet kann dann die Streckendaten darstellen und bietet diverse Anzeigemöglichkeiten, die dem Benutzer die Anwendung des Systems und die Betrachtung der Daten erleichtert.
Die Daten, welche der Webserver zur Verfügung stellt, stammen aus Datensätzen anderer IBS – Programme. Somit sind diese Datensätze zwischen den Programmen kompatibel. Aufgrund der enormen Größe der einzigen Daten – Datei (IBS – spezifisches Datenformat) wird diese mit Hilfe eines externen C++ – Programms aufgespaltet und es werden kleinere Fahrpläne für den gewünschten Verkehrstag erstellt. Diese Daten werden dann über das Intranet übertragen. Es wurde genau so realisiert um den Verkehr im lokalen Netz zu begrenzen, da sonst ein immenser Datenstrom verarbeitet werden müsste.
Folgende Illustration demonstriert den beschriebenen Ablauf:



Datenfluss – Diagramm 1: Prinzipielle Funktionsweise des Systems

3.2 Grund des Aufspaltens der Daten

Die angesprochene zentrale Daten – Datei umfasst bei den Referenzdaten beispielsweise ca. 30 Megabytes. Diese Datenmenge mit einem Mal über ein Intranet zu schicken wäre vielleicht gerade noch akzeptabel, aber man stelle sich nur einmal vor, es finden 100 simultane Zugriffe statt. Dann ist das Netz für einige Zeit extrem überfordert und die Benutzung des Systems wird unerträglich. Daraus folgt, dass die Kundenzufriedenheit sinkt und somit auch das Ansehen der Entwicklungsfirma.

3.3 Probleme bei der Inbetriebnahme des Servers

Zur Anwendung kam in der ursprünglichen Version ein Apache[2)] Webserver und ein JServ[2)] PlugIn für diesen Webserver. Mit Hilfe dieses PlugIns war es dann möglich Servlets in einem Container zu ‚deponieren', diese über den Apache[2)] ansprechen zu können und zu benutzen. Des Weiteren ist die JAVA – Archive – Datei (JAR – Datei, beinhaltet den Client) auch in diesem Webserver

untergebracht. Somit ist gewährleistet, dass das Client – Applet auch vom Webserver herunter geladen werden kann.
Zunächst hat es sich als ziemlich kompliziert herausgestellt, den Apache[2)] und danach den JServ[2)] zu installieren und diesen auch in dem Apache[2)] zu registrieren. Außerdem musste in dem JServ[2)] auch noch das Servlet, sowie in dem Apache[2)] das Client – Applet registriert werden.
Aufgrund dieser Schwierigkeiten, der relativ schwierigen Handhabung und der Unmengen an Fehlern, die daraus entstehen können, wurde beschlossen diese Konstruktion zu verwerfen und nicht den Apache[2)] in Verbindung mit dem JServ[2)] PlugIn zu verwenden. Die Entscheidung fiel auf den Tomcat[2)] – Webserver (der gleichen Firma, Apache Software Foundation), aus verschiedenen Gründen:

1. Vereinfachte Handhabung bei der Installation
2. Auch als ‚stand alone' – Server einsetzbar
3. Auch als Apache[2)] PlugIn verfügbar
4. JBuilder 7 Enterprise[3)] unterstützt Echtzeit Servlet Debugging (Besonders wichtig für die Entwicklung)
5. leichte Integration neuer Programmmodule in den Webserver,

Nach dieser Umstellung schien der Server problemlos zu laufen.

3.4 Probleme bei der Inbetriebnahme des Clients

Der Client lief auf keinem der Testrechner. Wie, normaler Weise gefordert, galt nicht der Grundsatz: möglichst keine Installationen bzw. benutzerdefinierte Einstellungen bei dem Client. Aufgrund der Konstruktion des Client – Applets werden Dateien lokal geöffnet (wenn diese auch auf einem Webserver liegen). Genau dies war der Fehler in der Konzeption, da hierdurch das Applet ohne clientseitige Konfiguration nicht lauffähig war. Es konnte nicht initialisiert werden, da dies durch ständige java.security.AccessControlExceptions[4)] verhindert wurde. Dies war absolut nicht akzeptabel und der erste Punkt, der geändert werden musste. Die Funktionsfähigkeit muss gesichert werden.

4. Grundsätzliche Änderungen im System

Wie bereits erwähnt wurde der Webserver von der Apache[2)] / JServ[2)] – Struktur auf den Tomcat[2)] Webserver umgestellt.

4.1 Installationen bzw. Einstellungen bei dem Client sind nicht erwünscht

Des Weiteren gab es aber immer noch das Problem des Clients, welcher sich nicht (ohne clientseitige Konfiguration) starten ließ. Hierbei handelt es sich um das ‚lokale' Öffnen von Dateien (in diesem Fall gif – Dateien welche für das Layout einzelner Buttons benötigt wurden). Es wurde in der Art geändert, dass diese Zugriffe alle über Standart URL-Connections[4)] der JAVA SDK erfolgen. Da diese Zugriffe dann auch über den Webserver laufen wird von dem selben die Verwaltung mehrerer gleichzeitiger Zugriffe geregelt. Dies ist die sicherste Variante um das Zugriffsmanagement zu implementieren. Aufgrund dieser Änderung werden die Dateien nicht mehr ‚lokal' geladen und es ist keine clientseitige Konfiguration nötig.

4.2 Sicherheitspolitik beim Programmieren

Bei Umstrukturierungen in diversen Klassen fiel auf, das eine Vielzahl von member – Variablen ‚public'[4)] deklariert waren. Dies hat durchaus einen Sinn, aber auch nicht an jeder Stelle. Es war teilweise möglich innerhalb einer GUI – Klasse datensensitive Variablen des ‚Workflows' zu manipulieren.
Genau dieser Designfehler zog sich durch den gesamten Quellcode. Solch eine Vorgehensweise ist absolut nicht akzeptabel und musste sofort geändert werden. Natürlich ist diese Art von Umstrukturierung des fast gesamten Quellcodes sehr aufwendig und mühselig, aber auch unvermeidbar. Schwachstellen in der Programmierung machen sich früher oder später bemerkbar.
Somit wurde jede einzelne Klasse angeschaut und als erstes jede wichtige member – Variable als private[4)] deklariert. Danach wurde geschaut, welche

Fehler bei der Kompilierung auftraten und wie genau diese Variablen genutzt wurden. Dort, wo nur Lesezugriffe stattfanden, wurde auch nur eine get – Methode implementiert. Von Fall zu Fall musste genau differenziert werden, wie der Zugriff auf das einzelne Attribut erfolgen sollte.
Ein ähnliches Vorgehen kann man auch bei Netzwerkadministratoren beobachten, die eine Firewall aufsetzen. Als erstes wird sie so konfiguriert, dass alles verboten ist. Danach wird überlegt, und bei Bedarf bestimmte Bereiche (in diesem Fall zu meist Ports) freigegeben. Ähnlich sollte man auch bei der Programmierung vorgehen. Zuerst werden die Zugriffe auf alle Variablen verboten, dann sollte man überlegen, wie Zugriffe erfolgen (meist ergibt sich auch im Laufe der Programmierung, dass es sinnvoller ist etwas freizugeben). Kommen die Zugriffe beispielsweise ausschließlich aus dem eigenen Objekt, dann bleibt das Attribut ‚private'[4)] und kein anderes Objekt kann darauf zugreifen. Ein sehr wichtiges Stichwort ist bei dieser Vorgehensweise das Prinzip des „Information Hiding". Informationen, die nicht unbedingt preisgegeben werden müssen sollten auch geheim gehalten werden. Dieses Vorgehen sichert auch die Portabilität einzelner Programmmodule. Sie kommunizieren nur über speziell definierte Schnittstellen und können somit bei Bedarf schnell ausgetauscht und ersetzt werden. Die Flexibilität eines Programms ist hierdurch gesteigert und erhöht auch die Erweiterbarkeit aller Komponenten.

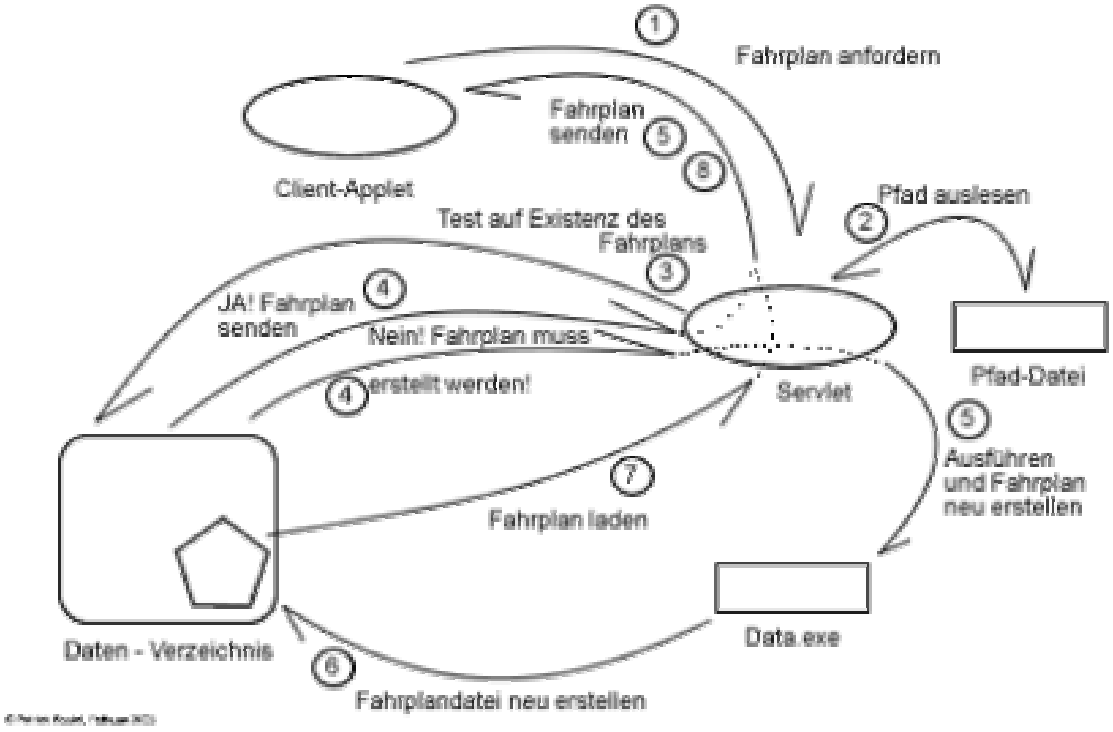
4.3 Komplettes Redesign des Servlets

Nach zwischenzeitlicher Besprechung und Rücksprache mit der Geschäftsführung und dem Projektbetreuer wurde entschieden das Servlet komplett neu zu entwickeln.

4.3.1 Bisheriger Aufbau

Das Servlet liest aus einer Text – Datei, welche in dem Servlet – Verzeichnis (auf dem Webserver) liegt, einen Pfad ein. Dieser Pfad gibt ein Verzeichnis an,

wo die Daten für die benötigten Fahrpläne liegen. Wird ein Fahrplan für einen bestimmten Verkehrstag (meist der aktuelle Tag) angefordert, dann wird überprüft, ob dieser Fahrplan bereits erstellt wurde. Letzten Endes wird getestet, ob eine Datei, dessen eindeutige Bezeichnung die gewünschte Strecke und den Verkehrstag enthält, existiert. Ist dies nicht der Fall dann, wird dieser Fahrplan mit Hilfe eines externen C++ – Programms erstellt und an das anfordernde Applet geschickt.
Folgende Illustration demonstriert den beschriebenen Ablauf:

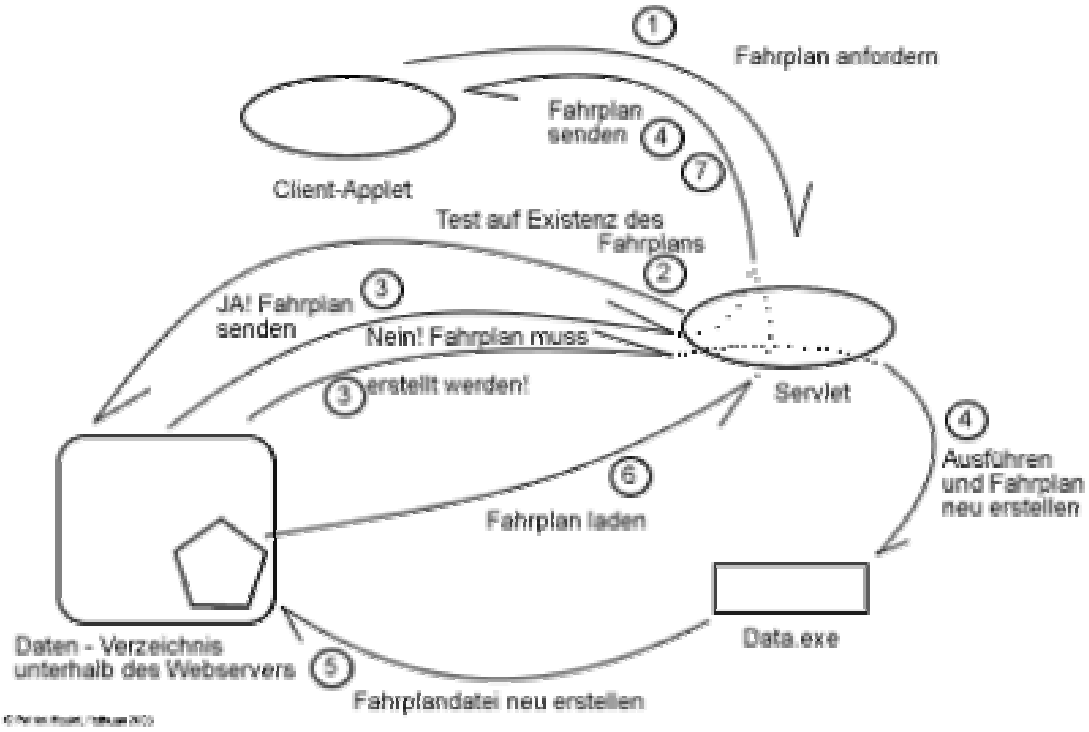


Datenfluss – Diagramm 2: Bisheriger Aufbau des Servlets

4.3.2 Aktueller Aufbau

Das Verfahren, den Pfad aus einer Datei auszulesen (Punkt 2), erschien sehr fehleranfällig. Diese Methode wurde somit entfernt.
Jetzt werden die Fahrplandaten direkt in einem Unterverzeichnis im Webserver abgelegt. Dies vermeidet auch die Benutzung von absoluten Pfaden (wie bei dem alten Verfahren → Punkt 2). Daraus entstand die Möglichkeit die gesamte Konstruktion des Servlets portabel zu gestalten. Relative Pfadangaben sind hierfür eine Notwendigkeit.

Das neue Design des Servlets kann wie folgt illustriert werden:



Datenfluss – Diagramm 3: Aktueller Aufbau des Servlets

Es gab jedoch eine weitere wichtige Eigenschaft, die das Servlet haben musste. Da das gesamte System ja eine Client – Server – Anwendung darstellt, sollen natürlich auch mehrere Instanzen des Servlets möglich sein. Also musste das Design auch darauf ausgerichtet werden. Hierbei musste man nur beachten, dass keine globalen Variablen verwendet wurden, die veränderbar sind. Wenn also globale Variablen benutzt wurden, dann mussten diese auch als ‚final' deklariert werden. Dies gewährleistet, dass einzelne Instanzen der Servlet – Methoden diese globalen Variablen nicht beeinflussen können. Ist dies nicht gewährleistet, dann können Inkonsistenzen entstehen und u. U. nicht nachvollziehbare Fehler auftreten.

Die Verwaltung der Instanzen übernimmt hierbei der Tomcat Webserver. Dieser Servlet – Container ist genau dafür entwickelt wurden und wird bei zahlreichen renommierten Firmen als Webserver eingesetzt. Seine Sicherheitspolitik ist hervorragend und die Konfigurierbarkeit erste Klasse. Die Dokumentation ist zwar etwas gewöhnungsbedürftig aber dennoch sehr komplex und beantwortet bei genauem Studieren alle wichtigen Fragen.

Auch die Ausgabe der Fehlernachrichten, welche ja immer einmal auftreten können, wurde an bestehenden Internetstandards orientiert. Beispielsweise wird

der ‚Error 404' ausgegeben, wenn die doGet() – Methode des Servlets (Standard – Servlet – Methode) mit einer falschen bzw. unbekannten ‚order' – Nummer aufgerufen wird. Darüber hinaus wurde auch sehr viel Wert auf die Fehlerbehandlung bzw. das Abfangen von verschiedensten Exceptions gelegt. Da in dem Servlet sehr oft (fast ausschließlich) Dateizugriffe erfolgen und hierbei immer ein Fehler auftreten kann, sollte dieser natürlich auch abgefangen und protokolliert werden. Bei dem Thema Dateizugriffe muss noch erwähnt werden, dass sie READONLY geschehen, also lediglich mit Leserechten. Hierdurch sind diese Zugriffe unproblematisch.

Die angesprochenen Fehlermeldungen werden in den Log – Dateien des Tomcat[2)] Webservers abgelegt, da die Ausgabekanäle von ihm standardmäßig auf die Tomcat[2)] – eigenen Log – Dateien umgelenkt werden. Dies ist sehr sinnvoll und eine gute Hilfe die Ausgaben kompakt und ständig zur Verfügung zu haben. Nichtsdestotrotz gibt es auch noch einen Ausgabekanal, der auf die Konsole des Tomcat[2)] – Webservers gelegt ist (auch Standard). Hier werden Statusmeldungen abgesetzt, die die aktuellen Zugriffe auf den Webserver zeigen. Somit hat der Administrator des Kunden die Möglichkeit die Zugriffe zu beobachten. Es wird angezeigt, wer als letztes auf das Servlet zugegriffen hat (Identifikation an Hand der IP), die Art des Zugriffs wird ausgegeben (z. B.: Test auf einen bestehenden Fahrplan, Erstellen eines neuen, oder Download eines Fahrplans) sowie der Fahrplan, auf den zugegriffen wurde. Natürlich auch noch diverse Fehler bzw. Exceptions, die abgefangen wurden, damit der Administrator des Webserver den Fehler lokalisieren kann.

5. Erweiterung / Überarbeitung des Systems

5.1 Überarbeiten der Zoomeinstellungen

Dies war eine relativ einfache Aufgabe, sie diente der Verbesserung der Benutzerfreundlichkeit und machte das System intuitiver. Zur Einarbeitung in die bestehenden Ressourcen wurden als erster Schritt die Zoomeinstellungen überarbeitet.
Ursprünglich war das Zoom in der Darstellung in fünf Schritten mittels entsprechenden Schaltflächen realisiert. Da weitere Abstufungen in dem Programm bereits realisiert waren, wurden diese Schaltflächen durch eine Combobox ersetzt, mit der alle verfügbaren Abstufungen wählbar sind.
Die daraus resultierende Bündelung und Erweiterung von Funktionalitäten erhöhte die Bedienbarkeit der graphischen Oberfläche.

5.2 Das farbliche Layout aller Dialoge und des Hauptfensters

Die bestehende Farbwahl war nicht wirklich clever gewählt. Es wurden zu viele grelle Farben verwendet (z.B.: Grün in Verbindung mit Rot). Solche Kombinationen sind nicht nur schlecht für die Augen, sondern verringern auch die Qualität des noch so guten Produktes. Man sollte diese Tatsache nicht unterbewerten. Designtechnische Konsistenz ändert den Eindruck und die Einstellung zu einem Programm und kann es somit attraktiver für den Kunden machen. Die Kundenzufriedenheit darf jedoch nicht außer Acht gelassen werden!
Diese farbtechnische Misere zog sich durch das gesamte System und fast alle Dialoge. Eine Überarbeitung war zwingend nötig.
Das gesamte Layout wurde auf eine standardisierte Farbgestaltung restrukturiert und in der Benutzerfreundlichkeit angepasst, so dass auch nach längerem Arbeiten mit dem Programm die Augen nicht den Dienst quittieren.

5.3 Benutzerdefinierte Einstellung der Zugfarben

Diese Funktion ist wiederum eine schon bestehende Funktion, welche jedoch in Sachen Bedienkomfort zu wünschen übrig ließ. Man musste die Rot-, Grün- und Blauwerte jeweils in dezimaler Form für die einzelnen Zugfarben eingeben. Des Weiteren ist auch noch aufgefallen, dass die Anordnung der Eingabefelder auch noch verdreht ist. So etwas ist keinem Anwender zu zumuten.
Dieser Dialog wurde entfernt und komplett neu gebaut. Somit gibt es jetzt eine Oberfläche, wo man direkt die gewünschte Farbe auswählen kann. Außerdem wurde auch noch die Möglichkeit integriert die Hintergrundfarbe der Hauptfläche den Benutzer wählen zu lassen. Dies schien daher sinnvoll, da manche Zugfarben auf weißem Hintergrund nicht sehr gut erkennbar waren (z.B. Gelb). Außerdem sollte man es dem Benutzer überlassen, wie er das farbliche Layout gestalten möchte.

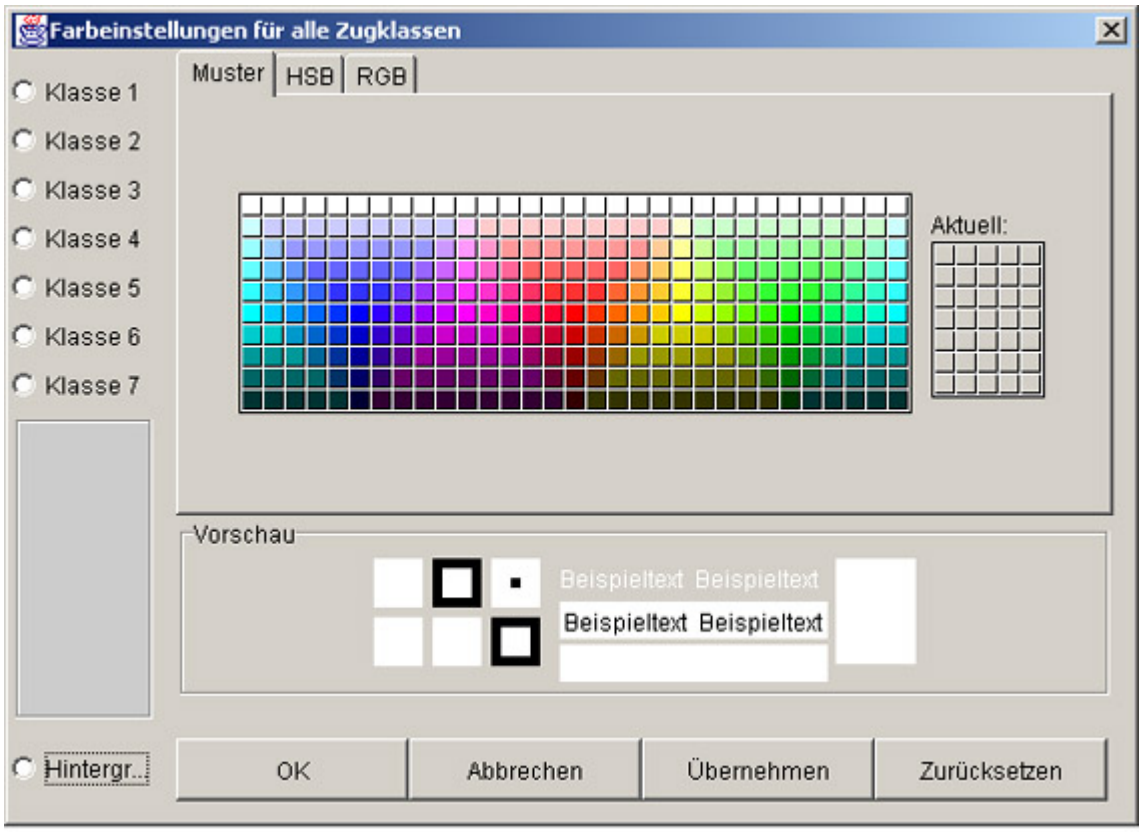


Bild 1: Farbeinstellungsdialog

5.4 Schriftartenvariabilität

Diese Funktion ist neu hinzukommen um dem Benutzer die Wahl der Schriftarten, der Größen sowie des Stils zur Verfügung zu stellen.
Wie in der Abbildung zu sehen gibt es drei verschiedene Bereiche in denen man die Schriftarten einstellen kann. Es war sinnvoll beispielsweise in der Werkzeugleiste alle Elemente zusammen zu fassen, da es sehr umständlich wäre jede Beschriftung gesondert einzustellen.
Des Weiteren sind die Schriftarten auch begrenzt. Es werden nur Standartschriftarten benutzt, da dieses Programm auch unter Linux – Systemen zum Einsatz kommen kann. Somit darf es nicht sein, dass windowsspezifische Schriften ausgewählt werden dürfen. Außerdem ist die Wahl der Schriftgröße auch begrenzt (Größe 8 bis 13), da größere bzw. kleinere Schriften selbst bei anderen Auflösungen nicht verwendbar sind. Die Lesbarkeit sollte in jedem Fall gewahrt bleiben.

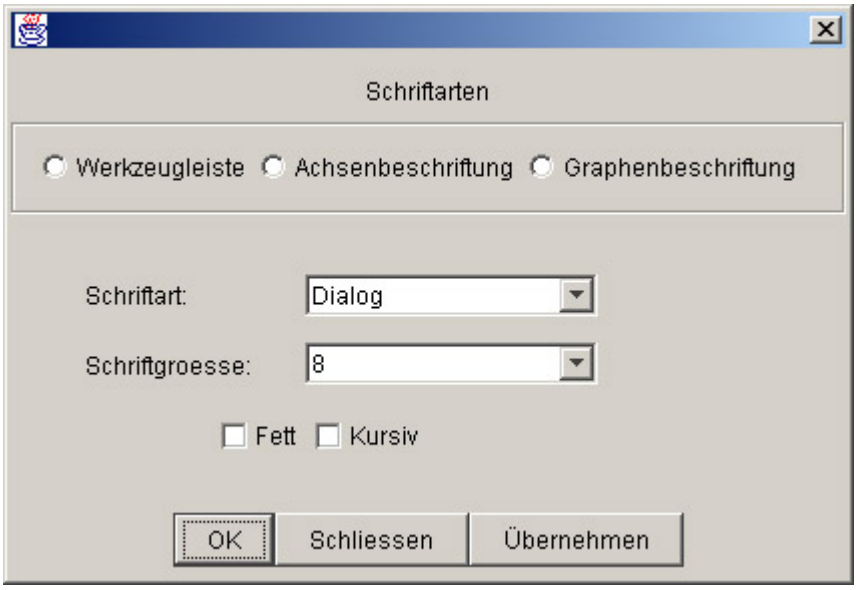


Bild 2: Schriftartendialog

5.5 Dialog für Zugklassendetails

Es wurde unter anderem auch ein Dialog realisiert, der den Kunden über die einzelnen Zugklassen informieren soll. Hierbei wird wiederum mit dem Servlet Kontakt aufgenommen. Die entsprechend benötigten Daten werden dann an

das Applet geschickt und in Form einer Liste ausgewertet und letztendlich visualisiert. Somit hat der Anwender stets einen Überblick über alle Klassen und deren einzelne Zugtypen. Beispielsweise ist ein Zug der ‚Klasse 1' ein ICE.
Hinzu kommt, dass dieser Dialog aus gutem Grund auch nicht als modal deklariert ist, da er während des Arbeitens mit dem System gleichzeitig auch verfügbar sein sollte.

5.6 Minutenanzeige

Diese Funktion ist wiederum komplett neu entwickelt.
Hierbei werden bei den Ein- bzw. Ausfahrten der Züge in bzw. aus einem Bahnhof die jeweilige Minute direkt an dem Zuggraphen angezeigt.
Diese Funktion stellte ein Problem in Bezug auf die Datenaufbereitung dar. Denn die Minuten der einzelnen Bahnhofsdurchfahrten waren zwar in den Rohdaten vorhanden allerdings noch nicht in der Form, die für das System nützlich wäre.
Es ergab sich, dass für jeden Zug ein eigenes Zug – Info – Frame beim Öffnen eines Fahrplans angelegt wird, welches alle Informationen über einen Zug enthält. Diese Vorgehensweise ist nicht besonders geschickt, da man die GUI und das Workflow stets von einander trennen sollte. Allerdings waren die Verwebungen so gravierend, dass sich eine Neuausrichtung nicht lohnte. Somit wurde das Zug – Info – Frame um eine weitere Komponente erweitert: Der Minuten – Vector (JAVA – Klasse ‚Vector').
Es musste von vorne herein darauf geachtet werden, dass die Minutenanzeige standardmäßig abgeschaltet ist und man sie bei Bedarf einschalten kann. Somit sollte dieser Vector nicht beim Einschalten der Anzeige neu gebaut werden. Da dies in Echtzeit sehr performancehemmend und vor allem für ein JAVA – System nicht tragbar ist. Die hier angewendete Vorgehensweise ist in jedem Fall zu empfehlen, da beim Benutzen der Minutenanzeige (bzw. der betreffenden Vectoren) der Aufbau in einem sehr geringen Zeitraum erfolgt, dies funktioniert auch für größere Datenmengen. Jetzt wird einfach nur bei jedem Bahnhofshalt die entsprechende Minute nacheinander an den Graphen geschrieben. Außerdem musste auch beachtet werden, dass die

Übersichtlichkeit verloren gehen kann, wenn ein Zug nur wenige Minuten in einem Bahnhof hält. Hierbei würden sich die Einfahrts- und Ausfahrtsminuten überschreiben und man könnte nichts erkennen. Dieser Tatsache zu Folge wurde eine weitere wählbare Funktion eingebaut, die es erlaubt: Jede Minute, jede zweite (gerade Anzahl) Minute und jede andere zweite (ungerade Anzahl) Minute anzuzeigen. Somit kann man immer umschalten und die Übersichtlichkeit wurde noch benutzerfreundlicher gestaltet.

Die X bzw. Y Koordinaten für die jeweiligen Minuten werden stets relativ von den aktuellen Zuggraphen – Koordinaten berechnet. Somit ist gewährleistet, dass die Darstellung der betreffenden Zahl immer an der richtigen Stelle erfolgt.

6. Fazit und Ausblick

Mir persönlich brachte dieses Praxissemester sehr viele praktische Einblicke in ein sehr gut und professionell agierendes Unternehmen. Ich habe viele programmierseitige Techniken, nicht nur aufgrund der selbständigen Arbeit sondern insbesondere auch in Zusammenarbeit mit meinem Projektbetreuer und anderen Mitarbeitern, erlernt. Die Arbeit hat mir auch Spaß gemacht und das Betriebsklima war hervorragend.

Nach dem Abschluss des Praxissemesters ist eine Software so weit weiterentwickelt und überarbeitet worden, dass sie nun reif für die Vorführung beim Kunden ist. Vor allem die Installation des gesamten Systems wurde extrem vereinfacht und kann somit sehr schnell vorgenommen werden.
Eine Funktion ist noch nicht komplett realisiert wurden. Hierbei handelt es sich um die PDF – Ausgabe eines Fahrplans. Dies ist angedacht und auch schon teilweise im Quellcode enthalten (Dialog ist komplett fertig, sowie clientseitige Klassen wurden teilweise implementiert, doGet() – Orders im Servlet sind schon vorgesehen, allerdings noch nicht vollständig implementiert). Jedoch reichte die Zeit für diesen Part der Aufgabenstellung nicht aus, da sich andere, wichtigere Aspekte im Laufe der Entwicklung ergaben (Sicherheitspolitik, Redesign etc.). Nichtsdestotrotz ist dieses Programm komplett einsatzfähig. IBS ist somit (wie in der Einleitung schon erwähnt) in der Lage einen wirtschaftlichen Nutzen daraus zu ziehen.
Natürlich müsste mit einem potentiellen Kunden verhandelt werden, welche Funktionen noch zusätzlich gewünscht bzw. verändert werden könnten. Dies ist ein Referenzprogramm. Es steht nun IBS zur Verfügung und gilt als Vorschlag an den Kunden, wie man sich ein Elektronisches Bildfahrplan Informationssystem vorstellt, bzw. wie es aussehen könnte.

7. Anhang

Quellcode, Dokumentation und andere programmspezifische Einzelheiten sind Eigentum der IBS GmbH Hannover und dürfen aus datenschutzrechtlichen Gründen nicht veröffentlicht werden.
Soweit nicht anders gekennzeichnet sind die hier erstellten Niederschriften und Diagramme geistiges Eigentum von Patrick Kosiol. Es wird auch versichert, dass dies alles aus dessen Feder stammt.

7.1 Quellen

[1] Firmenprofil: IBS GmbH Hannover http://www.simu.de

[2] Apache, JServ und Tomcat sind eingetragene Warenzeichen der Apache Software Foundation http://www.apache.org

[3] JBuilder ist ein eingetragenes Warenzeichen von Borland http://www.borland.com

[4] siehe JAVA – Dokumentation http://java.sun.com/apis.html

[5] JAVA ist ein eingetragenes Warenzeichen von SUN Microsystems http://java.sun.com

7.2 Benutzte Softwares

- JAVA 2 SDK SE – Version 1.3.1 http://java.sun.com
- JBuilder 7 Enterprise Edition http://www.borland.com
- Tomcat 4.0.4 http://jakarta.apache.org/tomcat/
- Mozilla Websuite (verschiedene Versionen) http://www.mozilla.org

7.3 Benutzte Literatur

- O' Reilly – JAVA Servlet Programmierung, 2nd Edition
  ISBN: 0-596-00040-5 http://www.oreilly.de
- O' Reilly – JAVA Enterprise in a Nutshell
  ISBN: 3-89721-334-6 http://www.oreilly.de
- MITP–Verlag – JAVA 2 Programmers Reference
  ISBN: 3-8266-0716-3 http://www.ge-packt.de
- Markt und Technik – JAVA 2 Kompendium
  ISBN: 3-8272-6039-6 http://www.mut.de
- JBuilder 7 Bücherdokumentation
  http://www.borland.com

7.4 Abbildungsverzeichnis

7.5 Glossar

Administrator

In diesem Fall ist hiermit eine Person gemeint, die einen Computer verwaltet und einrichtet. Sie besitzt sämtliche Rechte um auf diesem Rechner Konfiguration aller Art durchzuführen.

Applet / Client – Applet

Beide Bezeichnungen werden hier verwendet, dabei wird jedoch nicht differenziert. Es handelt sich hierbei um eine JAVA – Applikation, welche speziell für Intranets bzw. das Internet entwickelt wurde. Hierzu wird eine Anwendung (wie z.B.: ein Webbrowser) zum Start des Applets benötigt.

C++

Dies ist eine objektorientierte Programmiersprache. Sie wurde zum Erstellen des externen Programms benutzt, welche die Daten aufspaltet.

Client

Ein Computer, der die Dienste eines anderen Rechners (Server) in Anspruch nimmt.

Download

Vorgang des Herunterladens spezieller Daten, Programme o. ä.

Exception

Ausnahmebehandlung – Diese kann bei Fehlern während der Laufzeit eines Programms auftreten.

Firewall

Ein Firewall ist eine Schwelle zwischen zwei Netzen, die überwunden werden muss, um Systeme im jeweils anderen Netz zu erreichen. Es wird dafür gesorgt, dass jede Kommunikation zwischen den beiden Netzen über den Firewall geführt werden muss. Auf dem Firewall sorgen Zugriffskontrolle und Audit dafür, dass das Prinzip der geringsten Berechtigung durchgesetzt wird und potentielle Angriffe schnellstmöglich erkannt werden.

GIF – Dateien

Graphic Interchange Format – Formatspezifikation für Bilder. Wurde von Compuserve speziell für das Internet entwickelt.

GUI

Graphical User Interface – Graphische Benutzerschnittstelle, hiermit interagiert der Benutzer mit dem Programm.

HTML

Hypertext Markup Language – Standardisierte Sprache zur Beschreibung von Internetseiten (ISO 8879).

Information Hiding

Auch Geheimnisprinzip genannt. Ein Prinzip bei der Entwicklung strukturierter Software, bei der jede Softwareeinheit mit anderen ausschließlich über festgelegte Schnittstellen kommuniziert.
Kennzeichnend ist, dass eine Einheit von der anderen Einheit möglichst wenig weiß.
Die Spezifikation der Einheit definiert die Funktionalität, die Übergabeparameter usw. Ein Benutzer der Einheit benötigt keine Information über die intern verwendeten Algorithmen, Datenstrukturen, usw. Informationhiding erlaubt es, dass eine Einheit (bei unveränderter Schnittstelle) reimplementiert oder intern modifiziert werden kann (z.B. um einen Entwurfsfehler zu korrigieren), ohne dass andere Einheiten des Systems geändert werden müssen.

Inkonsistenz

Fehlende Übereinstimmung der systeminternen Daten mit den Fakten in der Realität, die sie repräsentieren sollen.

Instanz

Eine Instanz ist der konkrete Bezug einer Aktivität auf ein bestimmtes Objekt (z. B.Modul, Komponente, usw.).

IP – Nummer

Internet Protokoll – Nummer; Anhand dieser Nummer kann ein Rechner in einem Netzwerk eindeutig identifiziert werden.

JAVA

Dies ist eine objektorientierte Programmiersprache (entwickelt von SUN Microsystems). Sie wurde zum Erstellen Client – Applets sowie des Servlets angewandt.

Klasse

Eine Klasse beschreibt eine Menge von Objekten, die eine gemeinsame Menge von Attributen bzw. Eigenschaften, also eine gemeinsame Struktur, und eine gemeinsame Menge von Methoden, also ein gemeinsames Verhalten, haben.

Log – Dateien

Dateien in denen Statusmeldungen gespeichert werden, welche der Administrator dann später bei Bedarf kontrollieren kann.

Member – Variable

Eine Attribut, welches genau zu einem Objekt gehört und über dieses auch angesprochen werden kann. In diesem Attribut werden Informationen gespeichert.

Modal

Ein als modal deklarierter Dialog muss beendet sein, damit die Vater – Instanz (aufrufende Instanz, beispielsweise ein anderer Dialog) wieder benutzt werden kann.

PDF

Portable Document Format – Ist ein universelles Dateiformat, das alle Schriften, Formatierungen, Farben und Graphiken jedes Dokuments beinhaltet. Dies geschieht unabhängig von der Anwendung und der Plattform, wo dieses benutzt wird.

Port

In der IP – Terminologie eine Schnittstelle an einer IP – Adresse, auf der ein bestimmter Dienst laufen kann.

Server

Prozess bzw. Rechner, der IT – Dienstleistungen zentral für Client-Prozesse bzw. Client-Rechner zur Verfügung stellt. Beispiel: Webserver

Servlet

Ein JAVA – Programm, welches Dienste über einen Webserver zur Verfügung stellt. (Vergleichbar mit cgi – Skripten unter C++)

Strecke

Fahrtstrecke eines Zuges an einem bestimmten Werktag.

Verkehrstag

Genau der Tag, an dem ein Zug verkehrt.

Webserver

Ein Server, der Internetdienste in einem Netzwerk zur Verfügung stellt.

7.6 Quellcodeauszüge

[ZENSIERT]