Modernisierung von 3270-Host-Anwendungen mit Host Access Transformation Services

Diplomarbeit

Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften
Universität Tübingen

von

Stefan Steinhilber

Betreuer:

Prof. Dr.-Ing. Wilhelm G. Spruth

10. August 2009

Kurzfassung

Ziel dieser Diplomarbeit war es, für die im studentischen Praktikumsbetrieb verwendeten 3270-Host-Anwendungen TSO, ISPF und CICS mit Hilfe des Produkts IBM Rational Host Access Transformation Services (HATS) eine moderne Web-Oberfläche zu erstellen, welche allein über einen Webbrowser bedienbar ist. Die erstellte Web-Oberfläche wurde als Java EE-Anwendung auf einem dafür eingerichteten Websphere Application Server unter einer virtuellen Maschine mit z/Linux auf dem zSeries Mainframe der Universität Tübingen lauffähig gemacht. Zu Demonstrationszwecken wurde zusätzlich das Abrufen von Kundendaten der CICS-Transaktion NACT über einen SOAP Web Service ermöglicht. Grundlage für diese Arbeit war das HATS Information Center der Firma IBM. Nach einer kurzen Erläuterung der Grundlagen wird in dieser Arbeit das Produkt HATS im Hinblick auf Web-Anwendungen vorgestellt und anschließend die erwähnte Erstellung der Web-Oberfläche und des Web Services dokumentiert.

Positiv aufgefallen ist bei der Arbeit mit HATS, dass sich bereits in kurzer Zeit eine einfache Web-Oberfläche entwickeln lässt. Um jedoch die Web-Oberfläche optisch ansprechend zu gestalten und dem Anwender die Bedienung zu vereinfachen, sind Anpassungen nötig, wobei die aufzubringende Arbeit von Größe und Struktur der Host-Anwendung abhängig ist. Folgen alle Screens der Host-Anwendung einer einheitlichen Struktur, kann mit wenigen Umsetzungsregeln mit einer Standardtransformation die Web-Oberfläche optisch ansprechend erstellt werden. Folgen die Screens jedoch keiner einheitlichen Struktur, müssen Screens nach der Struktur gruppiert werden, wobei für jede Gruppe eine eigene Anpassung erstellt werden muss.

Die verwendeten Programme wurden auf einer auf Windows XP basierenden Workstation in Form einer virtuellen Maschine installiert, welche sich zusammen mit dem Projekt und den Dokumentationen im Anhang dieser Arbeit verteilt auf drei DVDs befindet.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen verwendet habe.

Tübingen, den 10. August 2009 _____

Stefan Steinhilber

Danksagungen

Mein herzlicher Dank gilt allen, die mir die Realisierung dieser Diplomarbeit ermöglicht haben. An erster Stelle möchte ich mich bei Herrn Prof. Dr.-Ing. Wilhelm G. Spruth bedanken, der meine Arbeit betreute und mich in vielfältiger Weise während der gesamten Umsetzung unterstützte. Weiterer Dank geht an Herrn Rolf Wiest (Firma IBM) für seine fachmännische Hilfe und seinen unermüdlichen Einsatz und an Kolja Treutlein für seine Motivation während der gesamten Umsetzungszeit.

Ebenso danke ich meiner Familie, die mir während meines gesamten Studiums zur Seite gestanden ist, sowie meiner Freundin Marion Schlegel für ihre Geduld und ihr Verständnis.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

3270-Host-Anwendungen existieren seit den 70er Jahren und werden auch heute noch sehr oft eingesetzt. Auf viele klassische Host-Anwendungen wird über Terminals zugegriffen, wobei die Terminals im Laufe der Zeit immer mehr durch Terminalemulationen, die auf einem PC laufen, ersetzt werden.

Aufgrund der enormen Entwicklung von Benutzeroberflächen in den letzten Jahrzehnten wird nun auch die Modernisierung der Oberflächen von Host-Anwendungen immer gefragter. Mainframe-Anwendungen bestehen oft aus sehr vielen Zeilen COBOL oder PL/1-Code und wurden bereits vor über 30 Jahren erstellt. Selbst heute wird immer noch ein überragender Anteil aller neuen Business-Anwendungen mit COBOL erstellt [SPRU]. Viele klassische Host-Anwendungen besitzen aufgrund ihres beachtlichen Alters jedoch keine saubere Trennung in Präsentationslogik und Geschäftslogik, weshalb sich die Erstellung einer neuen Oberfläche aus Zeit- und Kostengründen meist nicht realisieren lässt.

Für diese Modernisierung existieren verschiedene Lösungen von unterschiedlichen Herstellern. In dieser Diplomarbeit wird das Produkt IBM Rational Host Access Transformation Services (HATS) betrachtet und mit diesem Produkt eine moderne Web-Oberfläche als Ersatz für die klassischen zeilenorientierten Oberflächen für die im Client/Server-Praktikum an der Universität Tübingen verwendeten Host-Anwendungen erstellt.

Das Client/Server-Praktikum bietet Studenten die Möglichkeit, erste Erfahrungen im Umgang mit Mainframes zu sammeln. Die für die Verbindung zu einem Mainframe benötigten 3270-Terminalemulatoren wirken auf den ersten Blick für unerfahrene Anwender jedoch abschreckend, weshalb im Rahmen dieser Diplomarbeit mit einer modernen Web-Oberfläche versucht werden soll, die Kluft zwischen Anwender und Mainframe zu verkleinern.

1.2. Aufbau der Arbeit

Zuerst werden in Kapitel 2 die für diese Diplomarbeit grundlegenden Begriffe und Konzepte angesprochen und kurz erklärt. Danach wird in Kapitel 3 ein Überblick über das Produkt HATS gegeben, wobei auf das Prinzip und den Aufbau von HATS-Anwendung eingegangen wird. Im anschließenden Kapitel 4 wird die Einrichtung des für die Web-Oberfläche benötigten Websphere Application Servers beschrieben, worauf im Anschluss in Kapitel 5 die Erstellung der Web-Oberfläche für die Host-Anwendung dokumentiert wird. Darauf folgend wird in Kapitel 5.3

die Erstellung eines einfachen SOAP Web Service mit HATS erklärt, welcher auf eine CICS-Transaktion zugreift. Abschließend gibt es eine Zusammenfassung und einen Ausblick auf mögliche weitere Studien- und Diplomarbeiten, die basierend auf dieser Diplomarbeit denkbar sind. Im Anhang befinden sich Code-Auszüge der Web-Oberfläche, auf welche während der Dokumentation Bezug genommen wird. Die erstellten Projekte, sowie die dazu verwendeten Programme, befinden sich verteilt auf drei DVDs im Anhang dieser Diplomarbeit. Auf den DVDs befinden sich zusätzlich alle im Literaturverzeichnis erwähnten Arbeiten, soweit sie in digitaler Form verfügbar sind.

In den Verzeichnissen VM1, VM2 und VM3 befindet sich verteilt auf die drei DVDs eine virtuelle Maschine mit der für diese Arbeit verwendeten HATS-Entwicklungsumgebung. Um die virtuelle Maschine benutzen zu können, müssen alle Dateien aus den Ordnern VM1 auf DVD1, VM2 auf DVD2 und VM3 auf DVD3 in einen Ordner auf der Festplatte kopiert werden. Anschließend kann die virtuelle Maschine mit dem VMWare Player gestartet werden, welcher sich im Ordner *Software* auf DVD1 befindet. Beim Starten der virtuellen Maschine werden Sie eventuell gefragt, ob die virtuelle Maschine kopiert oder verschoben wurde. Wählen Sie in diesem Fall die Option „I copied it“ aus. Um die virtuelle Maschine kompakt zu halten, wurde die Nutzung von virtuellem Speicher deaktiviert. Für eine effektive Arbeit, sollte der virtuelle Speicher im Gastsystem aktiviert werden.

2. Grundlagen

2.1. Mainframe

2.1.1. IBM System z

System z ist eine 64-Bit Mainframe-Architektur der Firma IBM. Mainframes werden überall dort eingesetzt, wo ein hohes Maß an Zuverlässigkeit und hohe Ein-/Ausgabe-Leistung nötig ist. Gewährleistet wird dies durch sorgfältig aufeinander abgestimmte Hardware-Komponenten, welche zur Gewährleistung der Ausfallsicherheit redundant verbaut werden. Für weitere Informationen zu System z wird an dieser Stelle auf [IBM1] verwiesen.

2.1.2. 3270-Terminal und Screens

Der Zugriff auf einen Mainframe geschieht meist über ein IBM 3270-Terminal. Ein 3270-Terminal ist die Bezeichnung für ein Bildschirmterminal der Firma IBM, welches bereits seit 1972 auf dem Markt ist. Richtige Bildschirmterminals sind heutzutage kaum noch im Einsatz. Stattdessen wird auf klassischen PCs eine 3270-Terminalemulation eingesetzt um auf Host-Anwendungen zuzugreifen. Bekannte 3270-Terminalemulatoren sind unter anderen Personal Communications von IBM [IBM2] und der Open Source Terminal Emulator x3270 [X3270].

Das 3270-Terminal besitzt eine Auflösung von 24x80 Zeichen und war ursprünglich grün monochrom. Daher rührt auch die für einen Bildschirminhalt oft verwendete Bezeichnung Green Screen. Später wurden ebenso Farbbildschirme mit 8 Farben und unterstrichenem sowie blinkendem Text unterstützt. Die Terminals sind zusätzlich mit bis zu 24 Programmed function (PF) keys und drei Program attention (PA) keys zur Steuerung der Host-Anwendung ausgestattet, welche sich meist an der oberen Seite der Terminaltastaturen befinden.

Zur Kommunikation zwischen Mainframe und Client wurde früher das Übertragungsprotokoll VTAM, heutzutage TCP/IP benutzt. Über der vierten ISO/OSI-Schicht wird das angepasste Telnet-Protokoll tn3270 verwendet, welches anstatt zeilenorientiert, feldorientiert arbeitet und als Zeichenkodierung den für Mainframes üblichen Extended Binary Coded Decimals Interchange Code (EBCDIC) benutzt [RFC1].

Ähnlich ist das IBM 5250-Terminal, welches zum Zugriff auf IBM System i benutzt werden kann. Dies sei an dieser Stelle jedoch nur erwähnt. Weitergehende Informationen zu 5250-Terminals findet man in [IBM3].

In dieser Diplomarbeit wird der aktuell angezeigte Bildschirminhalt einer 3270-Host-Anwendung in Anlehnung an die HATS-Dokumentation [HATS1] als Screen bezeichnet. Positionen innerhalb eines Screens werden mit Hilfe der Notation (x,y) bezeichnet, wobei x die Zeile und y die Spalte innerhalb des Screens beschreibt.

2.1.3. z/OS Betriebssystem

Das IBM z/OS Betriebssystem ist als Nachfolger von OS/390 das momentan aktuelle 64-Bit Betriebssystem für IBM Mainframes, welches speziell auf die System z Hardware optimiert ist. Das Betriebssystem beinhaltet unter anderem die 3270-Host-Anwendungen CICS, DB2, ISPF, TSO und USS, auf die in dieser Diplomarbeit später im Abschnitt 2.1.4 genauer eingegangen wird. Für weitergehende Informationen zu z/OS sei an dieser Stelle auf [IBM4] verwiesen. In dieser Diplomarbeit werden manche z/OS-Konzepte erwähnt, die im Folgenden erklärt werden.

2.1.3.1. Datasets

Datasets werden zum Speichern von Daten benötigt und umfassen mehrere Records. Ein Record ist eine logische Einheit die aus mehreren Bytes besteht. Für Datasets existieren verschiedene Formen der Datenorganisation. Am Gebräuchlichsten sind die Formen sequentieller Dataset und partitionierter Dataset. Bei einem sequentiellen Dataset sind die Records sequentiell angeordnet und werden folglich auch so verarbeitet. Ein partitionierter Dataset besteht aus einem Inhaltsverzeichnis (Directory) und mehreren unabhängigen Teilen (Members), von denen jeder für sich die Struktur eines sequentiellen Datasets hat [TEUF, Kapitel 2.2].

2.1.3.2. Jobs und JCL

Ein Job beinhaltet den Aufruf eines oder mehrerer Programme und die Bereitstellung der dafür notwendigen Daten. Zur Definition solcher Jobs gibt es die Sprache Job Control Language (JCL). Die Ablaufkontrolle des Jobs übernimmt dabei das Job Entry System (JES). Die Ablaufkontrolle gliedert sich dabei in die drei Phasen Vorverarbeitung, Verarbeitung und Nachverarbeitung. In der Vorverarbeitungsphase wird das JCL-Script gelesen, auf Syntaxfehler überprüft und die für den Ablauf notwendigen Daten bereitgestellt. Falls Syntaxfehler auftreten, wird die Verarbeitung bereits in dieser Phase abgebrochen und dem Benutzer eine entsprechende Fehlermeldung angezeigt. Alle Fehler- und Erfolgsmeldungen werden zusätzlich für jeden Job in eine Log-Datei geschrieben. In der Verarbeitungsphase wird der Job ausgeführt und in der Nachverarbeitunsphase die Ausgabedateien entweder gedruckt oder dem Benutzer zur Weiterverarbeitung freigegeben. Ebenso werden alle vom Job verwendeten Datasets für andere Jobs freigebeben. [TEUF]

2.1.4. 3270-Host-Anwendungen

2.1.4.1. Time Sharing Option

Die Time Sharing Option (TSO) ist ein interaktiver Kommandozeileninterpreter, welcher es mehreren Benutzern erlaubt, einen Mainframe zeitgleich zu benutzen. Dies wird gewährleistet, indem jedem Benutzer periodisch eine gewisse Zeitscheibe für die Nutzung der CPU zugeteilt wird. Im Client/Server-Praktikum wird TSO lediglich zum An- und Abmelden und zum Starten der Interactive System Productivity Facility genutzt.

2.1.4.2. Interactive System Productivity Facility

Die Interactive System Productivity Facility (ISPF) für z/OS ist eine vielfältige Sammlung an Entwicklungswerkzeugen für System z. Unter anderem beherbergt die Sammlung Tools zum Bearbeiten von Datasets, sowie einem integrierten Texteditor zur Anwendungsentwicklung und zur Dokumentation. Im Client/Server-Praktikum wird dieser integrierte Texteditor häufig benutzt um ein C-, COBOL-, Assembler- oder ein REXX-Programm, sowie JCL-Scripte zu erstellen. Ebenso häufig wird das DSLIST-Utility benutzt um Datasets mitsamt den Membern zu bearbeiten.

2.1.4.3. Customer Information Control Center

Das Customer Information Control Center (CICS) ist ein weitverbreiteter Transaktionsmonitor, welcher für Transaktionen die ACID-Bedingungen garantiert. Im Fehlerfall wird durch ein Rollback der Zustand vor der Ausführung der Transaktion wiederhergestellt. CICS-Anwendungen sind teilweise mehrere Jahrzehnte alt und laufen zuverlässig und performant. Jedoch entspricht ihre Oberfläche meist nicht den heutigen Anforderungen, weshalb oft eine Modernisierung dieser Anwendungen gewünscht ist. Im Client/Server-Praktikum werden die Transaktion CEDA, NACT und CESF genutzt.

2.1.4.4. UNIX System Services

Die UNIX System Services (USS) sind eine vollwertige UNIX-Implementierung und erlauben es UNIX-Anwendungen auf einem System z Mainframe auszuführen. Einzige Inkompatibilität ist dabei die oft verwendete ASCII-Zeichenkodierung unter UNIX und die EBCDIC-Zeichenkodierung des Mainframes. Der Benutzer sieht nach der Anmeldung wie bei anderen UNIX-Implementierungen eine Shell, welche zum Absetzen von Kommandos benutzt wird. Im Client/Server-Praktikum wird USS benutzt, um ein Java Servlet für den Websphere Application Server zu

erstellen. Weitere Informationen zu USS findet man in [IBM7].

2.1.4.5. DB2

DB2 ist ein relationales Datenbank Management System. Im Client/Server-Praktikum werden unter anderem eine DB2-Datenbank und Tabellen erstellt und über verschiedene Wege auf die darin enthaltenen Daten zugegriffen. Zur Erstellung ruft der Benutzer innerhalb von ISPF das DB2 Primary Option Menü auf, aus welchem er einzelne DB2-Tools wie zum Beispiel SPUFI starten kann.

2.1.5. z/VM Betriebssystem

Ebenso wie z/OS ist z/VM ein Betriebssystem für IBM Mainframes, welches im Gegensatz zu z/OS auf die Erzeugung, Verwaltung und auf den parallelen Ablauf von virtuellen Maschinen optimiert ist. Mit z/VM ist es möglich mehrere virtuelle Maschinen mit den Betriebssystemen z/OS, z/VSE, TPF oder z/Linux parallel zu betreiben. Für weitergehende Informationen zu z/VM sei an dieser Stelle auf [IBM5] verwiesen.

2.1.6. z/Linux

z/Linux, oder auch Linux on System z genannt, ist die Portierung des Betriebssystems Linux auf die System z Plattform. z/Linux läuft entweder direkt auf der System z Hardware in einer LPAR oder innerhalb einer virtuellen Maschine unter z/VM. Auf dem Markt verfügbar sind verschiedene nicht kommerzielle Linux-Distributionen wie zum Beispiel Debian und CentOS, sowie die kommerziellen Distributionen Novell SUSE Linux Enterprise Server und Red Hat Enterprise Linux. Weitergehende Informationen zum Thema z/Linux kann in [IBM6] gefunden werden.

2.1.6.1. SUSE Linux Enterprise Server

SUSE Linux Enterprise Server (SLES) ist eine kommerzielle Linux Distribution von der Firma Novell. Im Unterschied zu nichtkommerziellen Distributionen bekommt man mit SLES Funktionsgarantien für einen längeren Zeitraum gewährleistet, was gerade im Unternehmensbereich eine wichtige Rolle spielt. Hierzu vergibt Novell Zertifizierungen für geeignete Hardware und Anwendungen, die nicht Teil des Standardpakets sind. Dadurch kann die Funktionsgarantie aufrecht erhalten werden [SLES]. Ebenso besteht für SLES die Option eines kostenpflichtigen Supports von Novell.

2.2. Websphere Application Server

Der IBM Websphere Application Server (WAS) ist ein Anwendungsserver mit einer Laufzeitumgebung für Java EE-Anwendungen. WAS wird in den Editionen Application Server, Application Server Express, Application Server Community Edition, Application Server Network Deployment und Application Server for z/OS angeboten, welche sich jeweils im Leistungsumfang und der Performance unterscheiden [WAS2]. Der WAS unterstützt zahlreiche Web Service Spezifikationen wie zum Beispiel JAX-RPC 1.1 und Web Services for Java EE 1.1 [RED1, Kapitel 12].

2.3. Java Platform Enterprise Edition

Die Java Platform Enterprise Edition (Java EE) definiert einen Standard um Multi-Tier Anwendungen zu entwickeln [SUN1]. Die Plattform profitiert von den Vorteilen der Java 2 Platform Standard Edition (J2SE) wie zum Beispiel der Portabilität, der Java Database Connectivity (JDBC) Schnittstelle für den Datenbankzugriff, der CORBA Technologie zur Kommunikation mit bestehenden Unternehmensanwendungen und dem Java Sicherheitsmodell. Darauf aufbauend unterstützt die Java EE-Plattform Enterprise Java Beans, Servlets, JavaServer Pages und die XML-Technologie.

Der Standard wird innerhalb des Java Community Process (JCP) von mehreren Unternehmen weiterentwickelt und der Öffentlichkeit anhand eines Dokuments und einer Referenzimplementierung zugänglich gemacht. Dies geschieht durch Java Specification Requests (JSRs), welche im Laufe der Entwicklung mehrere Status in Form von Drafts und Reviews annehmen [JSR5].

Java EE-Anwendungen benötigen zur Laufzeit einen Java EE Application Server welcher unter anderen einen Transaktionsmanagementdienst, Namens- und Verzeichnisdienste, Messagingdienste und einen Persistenzdienst zur Verfügung stellen muss. Der Server wird unterteilt in einen EJB-Container als Laufzeitumgebung für Enterprise Java Beans und einen Web-Container als Laufzeitumgebung für Servlets und JavaServer Pages.

2.3.1. JavaServer Page

JavaServer Page (JSP) ist eine von Sun Microsystems entwickelte Technik zur dynamischen Erzeugung von HTML- und XML-Ausgaben. JSPs sind Textdokumente, welche zu einem Teil aus statischem HTML-Code und zum anderen Teil aus dynamischem Java-Code bestehen. Um den Java-Code in das Dokument einzubetten werden spezielle XML-Elemente (Tags) verwendet, welche in Tag-Bibliotheken als Erweiterung der HTML- und XML-Tags definiert werden (siehe Abschnitt 2.3.1.1). Somit kann die Anwendungslogik in Java Beans gekapselt werden, wodurch

bei der Entwicklung das Model-View-Controller (MVC)-Prinzip effizient realisiert werden kann. JSPs werden zur Laufzeit vom Web-Container in ein Servlet umgewandelt und interpretiert [SUN3].

Um Java-Code in HTML-Seiten zu integrieren, existieren verschiedene JSP-Tags, welche im Folgenden kurz vorgestellt werden.

JSP-Direktive: Eine Direktive dient zum Übermitteln von speziellen Seiteninformationen an den JSP-Compiler. Hier werden zum Beispiel Tag-Bibliotheken und Java-Klassen eingebunden oder auch Fehlerseiten definiert.

JSP-Deklaration: Deklarationen dienen dazu Variablen und Methoden bekannt zumachen und zu initialisieren, welche innerhalb der ganzen JSP benutzt werden können.

JSP-Ausdruck: Ausdrücke werden verwendet um Variablen oder Rückgabewerte von Methoden auszugeben.

JSP-Skriptlet: Skriplets dienen dazu um sonstige Logik in Form von Java-Code einzubetten, wie zum Beispiel Schleifen und *if-else*-Verzweigungen.

2.3.1.1. Tag-Bibliothek

Im Folgenden werden die im vorherigen Abschnitt erwähnten Tag-Bibliotheken noch etwas genauer erklärt. Tag-Bibliotheken sind eine Möglichkeit um wiederverwendbaren Java-Code zu kapseln und somit JavaServer Pages nicht unnötig mit sich wiederholendem Code aufzublähen. Der Entwickler kann eigene Tags für JSPs entwerfen und diese mit selbst geschriebenen Java-Funktionen ausstatten. Wird eine JSP aufgerufen, welche ein selbst erstelltes Tag beinhaltet, wird vom Web-Container eine Instanz der zugehörigen Java-Klasse automatisch aufgerufen. Einmal geschriebene Tag-Bibliotheken können problemlos in andere Projekte portiert werden.

Um eigene Tags zu erstellen, muss für jeden Tag ein eigener Tag-Handler implementiert werden. Dieser Tag-Handler beinhaltet die Funktionalität des Tags. Dazu stellt Java mit dem Paket *javax.servlet.jsp.tagext* die nötigen Interfaces und Basisklassen bereit. Zu jedem Tag-Handler wird zur Konfiguration jeweils zusätzlich ein Tag Library Descriptor (TLD) benötigt, welcher in Form einer XML-Datei unter anderem den Namen des Tags mit der zugehörigen Implementierung verknüpft. Damit die Tag-Bibliothek in einer Web-Anwendung verwendet werden kann, muss der TLD im Deployment Deskriptor der Web-Anwendung eingebunden werden. Weiter Informationen dazu erhält der Leser in [STAR2].

2.3.2. Servlets

Servlets sind Java-Klassen, deren Instanzen innerhalb des Web-Containers eines Java EE Application Servers Anfragen von Clients entgegennehmen und diese dynamisch beantworten. Bei Bedarf werden die Servlets automatisch vom Web Container instanziiert [JSR1].

Servlets erben standardmäßig die Methoden *doGet()* und *doPost()* aus der Klasse *javax.servlet.-http.HttpServlet*. Diese Methoden werden passend überschrieben um die entsprechenden HTTP-Methoden *GET* und *POST* zu verarbeiten. Zusätzlich implementieren Servlets die Schnittstelle *javax.servlet.Servlet*.

Metadaten des Servlets werden in dem Deployment Deskriptor *web.xml* im XML-Format abgelegt. Der Deployment Deskriptor wird zusammen mit den kompilierten Java-Klassen in einer ZIP-Datei archiviert. Dieses Archiv wird als Web Archiv bezeichnet und kann in diesem Format dem Web-Container zur Ausführung übergeben werden.

2.3.3. Enterprise Java Beans

Enterprise Java Beans (EJBs) sind standardisierte, serverseitige Komponenten, welche die Geschäftslogik einer Anwendung implementieren. Analog zu Servlets werden Metainformationen der EJB in einem Deployment Deskriptor abgelegt. Für weitergehende Informationen sei auf [JSR2] verwiesen.

2.3.4. Enterprise Archive

Eine Enterprise Archive (EAR) Datei ist ein JAR-Archiv und wird dazu verwendet um die zu einer Java EE-Anwendung gehörenden Dateien zu bündeln und somit die Übertragung auf einen Java EE Application Server zu vereinfachen. Abbildung 2.1 zeigt den Aufbau einer EAR-Datei. Innerhalb einer EAR-Datei können weitere Archive existieren. So werden zum Beispiel JSPs und Servlets in einer Web Archive (WAR) Datei, EJBs in weiteren JAR-Archiven und Ressourcen-Adapter in Ressource Adapter Archive (RAR) Dateien untergebracht. Das Verzeichnis *META-INF* im Archiv dient als Inhaltsverzeichnis, welches den Deployment Deskriptor der Java EE-Anwendung beinhaltet. Dieser beschreibt die einzelnen Module und legt zusätzlich die Sicherheitsregeln festlegt [SUN2].

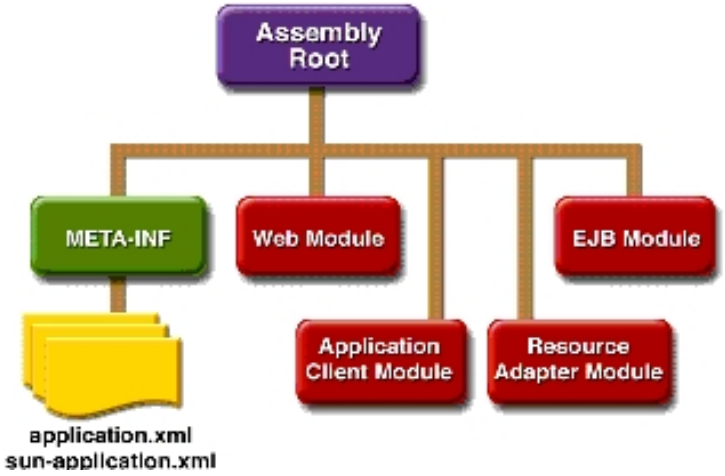

Abbildung 2.1.: Struktur einer EAR-Datei (Quelle: http://java.sun.com)

2.4. Web Services

Ein Web Service ist eine Anwendung, welche über ein Protokoll, wie zum Beispiel SOAP, XML-Nachrichten austauscht. Der Web Service verkörpert dabei eine Dienstleistung eines Servers, welche durch Clients über ein Netzwerk aufgerufen werden kann. Im weiteren Verlauf dieser Diplomarbeit ist mit Web Service standardmäßig ein SOAP Web Service gemeint. Die Grundlage für einen solchen Web Service bilden die Standards SOAP, WSDL und UDDI.

2.4.1. SOAP

SOAP ist ein auf der Anwendungsschicht angesiedeltes Protokoll welches vom W3C standardisiert wurde [W3C1]. Es definiert ein Anfrage-Antwort-Nachrichtenaustauschformat, bei dem alle ausgetauschten Nachrichten XML-kodiert sind. SOAP kann mit verschiedenen Transportprotokollen verwendet werden, wird aber meistens mit den Protokollen HTTP über TCP verwendet. Die Abkürzung SOAP stand ursprünglich für Simple Object Access Protocol [WIK2], wird jedoch seit der Version 1.2 nicht mehr als Akronym verwendet.

2.4.2. WSDL

Die Web Service Description Language (WSDL) beschreibt die Schnittstellen des Web Service basierend auf XML. Sie beschreibt die Funktionen, Parameter, Rückgabewerte jeweils mit Datentyp und die verwendeten Protokolle. Durch eine WSDL-Datei erhält der Client die zum Aufruf des Web Service notwendigen Informationen.

2.4.3. UDDI

Universal Description, Discovery and Integration (UDDI) ist ein Verzeichnisdienst, welcher die WSDL-Dateien dem Client zur Verfügung stellt. Somit besitzen Clients die Möglichkeit Web Services dynamisch aufzufinden. Falls UDDI nicht verwendet wird, kann die WSDL-Datei den Clients zum Beispiel über einen FTP-Server manuell zur Verfügung gestellt werden.

2.4.4. Bibliotheken

Zum Erstellen von Web Services mit Java existieren die Bibliotheken Java API for XML-based Web Services (JAX-WS), welche mit der Java EE Version 5 standardmäßig mitgeliefert wird, und deren Vorgänger Java API for XML-based RPC (JAX-RPC). Weitere Informationen über die Bibliotheken findet der Leser in [JSR3] und [JSR4]. Die Unterschiede und Gemeinsamkeiten der Bibliotheken findet man in [IBM9].

3. Host Access Transformation Services

3.1. Allgemeines

3.1.1. Produkt

IBM Rational Host Access Transformation Services ist eine Software, welche es ermöglicht 3270- und 5250-Host-Anwendungen ohne großes Risiko auf das Web, auf einen Rich-Client oder auf Webbrowser mobiler Endgeräte auszuweiten. Die Ausweitung geschieht risikolos, da weder eine Modifikation noch ein Zugriff auf den Quellcode der Host-Anwendung dafür notwendig ist. Unterstützt wird die Umsetzung durch eine Entwicklungsoberfläche, welche als Erweiterung für die Entwicklungsumgebungen IBM Rational Application Developer, IBM Rational Software Architect und IBM Rational Developer verfügbar ist [HATS2]. Für diese Diplomarbeit wurde HATS in der Version 7.1.0 und der IBM Rational Software Architect in der Version 7.0.0.8 verwendet.

3.1.1.1. IBM Rational Software Architect

Der IBM Rational Software Architect ist eine integrierte Entwicklungsumgebung, die den Entwickler bei der Erstellung von komplexen Web Services, Portal-, C++-, Java- und Java EE-Anwendungen unterstützt. Die Entwicklungsumgebung basiert auf der bewährten Eclipse Software Platform, wodurch die Oberfläche für viele Entwickler kein Neuland ist. Grundfunktionen wie zum Beispiel die Projektverwaltung, der Debugger, die Editoren mit Syntaxhervorhebung und Programmierunterstützung stammen aus der Eclipse Software Platform. Modellgetriebene Softwareentwicklung wird mit Hilfe der Unified Modeling Language (UML) unterstützt. Die Entwicklungsumgebung beinhaltet ebenso hilfreiche graphische Tools zum Bearbeiten von JSP-, UML- und XML-Dateien. Der Software Architect umfasst noch zahlreiche andere Funktionalitäten, die der Leser auf der IBM Produktseite unter [IBM8] nachschlagen kann.

3.1.2. HATS-Entwicklungsumgebung

Die HATS-Entwicklungsumgebung ist in Abbildung 3.1 abgebildet. Der zur Eclipse Software Platform ähnliche Aufbau ist sofort erkennbar, weshalb sich Entwickler, welche bereits Erfahrung mit der Eclipse-Entwicklungsumgebung gesammelt haben, schnell zurecht finden.

Innerhalb der beigefügten virtuellen Maschine lässt sich die HATS-Entwicklungsumgebung durch einen Doppelklick auf das Icon „HATS Toolkit 7.1“ starten.

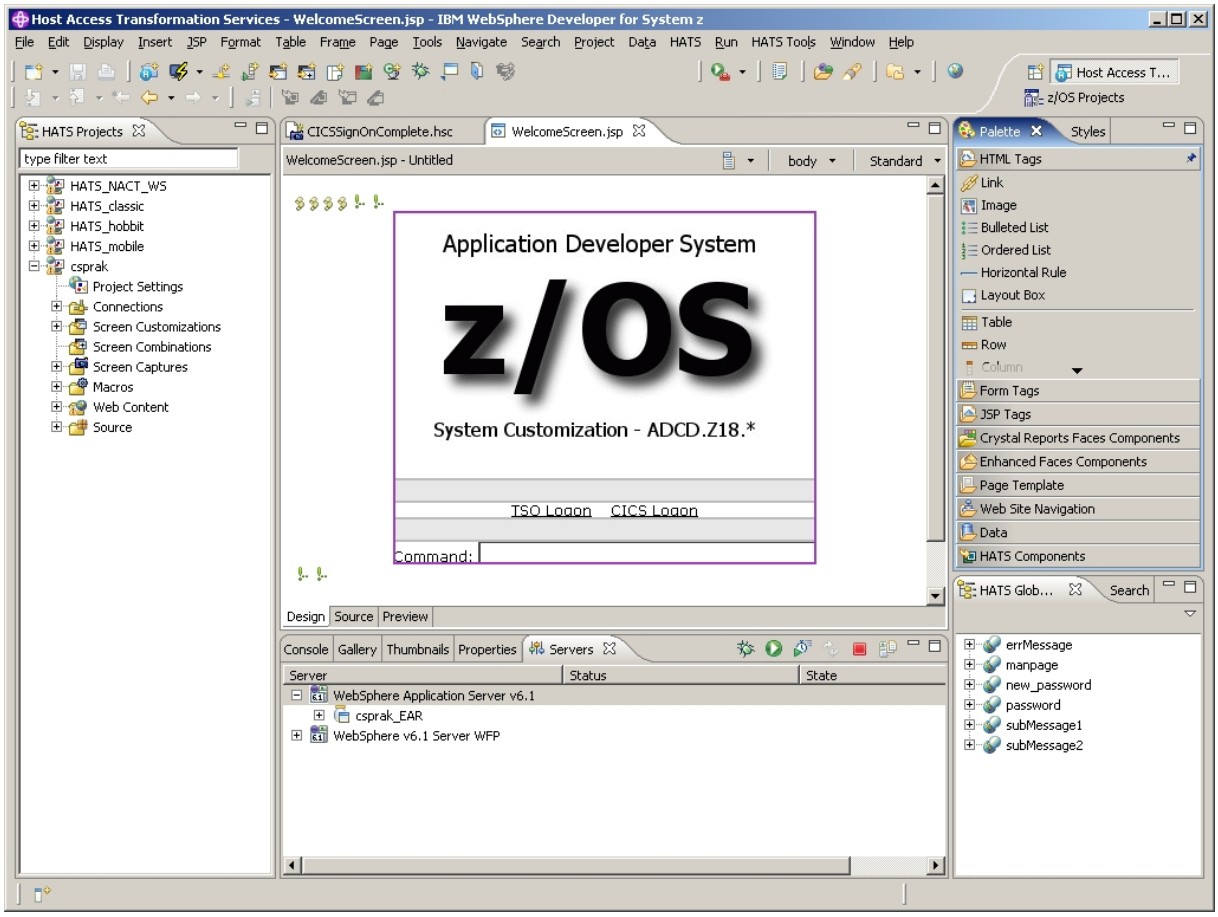

Abbildung 3.1.: HATS-Entwicklungsumgebung

In der Mitte des Fensters befindet sich die aktuell geöffnete Datei. Diese kann entweder im Entwurfsmodus oder direkt im Quelltextmodus bearbeitet werden. Im Entwurfsmodus kann bequem mit Drag & Drop und Mausklicks gearbeitet werden. Nur wenige Tastatureingaben sind hierbei nötig. Das Ergebnis der Änderung wird sofort im Editor sichtbar. Im Quelltextmodus unterstützt HATS den Entwickler durch aus der Eclipse-Entwicklungsumgebung bekannte Funktionen wie zum Beispiel Syntaxhervorhebung und Codevervollständigung.

Im linken Abschnitt des Fensters navigiert man durch die Projektstruktur, in der projektspezifische Dateien angelegt, bearbeitet, gelöscht und ausgeführt werden können.

Im unteren Abschnitt befinden sich mehrere Reiter, die Auskunft über den aktuellen Zustand des Projekts geben. Im Reiter „Console“ werden bei der Übersetzung und zur Laufzeit aktuelle Informationen und gegebenenfalls Fehlermeldungen ausgegeben. Der Reiter „Server“ beinhaltet eine Übersicht mit Steuerungsmöglichkeiten für die momentan eingebundenen Websphere Application Server.

Im rechten Abschnitt des Fensters werden in Abhängigkeit von der geöffneten Datei Elemente angezeigt, mit denen man die aktuelle Datei erweitern kann. Im Entwurfsmodus einer JavaServer Page können hier zum Beispiel mit wenigen Mausklicks HTML-Elemente hinzugefügt werden. Im Abschnitt rechts unten wird eine Übersicht über alle von der HATS-Anwendung momentan verwendeten globalen Variablen angezeigt, welche in Abschnitt 3.4.1 erklärt werden.

3.2. HATS-Anwendungen

3.2.1. Arten von HATS-Anwendungen

HATS-Anwendungen werden mit Hilfe der HATS-Entwicklungsumgebung erstellt und verschaffen 3270-Host-Anwendungen eine moderne Oberfläche. Ein separat installierter 3270-Terminalemulator wird für den Zugriff auf die Host-Anwendung nicht mehr benötigt. Es existieren verschiedene Arten von HATS-Anwendungen, über welche im Folgenden ein kurzer Überblick gegeben wird.

3.2.1.1. HATS Rich-Client-Anwendung

Eine HATS Rich-Client-Anwendung ist eine selbständig lauffähige Java-Anwendung mit moderner GUI, welche auf der Eclipse Rich Client Platform (RCP) oder auf einer Lotus Expeditor Client-Umgebung basiert. Der Benutzer arbeitet mit Menüs, Reitern, Eingabefeldern und Schaltflächen, wie sie ihm aus anderen Anwendungsoberflächen bekannt sind. Wird diese Art der Anwendung verwendet, spricht man in der Client-Server-Terminologie von einem Fat Client [HATS3].

3.2.1.2. HATS Web-Anwendung

Eine HATS Web-Anwendung ist eine Java EE-Anwendung, welche auf einem Websphere Application Server läuft und dem Benutzer über einen Webbrowser zugänglich ist. Unterstützt werden dabei alle gängigen Webbrowser die momentan auf dem Markt sind, wie zum Beispiel Mozilla Firefox ab Version 2.0, Microsoft Internet Explorer ab Version 6 mit SP1, Opera ab Version 7.6 und Apple Safari ab Version 1.2 [HATS6]. Der Benutzer bekommt beim Aufruf der HATS-Anwendung über eine URL eine gewohnte HTML-Seite zu sehen. Wird diese Art der Anwendung verwendet, spricht man in der Client-Server-Terminologie von einem Thin Client [HATS3].

Für die HATS Web-Anwendung existiert zusätzlich die Möglichkeit, eine Unterstützung für mobile Geräte wie zum Beispiel PDAs oder Mobilfunktelefone zu aktivieren. Somit kann der Zugriff auf die Web-Anwendung für mobile Geräte hinsichtlich der Darstellung optimiert werden. Jedoch muss dies zu Beginn der Entwicklung festgelegt werden, da eine automatische Konvertierung einer bestehenden HATS-Anwendung von der HATS-Entwicklungsumgebung nicht vorgesehen ist.

3.2.2. Prinzip einer HATS-Anwendung

Damit die 3270-Host-Anwendung nicht im Quellcode verändert werden muss, verfolgt HATS ein einfaches Prinzip. Die HATS-Anwendung emuliert serverseitig über das tn3270-Protokoll einen Zugriff auf die Host-Anwendung. Die komplette serverseitige Terminalemulation geschieht für den Benutzer transparent, wodurch die HATS-Anwendung als Zwischenschicht zwischen Benutzer und Host-Anwendung gesehen werden kann.

3.2.2.1. Prinzip einer HATS Web-Anwendung

Im Folgenden wird anhand der Abbildung 3.2 der Ablauf einer Kommunikation über eine HATS Web-Anwendung dargestellt. Im ersten Schritt wird die Anwendung in der HATS-Entwicklungsumgebung erstellt und im zweiten Schritt auf den Websphere Application Server übertragen und dort eingerichtet. Im dritten Schritt greift ein Client mit seinem Webbrowser auf die auf dem Server laufende HATS Web-Anwendung über das HTTP-Protokoll zu. Der WAS nimmt im vierten Schritt die HTTP-Anfrage entgegen und kontaktiert über eine Socket-Verbindung und das tn3270-Protokoll den Server, auf dem die 3270-Host-Anwendung läuft. Dieser Server antwortet nun ebenfalls über das tn3270-Protokoll an die HATS Web-Anwendung, welche die Antwort im fünften Schritt intern transformiert und die transformierte Nachricht anschließend dem Client als HTML-Seite mit dem HTTP-Protokoll über den Webbrowser präsentiert.

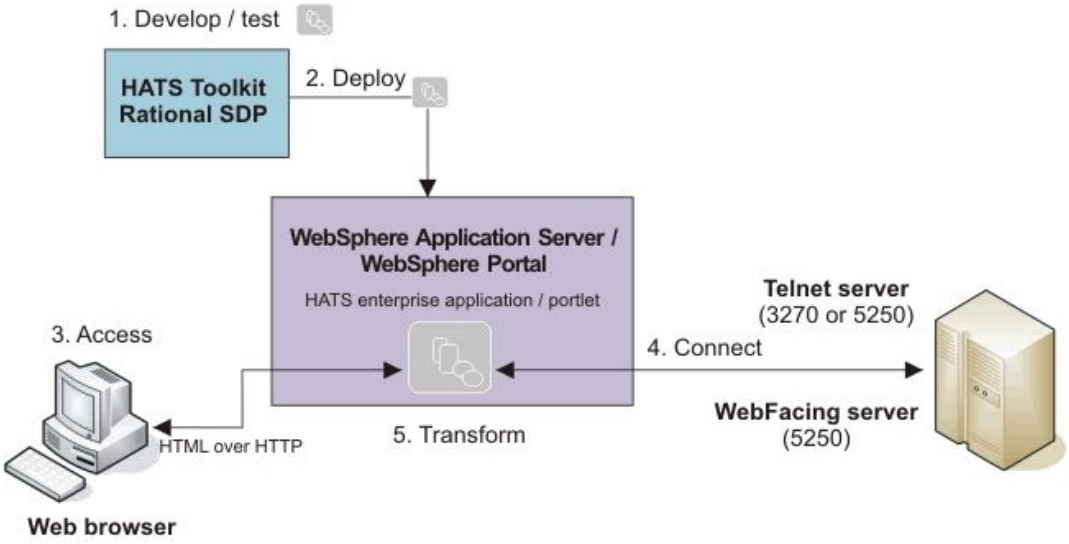

Abbildung 3.2.: Prinzip einer HATS Web-Anwendung (Quelle: [HATS3])

Wie bereits erwähnt benötigt der Benutzer für den Zugriff auf eine HATS Web-Anwendung lediglich einen Webbrowser. Hierbei kann der Benutzer von Vorteilen eines Webbrowsers, wie zum Beispiel der automatischen Formularvervollständigung oder einfachen Möglichkeiten zum Drucken, profitieren. Einziger Nachteil ist, dass der Benutzer die Vor- und Zurück-Schaltflächen des Webbrowsers nicht sinnvoll benutzen kann. Durch die Schaltflächen kann der Benutzer zwar durch die sich im Cache befindenden HTML-Seiten navigieren, jedoch wechselt die HATS Web-Anwendung intern nicht zu dem gewünschten Screen in der Host-Anwendung. Wird eine sich im Cache befindende HTML-Seite im Webbrowser aktualisiert oder ein HTML-Formular erneut abgesendet, wird im Webbrowser der von der HATS Web-Anwendung zuletzt dargestellte Screen angezeigt.

3.2.2.2. Prinzip einer HATS Rich-Client-Anwendung

Die Kommunikation über eine HATS Rich-Client-Anwendung ist in Abbildung 3.3 dargestellt und funktioniert ähnlich wie mit einer HATS Web-Anwendung. Zunächst wird die HATS-Anwendung über die HATS-Entwicklungsumgebung erstellt und dem Client im zweiten Schritt zum Beispiel über einen FTP-Server bereitgestellt. Der Anwender installiert und startet die Anwendung lokal auf dem Client und kann nun im fünften und sechsten Schritt mit der 3270-Host-Anwendung über die Oberfläche der HATS Rich-Client-Anwendung interagieren. Sämtliche Transformationen werden clientseitig vorgenommen.

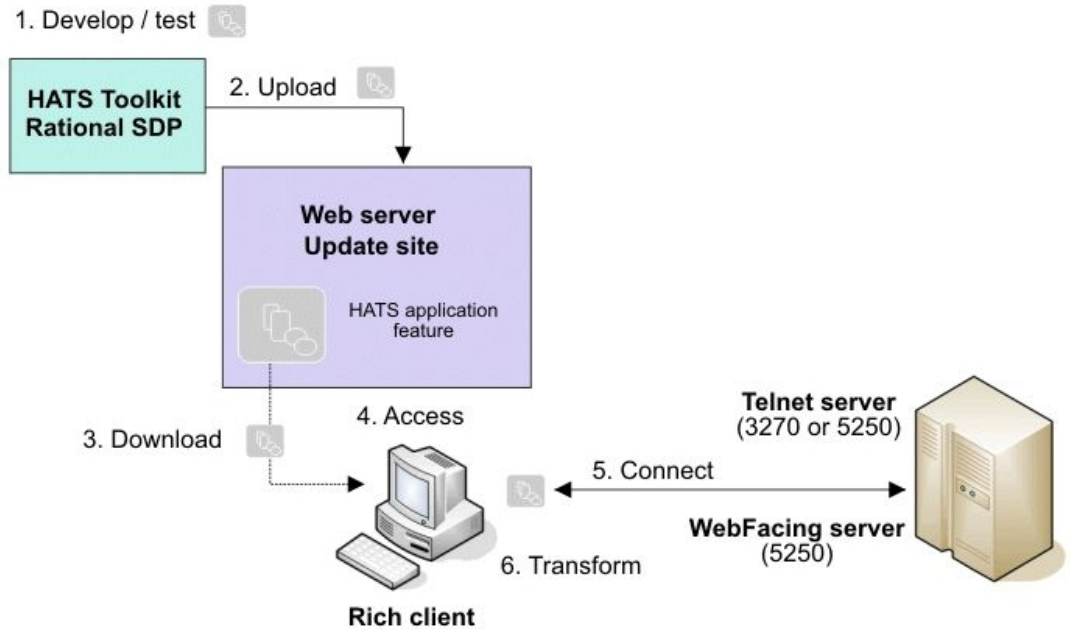

Abbildung 3.3.: Prinzip einer HATS Rich-Client-Anwendung (Quelle: [HATS3])

Durch dieses Prinzip kann mit Hilfe einer HATS-Anwendung für eine bestehende 3270-Host-Anwendung eine neue moderne Oberfläche erstellt werden, ohne die Host-Anwendung zu verändern. Nach der Bereitstellung einer HATS-Anwendung ist es für den Benutzer weiterhin möglich mit einem klassischen 3270-Terminalemulator die Host-Anwendung zu bedienen. Der Benutzer kann so je nach Fähigkeiten und Erfahrung eine bevorzugte Oberfläche benutzen.

Im Folgenden wird HATS nur hinsichtlich der Erstellung von HATS Web-Anwendungen näher betrachtet. Die Erstellung von HATS Rich-Client-Anwendungen funktioniert bis auf wenige Unterschiede ähnlich. Eine Einführung in die Entwicklung von HATS Rich-Client-Anwendungen befindet sich im vierten Kapitel des HATS User's and Administrator's Guide [HATS3].

3.3. Screen-Anpassung

Eine HATS-Anwendung gliedert sich intern grob in zwei Abschnitte. Im ersten Abschnitt versucht die HATS-Anwendung, den aktuell von der 3270-Host-Anwendung übermittelten Screen anhand verschiedener Erkennungskriterien zu erkennen. Erfüllt der Screen die definierten Erkennungskriterien, werden im nächsten Schritt eine oder mehrere HATS-Aktionen ausgeführt. Eine HATS-Aktion besteht zum Beispiel aus einer Transformation, welche den erkannten Screen mit Hilfe von HTML-Elementen darstellt. Alternativ oder zusätzlich zur Transformation können verschiedene andere Aktionen ausgeführt werden, wie zum Beispiel das Extrahieren einer Zeichenkette in eine globale Variable oder das Abspielen eines Makro.

Damit HATS für einen Screen eine oder mehrere Aktionen durchführen kann, muss für den Screen in der Entwicklungsumgebung eine Screen-Anpassung (Screen Customization) erstellt werden. Eine Screen-Anpassung setzt sich aus Erkennungskriterien für die Screens und den zugehörigen HATS-Aktionen zusammen. Screen-Anpassungen werden im XML-Format im Projektverzeichnis *Screen Customizations* abgespeichert.

3.3.1. Screen-Erkennung

Damit eine HATS-Anwendung Screens erkennen kann, müssen während der Anwendungsentwicklung Erkennungskriterien getroffen werden, die einen oder mehrere Screens anhand verschiedener Merkmale klassifizieren. In der HATS-Dokumentation wird dieser Schritt Screen Recognition genannt [HATS3, Kapitel 7]. Um die Auswahl und Definition der Kriterien zu vereinfachen, besitzt die HATS-Entwicklungsumgebung einen 3270-Terminalemulator, mit welchem es möglich ist, Screens der Host-Anwendung im XML-Format zu speichern (Screen Capture) und anschließend die Erkennungskriterien anhand des gespeicherten Screens zu definieren.

In Abbildung 3.4 ist der integrierte 3270-Terminalemulator Host Terminal abgebildet. Um einen Screen aufzuzeichnen verbindet man sich mit der Host-Anwendung über das tn3270-Protokoll und navigiert anschließend zum gewünschten Screen. Die Navigation funktioniert wie aus anderen 3270-Terminalemulatoren gewohnt über die Tastatur. Spezielle Tasten, wie zum Beispiel die PF-Tasten, werden unterhalb des Emulator eingeblendet und können über die Maus bedient werden. Es genügt ein Mausklick auf das Symbol „Create Screen Capture“ um den gewünschten Screen im HATS-Projekt innerhalb des Verzeichnisses *Screen Captures* im XML-Format zu speichern.

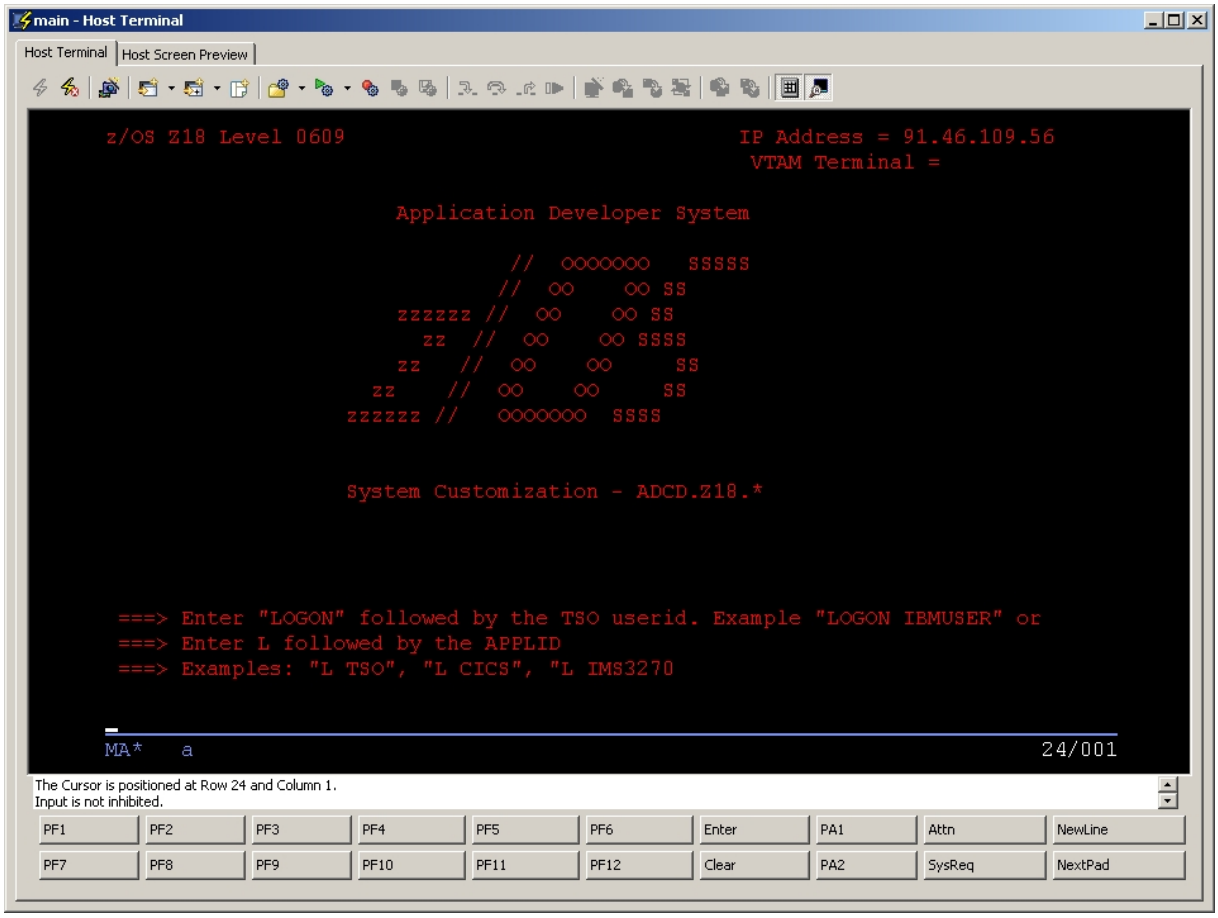

Abbildung 3.4.: Integrierter 3270-Terminalemulator der HATS-Entwicklungsumgebung

Um einen mit dem Host Terminal gespeicherten Screen zu klassifizieren, sind Kombinationen aus folgenden Erkennungskriterien zugelassen:

Felder	Die Anzahl der Felder, oder speziell die Anzahl der Eingabefelder, auf dem Screen
Cursor	Die aktuelle Cursorposition auf dem Screen
Text	Der Inhalt von Zeichenketten, die Textfarbe und die Position von Zeichenketten auf dem Screen
Globale Variablen	Der Inhalt von globalen Variablen (siehe Abschnitt 3.4.1)

Ein Screen muss in einer Screen-Anpassung mit Hilfe von einem oder mehreren oben genannter Erkennungskriterien klassifiziert werden. Zusätzlich ist es dabei möglich, Erkennungskriterien zu invertieren oder als optional zu markieren. Optionale Erkennungskriterien werden betrachtet, falls die anderen Erkennungskriterien auf den aktuellen Screen nicht zutreffen. In diesem Fall muss lediglich ein optionales Erkennungskriterium erfüllt sein, um den Screen zu erkennen.

Stehen die Erkennungskriterien für eine Screen-Anpassung fest, können folgende HATS-Aktionen festgelegt werden:

Apply transformation	Den aktuellen Screen über eine Transformation im HTML-Format darstellen
Execute business logic	Eigene in einer Java-Klasse implementierte Geschäftslogik ausführen
Extract global variable	Eine Zeichenkette des Screens in eine globale Variable einlesen
Insert data	Konstante Werte oder den Inhalt einer globalen Variablen in ein Eingabefeld des Screens einfügen
Set global variable	Eine globale Variable mit einem konstanten Wert oder dem Inhalt aus anderen globalen Variablen füllen
Remove global variable	Eine globale Variable entfernen
Show URL	Eine andere unabhängige Webseite laden und eine Schaltfläche zum Zurückkehren in die HATS Web-Anwendung bereitstellen
Forward to URL	Die Kontrolle an eine JSP weitergeben, welche Integration Objects aufruft (siehe Abschnitt 3.5)
Play macro	Ein Makro abspielen (siehe Abschnitt 3.4)
Macro transaction	Ein Makro mit einer neuen Host-Verbindung abspielen
Send key	Einen Tastendruck innerhalb der Host-Anwendung emulieren
Disconnect	Die Verbindung mit der Host-Anwendung beenden
Pause	Eine wählbare Zeitspanne warten

HATS-Aktionen können ebenso beim Starten, Beenden oder bei einer Fehlermeldung einer HATS-Anwendung ausgeführt werden. Die standardmäßige und am meisten genutzte HATS-Aktion ist das Anwenden einer Transformation, weshalb im folgenden Abschnitt darauf näher eingegangen wird.

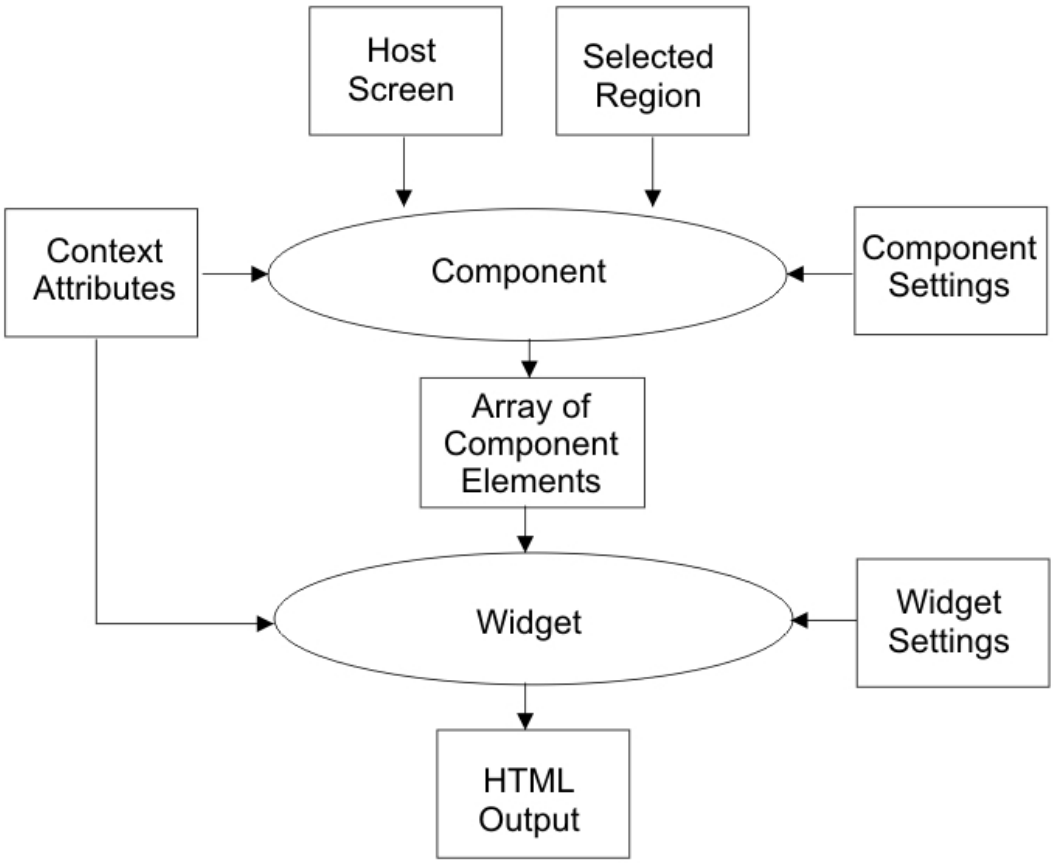

Abbildung 3.5.: Ablauf einer HATS-Transformation

3.3.2. Transformation

Mit einer Transformation wird festgelegt, wie die von der Screen-Anpassung erkannten Screens mit Hilfe einer JavaServer Page in das HTML-Format transformiert werden sollen. Die HATS-Terminologie führt für Elemente einer Transformation die Begriffe Komponenten und Widgets [HATS3, Kapitel 5] ein. Komponenten beschreiben Elemente der 3270-Host-Anwendung und definieren, wie sie von der HATS-Anwendung erkannt werden. Widgets hingegen beschreiben, wie erkannte Komponenten in das HTML-Format transformiert werden. Jede Komponente besitzt einen vorgefertigten Satz an zugehörigen Widgets. Ein Beispiel für eine Komponente ist ein Eingabefeld der 3270-Host-Anwendung und ein Beispiel für ein zur Komponente gehörendes Widget ist ein HTML-Eingabefeld innerhalb eines HTML-Formulars. HATS bietet in der Entwicklungsumgebung vorgefertigte Komponenten und Widgets an. Zusätzlich besteht die Möglichkeit, eigene Komponenten und Widgets mit der Programmiersprache Java zu entwickeln. Die Erstellung neuer Komponenten und Widgets wird in Abschnitt 3.10 näher erläutert.

Das Zusammenspiel zwischen Komponenten und Widgets und der Ablauf einer Transformation vom Host-Screen bis zur HTML-Seite wird in Abbildung 3.5 dargestellt. Der Screen oder ausgewählte Bereiche davon werden als Komponenten erkannt. Damit eine Komponente erkannt wird, muss der jeweils ausgewählte Bereich des Screens spezifische Kriterien der Komponente erfüllen. Diese können in der Entwicklungsumgebung über die Einstellungen der Komponente konfiguriert werden. Alle erkannten Komponenten werden von der HATS-Anwendung intern in einem Array gehalten und in dieser Form zur Darstellung einem oder mehreren Widgets übergeben, welche eine Transformation in das HTML-Format veranlassen. Die Darstellung der HTML-Ausgabe kann ebenfalls in der Entwicklungsumgebung durch Einstellungen der Widgets konfiguriert werden.

Transformationen werden standardmäßig als JSPs im HATS-Projekt innerhalb des Verzeichnisses *Web Content/Transformations* angelegt und gespeichert. Nachfolgend wird nun ein kurzer Überblick auf die in HATS vorgefertigten Komponenten gegeben.

3.3.2.1. Komponenten

Der Entwickler kann aus einem gespeicherten Screen Bereiche auswählen und diesen Komponenten zuordnen. Dazu muss der ausgewählte Bereich jedoch gewissen Kriterien gerecht werden. Die in HATS verfügbaren Komponenten und die dafür notwendigen Kriterien werden in nachfolgender Tabelle aufgelistet.

Command-Line	Dieser Komponente kann ein Eingabefeld der Host-Anwendung zugeordnet werden, welches durch eine voranstehende, eindeutige Zeichenkette wie zum Beispiel "===>" gekennzeichnet ist. Die voranstehende Zeichenkette kann in den Komponenteneinstellungen beliebig definiert werden.
Field	Jeder Screen besteht komplett aus Feldern wie zum Beispiel Eingabefeldern oder geschützten Textfeldern (welche ebenfalls leer sein können). Folglich ist die Zuordnung eines Screen-Bereichs zu dieser Komponente immer möglich.
Function-Key	Dieser Komponente können Host-Elemente der Form *Trennzeichen - Taste - Trennzeichen - Beschreibung - Trennzeichen* wie zum Beispiel " F3=Exit " zugeordnet werden. Das Trennzeichen kann hierbei beliebig definiert werden und ist in diesem Beispiel das Leerzeichen und einmal das Gleichheitszeichen. Meist befinden sich diese Host-Elemente am unteren Ende eines Screens und geben Hinweise wie die Host-Anwendung zu bedienen ist.
Input-Field	Dieser Komponente können Eingabefelder der Host-Anwendung zugeordnet werden. Zusätzlich ist es möglich voranstehende Zeichenketten des Eingabefeldes automatisch als Beschriftung zu extrahieren.
Item-Selection	Diese Komponente kann verwendet werden, falls im Host-Screen eine Gruppierung der Form *Eingabefeld - Text* mehrfach vorhanden ist. Als Interaktion beschreibt diese Komponente die Auswahl verschiedener Elemente.
Selection-List	Diese Komponente wird verwendet falls der Host-Screen eine Gruppierung der Form *Trennzeichen - Option - Trennzeichen - Beschreibung - Trennzeichen* aufweist. Das Trennzeichen kann dabei beliebig festgelegt werden. Die Zeichenkette im Bereich *Option* wird bei der Interaktion in ein Eingabefeld der Host-Anwendung geschrieben.
Table	Diese Komponente wird für tabellenähnliche Bereiche des Host-Screens verwendet. Eine Tabelle besteht aus mehreren Spalten, welche durch ein frei definierbares Trennzeichen, wie zum Beispiel ein Leerzeichen, getrennt sind.
Text	Diese Komponente kann analog zur Field-Komponente jedem Bereich zugeordnet werden. Hierbei gehen jedoch jegliche spezielle Host-Komponenten, wie zum Beispiel Eingabefelder, verloren. Lediglich der sichtbare Text wird übernommen.
URL	Diese Komponente kann allen Host-Elementen, die mit „http://“ oder „https://“ beginnen, zugeordnet werden.

3.3.2.2. Widgets

Wie bereits erwähnt kann jede Komponente durch verschiedene Widgets unterschiedlich im HTML-Format angezeigt werden. So kann die Table-Komponente zum Beispiel als normale HTML-Tabelle ausgegeben werden, alternativ aber auch in Form von verschiedenen visuell aufbereiteten Diagrammen. Die möglichen Kombinationen von Komponenten und Widgets findet man in [HATS3, Kapitel 9]. Eine Übersicht über alle in HATS für Komponenten existierenden Widgets bietet folgende Tabelle.

Button	Eine Schaltfläche, dargestellt durch das HTML-Element *<input type="button" ... >*.
Button-Table	Mehrere Schaltflächen die in einer Tabelle horizontal oder vertikal aufgelistet werden.
Calendar	Eine Schaltfläche, welche bei Betätigung ein kleines Popup-Fenster mit einem auf JavaScript basierenden Kalender öffnet. Nach der Auswahl des gewünschten Datums wird das Datum in das durch die zugehörige Komponente definierte Eingabefeld der Host-Anwendung eingetragen.
Checkbox	Ein Optionsfeld, dargestellt durch das HTML-Element *<input type="checkbox" ... >*. Mehrfachauswahl ist hierbei erlaubt.
Drop-Down	Eine Auswahlliste (Drop-Down-Box), dargestellt mit den HTML-Elementen *<select>* und *<option>*. Mehrfachauswahl ist nicht erlaubt.
Graph	Ein JPEG-Bild zum Visualisieren von Tabelleninhalten. Das JPEG-Bild wird zur Laufzeit automatisch erstellt.
Link	Ein Verweis, dargestellt durch das HTML-Element *<a href ... >*.
List	Eine Auswahlliste mit dem HTML-Element *<select>* mit Parameter *size* und dem HTML-Element *<option>*.
Popup	Eine Schaltfläche, dessen Betätigung die Anzeige eines kleinen Popup-Fensters mit Optionen bewirkt.
Radio-Button	Ein Optionsfeld, dargestellt durch das HTML-Element *<input type="radio"... >*. Mehrfachauswahl ist nicht erlaubt.
Table	Eine Tabelle, dargestellt durch das HTML-Element *<table>*.
Text-Input	Ein Eingabefeld eines HTML-Formulars, dargestellt durch das HTML-Element *<input type="text">*.

Komponenten und Widgets werden im Quelltext einer Transformation über das XML-Element *<HATS:Component>* eingebunden. Über verschiedene Attribute wird angegeben um welche Komponente es sich handelt, welches Widget benutzt wird, um welchen Screen-Bereich es sich handelt und welche Konfigurationen benutzt werden sollen.

3.3.2.3. Erstellung einer Transformation

Falls als HATS-Aktion eine Transformation gewünscht ist, kann bei der Erstellung einer Screen-Anpassung angegeben werden, automatisch eine leere Transformation in Form einer JSP mitzuerstellen. Die erstellte JSP kann entweder mit dem HATS Page Designer oder direkt im Quelltext bearbeitet werden. Der HATS Page Designer ist ein Editor, mit welchem HTML- und JSP-Elemente mit Drag & Drop der Transformation hinzugefügt werden können. Ebenfalls können damit Komponenten und die zugehörigen Widgets eingefügt werden, wie in Abbildung 3.6 zu sehen ist. Zuerst wird dazu auf dem zur Transformation gehörenden Screen mit Hilfe der Maus ein gelbes Rechteck über den zu transformierenden Bereich des Screens gezogen. In der Abbildung besteht der zu transformierende Bereich aus der Zeichenkette "ENTER ONE OF THE FOLLOWING" und einem Ausschnitt des darüber liegenden Eingabefeldes. Nach einem Mausklick auf „Next" bietet die HATS-Entwicklungsumgebung die zur Auswahl stehenden Komponenten mitsamt möglichen Widgets an (vgl. Abbildung 3.7). Wird eine Komponente ausgewählt, die für den ausgewählten Screen-Bereich nicht erkannt wurde, wird dies dem Benutzer durch eine Meldung im Feld „Component Preview" mitgeteilt. Zu den Komponenten passende Widgets, wie im Beispiel das Table-Widget, werden bei der Auswahl über eine Vorschau im Feld „Widget Preview" angezeigt. Mit den vertikal angeordneten Symbolen in der Mitte des Fensters können die Eigenschaften der Komponente und des Widgets definiert werden. Ebenfalls besteht über das Symbol „Text Replacement" die Möglichkeit, Zeichenketten des Screen-Bereichs, welche über reguläre Ausdrücke definiert werden können, zu löschen oder zu modifizieren.

3.3.2.4. Templates

Damit Transformationen einheitlich dargestellt werden, muss für die HATS-Anwendung ein Template definiert werden, in welches die Transformationen eingebettet werden. HATS bietet die Möglichkeit bereits vorgefertigte Templates zu verwenden oder eigene Templates zu erstellen. Das Template, welches ebenfalls eine JSP ist, kann analog zu einer Transformation im HATS Page Designer oder im Quelltext bearbeitet werden. Das Template besteht aus einem HTML-Grundgerüst, in welchem das XML-Element *<HATS:Transform>* den Bereich markiert, in welchen später die einzelnen Transformationen eingebettet werden.

Templates werden im HATS-Projekt unter dem Verzeichnis *Web Content/Templates* und zugehörende Bilder oder CSS-Dateien unter dem Verzeichnis *Web Content/Common/Images* beziehungsweise *Web Content/Common/Stylesheets* abgelegt.

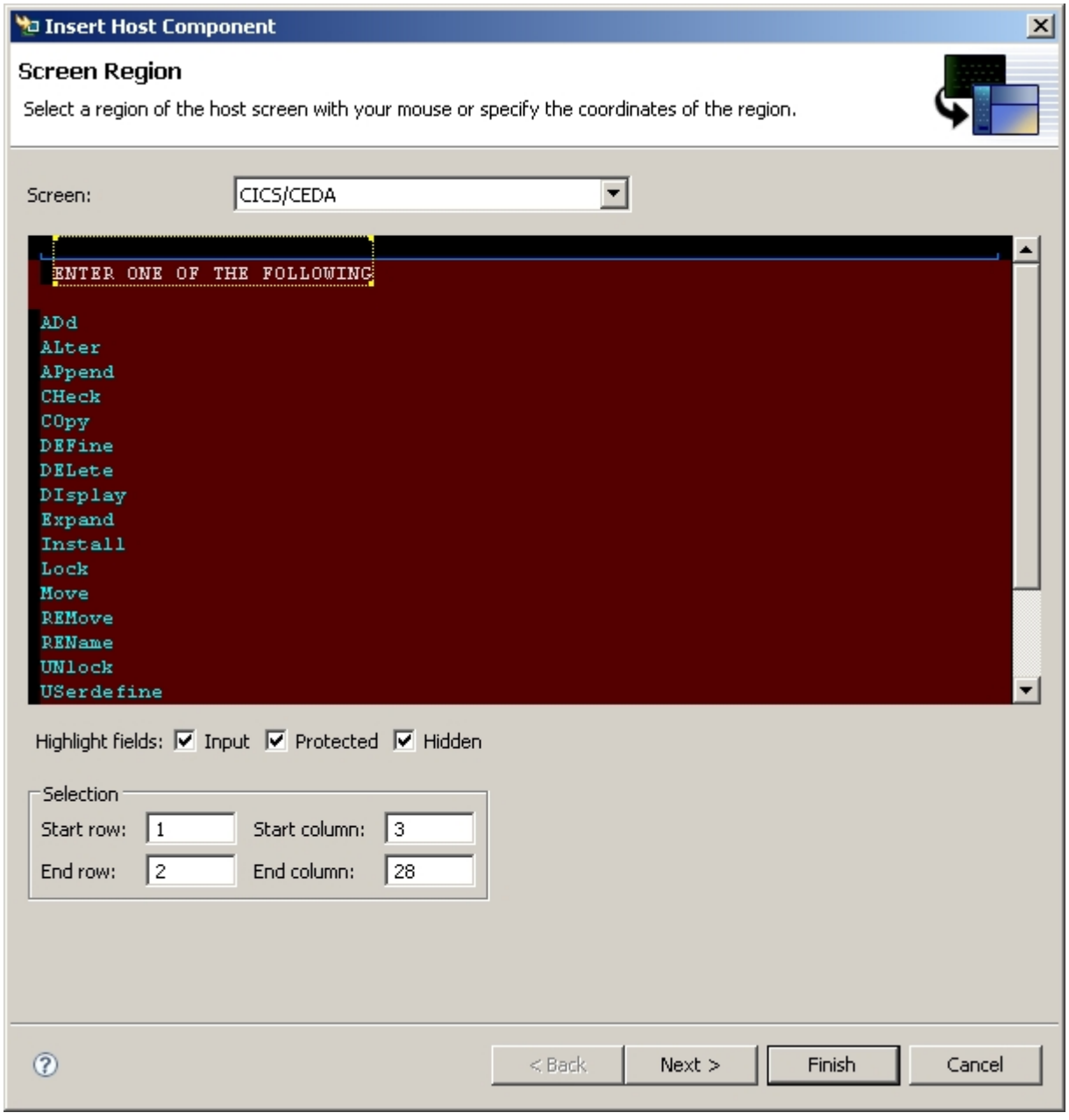

Abbildung 3.6.: Erster Schritt beim Einfügen eines Bereichs der Host-Anwendung

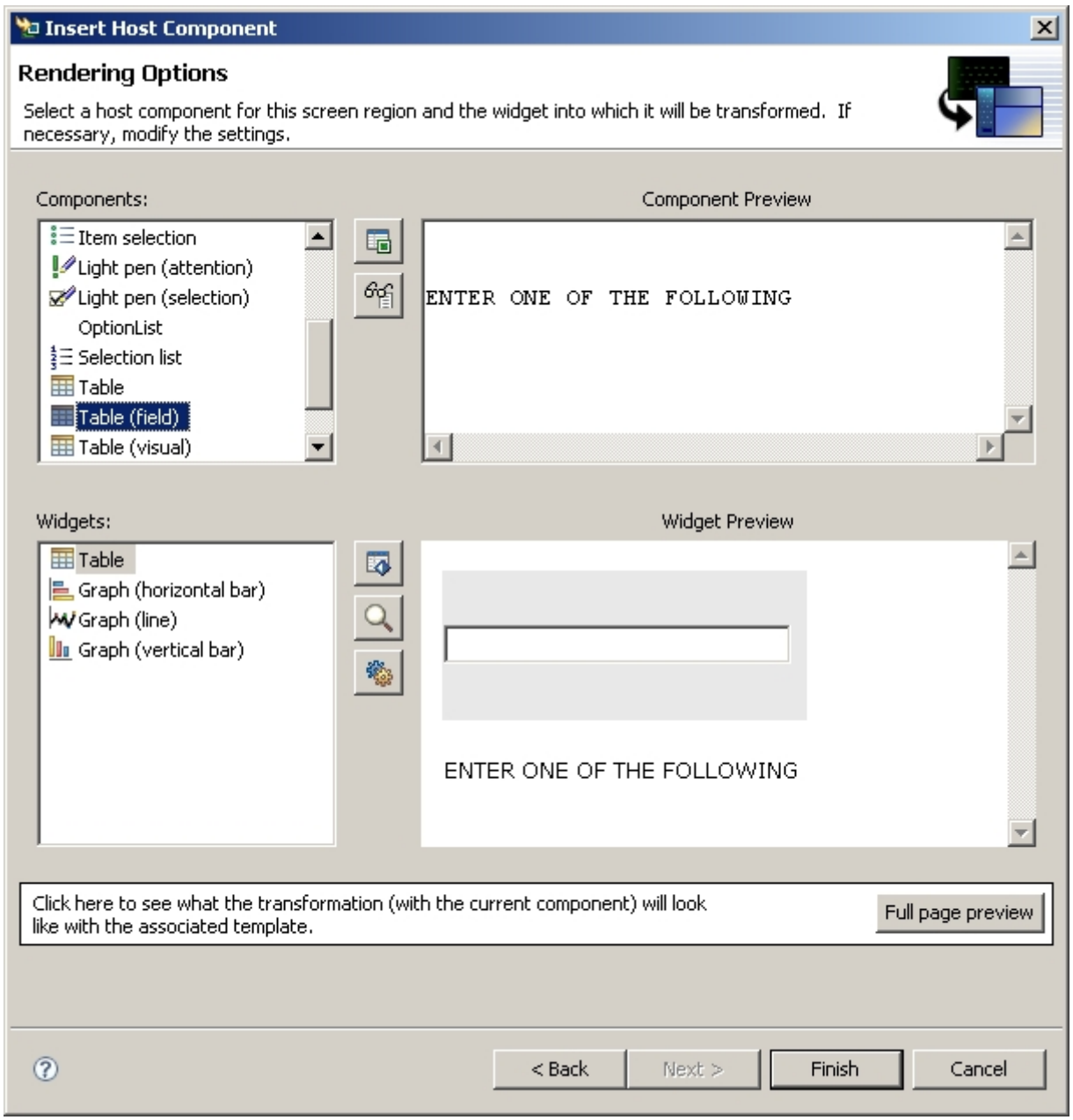

Abbildung 3.7.: Zweiter Schritt beim Einfügen eines Bereichs der Host-Anwendung

3.3.3. Funktionstasten

Damit der Benutzer die Möglichkeit hat, die Funktionstasten der 3270-Host-Anwendung innerhalb der HATS Web-Anwendung über die normale Tastatur zu benutzen, kann in den HATS-Projekteinstellungen unter dem Menüpunkt „Other/Keyboard Support“ die Tastaturunterstützung aktiviert werden. Die Funktionstasten F1-F12 werden abgebildet auf die Host-Funktionstasten PF1-PF12. Die Abbildung weiterer Host-Funktionstasten ist in [HATS3, Kapitel 16] zu finden.

Unter dem Menüpunkt „Rendering/Host Keypad“ kann in den Projekteinstellungen aktiviert werden, dass zu jeder Transformation der HATS Web-Anwendung eine Tastatur für die Funktionstasten angezeigt wird, wobei jede Taste durch eine HTML-Schaltfläche repräsentiert wird. Somit hat der Benutzer auf jeder Transformation vollständige Kontrolle über alle Funktionstasten mit der Maus. In der später vorgestellten HATS Web-Anwendung wurde hierauf verzichtet, da einzelne Funktionstasten kontextabhängig angezeigt werden sollen, um die wesentlichen Funktionstasten für den Benutzer besser hervorheben zu können.

Ebenso bietet HATS die Möglichkeit über den Menüpunkt „HATS Tools/Insert Host Keypad/Individual Key“ einzelne Tasten in Transformationen als Link oder Schaltfläche einzufügen. Dies geschieht im Gegensatz zu der Function-Key-Komponente unabhängig von den auf einem Host-Screen dargestellten Hinweisen zu Funktionstasten. Somit kann zum Beispiel auf jeder Transformation unabhängig vom jeweiligen Screen eine Schaltfläche eingefügt werden, welche bei einem Mausklick ein Drücken der Eingabetaste in der Host-Anwendung emuliert.

Das Deaktivieren von Funktionstasten wird von HATS momentan leider nicht vorgesehen. Um dennoch Funktionstasten zu deaktivieren oder umzubelegen, muss der Entwickler das für die Tastaturabwicklung zuständige JavaScript *KBS.js* im Projektverzeichnis *Web Content/common* entsprechend verändern. Um den JavaScript-Code im Projektverzeichnis sichtbar zu machen, muss in der Entwicklungsumgebung über den Menüpunkt „Window/Show View/Navigator“ in die Navigator-Ansicht gewechselt werden.

3.3.4. Prioritäten

Falls mehrere Screen-Anpassungen im HATS-Projekt existieren, kann es vorkommen, dass sich Erkennungskriterien der unterschiedlichen Screen-Anpassungen überlappen und somit Screens teilweise unerwünscht transformiert werden. Falls das Erkennungskriterium für eine Screen-Anpassung zum Beispiel die Zeichenkette “Menu“ beginnend an Position (1,1) und für eine andere Screen-Anpassung die Zeichenkette “Menulist“ ebenso beginnend an der Position (1,1) ist, gibt es eine Überlappung der Kriterien, bei welcher von HATS nur die älteste Screen-Anpassung aktiviert wird.

Um diesem unerwünschten Effekt vorzubeugen, bietet HATS die Möglichkeit Screen-Anpassungen verschiedene Prioritäten zu vergeben. Dabei wird während der Ausführung der HATS-Anwendung

ein Screen zuerst mit den Erkennungskriterien der Screen-Anpassung der höchsten Priorität verglichen und das Ganze, falls die Kriterien nicht zutreffen, mit Screen-Anpassungen absteigender Priorität fortgesetzt. Trifft keine Screen-Anpassung auf den aktuellen Screen zu, wird er mit Hilfe der Standardtransformation dargestellt. Die Vergabe der Prioritäten wird unter dem Reiter „Events“ in den Projekteinstellungen vorgenommen.

Damit nicht ständig die Prioritätsliste absteigend durchlaufen werden muss, kann der Entwickler zur Effizienzsteigerung bei der Erstellung einer Screen-Anpassung mögliche nachfolgende Screen-Anpassungen definieren. Wird diese Möglichkeit genutzt, vergibt HATS den auf die aktuelle Screen-Anpassung folgenden Screen-Anpassungen temporär höhere Prioritäten.

3.3.5. Standardtransformation

Für den Fall dass für einen Screen keine Screen-Anpassung existiert, wird der Screen über die Standardtransformation (Default Rendering) angezeigt. Die Standardtransformation versucht die Originalstruktur des Screens beizubehalten und diese mit HTML-Elementen nachzubilden. Dabei wird zum Beispiel versucht tabellarisch angeordnete Elemente des Screens als HTML-Tabelle oder Hinweise zu Funktionstasten als HTML-Links darzustellen.

Die Konfiguration der Standardtransformation geschieht unter dem Reiter „Rendering“ in den Projekteinstellungen. In der Konfiguration kann angegeben werden, welche Bereiche der Screens standardmäßig durch welche Komponenten und Widgets dargestellt werden sollen. Screen-Bereiche, die keinen Komponenten zugeordnet sind, werden standardmäßig durch die Field-Komponenten umgesetzt. Für Host-Anwendungen, welche nach einem gleichen Muster aufgebaut sind und einer einheitlichen Struktur verfolgen, kann mit Hilfe der Standardtransformation schnell eine ansehnliche HATS-Anwendung erstellt werden. Für Host-Anwendungen in welcher sich aufeinander folgende Screens jedoch wenig bis kaum ähnlich sind, erreicht man mit der Standardtransformation kein ansehnliches Ergebnis. Zum Beispiel transformiert die Standardtransformation jedes Eingabefeld der Host-Anwendung in ein HTML-Eingabefeld. In manchen Screens sind jedoch alle Felder der Host-Anwendung Eingabefelder, was mit der Standardtransformation zu einer unschönen HTML-Seite führt, welche gefüllt mit HTML-Eingabefeldern ist. In diesem Fall sollte mehr Aufwand in zusätzliche Screen-Anpassungen investiert werden.

3.3.5.1. Globale Regeln

Globale Regeln können ebenfalls in den Projekteinstellungen definiert werden und beschreiben wie die HATS-Anwendung bestimmte Host-Eingabefelder umsetzt. Eine globale Regel besteht aus Erkennungskriterien und einer Transformation für das Eingabefeld. So kann zum Beispiel definiert werden, dass alle Eingabefelder der Host-Anwendung, denen die Zeichenkette “Date“ vorangestellt ist, über das Calendar-Widget dargestellt werden. Globale Regeln sind, wie die Standardtransformation, sinnvoll, wenn die Host-Anwendung einer einheitlichen Struktur folgt.

Wird in der Host-Anwendung in einem Eingabefeld für ein Datum zum Beispiel eine Eingabe im Format DD.MM.YYYY und ein anderes Mal jedoch im Format YYYY.MM.DD erwartet, scheitert die globale Regel, da allein über eine voranstehende Zeichenkette des Eingabefeldes nicht genau definiert ist, in welchem Format die Eingabe erwartet wird.

3.3.6. Screen-Kombination

HATS erlaubt es den Inhalt mehrerer Screens zu lesen und dem Nutzer gebündelt auf einer einzelnen HTML-Seite zu präsentieren. Falls der Inhalt der zu lesenden Screens bei jedem Screen an der selben Position steht, ist es möglich eine Screen-Kombination [HATS3, Kapitel 7] zu verwenden. Ein Assistent hilft hierbei in der Entwicklungsumgebung Schritt für Schritt bei der Erstellung.

Falls sich der Inhalt der zu lesenden Screens jedoch nicht ständig an der selben Position oder teilweise sogar in unterschiedlichen Host-Anwendungen befindet, kann die Screen-Kombination nicht verwendet werden. Um den gewünschten Effekt dennoch zu realisieren, kann der Entwickler HATS-Makros und gegebenenfalls Integration Objects dazu einsetzen [HATS3, Kapitel 14]. Diese Konzepte werden in den nachfolgenden Abschnitten vorgestellt.

3.4. Makros

Ein Makro ist ein Ablauf, welcher eine fest vorgeschriebene Folge von Aktionen enthält. Mögliche Aktionen im Hinblick auf HATS-Makros sind den Benutzer zu einer Eingabe aufzufordern, eine Eingabe mit einer festgelegten Zeichenkette oder einer globalen Variable zu füllen, Tastendrücke in der Host-Anwendung zu emulieren, Zeichenketten an einer festgelegten Position auszulesen und falls gewünscht die Zeichenkette in einer globalen Variable zu speichern.

Ein Beispiel für den möglichen Einsatzort für ein HATS-Makro ist zum Beispiel eine Benutzeranmeldung in einer Host-Anwendung, die sich über mehrere Screens erstreckt. In der Host-Anwendung existiert dafür zum Beispiel ein Screen für die Eingabe der Benutzerkennung und ein separater Screen zur Eingabe des Kennwortes. Mit Hilfe eines HATS-Makros kann man diesen Anmeldevorgang dem Benutzer vereinfachen, indem der Benutzer in der HATS-Anwendung lediglich eine HTML-Seite mit zwei HTML-Eingabefeldern für die Benutzerkennung und das Kennwort angezeigt bekommt. Nach dem Ausfüllen dieser zwei Eingabefelder wird das Makro aktiv und speichert die eingegebenen Werte intern in zwei globalen Variablen. Anschließend navigiert das Makro selbständig in der Host-Anwendung zum jeweils passenden Screen und füllt dort automatisch die entsprechenden Eingabefelder mit den Daten des Benutzers. Somit kann dem Benutzer vom Makro Arbeit abgenommen und ein Vorgang vereinfacht werden.

3.4.1. Globale Variablen

Auf globale Variablen kann innerhalb einer Verbindung mit einer HATS-Anwendung ständig zugegriffen werden. Sie sind daher vergleichbar mit temporären Variablen, die während einer Verbindung definierte Inhalte von anderen Screens oder Benutzereingaben aufnehmen und bereitstellen können. Der Inhalt von globalen Variablen kann in Transformationen später als Text ausgegeben oder als Eingabe für Host-Eingabefelder verwendet werden. Eine globale Variable kann entweder durch eine Benutzereingabe oder in Folge einer Screen-Erkennung, mit automatisch extrahiertem Text aus einem Screen gefüllt werden. Hierbei kann spezifiziert werden, ob der Inhalt an die globale Variable angehängt oder ob die globale Variable damit überschrieben werden soll.

Globale Variablen können einen Integer, einen String oder einen indexierten Array aus Strings aufnehmen. Falls der indexierte Array verwendet wird, kann jeweils angegeben werden, ab welchem Index die globale Variable mit Daten gefüllt werden soll. Weitergehend kann unterschieden werden zwischen geteilten globalen Variablen und lokalen globalen Variablen. Auf geteilte globale Variablen können mehrere HATS-Anwendungen innerhalb der selben EAR-Datei zugreifen, wobei auf lokale globale Variablen nur innerhalb einer HATS-Anwendung zugegriffen werden kann. In dieser Diplomarbeit, sowie auch in der HATS-Dokumentation [HATS3, Kapitel 12], wird mit einer globalen Variablen immer eine lokale globale Variable gemeint. Ebenso ist es möglich über globale Variablen mit zugehöriger Geschäftslogik zu kommunizieren. Da für diese Diplomarbeit keine separate Geschäftslogik benötigt wurde, wird an dieser Stelle nicht näher auf die Integration von Geschäfslogik eingegangen. Mehr Informationen zu diesem Thema findet der Leser in [HATS4, Kapitel 2].

3.4.2. Makros aufzeichnen

Makros lassen sich in der HATS-Entwicklungsumgebung über das in Abschnitt 3.3.1 beschriebene Host Terminal aufzeichnen. Dazu müssen für jeden Screen, der in den Makroablauf involviert ist, Erkennungskriterien sowie gewünschte Aktionen definiert werden. Eine Übersicht über die involvierten Screens und die ausgewählten Aktionen liefert der Makro-Navigator, welcher sich am linken Rand des Host Terminals befindet (vgl. Abbildung 3.8). Für jeden Screen werden im Makro-Navigator baumförmig die zugehörigen Aktionen sowie die nachfolgenden Screens aufgelistet. Die jeweiligen Aktionen werden bei der Erstellung des Makros in Echtzeit aufgezeichnet und in die Baumstruktur eingegliedert. Spezielle Aktionen, wie zum Beispiel das Einfügen oder Extrahieren von Zeichenketten, initiiert man über die Symbole „Add Prompt Action“ beziehungsweise „Add Extract Action“ an der oberen Seite des Host Terminals. Nach der Aufzeichnung des Makros besteht die Möglichkeit in der Baumstruktur einzelne Aktionen zu verschieben, zu bearbeiten oder zu löschen. Anschließend wird das Makro als XML-Code mit der Endung *.hma* im HATS-Projektverzeichnis *Macros* gespeichert.

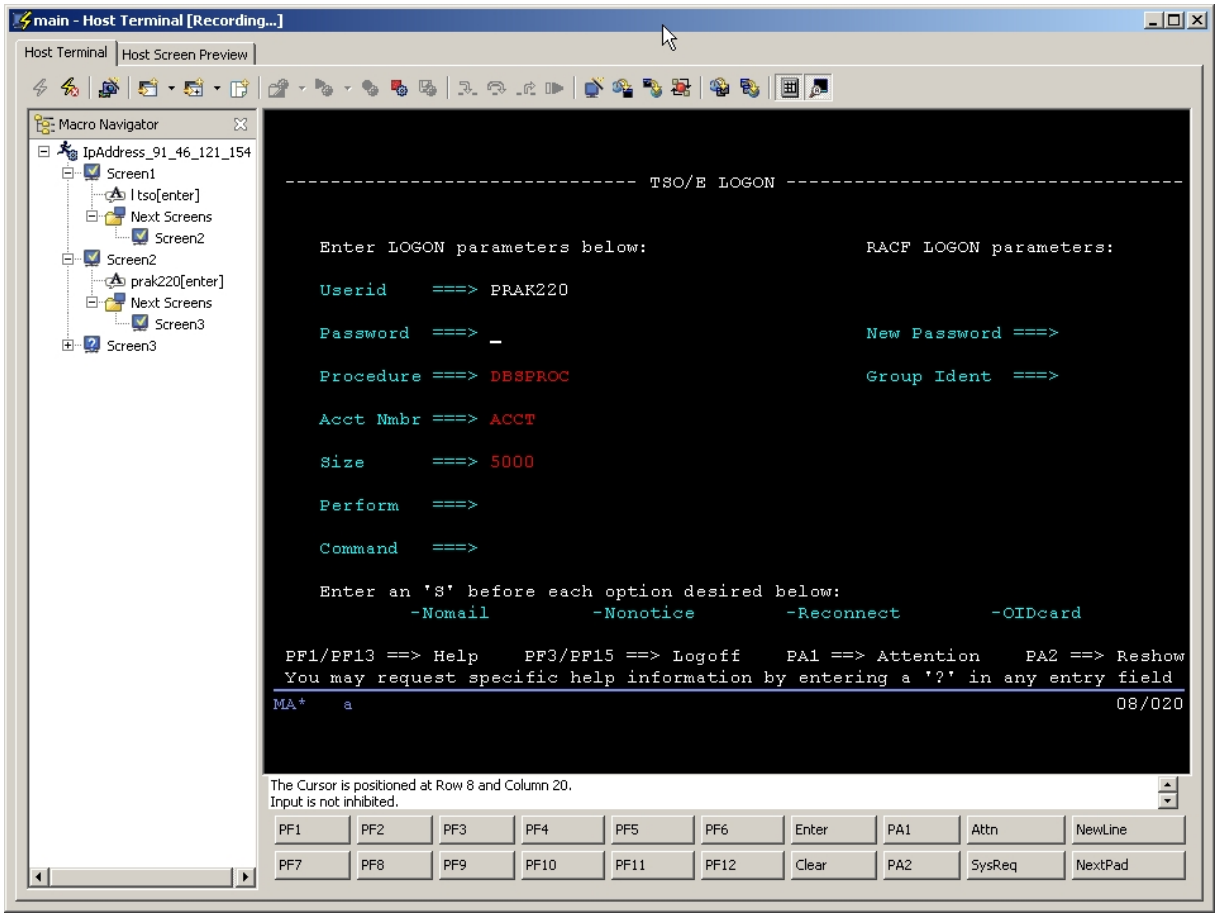

Abbildung 3.8.: Aufzeichnen eines HATS-Makros

3.4.3. Makros editieren

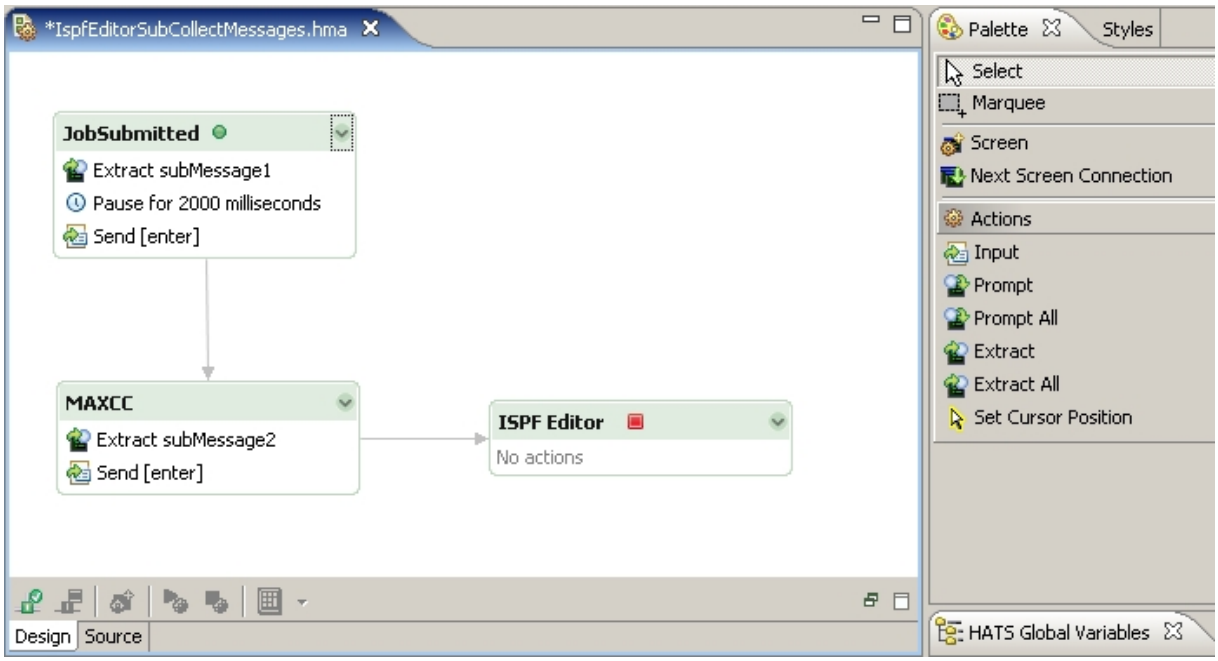

Abbildung 3.9.: HATS Visual Macro Editor

Um Makros zu editieren existieren mehrere Möglichkeiten. Man kann das Makro entweder direkt im XML-Quelltext mit dem Advanced Macro Editor oder mit dem Visual Macro Editor bearbeiten. Über einen Rechtsklick auf das Makro und anschließender Auswahl des Menüpunktes „Open With“ aus dem aufklappenden Menü kann das Makro im gewünschten Editor geöffnet werden.

Der Visual Macro Editor ist dabei mit Abstand der komfortabelste Editor, da hierbei der mögliche Ablauf des Makros sofort visuell ersichtlich ist. Jeder Screen wird als abgerundetes Rechteck dargestellt, welches die zum Screen gehörenden Aktionen beinhaltet. Möglicherweise nachfolgende Screens werden durch ausgehende Pfeile gekennzeichnet. Screens können zusätzlich als Start- oder Exit-Screens gekennzeichnet werden. Ein Makro startet immer in einem Start-Screen und endet immer in einem Exit-Screen. Falls dieser nicht vorhanden ist oder nie in der Host-Anwendung angesprungen wird, verweilt das Makro in einer Endlosschleife, bis eine dem Makro zugewiesene Zeitüberschreitung eintritt und das Makro beendet. Diese Zeitüberschreitung wird von HATS standardmäßig festgelegt, kann vom Entwickler jedoch im Advanced Macro Editor angepasst werden. Start-Screens erkennt man an einem grünen Punkt, Exit-Screens an einem roten Quadrat. In Abbildung 3.9 ist ein Beispiel eines Makros im Visual Macro Editor zu sehen. Sobald der Screen *JobSubmitted* erkannt wurde, wird als Aktion ein Bereich des Screens in die globale Variable *subMessage1* extrahiert, zwei Sekunden lang gewartet und anschließend ein Tastendruck auf die Eingabetaste in der Host-Anwendung emuliert. Im folgenden Screen *MAXCC* wird ein definierter Bereich des Screens in die globale Variable *subMessage2* gelesen und erneut

erneut ein Tastendruck auf die Eingabetaste emuliert. Im letzten Screen *ISPF Editor* hat das Makro alle wichtigen Informationen gesammelt und wird beendet.

Mit der Visualisierung des Makros lässt sich ein Makroablauf im Vergleich zu reinem XML-Quelltext sehr effektiv bearbeiten. Um Screens zu verbinden oder um Screens Aktionen hinzuzufügen, kann der Entwickler die im Reiter „Palette“ angebotenen Symbole verwenden. Um Makros zu testen, verwendet der Entwickler das Symbol „Connect“ an der unteren Seite des Fensters. Damit ist es möglich in der Host-Anwendung zum Start-Screen des Makros zu navigieren und an dieser Stelle das Makro ablaufen zu lassen. Vor dem Ablauf des Makros wird der Entwickler nach eventuell verlangten Benutzereingaben gefragt. Extrahierte globale Variablen werden dem Entwickler nach dem Ablauf des Makros angezeigt.

Der Advanced Macro Editor ist weniger übersichtlich, bietet dafür aber im Gegensatz zum Visual Macro Editor mehr Freiheiten bei den Einstellungen des Makros. Der in Abbildung 3.10 dargestellte Editor besteht aus den vier Reitern „Makro“, „Screens“, „Links“ und „Variables“. Der Reiter „Makro“ dient dazu, generelle Einstellungen des Makros festzulegen, wie zum Beispiel eine Zeitüberschreitung der Laufzeit oder die Zeit, die nach der Ausführung der Aktionen eines Screens in Millisekunden gewartet werden soll. Im Reiter „Screen“ kann über eine Auswahlliste für jeden am Makro beteiligten Screen die Einstellung geladen und angezeigt werden. Neben Erkennungskriterien für Screens und den in Screens auszuführenden Aktionen finden sich an dieser Stelle zahlreiche weitere Optionen. Im Reiter „Links“ können die Verbindungen zwischen den einzelnen Screens definiert werden. Ebenso kann hier für jeden Screen eine Zeitüberschreitung festgelegt werden, für den Fall dass kein gültiger nachfolgender Screen erkannt wird. Im Reiter „Variables“ können Makro-Variablen definiert werden, welche im Gegensatz zu globalen Variablen nur innerhalb eines Makros gültig sind. Da die zahlreichen Makroeinstellungen sehr detailliert und umfangreich sind, sei an dieser Stelle für mehr Informationen auf [HATS5] verwiesen.

3.4.4. Format

Ein Makro wird, wie bereits erwähnt, in der HATS-Entwicklungsumgebung als XML-Datei mit der Endung *.hma* gespeichert. Zur Erläuterung des Formats befindet sich in Listing 3.1 der Quelltext eines Makros. Dieses Makro wird aktiviert, sobald auf einem Screen die Zeichenkette “SUBMITTED“ erkannt wird. Es extrahiert dann aus dem Screen eine definierte Zeichenkette in die globale Variable *subMessage1* und emuliert in der Host-Anwendung ein Drücken der Eingabetaste.

Das Wurzelelement stellt bei einem Makro das Element <*macro*> dar, welches die Kind-Elemente <*associatedConnections*>, <*extracts*>, <*prompts*> und <*hascript*> enthält. Das Element <*associatedConnections*> in der dritten Zeile wird verwendet um dem Makro die zugehörige Host-Verbindung mitzuteilen. In HATS wird zu Projektbeginn eine Host-Verbindung unter anderem mit Name, IP-Adresse, Port und Codepage definiert. Der Name dieser Host-Verbindung wird in

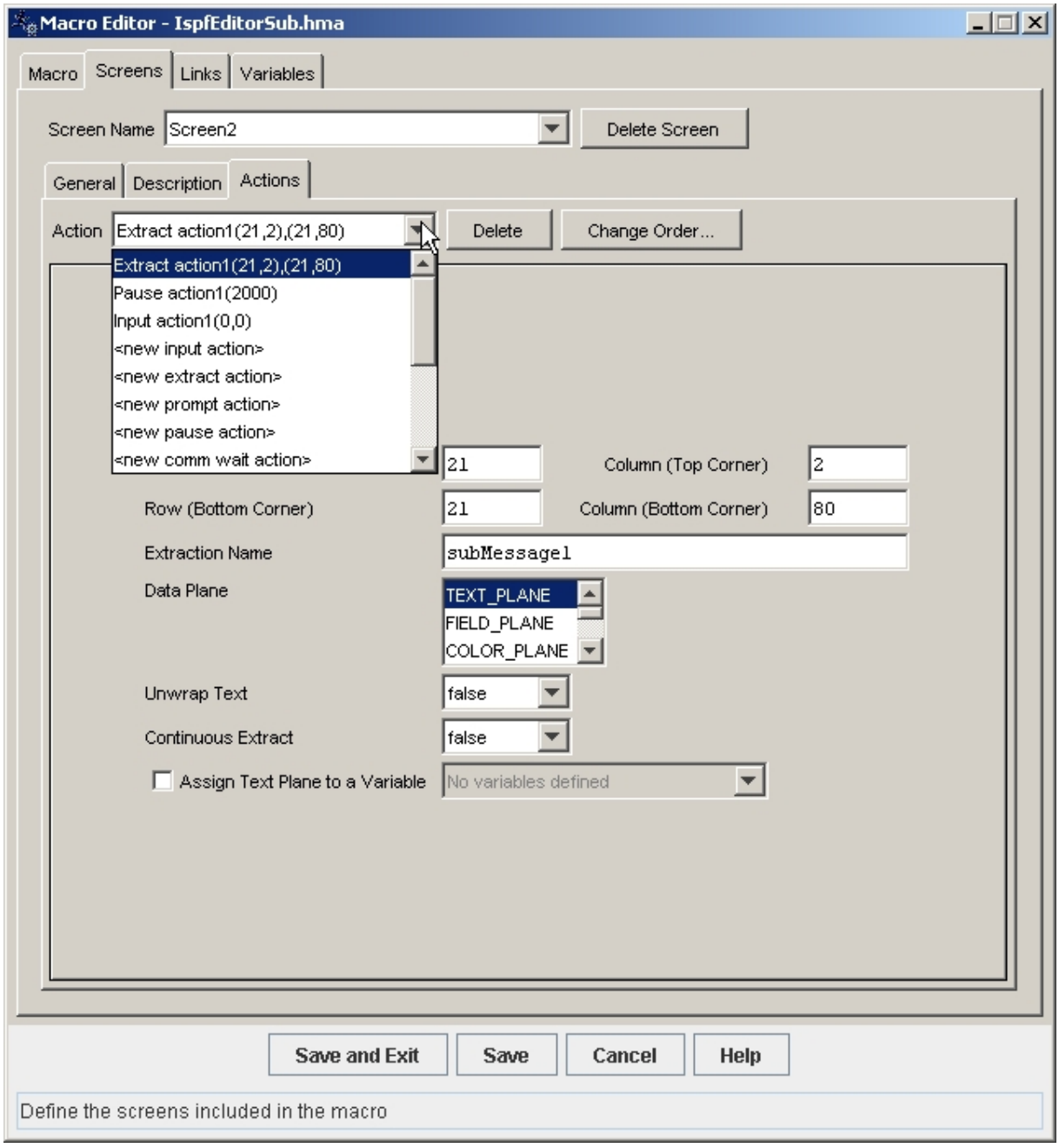

Abbildung 3.10.: HATS Advanced Macro Editor

das Kind-Element <*connection*> über das Attribut *name* eingetragen.

Über das Element <*extracts*> in Zeile 7 wird definiert, wie das Makro Daten extrahiert. Für jede Extraktion wird ein Kind-Element <*extract*> angelegt. Über Die Attribute *handler* und *showHandler* kann angegeben werden, ob nach Beendigung des Makros eine JavaServer Page geladen werden soll, welche die extrahierten Elemente anzeigt oder weiterverarbeitet. Ebenso kann über die Attribute *variableName* und *save* eine globale Variable als Zielort angegeben werden. Mit den Attributen *index* und *indexed* lässt sich die Extraktion in indexierte globale Variablen steuern. Das Attribut *override* gibt dabei an, ob die globale Variable überschrieben werden soll.

Analog zum Element <*extracts*> existiert für Benutzereingaben das Element <*prompts*> mit den zugehörigen Kind-Elementen vom Typ <*prompt*>. Wichtig hierbei sind die Attribute *handler*, *source* und *value*. Mit dem Attribut *handler* kann eine JavaServer Page angegeben werden, welche für die Benutzereingaben zuständig ist. Das Attribut *source* legt fest, ob die Benutzereingabe aus einer globalen Variable oder aus einem Eingabefeld gelesen werden soll. Das Attribut *value* beschreibt abhängig vom Attribut *source* den Namen der globalen Variable oder einen Standardwert für die Benutzereingabe. Das in Listing 3.1 abgebildete Makro verwendet keine Benutzereingaben, weshalb das Element <*prompts*> in Zeile 14 in sich abgeschlossen ist.

Das Element <*hascript*> in Zeile 16 enthält die Kind-Elemente <*comment*>, <*import*>, <*variables*> und <*screen*>. Das Element <*import*> ist optional und dient zur Einbindung von Java-Klassen, worauf an dieser Stelle jedoch nicht weiter eingegangen wird. Im Element <*variables*> können Makro-Variablen definiert und initialisiert werden. Das Element <*screen*> in Zeile 20 existiert für jeden in das Makro involvierten Screen und besitzt wiederum vier weitere Kind-Elemente <*comment*>, <*description*>, <*actions*> und <*nextscreens*>.

Das Element <*description*> in Zeile 22 beschreibt die Kriterien, die benötigt werden um den Screen zu erkennen. Dabei beschreibt das Attribut *uselogic*, welche nachfolgenden Kind-Elemente in welcher logischen Kombination als Kriterium verwendet werden. Das Kind-Element <*oia*> dient zur Unterscheidung von blockierten und nicht blockierten Screens, worauf in Abschnitt 5.1.8.1 näher eingangen wird. Das Kind-Element <*string*> beschreibt über seine Attribute das Erkennungskriterium, das besagt dass die Zeichenkette “SUBMITTED“ an beliebiger Position auf dem Screen auftreten muss. In Zeile 29 beschreibt das Element <*actions*> die auszuführenden Aktionen, falls der Screen erkannt wurde. In diesem Makro sind dies die über die Kind-Elemente beschriebenen Aktionen <*extract*> und <*input*>. Das Element <*nextscreens*> in Zeile 37 beschreibt über die Kind-Elemente vom Typ <*nextscreen*> die dem Makroablauf logisch folgenden Screens.

Das Element <*comment*> in Zeile 17 und Zeile 21 ist bezüglich des Makroablaufs nicht relevant und kann vom Entwickler zur Dokumentation des Quellcodes benutzt werden.

```
<macro>

  <associatedConnections default="main">
    <connection name="main"/>
  </associatedConnections>

  <extracts>
    <extract handler="default.jsp" index="-1" indexed="false"
            name="subMessage1" overwrite="true" save="true"
            separator="" showHandler="false"
            variableName="subMessage1"/>
  </extracts>

  <prompts />

  <hascript>
    <comment />
    <import />
    <variables />
    <screen>
      <comment>screen 1</comment>
      <description uselogic="1 and 2">
        <oia invertmatch="false"
             optional="false" status="NOTINHIBITED"/>
        <string casesense="false" col="1" ecol="-1"
               erow="-1" invertmatch="false" optional="false"
               row="1" value="SUBMITTED" wrap="false"/>
      </description>
      <actions>
        <extract assigntovar="" continuous="false"
                 ecol="78" erow="1" name="subMessage1"
                 planetype="TEXT_PLANE" scol="2"
                 srow="1" unwrap="false"/>
        <input col="0" encrypted="false" movecursor="true" row="0"
              value="[enter]" xlatehostkeys="true"/>
      </actions>
      <nextscreens>
        <nextscreen name="Screen 2"/>
      </nextscreens>
    </screen>
  </hascript>

</macro>
```

Listing 3.1: Format eines HATS-Makros

3.5. Integration Objects

Integration Objects sind Java Beans, welche Interaktionen mit einer Host-Anwendung kapseln. Benötigt werden Integration Objects, falls eine HATS-Anwendung mehrere gleichzeitige Verbindungen zu Host-Anwendungen auf einem oder auf verschiedenen Hosts aufbauen muss oder der Entwickler eine Interaktion mit einer Host-Anwendung in EJBs oder Web Services kapseln möchte. Integration Objects basieren immer auf Makros, welches die zu kapselnden Aktionen beschreiben [HATS3, Kapitel 13].

Ein Integration Object wird zum Beispiel verwendet, falls dem Benutzer innerhalb einer HATS-Anwendung, welche momentan ISPF umsetzt, zusätzlich Informationen aus einer auf einem anderen Host laufenden CICS-Transaktion dargestellt werden sollen. Das Integration Object kapselt dabei die folgenden Aktionen: Einloggen in CICS, Speichern der gewünschten Informationen in globale Variablen und das Abmelden von CICS. Denkbar wäre ebenso diesen Vorgang in einen Web Service zu kapseln, über den die extrahierten Informationen abrufbar sind.

Erstellt wird ein Integration Object durch einen Rechtsklick auf ein bestehendes Makro und anschließender Auswahl des Menüpunktes „Create Integration Object“ im aufklappenden Menü. Dadurch wird im Projektverzeichnis *Source/IntegrationObject* eine Java Bean mit dem Namen des Makros angelegt. Aus einem erstellten Integration Object können für die Benutzereingabe und für die Ausgabe der gesammelten Informationen Web-Anwendungen erstellt werden. HATS bietet zur Erstellung die verschiedene Ansätze Model 1, JavaServer Faces (JSF) und Struts an. Der Ansatz Model 1 erstellt jeweils eine JSP für die Ein- und Ausgabe, wobei jede JSP die komplette Logik enthält. Die Ansätze JSF und Struts erstellen zusätzlich Servlets und Java Beans, um die Web-Anwendung nach dem MVC-Prinzip zu strukturieren. Für nähere Informationen zu den verschiedenen Ansätzen sei auf [WIK1] und [STAR1] verwiesen.

Mehrere Intergration Objects mit zusammenhängenden Teilaufgaben können miteinander verkettet werden. In der HATS-Terminologie spricht man hierbei von einer Integration Objects Chain (vgl. Abbildung 3.11). Die Integration Objects Chain startet Integration Object A, welches seine gekapselten Aktionen durchführt und den Zustand der aktuellen Verbindung speichert. Integration Object B lädt den gespeicherten Zustand und beginnt seine Aktionen dort, wo das Integration Object A aufgehört hat. Dies zieht sich durch die komplette Kette fort, bis das letzte Integration Object die Verbindung wieder trennt. Durch diesen Vorgang kann ein komplexer Vorgang zur besseren Übersicht in Teilaufgaben zerlegt und in einer Integration Objects Chain gebündelt untergebracht werden.

3.6. HATS-Anwendungen debuggen

Bei umfangreichen Makros kommt es während der Entwicklung oft vor, dass ein Makro in einer Endlosschleife landet und nie den als Exit-Screen gekennzeichneten Screen erreicht. Ebenso

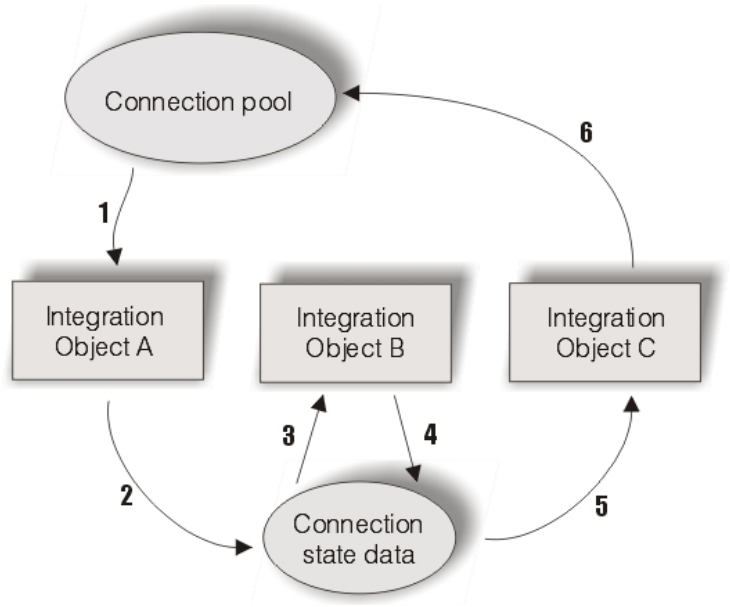

Abbildung 3.11.: HATS Integration Objects Chain (Quelle: [HATS3])

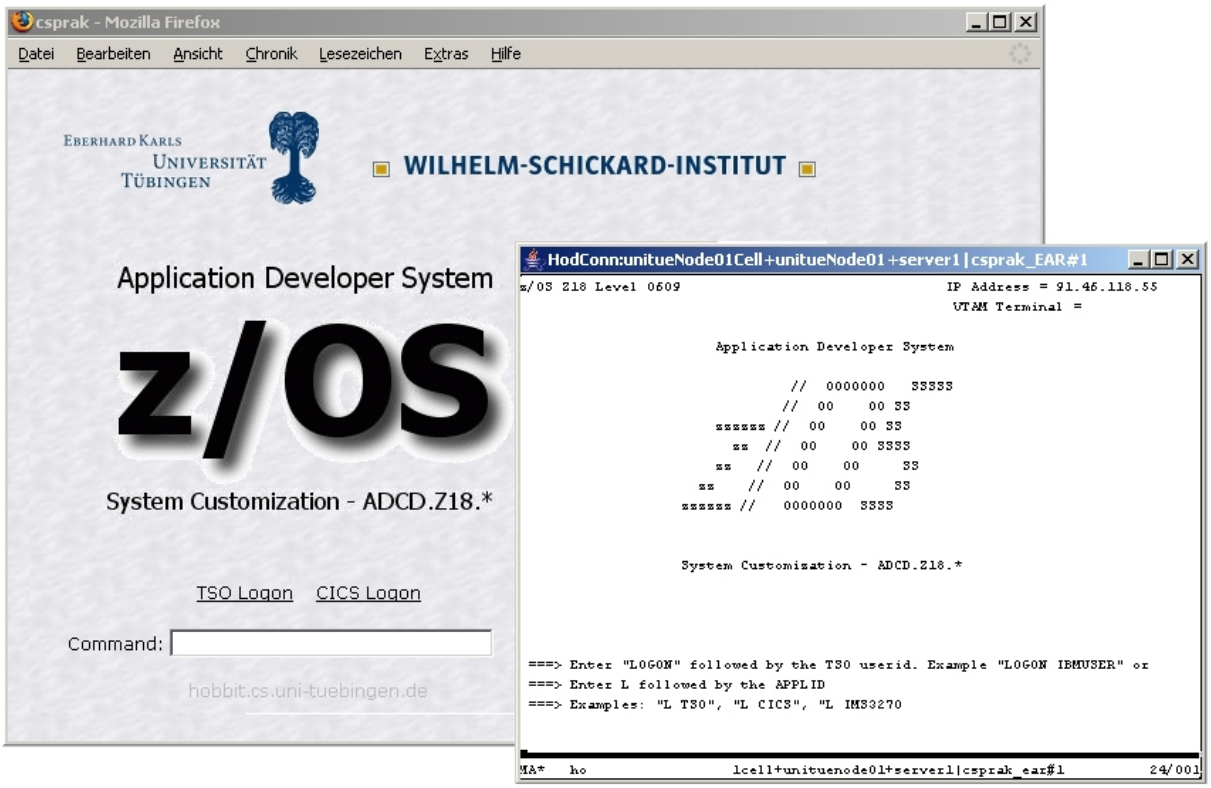

Abbildung 3.12.: HATS Debug-Ansicht

kann es bei zahlreichen Screen-Anpassungen Überlappungen in den Erkennungskriterien geben, so dass Screen-Anpassungen, aufgrund falsch gesetzter Prioritäten, nicht wie geplant aktiviert werden. Um dem Entwickler in solchen Fällen die Problemsuche zu vereinfachen, bietet die HATS-Entwicklungsumgebung die Möglichkeit, durch einen Rechtsklick auf das Projekt und anschließender Auswahl des Menüpunktes „Debug on Server“ eine Web-Anwendung während der Ausführung zu debuggen.

Dazu bekommt der Entwickler die in Abbildung 3.12 dargestellte Debug-Ansicht mit zwei Fenstern angezeigt. Im rechten Fenster wird der ursprüngliche Screen der Host-Anwendung gezeigt und im linken Fenster zeitgleich die Umsetzung in der HATS Web-Anwendung. Der Entwickler kann durch beide Fenster navigieren, wobei beide Fenster synchron gehalten werden. Wird in der Web-Anwendung durch eine Screen-Anpassung ein Makro aktiviert, werden im rechten Fenster alle Screens, welche das Makro durchläuft, mitsamt der vom Makro darauf ausgeführten Aktionen, in Echtzeit angezeigt.

Zusätzlich werden in der Konsole der Entwicklungsumgebung laufend Meldungen über den aktuellen Zustand der HATS-Anwendung ausgegeben. Wird während der Ausführung ein Screen erkannt und in Folge der Screen-Anpassung eine Transformation angewendet, wird dies in der HATS-Konsole zum Beispiel durch die in Listing 3.2 abgebildeten Meldungen angezeigt. Die Information *Processing Event* beschreibt dabei, welche Screen-Anpassung für den Screen erkannt wurde. Die nachfolgende Information *Applying Transformation* beschreibt, welche Transformation infolge der Screen-Anpassung auf den Screen angewandt wurde.

```
1  SystemOut    0   [csprak] -
2    Processing event: ISPF/IspfPrimaryOptionMenu
3
4  SystemOut    0   [csprak] -
5    Applying transformation: IspfPrimaryOptionMenu.jsp
6
7  ServletWrappe I SRVE0242I: [csprak_EAR] [/csprak]
8    [/transformations/IspfPrimaryOptionMenu.jsp]: Initialization successful.
```

Listing 3.2: HATS Konsole

3.7. Asynchrone Updates

Manche 3270-Host-Anwendungen aktualisieren selbständig Screens ohne Interaktion des Benutzers. Diesen Vorgang bekommt die HATS Web-Anwendung nicht mit, da die Host-Anwendung nur synchron bei Änderungen in der Web-Anwendung kontaktiert wird. Lediglich ein Benutzen der „Neu Laden"-Funktion des Web-Browsers würde in diesem Fall eine Aktualisierung der HATS Web-Anwendung bewirken.

Um eine asynchrone Aktualisierung zu ermöglichen, bietet HATS an, das Java Applet Asynchronous Update über die Projekteinstellungen einzubinden (Abbildung 3.13). Das Applet baut über einen einstellbaren Port eine Socket-Verbindung zum Websphere Application Server auf, durch welchen der Client über Aktualisierungen der Host-Anwendung informiert werden kann. Wird keine Portnummer angegeben, wählt die HATS-Anwendung zufällig einen unbelegten Port oberhalb des Ports 1024.

Falls der Benutzer einen Webbrowser schließt, welcher aktuell eine HATS Web-Anwendung geöffnet hat, besteht über das Applet und die Socket-Verbindung zudem die Möglichkeit, dem WAS mitzuteilen, dass die Verbindung zum Client geschlossen werden soll. Dadurch können der Verbindung zugeteilte Ressourcen sofort wieder freigegeben werden, was sich positiv auf die Auslastung des Servers auswirkt.

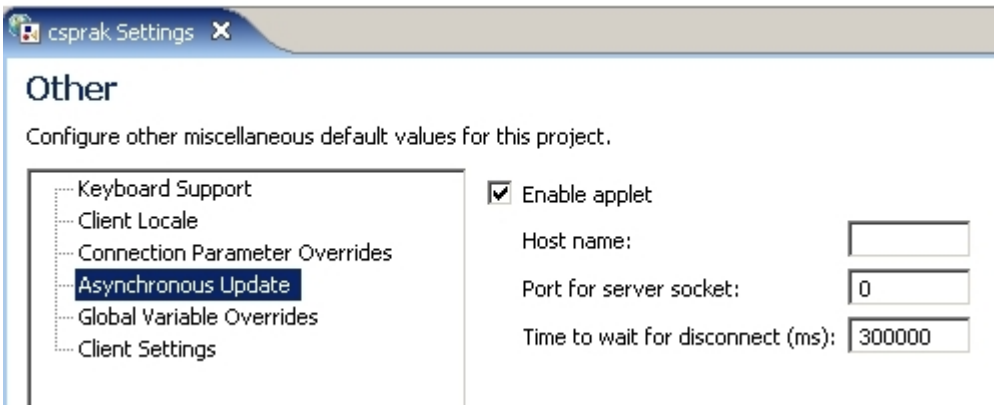

Abbildung 3.13.: HATS Asynchronous Update Applet

Aufgrund dieses Applets, muss ein Webbrowser für die lückenlose Funktion einer HATS Web-Anwendung, eine lokale JAVA Laufzeitumgebung besitzen. Ist auf dem Client keine JAVA-Laufzeitumgebung vorhanden, funktioniert die Web-Anwendung lediglich mit Einschränkungen. Falls sich in diesem Fall die Host-Anwendung selbstständig aktualisiert, muss der Benutzer die „Neu Laden"-Funktion des Web-Browsers benutzen, um die Änderung in der HATS Web-Anwendung angezeigt zu bekommen.

3.8. HATS-Administrationskonsole

Für jede HATS Web-Anwendung existiert die Möglichkeit eine Administrationskonsole zu aktivieren. Die Administrationskonsole ist über den Zusatz */hatsadmin/admin* an die URL der HATS Web-Anwendung direkt auf dem Websphere Application Server erreichbar und bietet von dort die Möglichkeit Verbindungen, Lizenzen und Benutzer zu verwalten. Ebenso lässt sich die Aufzeichnung von Protokollen aktivieren, welche zum Beispiel zur Fehlersuche verwendet werden können. Durch die Zuweisung von Sicherheitsrollen im Websphere Application Server, kann die Nutzung der Administrationskonsole feingranular auf ausgewählte Benutzer beschränkt werden. In dieser Diplomarbeit wurde für die entwickelte HATS-Anwendung der Zugriff auf die Administrationskonsole eingeschränkt und kann nur von Benutzern aufgerufen werden, welche einen Account auf dem darunterliegenden z/Linux-Betriebssystem besitzen.

Abbildung 3.14 zeigt die Administrationskonsole. Dabei werden im Menüpunkt „Verbindungsverwaltung/Hostverbindungen“ alle momentan mit einer HATS Web-Anwendung verbundenen Benutzer mit ihrer IP-Adresse angezeigt. An dieser Stelle können bestehende Verbindungen auch getrennt werden, falls zum Beispiel durch eine fehlerhaft entwickelte Web-Anwendung der Benutzer in einer Endlosschleife landet und die der Anwendung zugewiesene Zeitüberschreitung zu hoch eingestellt ist.

Für weitergehende Informationen zur HATS-Administrationskonsole sei auf [HATS3, Kapitel 18] verwiesen.

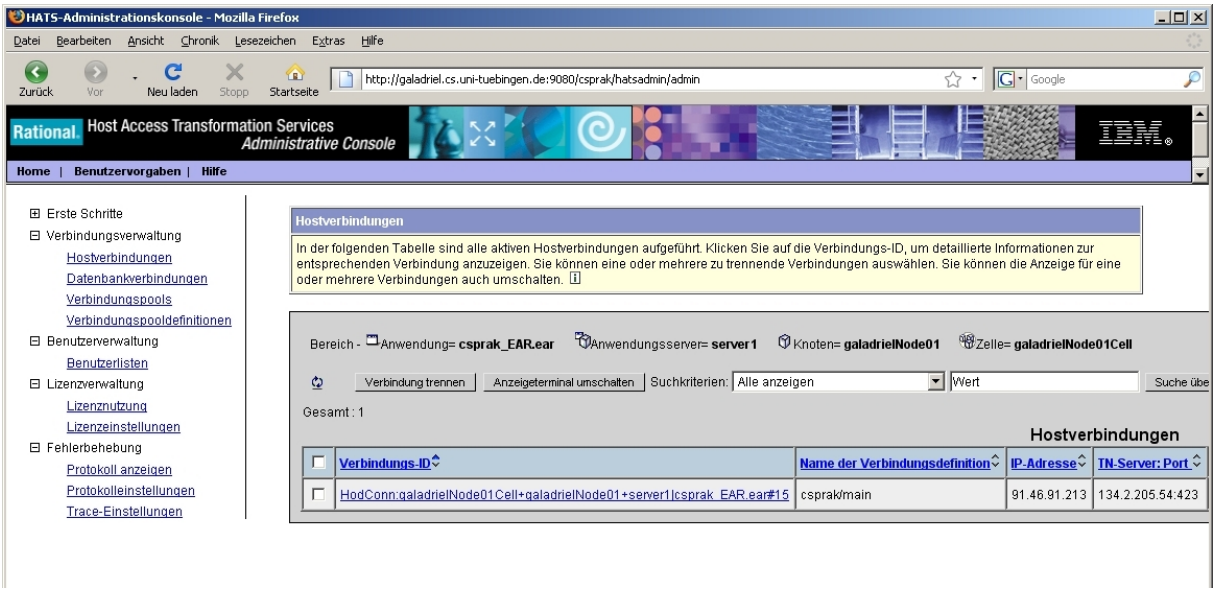

Abbildung 3.14.: HATS-Administrationskonsole

3.9. Analyse einer HATS Web-Anwendung

Um eine HATS Web-Anwendung auf einem Websphere Application Server zu installieren, muss die Anwendung aus der Entwicklungsumgebung im EAR-Format exportiert werden. Dies geschieht durch einen Rechtsklick auf das Projekt und anschließender Auswahl des Menüpunktes „Export Project“. Mit den nachfolgenden Eingabefeldern definiert man das Zielverzeichnis und den Namen der EAR-Datei. Diese EAR-Datei muss anschließend auf dem Websphere Application Server über die Verwaltungskonsole installiert werden, was in Abschnitt 4.4 näher erläutert wird.

Für den Inhalt einer solchen EAR-Datei existiert keine HATS-Dokumentation. Daher soll nun nachfolgend versucht werden den Inhalt, soweit möglich, vorzustellen.

Datei 1 myhatsapplication.ear

```
META-INF/                    HatsService.jar
  ibmconfig/                 hodwel.jar
  application.xml            hostsim.jar
  was.policy                 hsrendering.jar
  ...                        ibmjlog.jar
bidibean.jar                 logs
myhatsapplication.war        product.xml
habeansnlv2.jar              runtime.properties
hasslite.jar                 runtime-debug.properties
hatscommon.jar               WFCommon.jar
hatsruntime.jar
```

Der Ordner *META-INF* beinhaltet die Datei *application.xml*, welcher den Deployment Deskriptor der Web-Anwendung darstellt. Die Datei *was.policy* enthält Sicherheitsregeln, mit denen für den Application Server feingranular die Zugriffsrechte der Web-Anwendung spezifiert werden. Im Ordner *ibmconfig* befinden sich zusätzliche WAS-spezifische Konfigurationsdateien.

Die Datei *product.xml* beinhaltet die zur Entwicklung verwendete Version von HATS. Die Datei *runtime.properties* fungiert als Konfigurationsdatei, in welcher Informationen über die HATS-Lizenz und Informationen über eine optionale Aufzeichnung von Log-Dateien angegeben werden. Die Datei *runtime-debug.properties* ist analog zur Datei *runtime.properties* aufgebaut, beschränkt ihre Gültigkeit jedoch nur auf die lokalen Testumgebung.

Alle JAR-Dateien in der EAR-Datei beherbergen Java Klassen, welche mit jeder HATS Web-Anwendung ausgeliefert werden. Zusammen ergeben die Klassen in der Größe knapp über 20 Megabyte und bilden das Fundament jeder HATS Web-Anwendung. Diese Klassen können an dieser Stelle nur erwähnt werden, da zu den Klassen weder eine öffentliche HATS-Dokumentation existiert, noch der Quelltext von IBM mit dem Produkt mit ausgeliefert wird. Der Entwickler

einer HATS-Anwendung hat keine weitergehenden Möglichkeiten dieses Fundament seinen Wünschen anzupassen, was im Normalfall jedoch auch nicht nötig ist.

Der selbst entwickelten Dateien der HATS Web-Anwendung liegen in der WAR-Datei *myhatsapplication.war*, welche jeweils den Namen des betrachteten HATS-Projekts enthält, in diesem Fall *myhatsapplication*. Der Inhalt der WAR-Datei wird nachfolgend aufgelistet.

Datei 2 myhatsapplication.war

```
common/                          hatsadmin/
  images/                        iojsp/
  stylesheets/                   macroHandlers/
  bidishape.js                   META-INF/
  dbcs.js                        templates/
  env.js                         transformations/
  HatsJS.js                      AdvancedIOErrorPage.jsp
  KBS.js                         BasicIOErrorPage.jsp
  lxgwfunctions.js               busy.jsp
  PrintERR.jsp                   checkTerm.jsp
  PrintJobs.jsp                  error.jsp
  TabbedFolder.js                HATSApplet.cab
  vif_jsf.js                     HATSApplet.jar
  vif_jsp_ad.js                  hiddenframe.jsp
  visualfield.js                 HPubConvertToTableFormat.xsl
  xhtmlbasic.js                  index.jsp
WEB-INF/                         login.jsp
  classes/                       login_error.jsp
  lib/                           LUName.jsp
  profiles/                      showURL.jsp
    connections/                 SignedJs.jar
    events/                      stop.jsp
    macros/                      struts
    ...                          workstationID.jsp
  tld/                           wsdl
  web.xml
  ...
```

Im Ordner *common* beherbergt HATS die von der Web-Anwendung benutzten CSS-Dateien und Grafiken. Ebenso wird ein Satz an JavaScript-Dateien in diesem Ordner mitgeliefert, welche die Web-Anwendung für manche Funktionalitäten benötigt. Die Datei *env.js* unterscheidet durch verschiedene JavaScript-Methoden verschiedene Webbrowser und speichert den aktuell verwendeten Browser in einer Variable. Dies ist nötig, da manche Webbrowser HTML-Elemente und JavaScript-Code unterschiedlich interpretieren. Die durch die Datei *env.js* realisierte Browserweiche [WIK3] wird beim Starten der HATS Web-Anwendung aufgerufen und kann die Web-Anwendung bei der Auswahl von webbrowserspezifischen Implementierungen einer Funktionalität unterstützen.

Von der Datei *HatsJS.js* werden verschiedenen Interaktionen mit dem Webbrowser, wie zum Beispiel ein Tastendruck oder ein Mausklick, abgefangen und zur Behandlung an die jeweils passende Funktion delegiert. Die zur Behandlung zuständigen Funktionen (Handler) sind in der Datei *KBS.js* implementiert. Die Datei *lxgwfunctions.js* ist mit knapp 4000 Zeilen JavaScript-Code die Umfangreichste. Sie beinhaltet Code für die Funktionalität von manchen Widgets, wie zum Beispiel für das in Abschnitt 3.3.2.2 erwähnte Calender-Widget.

Der optionale Ordner *hatsadmin* beinhaltet die HATS-Administationskonsole, die in Absatz 3.8 erklärt wurde. Die Ordner *iojsp*, *macroHandlers*, *templates* und *transformations* beinhalten die JSPs, welche vom Entwickler in der HATS-Entwicklungsumgebung angelegt wurden. Im Ordner *iojsp* befinden sich JSPs, welche bei Bedarf die Ein- und Ausgabe für Integration Objects regeln. Das Verzeichnis *macroHandler* beinhaltet, falls vorhanden, JSPs welche vom Makro extrahierte Informationen dem Benutzer anzeigen. Der Ordner *templates* beinhaltet das von der Web-Anwendung benutzte Template und der Ordner *transformations* die vom Entwickler in Folge einer Screen-Anpassung angelegten Transformationen, beides mal in Form von JSP-Dateien.

Der Ordner *META-INF* beinhaltet die Datei *MANIFEST.MF*, welche den Klassenpfad zu den sich in der EAR-Datei befindenden JAR-Archiven angibt. Im Ordner *WEB-INF* befindet sich der Deployment Deskriptor *web.xml*, welcher unter anderem die von der Web-Anwendung genutzten Servlets sowie auch Tag-Bibliotheken definiert. Im Unterordner *classes* befinden sich die kompilierten Java-Klassen von eventuell selbst erstellten Komponenten und Widgets, sowie auch von Integration Objects. Noch zusätzlich benötigte Hilfsbibiliotheken befinden sich in dem Unterordner *lib*. Der Unterordner *tld* enthält die von HATS verwendete Tag-Bibliothek. Im Unterordner *profiles* befinden sich in weiteren Unterordnern die in der HATS-Entwicklungsumgebung erstellten und im XML-Format gespeicherten Verbindungen, Screen-Anpassungen und Makros wieder.

Die JSPs *AdvancedIOErrorPage.jsp*, *BasicIOErrorPage.jsp* und *error.jsp* sind für die Darstellung von eventuell auftretenden HATS-Fehlermeldungen zuständig. Die JSP *busy.jsp* informiert den Benutzer, falls die HATS Web-Anwendung momentan noch beschäftigt ist und bietet ihm an die Seite über eine Schaltfläche erneut nachzuladen. Die Datei *checkTerm.jsp* beinhaltet die JavaScript-Methode *disconnectIfParentClosed()*, welche sich vermutlich über das asynchrone Updates Applet um das Beenden einer noch bestehenden Verbindung kümmert, falls der Webbrowser geschlossen wurde. Bei jedem Start der Web-Anwendung wird die Datei *index.jsp* aufgerufen, welche das von HATS vorgefertigte Servlet *entry* aufruft.

Falls als Browser der Internet Explorer genutzt wird, wird zusätzlich über ein HTML Frameset die JSP *checkTerm.jsp* im Hintergrund geladen, wodurch die von dieser JSP bereits erwähnte, bereitgestellte Funktionalität aktiviert wird. Die Dateien *login.jsp* und *login_ error.jsp* dienen dazu, Benutzer über eine Benutzerkennung und Passwort zu authentifizieren, falls die HATS Web-Anwendung entsprechend geschützt wurde. Die JSP *stop.jsp* wird aufgerufen falls die HATS Web-Anwendung beendet wird. Eine Meldung und eine Schaltfläche zum erneuten Verbinden werden angezeigt.

Über die Aufgabe des erwähnten, von HATS vorgefertigten Servlets *entry* findet man in der Dokumentation keine Hinweise. Da das Servlet zu Beginn der HATS Web-Anwendung aufgerufen wird, übernimmt es vermutlich die Aufgabe des Controllers im MVC-Prinzip und steuert den kompletten Ablauf der Web-Anwendung. Über die Funktion der restlichen, nicht erwähnten Dateien in diesem Verzeichnis kann ebenfalls nur spekuliert werden.

3.10. Entwicklung von Komponenten und Widgets

HATS bietet wie bereits erwähnt die Möglichkeit an, neue Komponenten und Widgets zu erstellen oder bestehende zu erweitern. Um eigene Komponenten beziehungsweise Widgets zu entwickeln, muss dem Entwickler hierfür bekannt sein, wie die HATS-Anwendung intern ein *<HATS:Component>*-Element in einer JavaServer Page behandelt.

Zuerst wird eine zum *<HATS:Component>*-Element über das Attribut *type* zugeordnete Komponente als Objekt instanziiert. Die zugehörigen Klassen befinden sich im Java-Package *com.ibm.hats.transform.components*. Falls die HATS-Anwendung in dem Package kein dem *type*-Attribut zugeordnetes Objekt findet, wird das Attribut als qualifizierende Pfadangabe für die Klasse angesehen. Aus dieser Klasse wird dann versucht das gewünschte Objekt zu instanziieren. Anschließend wird die *recognize()*-Methode des Komponenten-Objekts aufgerufen, welche nach festgelegten Kriterien Bereiche des Screens erkennt und diese in Form von *ComponentElement*-Objekten in einem Array zurückliefert. Anschließend wird auf jedem *ComponentElement*-Objekt des Arrays die Methode *doTextReplacement()* aufgerufen, um die vom Entwickler in den Komponenteneinstellungen über reguläre Ausdrücke definierten Zeichenkettenersetzungen vorzunehmen.

Ähnlich wie bei der Komponente wird nun über das *widget*-Attribut ein zugeordnetes Widget als Objekt instanziiert. Zusätzlich wird dem Objekt das Array mit den *ComponentElement*-Objekten übergeben. Anschließend wird auf dem Widget-Objekt die Methode *drawHTML()* aufgerufen, welche die einzelnen *ComponentElement*-Objekte über HTML-Elemente darstellt und diese in Form eines Java *StringBuffers* zurückgibt. Der Inhalt dieses *StringBuffers* wird anschließend in der JavaServer Page anstelle des *<HATS:Component>*-Elements ausgegeben.

Um eigene Komponenten oder Widgets anzulegen, bietet die Entwicklungsumgebung die Möglichkeit, die Grundstruktur der entsprechenden Java-Klassen automatisch zu erstellen. Dies geschieht über den Menüpunkt „HATS/New/Component“ beziehungsweise „HATS/New/Widget“.

3.10.1. Struktur einer Komponente

Für eine eigene Komponente wird die in Listing 3.3 gezeigte Grundstruktur von der HATS-Entwicklungsumgebung automatisch erstellt.

```
1  public class MyComponent {
2
3    public MyComponent(HsrScreen arg0) {
4      super(arg0);
5    }
6
7    public ComponentElement[] recognize(BlockScreenRegion region,
8                                        Properties settings)
9    {
10     return null;
11   }
12
13   public int getPropertyPageCount() {
14     return (1);
15   }
16
17   public Vector getCustomProperties(int iPageNumber,
18                                     Properties properties,
19                                     ResourceBundle bundle)
20   {
21     return null;
22   }
23
24   public Properties getDefaultValues(int iPageNumber) {
25     return (super.getDefaultValues(iPageNumber));
26   }
27
28 }
```

Listing 3.3: Struktur einer eigenen Komponente

Im Folgenden wird die Struktur erläutert. Wie bereits im vorherigen Abschnitt erklärt, dient die Methode *recognize()* in Zeile 7 dazu, die in dem mit dem Objekt *BlockScreenRegion* definierten Screen-Bereich erkannten Komponenten in Form von *ComponentObjects* in einem Array zurückzugeben. Über das Objekt *Properties* kann dabei auf in der HATS-Entwicklungsumgebung angegebene Komponenten-Einstellungen zurückgegriffen werden.

Um die neue Komponente über die HATS-Entwicklungsumgebung über Einstellungen konfigurierbar zu machen, existiert die Methode *getCustomProperties()* in Zeile 17. In ihr können Objekte vom Typ *HCustomProperty* angelegt werden, welche später in den Komponenten-Einstellungen als Eingabefelder dargestellt werden. Alle die Komponenten-Einstellungen betreffenden Objekte

müssen in dieser Methode in einem Java *Vector* zurückgegeben werden.

Die Methode *getDefaultValues()* in Zeile 24 liefert die für die Komponenten-Einstellungen definierten Standardwerte und die Methode *getPropertyPageCount()* in Zeile 13 die Anzahl der Seiten, über die sich die Komponenten-Einstellung erstreckt, zurück. Laut der HATS Web API Referenz 7.0 (Abschnitt 3.10.3) ist momentan jedoch maximal eine Seite erlaubt, wodurch diese Methode wahrscheinlich erst in späteren Versionen nützlich sein wird.

3.10.2. Struktur eines Widgets

Für ein eigenes Widget wird die in Listing 3.4 gezeigte Grundstruktur von der HATS-Entwicklungsumgebung automatisch erstellt.

```
1  public class MyWidget extends Widget implements HTMLRenderer {
2
3    public MyWidget(ComponentElement[] arg0, Properties arg1) {
4      super(arg0, arg1);
5    }
6
7    public StringBuffer drawHTML() {
8      StringBuffer buffer = new StringBuffer();
9      return (buffer);
10   }
11
12 }
```

Listing 3.4: Struktur eines eigenen Widgets

Der von der Komponente erstellte Array *ComponentElement[]* und die in der HATS-Entwicklungsumgebung definierten Widget-Einstellungen im Objekt *Properties*, werden in der dritten Zeile im Konstruktoraufruf dem Widget übergeben.

In der Methode *drawHTML()* in Zeile 7 können die *Component*-Objekte ausgelesen und über HTML-Elemente dargestellt, in den *StringBuffer* geschrieben werden, welcher von der Methode zurückgegeben wird. Um *Component*-Objekte als HTML-Elemente darzustellen, kann man mit den Klassenmethoden der *Component*-Objekte die jeweiligen Eigenschaften des Objekts auslesen und diese, aufbereitet durch HTML-Elemente, dem StringBuffer übergeben. Alternativ bietet die HATS-API das Objekt *HTMLElementFactory* an, welches *Component*-Objekte automatisch über vorgefertigte CSS-Dateien und JavaScript-Code darstellt. Der Inhalt dieses StringBuffers wird anschließend in einer Transformation anstelle des <*HATS:Component*>-Elements ausgegeben.

3.10.3. HATS-API

Für die Entwicklung eigener Komponenten und Widgets ist es empfehlenswert sich die HATS-API Referenz anzuschauen. Die HATS-API Referenz für Web-Anwendungen umfasst über 200 Java-Klassen, wobei die für den Einstieg wichtigen Klassen in den vorherigen Abschnitten erwähnt wurden. Weitere Klassen einzeln zu erwähnen und zu erklären würde den Umfang dieser Diplomarbeit sprengen. Aus diesem Grund wird hier lediglich auf die umfassende HATS-API Referenz im HATS Information Center [HATS7] verwiesen.

4. Einrichtung des Websphere Application Servers

4.1. Virtuelle Maschine mit z/Linux

Eine HATS Web-Anwendung benötigt als Laufzeitumgebung einen Websphere Application Server. Für die Installation des WAS wurde auf dem System z Mainframe der Universität Tübingen, die virtuelle Maschine *galadriel.cs.uni-tuebingen.de* unter z/VM eingerichtet und darauf das z/Linux Betriebssystem SUSE Linux Enterprise Server (SLES) 10 mit Service Pack 2 installiert. Auf dieser virtuellen Maschine mit z/Linux wurden alle weiteren Installationen und Konfigurationen durchgeführt.

4.2. Installation des WAS

Für diese Diplomarbeit wurde der Websphere Application Server Network Deployment in der Version 6.1 installiert. Die dabei verwendeten Installationsdateien befinden sich auf der DVD1 im Anhang. Da der WAS das Betriebssystem SLES 10 ohne Weiteres nicht unterstützt [WAS3], musste eine kleine Modifikation an den Installationsquellen vorgenommen werden [WAS4]. Die Datei */WAS/was.primary.pak/maintenance.xml* im WAS-Installationsverzeichnis wurde dabei durch die *maintenance.xml* Datei, welche sich ebenfalls im Anhang dieser Diplomarbeit auf DVD1 befindet, ersetzt, wodurch der WAS das Betriebssystem SLES 10 akzeptiert und sich problemlos installieren ließ.

Da für die Installation keine graphische Oberfläche, sondern nur eine entfernte Konsole über SSH verfügbar war, musste der WAS im sogenannten Silent-Modus [WAS5] installiert werden, bei welchem eine Rückmeldung nur in Form von Log-Dateien geschieht. Um die Installationsroutine mit den für die Installation nötigen Informationen zu versorgen, wurde eine Response-Datei *myresponsefile.txt* auf Basis der mit WAS augelieferten Datei *WAS/responsefile.nd.txt* angelegt. Diese Datei ist in Listing 4.1 abgebildet. In ihr befinden sich die Lizenzzustimmung, das Zielverzeichnis für die Installation, der Hostname und weitere Konfigurationsparameter.

```
-OPT silentInstallLicenseAcceptance="true"
-OPT installType="installNew"
-OPT feature="noFeature"
-OPT installLocation="/opt/IBM/WebSphere/AppServer"
-OPT profileType="standAlone"
-OPT PROF_enableAdminSecurity="false"
-OPT PROF_hostName=galadriel.cs.informatik.uni-tuebingen.de
```

Listing 4.1: WAS myresponsefile.txt

Um die Installation im Silent-Modus mit der erstellten Response-Datei zu starten, wird das folgende Kommando aus Listing 4.2 im WAS-Installationsverzeichnis abgesetzt.

```
WAS/install -silent -options WAS/myresponsefile.txt
```

Listing 4.2: WAS installieren

Nach der Installation bekommt der Benutzer keine Erfolgs- oder Fehlermeldung. Die Installationsroutine des WAS schreibt alle Meldungen in Log-Dateien, die nach der Installation überprüft werden müssen. Bei einer erfolgreichen Installation kann der WAS über die Kommandos in Listing 4.3 gestartet beziehungsweise gestoppt werden.

```
/opt/IBM/WebSphere/AppServer/bin/startServer.sh -server1
/opt/IBM/WebSphere/AppServer/bin/stopServer.sh -server1
```

Listing 4.3: WAS starten und stoppen

4.3. Autostart einrichten

Nach der Installation muss der Websphere Application Server standardmäßig nach jedem Neustart des Systems manuell über die Konsole mit den Kommandos aus Listing 4.3 neu gestartet werden. Um den WAS automatisch bei Systemstart zu laden, sind nachfolgende Schritte nötig.

Zuerst muss ein Shell-Script für den automatischen Start generiert werden. Dies erledigt der WAS mit dem Kommando aus nachfolgendem Listing 4.4, welches vom Benutzer *root* ausgeführt werden muss. Dadurch wird die Datei *start_ server.sh*, welche im Anhang in Listing A.7 abgebildet ist, im aktuellen Verzeichnis erstellt.

```
1  /opt/IBM/WebSphere/AppServer/bin/startServer.sh -server1
2                                                  -script
3                                                  -background
```

Listing 4.4: WAS Shell-Script für den automatischen Start erstellen

Das erstellte Shell-Script muss anschließend einem geeigneten Run-Level des Betriebssystems zugeordnet werden. Ein Run-Level ist ein Zustand, in welchem jeweils verschiedene Systemdienste gestartet, beziehungsweise gestoppt werden. Beim Starten und Herunterfahren des Betriebssystems durchläuft das System mehrere Run-Level. Für unseren Fall wurde der Run-Level 3 gewählt, in welchem standardmäßig Netzwerkdienste gestartet werden. Um das Script diesem Run-Level zuzuordnen, wird es in das Verzeichnis */etc/ init.d* verschoben. Anschließend werden im Verzeichnis */etc/init.d/rc3.d/* die symbolischen Links *S[XX]was* und *K[YY]was* erstellt, welche auf das Script verweisen. *S[XX]was* wird in diesem Fall vom Betriebssystem zum Starten und *K[YY]was* zum Beenden des Dienstes benutzt. Für jedes vom System auszuführende Script existiert ein solcher symbolischer Link. Die Variablen XX und YY sind dabei ausschlaggebend für die Reihenfolge der Ausführung. Beim Systemstart werden Scripte anhand dieser Werte nacheinander in aufsteigender Reihenfolge ausgeführt. Bei der Wahl von XX und YY ist zu beachten, dass *S[XX]was* erst ausgeführt werden darf, wenn der Netzwerkdienst bereits gestartet wurde. *K[XX]was* muss hingegen bereits ausgeführt werden, bevor der Netzwerkdienst gestoppt wird. In dieser Arbeit wurde für XX der Wert 13 und für YY der Wert 15 gewählt.

Abschließend wird mit der Kommandosequenz in Listing 4.5 das Shell-Script als Systemdienst eingerichtet.

```
1  chkconfig --add was
2  chkconfig --set was 3
```

Listing 4.5: WAS Systemdienst einrichten

4.4. HATS Web-Anwendung installieren

Um eine HATS Web-Anwendung auf dem WAS zu installieren, muss das Projekt im EAR-Format aus der HATS-Entwicklungsumgebung exportiert werden. Anschließend kann die Web-Anwendung in der WAS-Verwaltungsoberfläche Integrated Solutions Console (Abbildung 4.1) mit dem Menüpunkt „Anwendungen/Neue Anwendung installieren“ installiert werden. Über die URL `https://134.2.205.55:9043/ibm/console/` erreicht man die Verwaltungsoberfläche des installierten WAS. Die Installation erstreckt sich über mehrere Dialoge mit Abfragen zur Konfiguration. Zur Installation einer HATS Web-Anwendung kann jedes Mal die Standardeinstellung verwendet werden

Nach der erfolgreichen Installation befindet sich im Menüpunkt „Anwendungen/Enterprise Anwendungen“ eine Übersicht über alle installierten Web-Anwendungen. Das Feld „Anwendungsstatus“ zeigt über ein Symbol an, ob die Anwendung gestartet oder gestoppt ist. Standardmäßig ist das Symbol nach der Installation rot, was bedeutet dass die Anwendung gestoppt ist. Um die Anwendung zu starten muss die Web-Anwendung manuell ausgewählt und über die Schaltfläche „Starten“ gestartet werden. Wechselt das Symbol seine Farbe zu grün, verlief das Starten erfolgreich und die Web-Anwendung kann nun über einen Webbrowser aufgerufen werden.

Abbildung 4.1.: Integrated Solutions Console

4.5. Sicherheit konfigurieren

Standardmäßig werden mit der vorgenommenen Installation des WAS kaum Sicherheitsvorkehrungen getroffen. Die WAS-Verwaltungsoberfläche Integrated Solutions Console kann ohne Authentifizierung von jedem verwendet werden, was ein erhebliches Sicherheitsrisiko darstellt. Um eine Authentifizierung zu aktivieren muss in der Verwaltungsoberfläche über den Menüpunkt „Sicherheit/Sichere Verwaltung, Anwendungen und Infrastruktur“ die Verwaltungssicherheit mit der Option „Verwaltungssicherheit aktivieren“ ausgewählt werden. Über das Auswahlfeld „Verfügbare Realm-Definitionen“ kann zusätzlich festgelegt werden, welche Benutzerkonten Zugang erhalten. Über die Auswahl „Lokales Betriebssystem“ wird definiert, dass die Verwaltungsoberfläche nur noch über Benutzerkonten des z/Linux-Betriebssystems zugänglich ist.

Um den Zugriff auf die in Abschnitt 3.8 erwähnte HATS-Adminisationskonsole ebenfalls einzuschränken, muss in der Verwaltungsoberfläche unter dem Menüpunkt „Sicherheit/Sichere Verwaltung, Anwendungen und Infrastruktur“ die Option „Anwendungssicherheit aktivieren“ ausgewählt werden. Um nun lediglich den Benutzeraccounts des Betriebssystems den Zugriff zu ermöglichen, wird unter dem Menüpunkt „Anwendungen/Enterprise Anwendungen“ die installierte HATS Web-Anwendung ausgewählt und anschließend über den Menüpunkt „Zuordnung

von Sicherheitsaufgabenbereichen zu Benutzern/Gruppen“ für die Sicherheitsrollen „HATSAdministrator“, „HATSMonitor“ und „HATSOperator“ jeweils die Option „Alle Authentifizierten?“ aktiviert (vgl. Abbildung 4.2).

Abbildung 4.2.: Zuordnung von Sicherheitsrollen in der Integrated Solutions Console

Bei der Installation legt der Websphere Application Server standardmäßig Secure Sockets Layer (SSL) Konfigurationen mit einem selbst erstellten Zertifikat an. In der Verwaltungsoberfläche kann die Konfiguration unter dem Menüpunkt „Sicherheit/Verwaltung von SSL-Zertifikaten und Schlüsseln/Sicherheitskonfigurationen für Endpoint verwalten“ kontrolliert und gegebenenfalls bearbeitet werden. Der Port-Name *WC_defaulthost_secure* repräsentiert dabei den SSL-Zugriff auf die Web-Anwendungen. Mit den Standardeinstellungen kann eine HATS Web-Anwendung nach der Installation über den Port 9443 geschützt durch SSL aufgerufen werden. Dadurch wird die Kommunikation zwischen den Clients und dem WAS mit einem ausgehandelten symmetrischen Verschlüsselungsverfahren geschützt, wodurch in einer HATS Web-Anwendung zum Beispiel die Passwörter von TSO- und CICS-Accounts bei einer Anmeldung nicht mehr im Klartext über das Internet übertragen werden.

Für weitere Informationen über Sicherheitseinstellungen des Websphere Application Servers sei auf [RED2] verwiesen.

5. Erstellung der Web-Oberfläche

Im Client/Server-Praktikum an der Universität Tübingen haben Studenten die Möglichkeit, erste Erfahrungen im Umgang mit einem Mainframe zu sammeln. Neben Aufgaben über CORBA und Java RMI existieren Aufgaben, welche Grundlagen von ISPF vermitteln oder das Erstellen einer einfachen CICS-Anwendung zum Ziele haben. 3270-Terminalemulatoren mit ihren Green Screens, welche sich nur mit der Tastatur bedienen lassen, schrecken unerfahrene Anwender jedoch auf den ersten Blick ab. Um daher den Studierenden den Einstieg zu erleichtern, wurde die Oberfläche für die benötigten Host-Anwendungen im Rahmen dieser Diplomarbeit mit einer HATS Web-Anwendung modernisiert. Studierende können nun auf die Host-Anwendungen über einen Webbrowser und einer ihnen vertrauten Oberfläche zugreifen.

Folgende Praktikumsaufgaben können mit der erstellten HATS Web-Anwendung durchgeführt werden:

- Tutorial 1a: Dateiverwaltung unter TSO und ISPF
- Tutorial 1b: Benutzung von ISPF
- Tutorial 2a: Erstellung eines C-Programms
- Tutorial 2b: Erstellung eines COBOL-Programms
- Tutorial 3: CICS
- Tutorial 4: DB2
- Tutorial 6: REXX

Im Folgenden wird nun die Erstellung der HATS Web-Anwendung dokumentiert.

5.1. HATS Web-Anwendung

HATS bietet, wie in Kapitel 3.3.5 erklärt, die Möglichkeit, eine Standardtransformation zu definieren. Dabei wird über verschiedene Regeln festgelegt, welche Bereiche über welche Komponenten und Widgets dargestellt werden. Dies bietet sich effizient an, wenn alle Screens einer Host-Anwendung nach dem selben Muster aufgebaut und einheitlich strukturiert sind.

Da in unserem Fall die Host-Anwendungen TSO, ISPF, CICS, USS und DB2 aus sehr vielen Screens bestehen, welche keine analoge Struktur zeigen, bietet sich die Standardtransformation allein nicht an. Folglich war es nötig für verschiedene Bereiche der Host-Anwendung verschiedene Screen-Anpassungen zu definieren. Im Rahmen dieser Diplomarbeit sind dadurch 53 Screen-Anpassungen, 44 Transformation durch JavaServer Pages, 14 Makros, 6 eigene Widgets, zwei eigene Komponenten, ein Integration Object und ein Template entstanden, über die im Folgenden ein kurzer Überblick gegeben wird.

Die erstellte HATS Web-Anwendung ist über nachfolgende URLs aufrufbar:
`http://galadriel.cs.uni-tuebingen.de:9080/csprak/`
`https://galadriel.cs.uni-tuebingen.de:9443/csprak/`

Innerhalb der beigefügten virtuellen Maschine befindet sich der Quelltext der HATS Web-Anwendung in der Entwicklungsumgebung im HATS-Projekt *csprak*. Durch einen Rechtsklick auf das Projekt und anschließender Auswahl des Menüpunktes „Debug on Server“ im aufklappenden Menü, kann die HATS Web-Anwendung innerhalb der virtuellen Maschine getestet werden. Im erscheinenden Dialog muss als Server „Websphere Application Server v6.1“ ausgewählt werden. Mit der Schaltfläche „Finish“ wird die Auswahl bestätigt und die HATS Web-Anwendung gestartet.

5.1.1. Template

Alle nachfolgend erwähnten Transformationen werden innerhalb einer HTML-Tabelle in das erstellte Template *Web Content/Templates/unituebingen.jsp* eingebettet. Das Template besteht aus einer einfachen Hintergrundgrafik und einem Logo der Universität Tübingen. Als CSS wurde die Datei *Web Content/common/stylesheets/uni-tuebingen.css* eingebunden, welche unter anderem die von der HATS Web-Anwendung verwendeten Schriftgrößen, Schriftarten, Abstände und Farben definiert. Das CSS definiert die Schriftart Verdana mit einer Schrifthöhe von 12 Pixeln. Als Hintergrund für Inhalte wurde ein helles Gelb (#ffffc4) definiert. Das Template ist auf später gezeigten Abbildungen von erstellten Transformationen zu sehen.

5.1.2. z/OS Willkommen-Screen Umsetzung

Der z/OS Willkommen-Screen ist in Abbildung 5.2 dargestellt und begrüßt den Benutzer bei jeder neuen Verbindung mit dem Host. Über ASCII-Zeichen wird dem Benutzer die Grafik *z/OS* angezeigt. Ebenso erfährt er, wie er sich bei den Host-Anwendungen TSO oder CISC anmelden kann. Da jede Zeile dieses Screens durch die Standardtransformation als HTML-Eingabefeld dargestellt werden würde, wurde für diesen Screen die Screen-Anpassung *WelcomeScreen/WelcomeScreen.evnt* erstellt. Als Erkennungskriterium wurde die Zeichenkette „z/OS“ an Position (1,1) gewählt.

Als Aktion stellt die Screen-Anpassungen den Screen über die erstellte Transformation *WelcomeScreen.jsp* dar. Die Transformation ist in Abbildung 5.3 zu sehen. Sie ersetzt die ASCII-Grafik durch die modernere GIF-Grafik *welcome.gif*. Damit die GIF-Grafik mit dem Hintergrundbild des Templates harmoniert, wurde die weiße Farbe der Grafik bei der Erstellung transparent gesetzt. Als Eingabefeld wird die Zeile 24 des Screens in der Transformation durch die Input-Field-Komponente mit dem Text-Input-Widget dargestellt. Um dem Benutzer die Navigation mit der Maus zu ermöglichen, wurden zwei Makros *LTSOButton.hma* und *LCICSButton.hma* erstellt, welche über die in die Transformation eingefügten Links („TSO Logon“ beziehungsweise „CICS Logon“) aktiviert werden. Die Makros übergeben an das Eingabefeld das zugehörige Kommando „L TSO“ bzw. „L CICS“ und emulieren nach einer kurzen Zeitspanne in der Host-Anwendung ein Tastendruck auf die Eingabetaste.

Abbildung 5.1.: Makro zum Ausführen des Kommandos „L TSO“

Das Makro *LTSOButton.hma* ist in Abbildung 5.1 zu sehen. Wird das Makro durch einen Klick auf die Schaltfläche aktiviert, wird das Kommando „L TSO“ in das Eingabefeld an der aktuellen Cursorposition geschrieben. Anschließend wird in der Host-Anwendung vom Makro ein Tastendruck auf die Eingabetaste emuliert, was das Ausführen des Kommandos und somit ein Wechsel des Screens veranlasst. Das Makro wird beendet sobald der Screen für die TSO-Anmeldung erscheint. Das Makro *LCICSButton.hma* unterscheidet sich lediglich an dem auszuführenden Kommando, weshalb das Makro hier nicht separat abgebildet wird.

Auf dem Mainframe der Universität Tübingen existieren zwei nahezu identische z/OS-LPARs mit dem Namen Hobbit (134.2.205.54) und Frodo (134.2.14.211). Standardmäßig wird das Client/Server-Praktikum mit der LPAR Hobbit betrieben. Da es sich bei Hobbit und Frodo um identische LPARs handelt, kann die im Rahmen dieser Arbeit erstellte HATS Web-Anwendung für den Zugriff auf beide LPARs verwendet werden. Um die HATS Web-Anwendung für den Zugriff auf Frodo zu konfigurieren, muss in der HATS-Entwicklungsumgebung in den Projekteinstellungen im Abschnitt „Connections“ die IP-Adresse entsprechend geändert werden. Weitere Änderungen

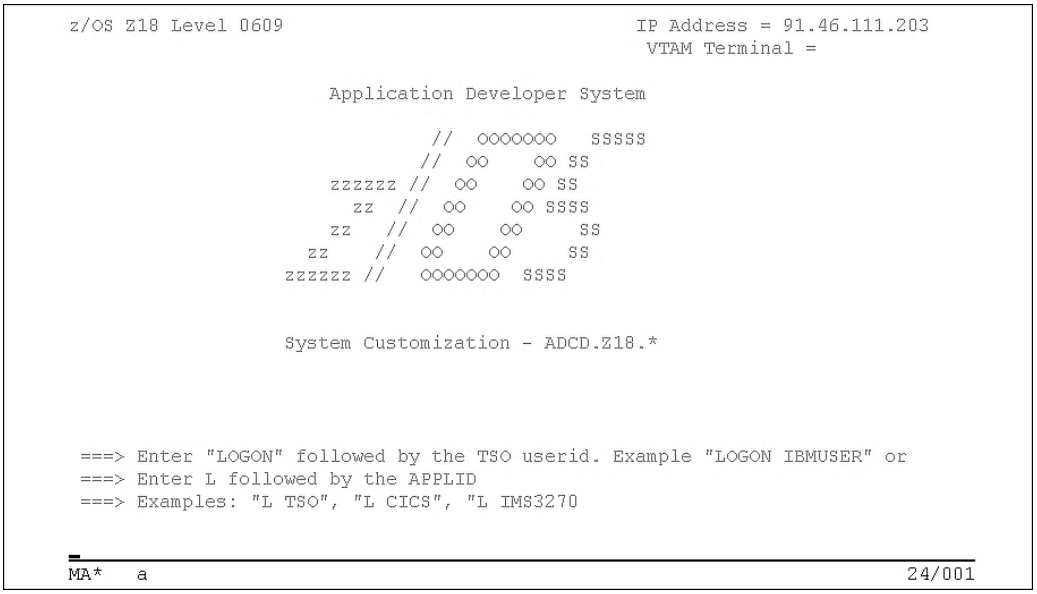

Abbildung 5.2.: z/OS Willkommen-Screen

Abbildung 5.3.: z/OS Willkommen-Screen in der HATS Web-Anwendung

in den Einstellungen werden nicht benötig. Um den Benutzer der HATS Web-Anwendung über die aktuelle LPAR zu informieren, wird auf dem Willkommen-Screen unter dem Eingabefeld der jeweilige Hostname der aktuellen LPAR ausgegeben. Um dies zu realisieren wurde der Codeabschnitt aus Listing 5.1 in die Transformation *WelcomeScreen.jsp* integriert.

```
1  TransformInfo tInfo = (TransformInfo)request.getAttribute(CommonConstants.
       REQ_TRANSFORMINFO);
2  String hostname = "";
3
4  if (tInfo != null) {
5    HostScreen hScreen = tInfo.getHostScreen();
6    hostname = hScreen.getHost();
7  }
8
9  if (hostname.equals("134.2.14.211"))
10   hostname = "frodo.cs.uni-tuebingen.de";
11 else if (hostname.equals("134.2.205.54"))
12   hostname = "hobbit.cs.uni-tuebingen.de";
```

Listing 5.1: Anzeigen des Hostnamens

Über das Objekt vom Typ *TransformInfo* wird in der fünften Zeile über die Methode *getHostScreen()* der aktuelle Screen der Host-Anwendung als Objekt vom Typ *HostScreen* instanziiert. Dieses Objekt liefert in der sechsten Zeile mit der Methode *getHost()* die zur Host-Anwendung gehörende IP-Adresse als Zeichenkette. Durch eine Fallunterscheidung wird nun die IP-Adresse mit den IP-Adressen der LPARs Hobbit und Frodo verglichen und der zugehörige Hostname in der Variable *hostname* gesetzt. Diese Variable wird später in der Transformation in grauer Farbe unter dem Eingabefeld ausgegeben.

Die Erstellung dieser Screen-Anpassung wurde durch einen Bug, welcher das Anzeigen des zuvor im XML-Format gespeicherten z/OS Willkommen-Screens mit einer Java NullPointer-Ausnahme quittiert, in der hier benutzen Version der HATS-Entwicklungsumgebung erschwert. Dies liegt wahrscheinlich daran, dass die Entwicklungsumgebung beim Speichern des z/OS Willkommen-Screens das XML Attribut *penType* eines *field*-Elements auf einen ungültigen Wert setzt. Die XML-Datei mit dem ungültigen Wert ist in Abbildung 5.4 dargestellt.

Der Bug konnte durch manuelles Bearbeiten der XML-Datei behoben werden, indem das in der Abbildung markierte Attribut *penType* vollständig entfernt wurde. Dieser Bug wurde in der Version 7.5 von HATS behoben.

?=? xml	version="1.0" encoding="UTF-8"
⊟ e xml_app_data	
⊟ e screen	
a cursor	1841
e interaction	
⊞ e display	
⊟ e content	
⊞ e field	
⊞ e field	
⊟ e field	
a colorarray	2
a dbcsarray	0
a extarray	0
a fieldarray	0
a foreground	4
a gridarray	0
a highintensity	true
a index	2
a length	0
a penType	&#0;
a position	1920
a selectable	true
⊞ e plane	
e enhancedinputattributes	
⊞ e session	

Abbildung 5.4.: WelcomeScreen.hsc

5.1.3. TSO Umsetzung

Alle nachfolgend erwähnten TSO Screen-Anpassungen befinden sich innerhalb des HATS-Projekts im Verzeichnis *Screen Customizations/TSO*.

5.1.3.1. TSO-Anmeldung

Zur Anmeldung von TSO gelangt man wie bereits erwähnt aus dem z/OS Willkommen-Screen durch das Kommando „L TSO“. Die TSO-Anmeldung durchläuft mehrere Screens. Zuerst wird der Benutzer über einen Screen gebeten, seine Benutzerkennung einzugeben und diese mit der Eingabetaste zu bestätigen (vgl. Abbildung 5.5). Anschließend folgt ein Screen, in welchem der Benutzer gebeten wird, sein Passwort einzugeben und dieses ebenfalls mit der Eingabetaste zu bestätigen (vgl. Abbildung 5.6). Wird das Passwort akzeptiert, gelangt der Benutzer aufgrund der z/OS-Konfiguration des Tübinger Mainframes zur TSO-Shell (vgl. Abbildung 5.7).

Im Client/Server-Praktikum wird TSO hauptsächlich dazu benutzt, die Host-Anwendung ISPF zu starten. Dies geschieht durch die Eingabe des Kommandos „ISPF“ welches daraufhin den Screen in Abbildung 5.8 zum Symbolisieren eines Ladevorgangs anzeigt. Drückt der Benutzer anschließend die Eingabetaste, gelangt er in das ISPF Primary Option Menü, sieht zuvor aber noch den Copyright-Hinweis in Abbildung 5.9, welchen er ebenfalls mit der Eingabetaste bestätigen muss. Somit erstreckt sich die Anmeldung bei TSO und das Starten von ISPF über insgesamt fünf Screens.

Um dem Benutzer diese mühselige Anmeldung zu ersparen, wurden Screen-Anpassungen und Makros erstellt, welche den Vorgang in der HATS Web-Anwendung nach Eingabe von Benutzerkennung und Passwort automatisieren. Zuerst wurde dazu für den Screen in Abbildung 5.5 die Screen-Anpassung *EnterUserID.evnt* erstellt, welche aktiv wird, sobald die Zeichenkette „IKJ56700A ENTER USERID“ an beliebiger Stelle auf einem Screen erkannt wird. Diese Zeichenkette konnte als Erkennungskriterium nicht an eine feste Position gebunden werden, da der Benutzer bei dem z/OS Willkommen-Screen ungültige Kommandos absetzen kann, wodurch sich die erwähnte Zeichenkette zeilenweise verschiebt. Als Aktion wurde der Screen-Anpassung die erstellte Transformation *EnterUserID.jsp* zugewiesen.

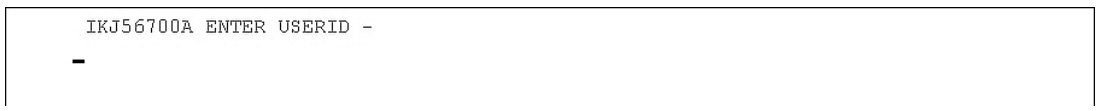

Abbildung 5.5.: Eingabe der Benutzerkennung bei der TSO-Anmeldung

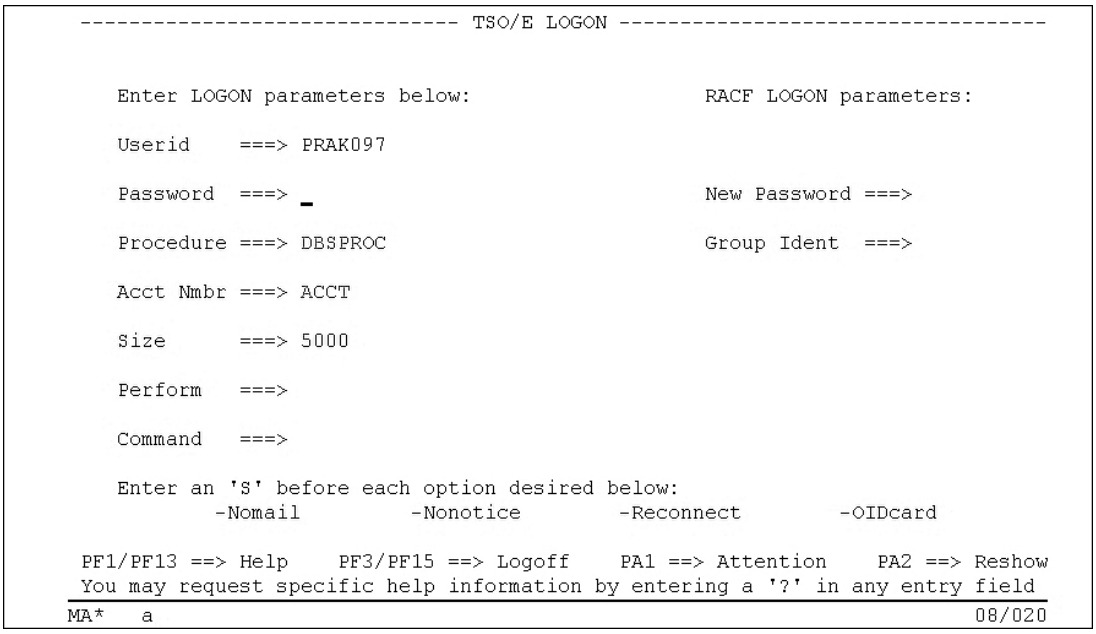

Abbildung 5.6.: Eingabe des Passworts bei der TSO-Anmeldung

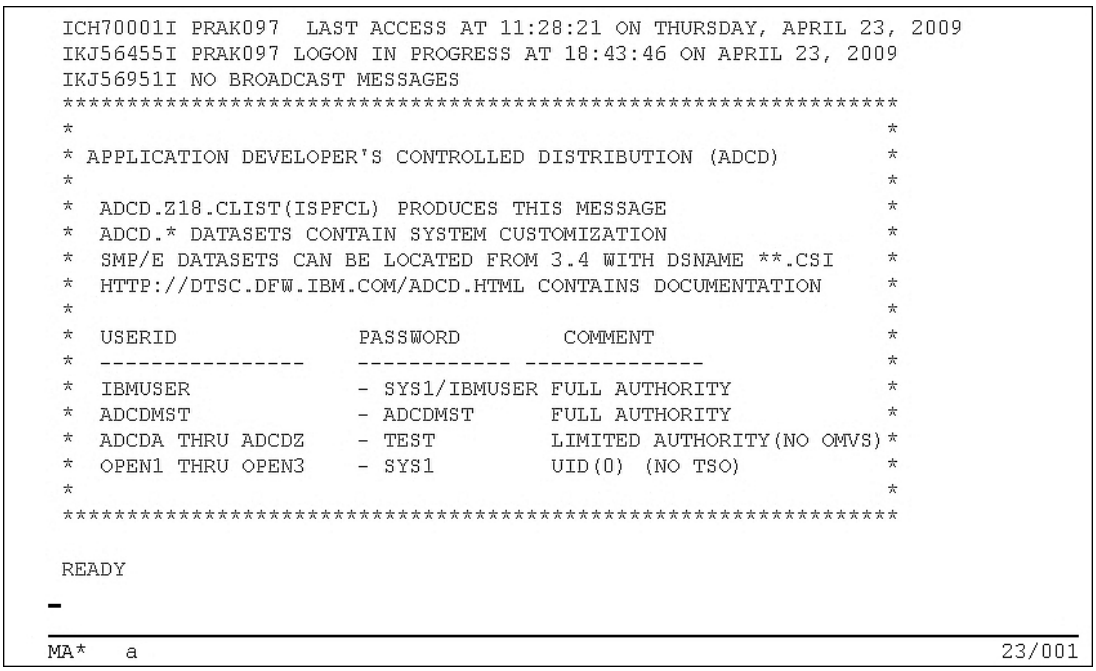

Abbildung 5.7.: TSO-Shell

Abbildung 5.8.: Ladevorgang von ISPF

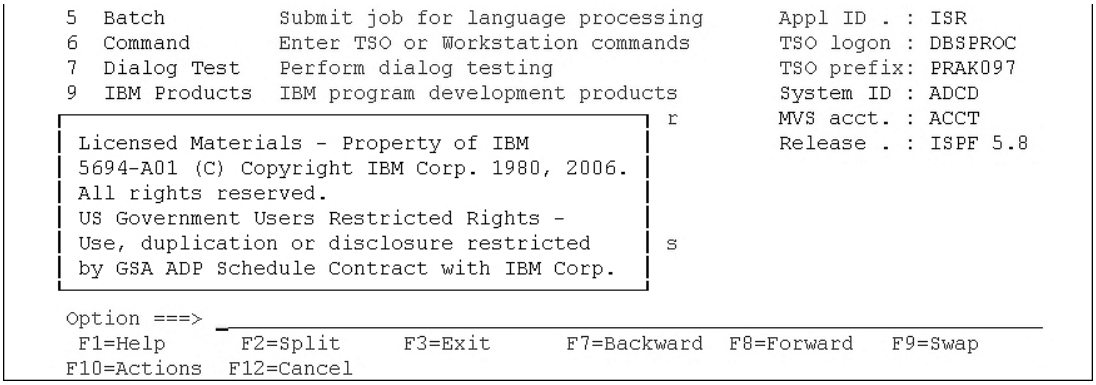

Abbildung 5.9.: Copyright-Hinweis beim Starten von ISPF

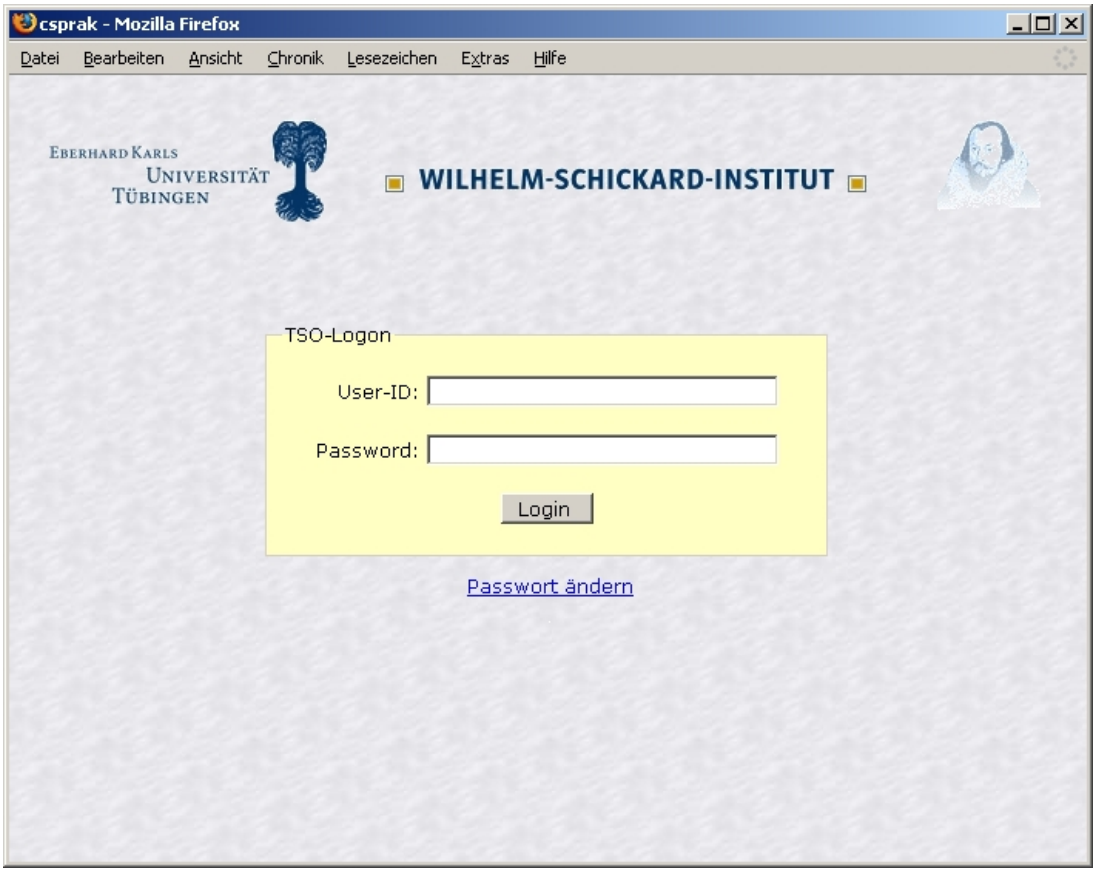

Abbildung 5.10.: TSO-Anmeldung in der HATS Web-Anwendung

Die Transformation ist in Abbildung 5.10 zu sehen. Das Eingabefeld für die Benutzerkennung wird durch die Input-Field-Komponente mit dem Text-Input-Widget transformiert. Das Eingabefeld für das Passwort existiert nicht auf dem Screen zur Eingabe der Benutzerkennung, weshalb eine Umsetzung dieses Eingabefeldes mit Komponenten und Widgets nicht möglich ist. Eingefügt wurde dieses Eingabefeld daher in der HATS-Entwicklungsumgebung über den Menüpunkt „HATS Tools“ mit der Aktion „Insert Global Variable“ und der aktivierten Option „Prompt for global variable with input box“. Hierbei wurde zusätzlich die Option „Mask as password field“ gewählt, um das transformierte Eingabefeld auf den Typ *password* zu setzen und somit die Eingabe zu verstecken. Das in dieses Eingabefeld eingegebene Passwort wird in der globalen Variable *password* gespeichert. Zusätzlich wurde eine Schaltfläche „Login“ eingefügt, welche ein Drücken der Eingabetaste emuliert und somit die eingebenen Daten an die Host-Anwendung sendet. Der Link „Passwort ändern“ verweist auf die zu einem Integration Object gehörende JSP *iojsp/ChangePasswordInput.jsp*. Auf dieses Integration Object wird in Abschnitt 5.1.3.2 genauer eingangen.

Nachdem der Benutzer in der HATS Web-Anwendung die Schaltfläche „Login“ betätigt hat, wechselt die Host-Anwendung im Hintergrund zu dem in Abbildung 5.6 gezeigten Screen. Dieser Screen wird von der Screen-Anpassung *Tso_ eLogon.evnt* durch die ersten fünf Zeilen erkannt. Hierbei wurden bewusst die ersten fünf Zeilen gewählt, da eventuell auftretende Fehlermeldungen in diesem Screen-Bereich angezeigt werden und in diesem Fall eine andere HATS-Aktion ausgeführt werden muss. Wird keine Fehlermeldung angezeigt, wird von der Screen-Anpassung als Aktion das Makro *LogonTSO.hma* ausgeführt. Das Makro ist in Abbildung 5.11 dargestellt und fügt in der Host-Anwendung durch die Makro-Aktion „Prompt“ an Position (20,8) die vorher gefüllte globale Variable *password* ein. Anschließend emuliert das Makro ein Tastendruck auf die Eingabetaste. Das Makro kann nun in drei unterschiedlichen Screens enden: Falls das Passwort nicht akzeptiert wird, da es entweder falsch eingegeben wurde bzw. abgelaufen ist, endet das Makro auf dem Screen *TSO Password Not Authorized* bzw. *TSO Password Expired*. Bei Eingabe eines gültigen Passworts terminiert das Makro auf dem Screen *TSO Shell*.

Falls das Passwort falsch eingegeben wurde, bekommt der Benutzer eine erneute Möglichkeit zur Eingabe des Passworts. Für diesen Fall wurde die Screen-Anpassung *Tso_ eLogon_ PasswordNotAuthorized.evnt* erstellt. Als Erkennungskriterium wurde definiert, dass der Text „PASSWORD NOT AUTHORIZED FOR USERID“ an Position (12,2) auf einem Screen erscheint. Als Aktion wird der Screen durch die Transformation *Tso_ eLogonNotAuthorized.jsp* angezeigt, welche es dem Benutzer ermöglicht das Passwort erneut einzugeben oder die Anmeldung über eine Schaltfläche „Exit“ abzubrechen. Das Eingabefeld für das Passwort wird in diesem Fall mit der Input-Field-Komponente über das Text-Input-Widget dargestellt. Ein Umweg über eine globale Variable wie in der vorherigen Screen-Anpassung ist nicht mehr nötig, da die Benutzereingabe in der Host-Anwendung nun komplett auf dem Screen in Abbildung 5.6 stattfindet. Für den Fall dass der Benutzer die Anmeldung abbrechen möchte, wurde die Schaltfläche „Exit“ eingefügt, welche in der Host-Anwendung den Druck der Funktionstaste PF3 emuliert.

Falls das Passwort abgelaufen ist, ist es erforderlich ein neues Passwort zu setzen. Dazu wurde die Screen-Anpassung *Tso_ eLogon_ PasswordExpired.evnt* erstellt, welche aktiv wird sobald das Erkennungskriterium „*New Password“ beginnend an Position (52,8) auf einem Screen erscheint.

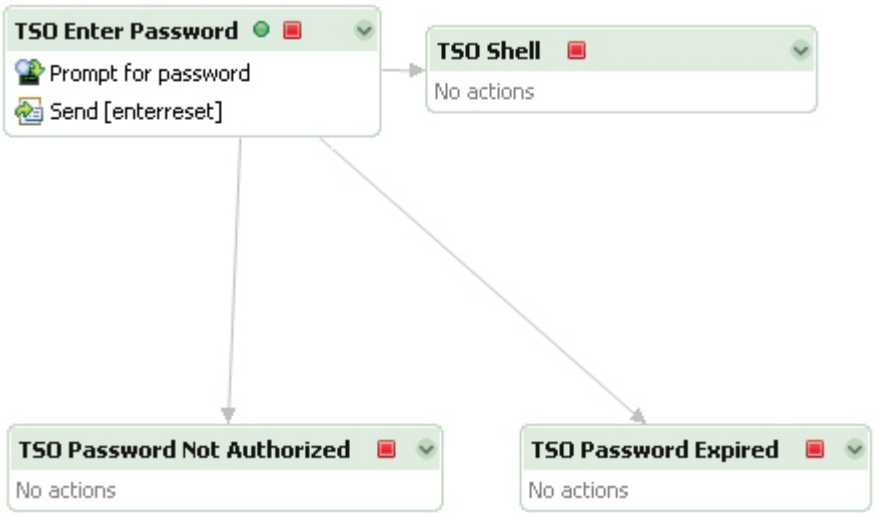

Abbildung 5.11.: Makro zum Einfügen des TSO-Passworts

Als Aktion wird der Screen über die erstellte Transformation *Tso_eLogon_PasswordExpired.jsp* angezeigt, welche es dem Benutzer mittels einem Eingabefeld erlaubt, ein neues Passwort zu setzen. Zur Darstellung von dem Eingabefeld wurde wieder die Input-Field-Komponente mit dem Text-Input-Widget verwendet. Die TSO-Meldung, die den Benutzer darüber informiert dass das Passwort abgelaufen ist und neu eingegeben werden muss, wird an Position (2,2) aus dem Screen extrahiert und dem Benutzer über das Label-Widget unter dem Eingabefeld in roter Farbe angezeigt. Da TSO eine zweimalige Eingabe des Passworts verlangt, wird nach der ersten Eingabe die selbe Transformation mit einer angepassten TSO-Meldung erneut angezeigt. Dies ist auch der Grund dafür, warum die TSO-Meldung nicht als Erkennungskriterium gewählt wurde. Ansonsten hätte mit optionalen Erkennungskriterien oder mit einer zusätzlichen Screen-Anpassung gearbeitet werden müssen.

Für den Fall dass der Benutzer eine ungültige Benutzerkennung eingegeben hat, wurde die Screen-Anpassung *Tso_eLogon_UserIDNotAuthorized.evnt* angelegt. Aktiviert wird sie, wenn die Zeichenkette „IKJ56420I Userid“ auf einem Screen erkannt wird. Auch hier führt die Screen-Anpassung eine Transformation aus. Die Transformation *Tso_eLogon_UserIDNotAuthorized.jsp* verwendet die Input-Field-Komponente mit dem Text-Input-Widget um Eingabefelder für eine erneute Eingabe der Benutzerkennung und des Passworts darzustellen.

Ebenso muss der Fall berücksichtigt werden, dass Benutzer versuchen sich mit einer bereits angemeldeten Benutzerkennung anzumelden. Dieser Fall kann auftreten falls mehrere Personen denselben Account verwenden oder ein Benutzer sich nicht ordnungsgemäß abgemeldet hat und ein automatisches Abmelden des Benutzers vom System oder durch Administratoren abgewartet werden muss. Tritt dieser Fall ein, wird dem Benutzer in den ersten zwei Zeilen des Screens die Meldung „IKJ56425I LOGON rejected, UserId PRAKxxx already logged on to sys-

tem ADCD, IKJ56400A ENTER LOGON OR LOGOFF“ angezeigt, was als qualifizierendes Erkennungskriterium für die erstellte Screen-Anpassung *Tso_ eLogon_ LogonRejected.evnt* gewählt wurde. Das Erkennungskriterium wurde dabei in zwei Teile aufgeteilt, da die Zeichenkette „PRAKxxx“ jeweils abhängig vom aktuellen Benutzer ist und somit nicht als Kriterium verwendet werden kann. In Folge der Screen-Anpassung wird der Screen über die Transformation *Tso_ eLogon_ LogonRejected.jsp* angezeigt. Diese gibt dem Benutzer die erwähnte TSO-Meldung über die Text-Komponente mit dem Text-Widget aus. Zusätzlich wurden in die Transformation zwei Schaltflächen „Logoff“ und „Retry“ eingefügt, welche jeweils ein Makro aktivieren. Das Makro *TSOLogonRejectedLogoff.hma* ist in Abbildung 5.12 dargestellt. Seine Funktion besteht darin, das Kommando „LOGOFF“ an der aktuellen Cursorposition in das Eingabefeld der Host-Anwendung einzutragen und die Eingabe mit der Eingabetaste zu bestätigen. Der darauf folgende Exit-Screen des Makros wird durch die Zeichenkette „****** “ an beliebiger Stelle auf dem Screen erkannt. Das Makro *TSOLogonRejectedLogon.hma* wird durch die Schaltfläche „Retry“ aktiviert und führt analog zum Makro *TSOLogonRejectedLogoff.hma* das Kommando „LOGON“ aus.

Abbildung 5.12.: Makro zum Ausführend des Kommandos „LOGOFF“

Somit wurden nun alle Sonderfälle, die bei der TSO-Anmeldung auftreten können, in der HATS Web-Anwendung berücksichtigt. Um den Benutzer bei korrekter Anmeldung von der TSO-Shell in das ISPF Primary Option Menü weiterzuleiten, wurden die Screen-Anpassungen *ADCD.evnt* und *IspfPrimaryOptionMenuSplash.evnt* mit den Makros *ForwardADCD.hma* und *ForwardISPFSplash.hma* erstellt. Die Screen-Anpassung *ADCD.evnt* wird aktiv, sobald der Screen aus Abbildung 5.7 erscheint. Erkannt wird der Screen durch die Zeichenkette „APPLICATION DEVELOPER'S CONTROLLED DISTRIBUTION (ADCD)“ in der sechsten Zeile des Screens. Als Aktion führt die Screen-Anpassung das Makro *ForwardADCD.hma* aus, welches das Starten von ISPF veranlasst. Die Screen-Anpassung *IspfPrimaryOptionMenuSplash.evnt* startet als Aktion das Makro *ForwardISPFSplash.hma*, welches die ISPF Copyright-Meldung in der Host-Anwendung mit der Eingabetaste bestätigt. Aktiv wird diese Screen-Anpassung, wenn der in Abbildung 5.9 dargestellte Screen erkannt wird. Als Erkennungskriterium wurde der Inhalt der Copyright-Meldung gewählt.

Das Makro *ForwardADCD.hma* ist in Abbildung 5.13 abgebildet. In der TSO-Shell der Host-Anwendung fügt das Makro an der aktuellen Cursorposition das Kommando „ISPF“ ein und bestätigt dies mit der Eingabetaste. Der ISPF-Ladevorgang wird nur sporadisch angezeigt, weshalb eine Fallunterscheidung im Makro nötig ist. Entweder der Ladevorgang wird auf einem Screen angezeigt und das Makro landet im Zustand *Loading ISPF* oder der Ladevorgang läuft so schnell ab, dass der Ladevorgang von der Host-Anwendung nicht angezeigt wird. Falls der Ladevorgang auf einem Screen angezeigt wird, emuliert das Makro in der Host-Anwendung ein

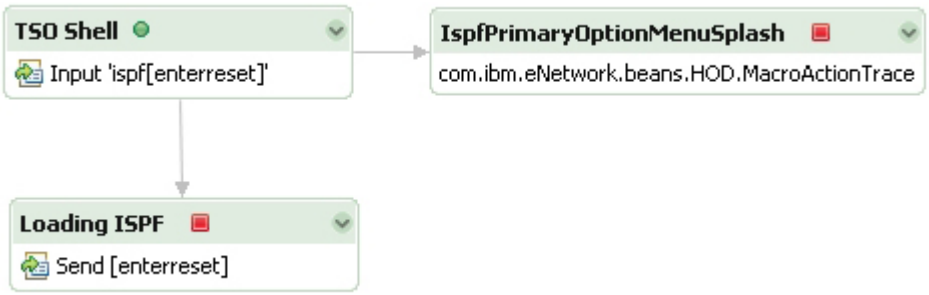

Abbildung 5.13.: Makro zum Starten von ISPF

Tastendruck auf die Eingabetaste um einen Wechsel des Screens zu veranlassen. Nachdem ISPF gestartet wurde befindet sich das Makro auf dem Exit-Screen *IspfPrimaryOptionMenuSplash*, in welchem die Aktion *com.ibm.eNetwork.beans.HOD.MacroActionTrace* ausgeführt wird. Diese Aktion bewirkt, dass in der Konsole der Entwicklungsumgebung ein im Quelltext des Makros festgelegter Text ausgegeben wird. Diese Aktion wird lediglich zu Debug-Zwecken verwendet und hat auf den Ablauf der HATS Web-Anwendung keine Auswirkung. Diese Aktion muss im Quelltext des Makros, wie in Listing 5.2 gezeigt, eingefügt werden. Das Attribut *value* beschreibt den Text, welcher in der Konsole der Entwicklungsumgebung während des Ablaufs des Makros ausgegeben werden soll.

```
1  <actions>
2      <trace type="SYSOUT" value="'screen recognized'"/>
3  </actions>
```

Listing 5.2: HATS Makro Trace-Aktion

Nachdem das Makro *ForwardADCD.hma* beendet wurde, wird das Makro *ForwardISPFSplash.hma* aktiv, welches die Copyright-Meldung von ISPF mit der Eingabetaste bestätigt (Abbildung 5.14). Diese Funktionalität hätte alternativ in das Makro *ForwardADCD.hma* mit aufgenommen werden können. Um die Makros überschaubar zu halten, wurde an dieser Stelle jedoch darauf verzichtet

Abbildung 5.14.: Makro zum Bestätigen der ISPF Copyright-Meldung

5.1.3.2. TSO-Anmeldung - Passwort ändern

Möchte ein Benutzer sein Passwort ändern, muss er das auf dem Screen der TSO-Anmeldung erledigen, welcher in Abbildung 5.6 zu sehen ist. Dazu existiert auf dem Screen neben den Feldern „User-ID“ und „Password“ das Feld „New Password“. Werden die bisherigen Anmeldedaten korrekt eingegeben und dieses Feld mit einem neuen Passwort gefüllt, gelangt der Benutzer im nächsten Schritt zu einem weiteren Screen in welchem zur Bestätigung des neuen Passworts, eine erneute Eingabe verlangt wird. Wird dieses korrekt eingegeben, so wird das Passwort geändert und der Benutzer automatisch in TSO eingeloggt.

Da bei der Erstellung der modernen Oberfläche für die TSO-Anmeldung die Anmeldung einfach gehalten werden sollte und das Passwort im Vergleich zur Anmeldehäufigkeit nicht so oft geändert wird, ist es nicht sinnvoll, dem Benutzer bei der Anmeldung in der HATS Web-Anwendung neben den Eingabefeldern „User-ID“ und „Password“ automatisch ein Eingabefeld „New Password“ anzuzeigen. Stattdessen wurde bei der TSO-Anmeldung in der HATS Web-Anwendung ein Link „Passwort ändern“ eingefügt (Abbildung 5.10), über den der Benutzer bei Bedarf auf eine neue Seite mit Eingabefeldern zum Ändern des Passworts weitergeleitet wird.

Die Umsetzung war problematisch, da dem Benutzer dazu zu Beginn eine Transformation des Screens für die Eingabe des Passworts (Abbildung 5.6) anstatt der Transformation des Screens für die Eingabe der Benutzerkennung (5.5) angezeigt werden muss. Da der Benutzer jedoch zuerst die Benutzerkennung eingeben muss um zu diesem Screen zu gelangen, konnte die Umsetzung allein mit Screen-Anpassungen und Makros nicht realisiert werden. Aus diesem Grund wurde eine Lösung mit einem Integration Object erstellt, dessen zugehöriges Makro in Abbildung 5.15 zu sehen ist.

Das Makro startet im z/OS Willkommen-Screen, wechselt durch das Kommando „L TSO“ zum Screen der TSO-Anmeldung und füllt das Feld „User-ID“ automatisch mit der durch die Variable *username* bereitgestellten Benutzerkennung. Nach einer Bestätigung mit der Eingabetaste füllt das Makro im folgenden Screen die Eingabefelder „Password“ und „New Password“ mit den in den Variablen *old_password* und *new_password* bereitgestellten Passwörtern. Nach dem Absenden der Daten durch die Eingabetaste bestätigt das Makro das neue Passwort, indem das Feld „New Password“ erneut mit dem Inhalt der Variablen *new_password* gefüllt wird. Aufgrund der verwendeten Makro-Aktionen vom Typ „Prompt“ muss der Benutzer vor Ablauf des Makros die Variablen *username*, *old_password* und *new_password* entsprechend füllen. Dazu wurde eine JSP erstellt, welche später in diesem Abschnitt erklärt wird. Fehlermeldungen, wie zum Beispiel ein falscher Benutzername, ein ungültiges Passwort oder nicht übereinstimmende neue Passwörter, veranlassen das Makro dazu, die Fehlermeldung der Host-Anwendung in die globale Variable *errMessage* zu extrahieren und das Makro dann zu beenden. Bei normalem Ablauf wird die TSO-Sitzung ordnungsgemäß durch das Kommando „LOGOFF“ vom Makro beendet.

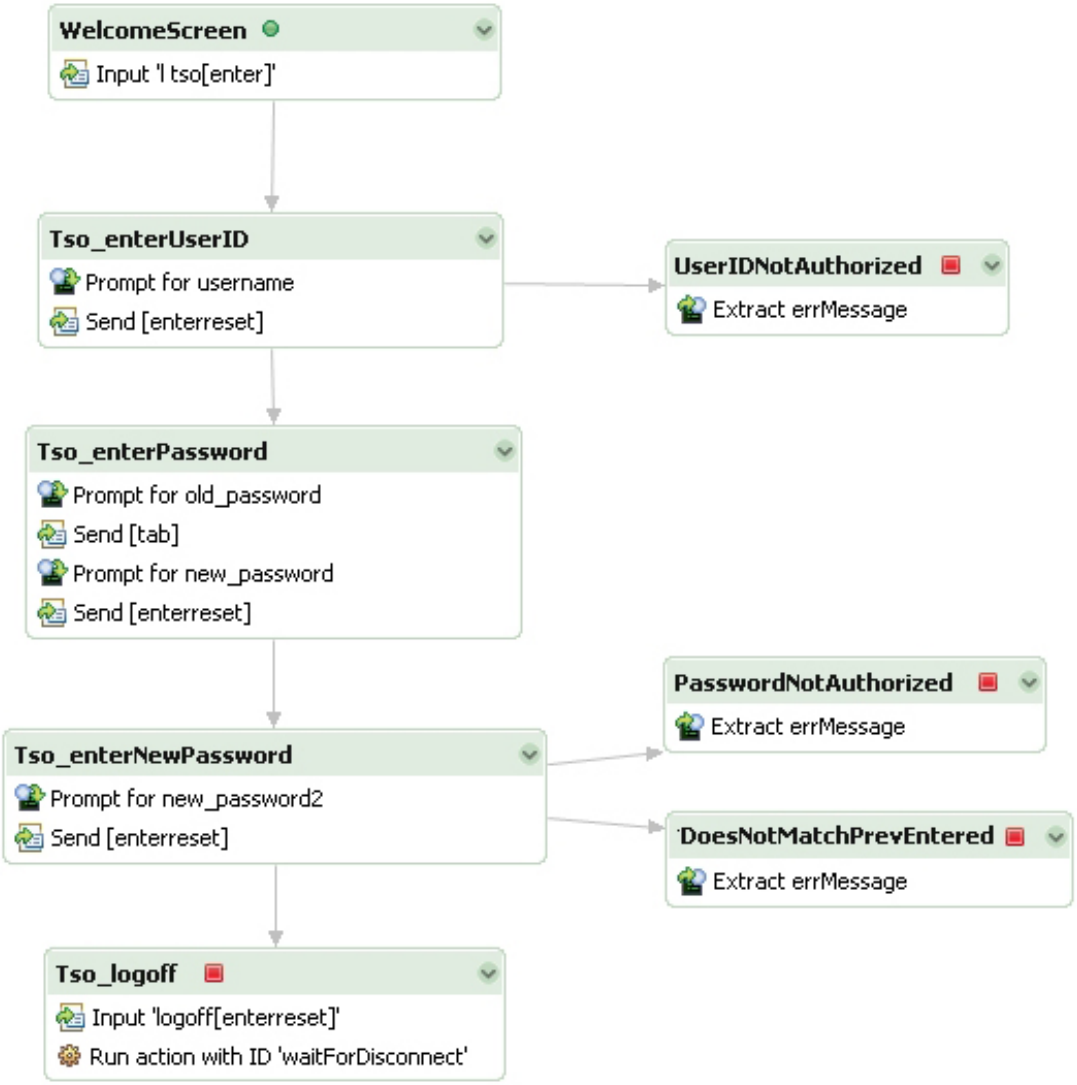

Abbildung 5.15.: Makro für die Änderung des TSO-Passworts

Durch einen Recktsklick auf das Makro und die Auswahl von „Create Integration Object“ aus dem aufklappenden Menü, wird das zum Makro gehörende Integration Object *ChangePassword.java* im Projektverzeichnis *Source/Integration Object* erstellt. Auszüge des erstellten Integration Objects sind im Anhang in Listing A.1 dargestellt. Daraus ist ersichtlich, dass Integration Objects im Wesentlichen nur mit Makros interagierende Java Beans sind, welche die nötigen Daten, in diesem Fall die Benutzerkennung, die Passwörter und die eventuell auftretende Fehlermeldung, beinhalten.

Für die Eingabe der vom Makro benutzten Variablen wurde die JSP *ChangePasswordInput.jsp* erstellt. Die JSP ist in Abbildung 5.16 zu sehen und besteht aus einem HTML-Formular mit Eingabefeldern für die Benutzerkennung und die Passwörter. Durch ein Klick auf die Schaltfläche „Ändern“, ändert das Integration Object das Passwort auf die vom Benutzer in der JSP eingegebenen Daten. Um dem Benutzer eine Erfolgs- oder Fehlermeldung anzuzeigen, wurde die JSP *ChangePasswordOutput.jsp* erstellt, welche in Abbildung 5.17 zu sehen ist. Falls die globale Variable *errMessage* vom Makro gefüllt wurde, wird sie durch den Code in Listing 5.3 auf dieser JSP ausgegeben. In der ersten Zeile wird die Variable *errMessage* aus dem Integration Object ausgelesen. Anschließend wird je nach Inhalt der globalen Variable eine Erfolgs- oder Fehlermeldung ausgegeben. Das Grundgerüst dieser JSPs wurde von der HATS-Entwicklungsumgebung durch Rechtsklick auf das Integration Object und der Auswahl „Create Model 1 Web Pages“ aus dem aufklappenden Menü automatisch erstellt. Lediglich kleine Anpassungen im Quellcode und in der Darstellung waren notwendig. Die erstellten JSPs befinden sich im Verzeichnis *Web Content/Model 1 Pages* des HATS-Projekts.

```
1  String status = ChangePassword.getErrMessage();
2  if (status.equals(""))
3    status = "Das Passwort wurde erfolgreich geändert.";
4  else
5    status = "<p style=\"color: red;\">" + status + "</p>";
```

Listing 5.3: Erfolgs- oder Fehlermeldung in der JSP ausgeben

Abbildung 5.16.: JavaServer Page zum Ändern des TSO-Passworts

Abbildung 5.17.: JavaServer Page für die Erfolgsmeldung der Änderung

5.1.3.3. TSO

Wie bereits erwähnt ist TSO ein interaktiver Kommandozeileninterpreter, welchen man sich von der Darstellung, ähnlich wie eine Linux-Shell oder eine Windows Eingabeaufforderung vorstellen kann. Der Benutzer gibt Kommandos mit verschiedenen Parametern ein und wartet auf das Ergebnis.

Die TSO-Shell besteht vollständig aus Eingabefeldern, weshalb die Standardtransformation von HATS ein unschönes Ergebnis liefern würde (vgl. Abbildung 5.18). Um die TSO-Shell übersichtlich darzustellen, wurde daher die Screen-Anpassung *ReadyTSO.evnt* mit der Transformation *ReadyTSO.jsp* angelegt. Die Screen-Anpassung wird aktiviert, sobald die Zeichenkette „READY“ in der zweiten Spalte einer beliebigen Zeile eines Screens erkannt wird. Die Transformation ist in Abbildung 5.19 zu sehen. Als Eingabefeld konnte eine beliebe Stelle des Screens extrahiert werden. Das Eingabefeld wird über die Input-Field-Komponente mit dem Text-Input-Widget dargestellt. Der komplette Screen wird in der Web-Anwendung mit Hilfe der Text-Komponente und dem Label-Widget dargestellt. Die Eingabe der im Praktikum verwendeten TSO-Kommandos „ISPF“ und „LOGOFF“ kann alternativ über einen Link geschehen, der das dafür erstellte Makro *LogoffTSO.hma* beziehungsweise *LogonISPF.hma* aktiviert, geschehen. Die Makros sind analog zu dem Makro in Abbildung 5.1 aufgebaut. Sie übergeben das jeweilige Kommando an ein Eingabefeld und bestätigen die Eingabe mit der Eingabetaste.

5.1.3.4. TSO-Abmeldung

Zum Abmelden von TSO verwendet der Benutzer das Kommando „LOGOFF“. Wird das Kommando abgesendet, zeigt die Host-Anwendung auf dem Screen eine Meldung der Form „IKJ56470I PRAK097 LOGGED OFF TSO AT 09:21:17 ON NOVEMBER 13, 2008“ mit der aktuellen Benutzerkennung und der entsprechenden Uhrzeit mit Datum an. Anschließend wird die Verbindung von der Host-Anwendung nach wenigen Millisekunde getrennt. Die HATS Web-Anwendung bekommt dies automatisch mit und schließt die Verbindung ebenfalls. Über eine JSP wird dem Benutzer angeboten, die Verbindung erneut zu initiieren. Hierfür handelt es sich um eine Grundfunktionalität von HATS. Es musste keine Screen-Anpassung oder ein Makro dafür angelegt werden.

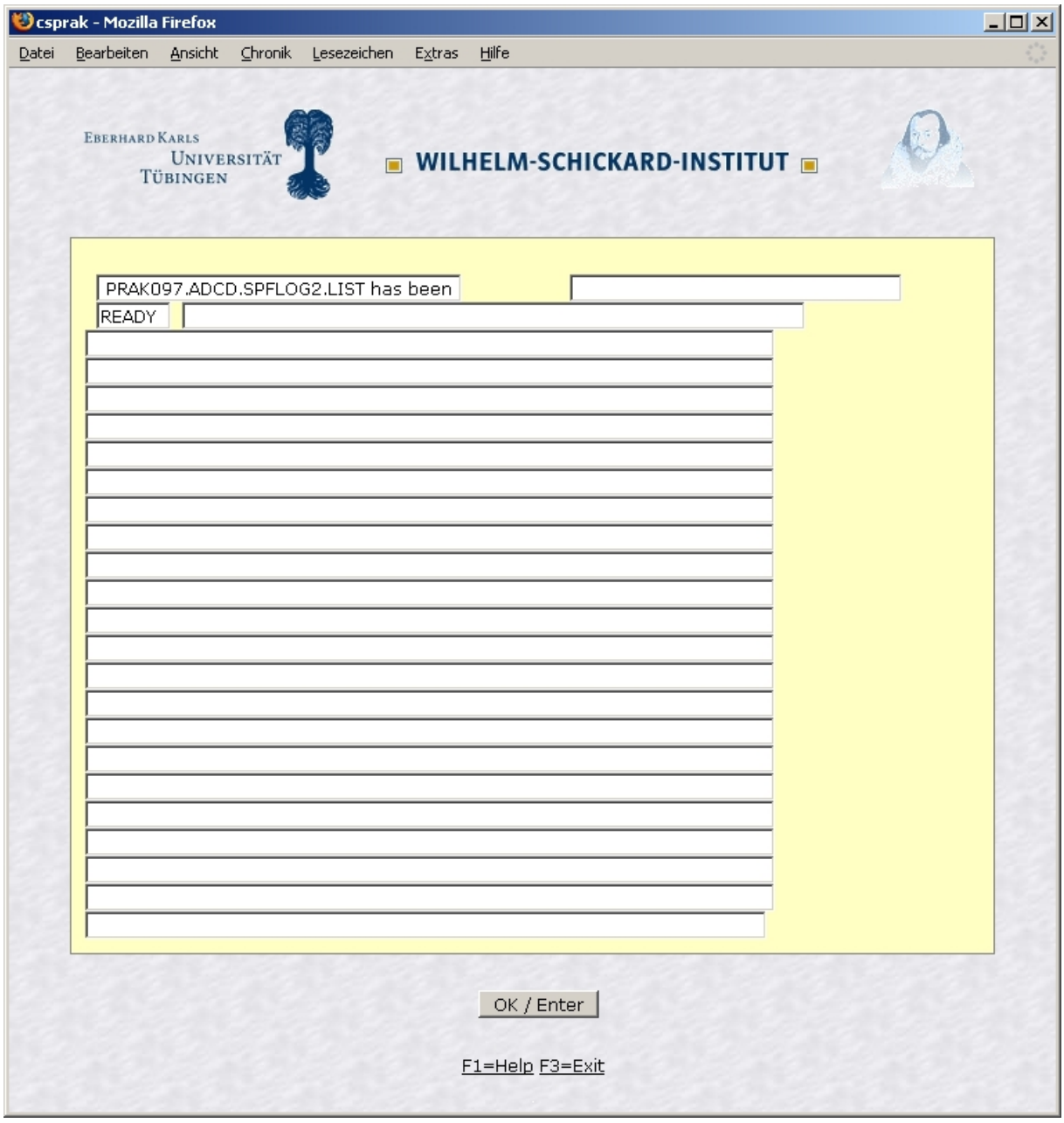

Abbildung 5.18.: TSO-Shell bei reiner Verwendung der Standardtransformation

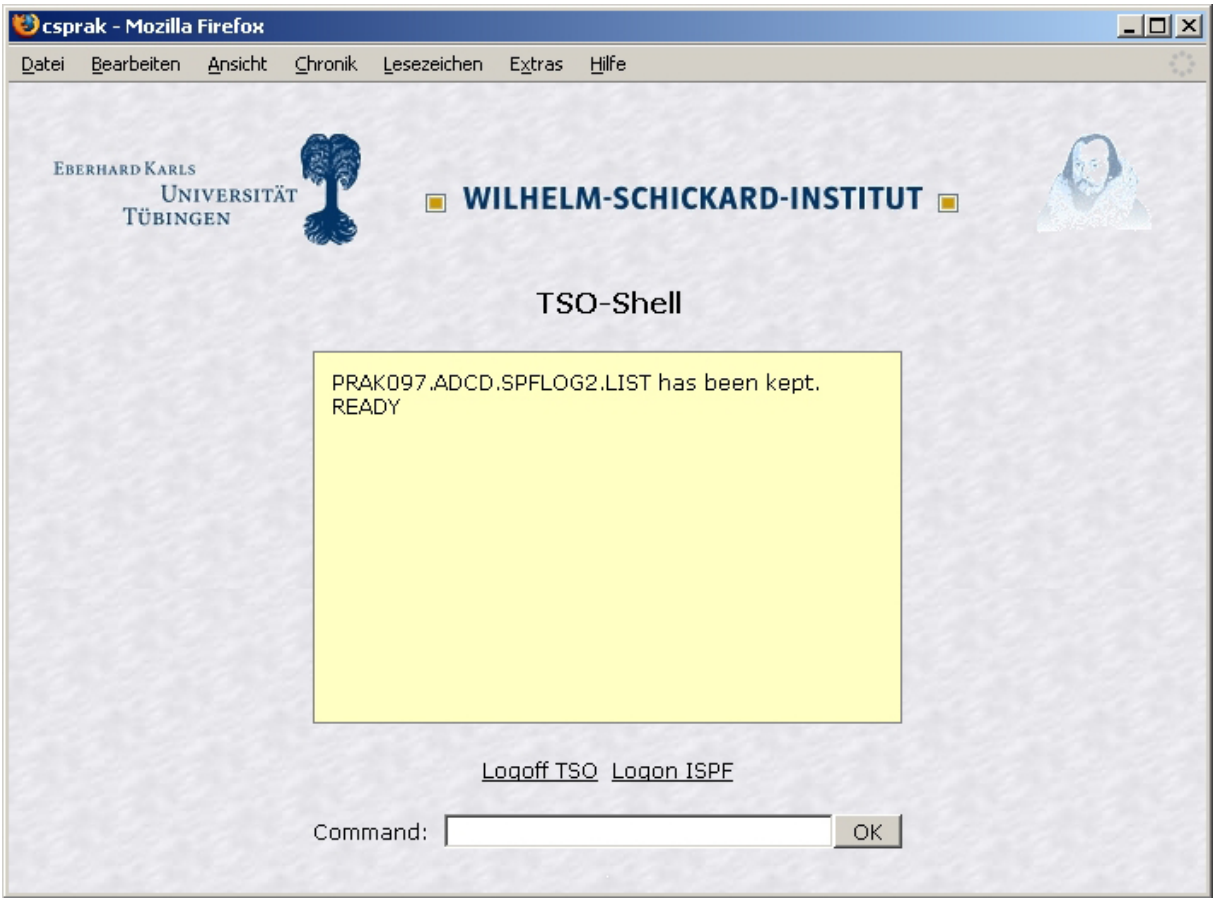

Abbildung 5.19.: TSO-Shell in der HATS Web-Anwendung

5.1.4. ISPF Umsetzung

ISPF ist im Client/Server-Praktikum die mit Abstand am häufigsten benutzte Host-Anwendung. Jeder ISPF-Screen kann in die vier Bereiche Menüleiste (Action Bar), Eingabefeld, Hinweise für die Belegung der Funktionstasten und eigentlicher Inhalt gegliedert werden.

Da alle für das Client/Server-Praktikum wichtigen Elemente aus der Menüleiste über Funktionstasten oder über ISPF-Kommandos zugänglich sind, wurde in der HATS Web-Anwendung auf eine Umsetzung der Menüleiste verzichtet. Die Umsetzung der Menüleiste wäre zusätzlich problematisch zu realisieren, da ein aus der Menüleiste aufgeklapptes Untermenü Teile des ursprünglichen Host-Screens überlappt und somit für eine Screen-Anpassung eventuell wichtige Erkennungskriterien verdeckt.

Die Funktionstaste PF2, welche einen Screen an der aktuellen Cursorposition in zwei unabhängige Screens auftrennt, konnte ebenfalls nicht in der HATS Web-Anwendung umgesetzt werden. Durch die Auftrennung eines Screens geht ein Großteil der Erkennungskriterien für einen Screen verloren, was Screen-Anpassungen extrem schwierig, wenn nicht sogar unmöglich machen. Aus diesem Grund wurde, wie in Abschnitt 3.3.3 erklärt, die Funktionstaste PF2 für die HATS Web-Anwendung deaktiviert.

Alle nachfolgend aufgeführten Screen-Anpassungen wurden bei den Screen-Anpassungen im HATS-Projekt im Unterordner *ISPF* erstellt.

5.1.4.1. ISPF Primary Option Menü

Das ISPF Primary Option Menü, welches in Abbildung 5.20 dargestellt ist, wird dem Benutzer beim Starten von ISPF angezeigt und dient als Hauptmenü. Der Benutzer navigiert durch die Eingabe von Optionen in Form von alphanumerischen Ziffern zum gewünschten Ziel. Eine Umsetzung dieses Screens mit Hilfe der Standardtransformation hätte bewirkt, dass der Benutzer weiterhin durch die Eingabe von Optionen in ein Eingabefeld navigieren muss. Daher wurde die Screen-Anpassung *IspfPrimaryOptionMenu.evnt* und die Transformation *IspfPrimaryOptionMenu.jsp* erstellt, wodurch der Benutzer die alternative Möglichkeit bekommt, mit der Maus über Links zu navigieren. Als Erkennungskriterium für diese Screen-Anpassung wurde definiert, dass die Zeichenkette „ISPF Primary Option Menu“ beginnend an Position (29,3) auf dem Host-Screen erscheint. Als auszuführende Aktion wurde die Darstellung der Transformation definiert.

In Abbildung 5.21 wird die Transformation gezeigt. Das Hauptmenü von ISPF wird in der Transformation durch die selbst entwickelte OptionList-Komponente und das selbst entwickelte OptionList-Widget dargestellt. Die Erstellung der Komponente und des Widgets wird in Abschnitt 5.2 näher erläutert. Durch die erstellte Komponente und das zugehörige Widget wird ein Bereich des Hauptmenüs durch Text und Links dargestellt. Ein Mausklick auf den Link „Settings“ bewirkt zum Beispiel die Eingabe des Zeichens „0“ in das Eingabefeld, wodurch der

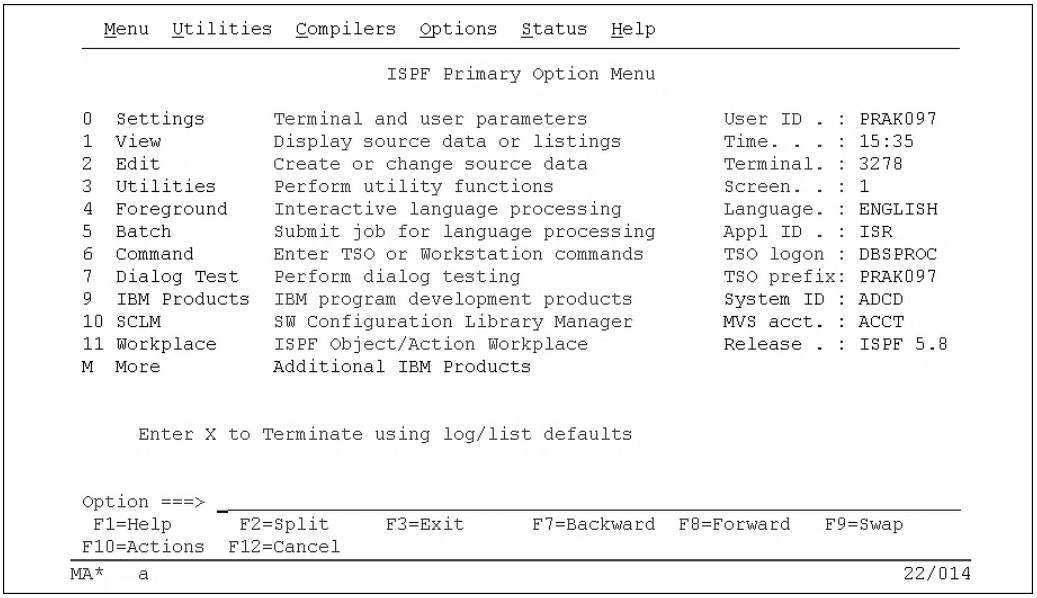

Abbildung 5.20.: ISPF Primary Option Menü

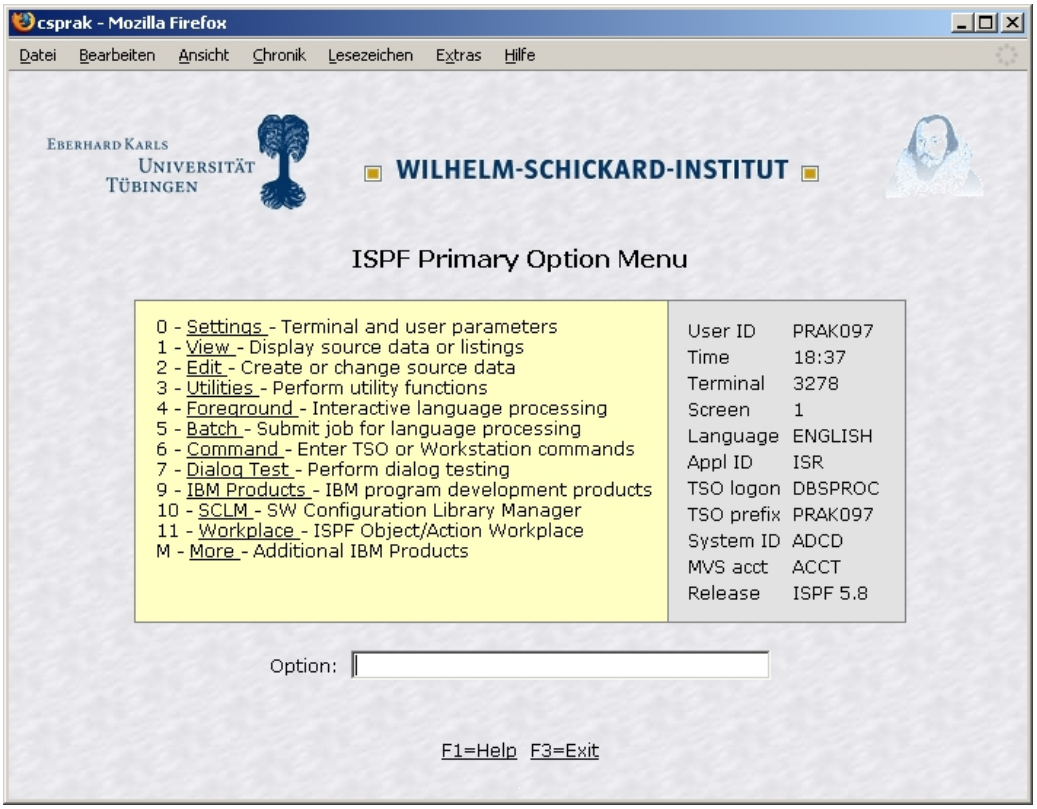

Abbildung 5.21.: ISPF Primary Option Menü in der HATS Web-Anwendung

Benutzer allein durch Bedienung mit der Maus zu den ISPF-Einstellungen gelangen kann. Damit der Benutzer durch alleinige Navigation mit der Maus zusätzlich die Bedienung über die Tastatur erlernen kann, wird das aktuell durch den Mausklick bewirkte alphanumerische Zeichen mit Hilfe von mit HATS mitgeliefertem JavaScript automatisch in das Eingabefeld eingefügt. Erst nach Ablauf von knapp einer Sekunde wird durch das automatische Bestätigen mit der Eingabetaste das eingefügte Zeichen an die Host-Anwendung abgeschickt. Somit erfährt und erlernt der Benutzer was der Mausklick bewirkt.

Weitere auf dem Screen erscheinende Informationen wie zum Beispiel die aktuelle Benutzerkennung und Zeit werden durch die Table-Komponente in Verbindung mit dem Table-Widget transformiert. Das Eingabefeld in Zeile 22 des Screens wird mit der Input-Field-Komponente und dem Text-Input-Widget dargestellt. Die Hinweise auf Funktionstasten werden mit Hilfe der Function-Key-Komponente und dem Link-Widget als Links zur Verfügung gestellt. Dabei werden lediglich die Funktionstasten PF1 und PF3 dargestellt. Andere Funktionstasten haben auf diesem Screen für das Client/Server-Praktikum keine bedeutenden Funktionen. Eventuelle ISPF-Fehlermeldungen erscheinen auf dem Screen an Position (54,3) bis (79,3). Diese Fehlermeldungen werden zur besseren Sichtbarkeit in der HATS Web-Anwendung rot gefärbt und unter dem Eingabefeld platziert. Das gerade erwähnte Eingabefeld und die Hinweise auf Funktionstasten wurden für alle ISPF-Screens in nachfolgend beschriebenen Transformationen analog umgesetzt und werden nicht mehr gesondert erklärt.

Das IBM Products Panel, welches über die Option „M“ im ISPF Primary Option Menü zugänglich ist, und das Utility Selection Panel, welches über die Option „3“ zugänglich ist, sind ähnlich wie das ISPF Primary Option Menü aufgebaut. Daher wurden für diese Screens ebenfalls Screen-Anpassungen und Transformationen erstellt, welche die OptionList-Komponente verwenden. Da bezüglich der Umsetzung in den Dateien *IbmProductsPanel.evnt*, *IbmProductsPanel.jsp*, *UtilitySelectionPanel.evtn* und *UtilitySelectionPanel.jsp* keine großen Unterschiede zur Screen-Anpassung und Transformation des Primary Options Menüs bestehen, werden diese hier nicht näher erläutert.

5.1.4.2. ISPF Primary Option Menü - TSO CALL und EXEC

Im Client/Server-Praktikum werden in Tutorial 2 und Tutorial 6 einfache Anwendungen mit C, COBOL oder Assembler und mit REXX entwickelt, wobei der Benutzer anschließend angeleitet wird, diese Anwendungen im ISPF Primary Option Menü mit dem Befehl „TSO EXEC dataset(member) EXEC“ oder „TSO CALL 'dataset(member)' “ zu starten. Die Ausgabe der Anwendung wird hierbei in der Host-Anwendung an das untere Ende des ISPF Primary Option Menüs angefügt und gegebenenfalls auf einem weiteren leeren Screen weitergeführt. Dadurch werden die Hinweise für die Funktionstasten im ISPF Primary Option Menü Screen, wie in Abbildung 5.22 gezeigt, überschrieben. Folglich musste, um die Ausgabe der selbst erstellten Anwendung korrekt anzuzeigen, die Screen-Anpassung *IspfPrimaryOptionMenu_ TsoExec.evnt* erstellt werden. Für die Screen-Anpassung wurde als Erkennungskriterium die Zeichenkette „ISPF Primary Option

Menu“ beginnend an der Position (29,3) und zusätzlich die Zeichenkette „*** “ in dem Screen-Bereich (1,21) bis (80,24) festgelegt. Als auszuführende Aktion wurde wieder die Darstellung einer Transformation definiert.

Die zugehörige Transformation *IspfPrimaryOptionMenu_ TsoExec.jsp* ist in Abbildung 5.23 dargestellt und zeigt lediglich die Ausgabe der ausgeführten Anwendung über die Field-Komponenten mit dem Field-Widget an. Um Fortzufahren wurde zusätzlich die Schaltfläche „OK“ eingefügt, welche in der Host-Anwendung ein Tastendruck auf die Eingabetaste emuliert. Auf eine zusätzliche Darstellung des ISPF Primary Option Menüs analog zum Host-Screen wurde verzichtet, da somit dem Benutzer keine überflüssigen Informationen angezeigt werden.

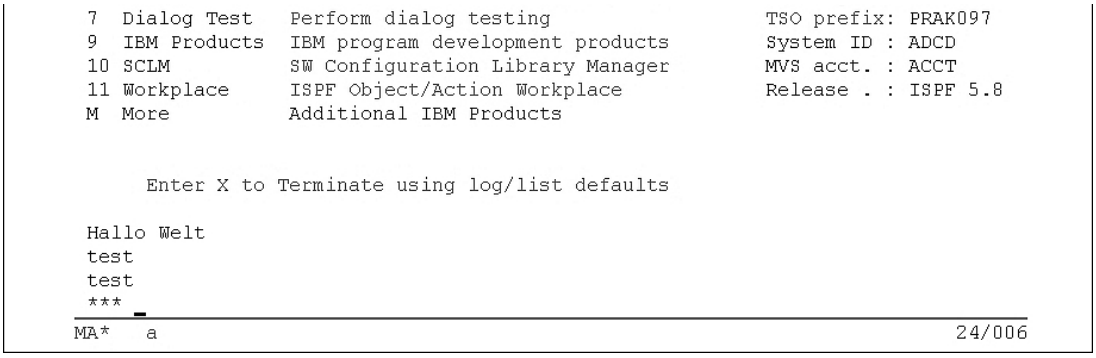

Abbildung 5.22.: Host-Anwendung nach Ausführung von TSO EXEC

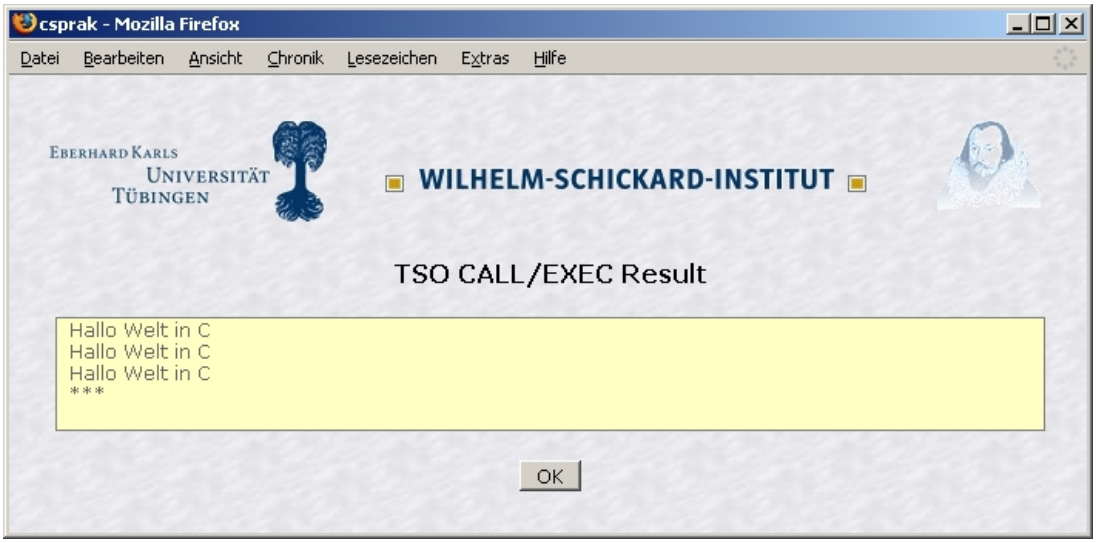

Abbildung 5.23.: Ausführung von TSO EXEC in der HATS Web-Anwendung

5.1.4.3. ISPF Dataset Utility

Das ISPF Dataset Utility ist über die Option „4.2“ aufrufbar und wird unter anderem dazu verwendet um Datasets zu erstellen oder Informationen über Datasets einzusehen. Dabei existieren zwei unterschiedliche Möglichkeiten einen Dataset-Namen anzugeben. Für dreiteilige Dataset-Namen ist die vollständige Angabe der drei Bestandteile Projekt, Gruppe und Typ im Feld „ISPF Library“ möglich. Ist der Dataset-Name nicht dreiteilig, muss der Name über das Feld „Other Partitioned, Sequential or VSAM Data Set“, mit Hochkommata umgeben, angegeben werden. Das ISPF Dataset Utility der 3270-Host-Anwendung ist in Abbildung 5.24 dargestellt. Um diesen Screen, der bei der Arbeit mit ISPF oft verwendet wird, graphisch aufzubereiten, wurde die Screen-Anpassung *DataSetUtility.evnt* erstellt. Die Screen-Anpassung wird aktiv, falls die Zeichenkette „Data Set Utility“ auf einem Screen, beginnend an der Position (32,3), erscheint.

Als zugehörige Aktion wird der Screen über die Transformation *DataSetUtility.jsp* angezeigt. Für die Umsetzung werden mehrere verschiedene Komponenten verwendet. Der obere Abschnitt des Screens, mit Abkürzungen für die möglichen Aktionen, wird analog zum ISPF Primary Option Menü mit der eigenen OptionList-Komponente über das OptionList-Widget dargestellt. Dadurch bewirkt zum Beispiel ein Mausklick auf den Link „Allocate“, dass die zugehörige Option „A“ in das Eingabefeld geschrieben und abgesendet wird. Wurde ein Dataset-Name angegeben, wird ein Dataset angelegt.

Der restliche Screen besteht aus einzelnen Field-Komponenten, die durch Field-Widgets dargestellt werden. Lediglich das Feld „Confirm Data Set Delete“, welches in der Host-Anwendung durch das vorangestellte Zeichen „/“ aktiviert werden kann, wurde mit Hilfe der Item-Selection-Komponente über ein Check-Box-Widget dargestellt. Dadurch ist dem unerfahrenen Anwender sofort ersichtlich wie das Auswahlfeld zu bedienen ist. Die Transformation ist in Abbildung 5.25 zu sehen.

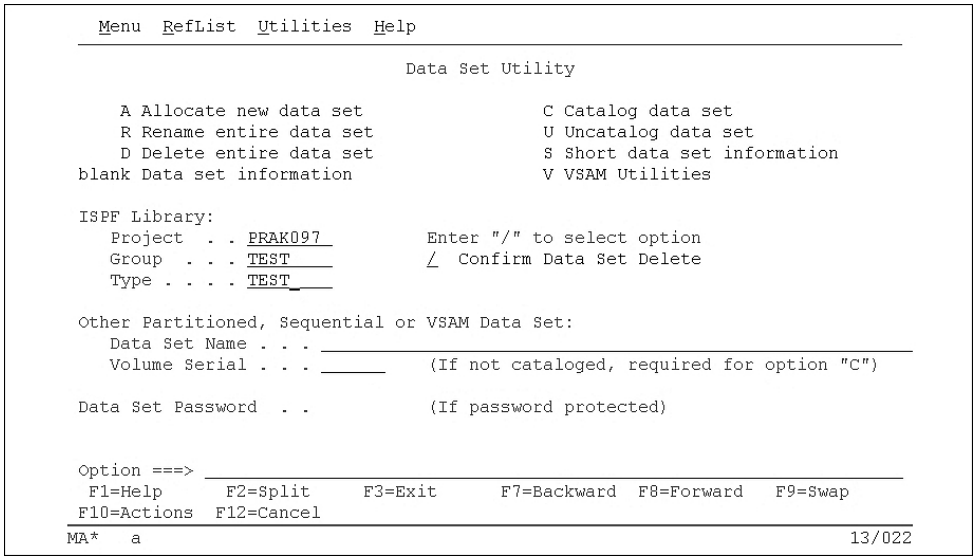

Abbildung 5.24.: ISPF Dataset Utility

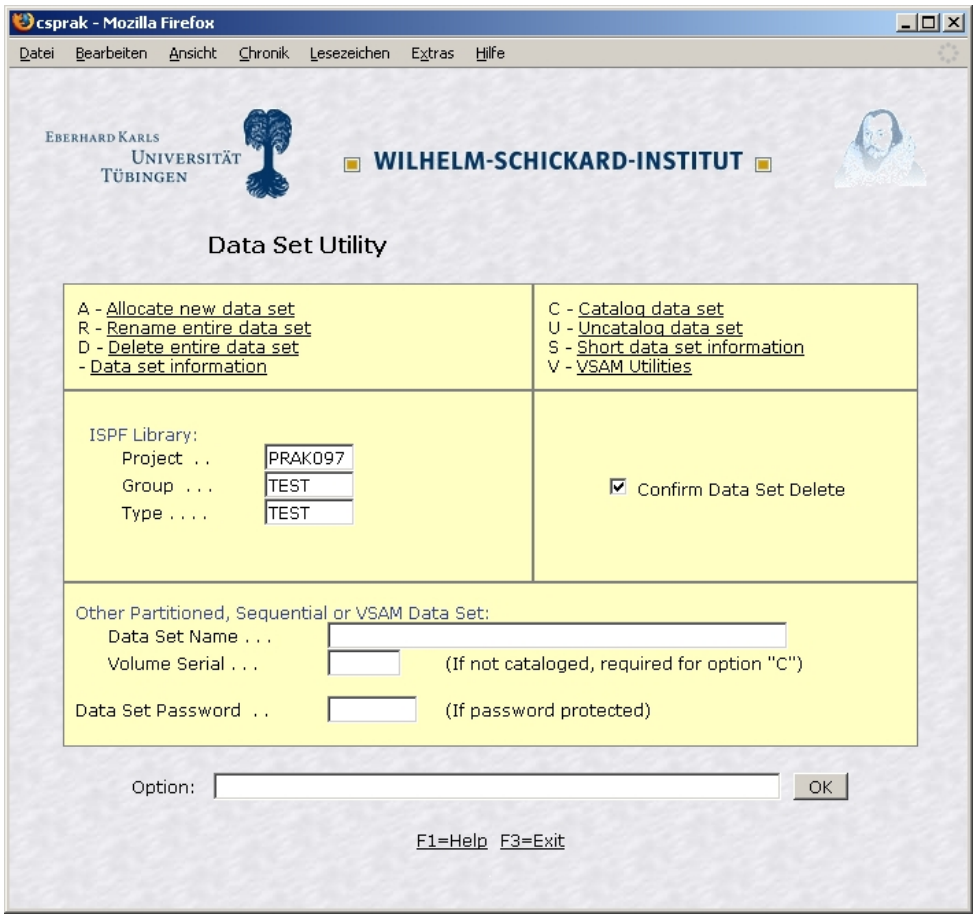

Abbildung 5.25.: ISPF Dataset Utility in der HATS Web-Anwendung

5.1.4.4. ISPF Dataset List Utility

Das ISPF Dataset List Utility (DSLIST) ist über die Option „3.4“ aufrufbar und wird benutzt um Datasets nach bestimmten Kriterien aufzulisten. In Abbildung 5.26 wurde das Kriterium zum Beispiel so definiert, dass alle Datasets mit dem Präfix *PRAK097* aufgelistet werden. Anschließend können auf den aufgelisteten Datasets diverse Operationen, wie zum Beispiel Löschen, Kopieren und Editieren ausgeführt werden. Um für den unerfahrenen Anwender die wesentlichen Elemente des Screens hervorzuheben und um die Auswahl der für die Auflistung möglichen Optionen zu vereinfachen, wurde die Screen-Anpassung *DataSetListUtility.evnt* erstellt, welche aktiv wird, sobald auf einem Screen beginnend an Position (30,3) die Zeichenkette „Data Set List Utility“ erscheint.

Als Aktion stellt die Screen-Anpassung den Screen über die in Abbildung 5.27 gezeigte Transformation *DataSetListUtility.jsp* dar. Der obere Abschnitt des Screens wird über die Field-Komponente mit dem Field-Widget dargestellt. Der Screen-Bereich „Initial View“ wird durch eine Selection-List-Komponente über das Drop-Down-Widget dargestellt, wobei die Einstellungen der Komponente manuell angepasst wurden. Als Ziel-Eingabefeld für das Drop-Down-Widget wurde das Eingabefeld an Position (24,13) definiert, wodurch jede Auswahl automatisch in das angegeben Eingabefeld übertragen wird. Die Auswahlfelder im rechten Bereich des Screens wurden analog zur vorherigen Screen-Anpassung über die Item-Selection-Komponente mit dem Check-Box-Widget dargestellt. Hierbei war keine Modifikation der Komponenteneinstellung nötig.

Im nachfolgenden Screen werden die durch die Suchkriterien gefundenen Datasets in einer Dataset-Liste dargestellt. Dieser Screen ist in Abbildung 5.28 dargestellt. Hier hat der Benutzer die Möglichkeit den Inhalt von Datasets anzuzeigen oder zu manipulieren, indem diesem Kommandos vorangestellt und mit der Eingabetaste bestätigt werden. Mögliche Kommandos sind zum Beispiel „B“ um ein Dataset zu durchsuchen, „CO“ um ein Dataset zu kopieren, „MO“ um ein Dataset zu verschieben oder „Z“ um ein Dataset zu komprimieren. Insgesamt existieren über 10 Kommandos, welche der unerfahrene Benutzer oft in der Dokumentation nachschlagen muss. Der Benutzer hat zusätzlich die Möglichkeit sich weitere Informationen über ein Dataset, wie zum Beispiel die verwendete Blockgröße, anzuschauen. Hierzu navigiert der Benutzer mit den Funktionstasten PF10 und PF11 nach links beziehungsweise nach rechts, bis die gewünschte Information in einer Spalte angezeigt wird. Problematisch für unerfahrene Anwender ist an dieser Stelle, dass die Informationen aufgrund der kleinen Größe eines Screens abgekürzt werden. Zum Beispiel steht die Abkürzung „Blksz“ für Blockgröße (Block Size), was für manche Anwender nicht sofort ersichtlich ist.

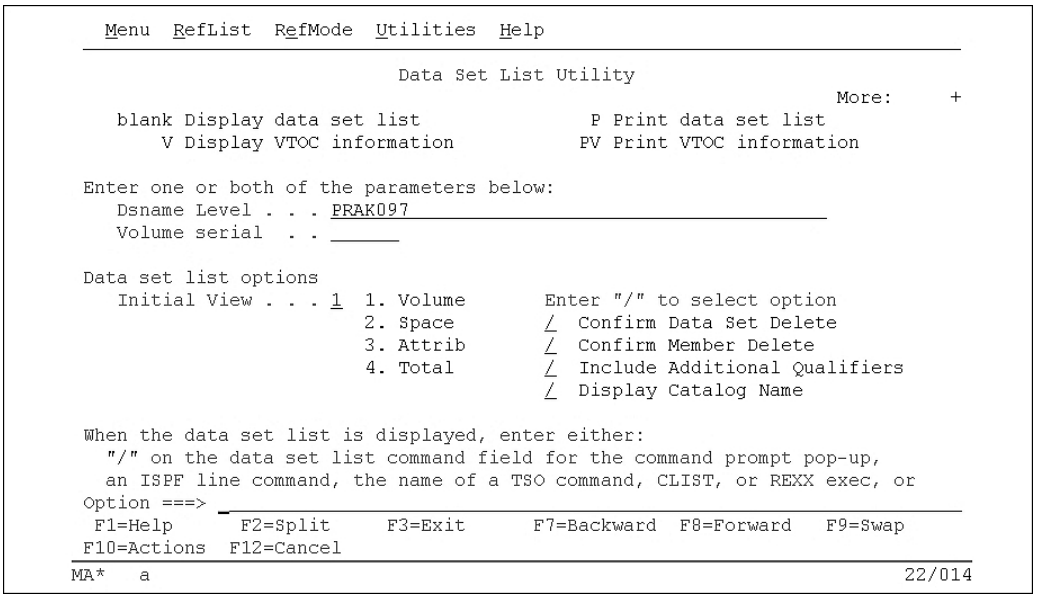

Abbildung 5.26.: ISPF Dataset List Utility

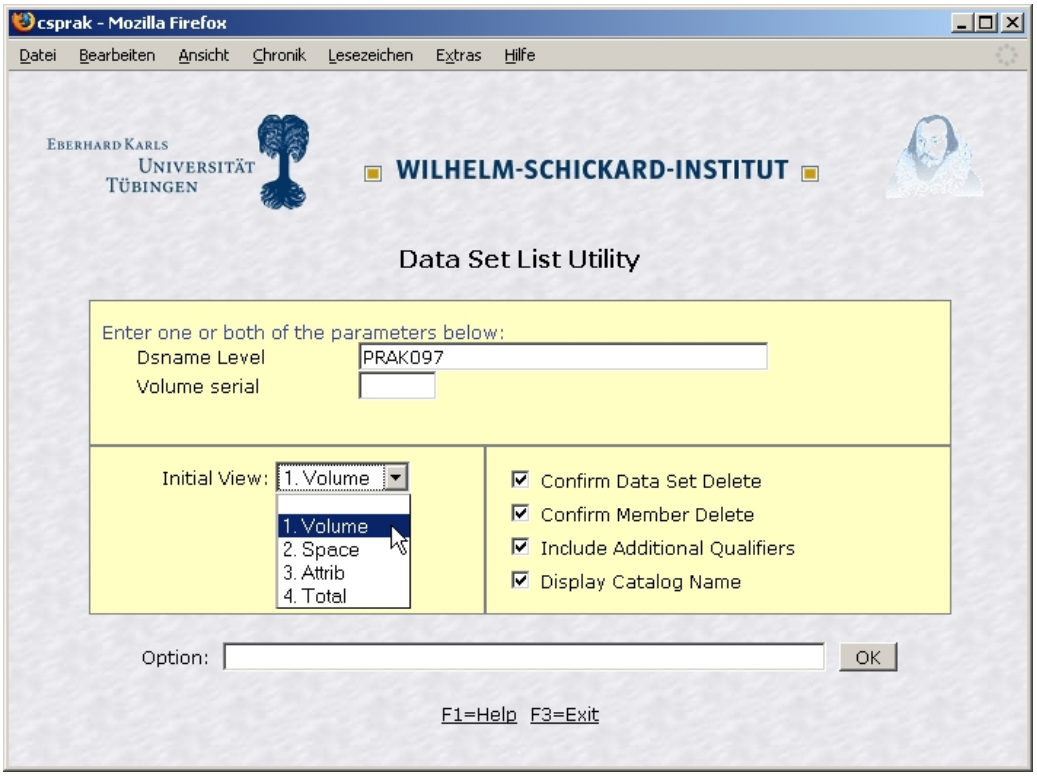

Abbildung 5.27.: ISPF Dataset List Utility in der HATS Web-Anwendung

```
  Menu  Options  View  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
 DSLIST - Data Sets Matching PRAK097                          Row 1 of 19

 Command - Enter "/" to select action                Dsorg  Recfm  Lrecl  Blksz
 ------------------------------------------------------------------------------
          PRAK097
          PRAK097.ADCD.SPFLOG1.LIST                   PS     VA       125    129
          PRAK097.CICS.TEST                           PO     FB        80    320
          PRAK097.HCD.MSGLOG                          PS     FB       133   1330
          PRAK097.HCD.TERM                            PS     FB        80   1600
          PRAK097.HCD.TRACE                           PS     FB        80   6160
          PRAK097.ISPF.ISPPROF                        PO     FB        80   3120
       B_ PRAK097.LIB                                 PO     FB        80    320
          PRAK097.REXX.EXEC                           PO     FB        80    320
          PRAK097.SPFLOG1.LIST                        PS     VA       125    129
          PRAK097.SPUFI.IN                            PO     FB        80    320
          PRAK097.SPUFI.OUT                           PS     VB      4092   4096
          PRAK097.TEST                                PO     FB        80  27920
          PRAK097.TEST.C                              PO     FB        80    320
          PRAK097.TEST.CNTL                           PO     FB        80    320
 Command ===> _____________________________________________ Scroll ===> PAGE
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
 MA*   a                                                          14/009
```

Abbildung 5.28.: Auflistung von Datasets durch das ISPF Dataset List Utility

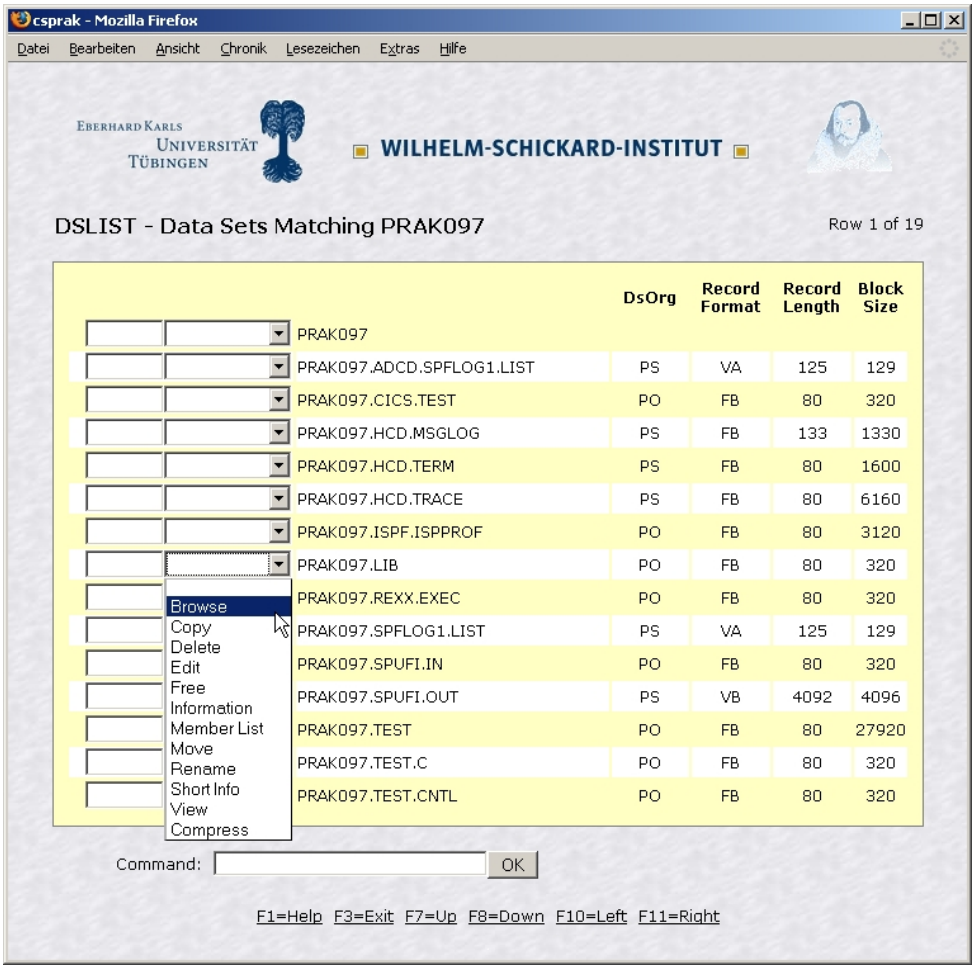

Abbildung 5.29.: Auflistung von Datasets in der HATS Web-Anwendung

Um dem Benutzer die Handhabung dieser Informationen zu vereinfachen, wurden vier weitere Screen-Anpassungen mit dem Namen *DSListOverview.evnt*, *DSListOverview4ColA.evnt*, *DSListOverview4ColB.evnt* und *DSListOverviewCompl.evnt* erstellt, welche als Aktion jeweils eine Transformation aufrufen. Vier Screen-Anpassungen sind nötig, da an dieser Stelle mit den Funktionstasten PF10 und PF11 genau durch vier Screens navigiert werden kann und sich dabei jeder Screen im Aufbau durch unterschiedliche Spaltenbreiten und Spaltenüberschriften unterscheidet. Als Erkennungskriterium für die Screen-Anpassungen wurde jeweils die Zeichenkette „DSLIST - Data Sets“ an Position (2,3) und die Spaltenüberschriften definiert.

Für die Transformation wurde die Komponente *DataSetList* mit den Widgets *DataSet2Columns* und *DataSet4Columns* entwickelt, dessen Erstellung in Abschnitt 5.2 näher betrachtet wird. Der Benutzer bekommt dadurch vor jedem Dataset-Namen zusätzlich zum Eingabefeld für Befehle eine Auswahlliste angezeigt, in welcher er für den Dataset die gewünschte Aktion über einen Mausklick auswählen kann (vgl. Abbildung 5.29). In der Auswahlliste wird nicht das wenig aussagekräftige Kommando, sondern die komplett ausgeschriebene Aktion angezeigt. Wählt der Benutzer eine Aktion aus, wird in das zu dem Dataset gehörende Eingabefeld automatisch das entsprechende Kommando eingefügt. Hinter dem Eingabefeld und der Auswahlliste werden in einer HTML-Tabelle verschiedene Informationen über das Dataset angezeigt, welche aus dem Host-Screen extrahiert wurden. Die Komponente erkennt die Informationen anhand der in den Komponenteneinstellungen definierten Koordinaten für das Eingabefeld und für jeweils Anfang und Ende jeder Spalte. Zusätzlich kann hier im Format „Kommando=Aktion;“ angegeben werden, welche Aktionen über welche Kommandos ausgeführt werden können. In den Einstellungen der Widgets *DataSet2Columns* und *DataSet4Columns* kann jeweils die Überschrift der einzelnen Spalten definiert werden. In Abbildung 5.29 wurde dadurch zum Beispiel für die letzte Spalte die aussagekräftigere Spaltenüberschrift „Block Size“ festgelegt. Somit ist dem Benutzer sofort ersichtlich welche Information die Spalte beschreibt.

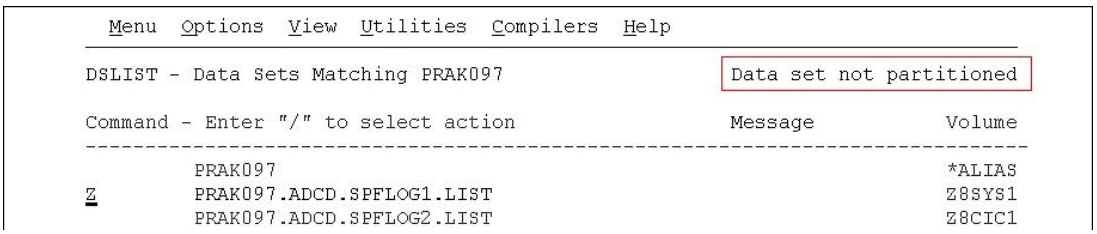

Abbildung 5.30.: Blockierende Fehlermeldung im ISPF Dataset List Utility

Manche Kommandos können vom Benutzer nur auf bestimmten Datasets ausgeführt werden. So kann ein nicht partitionierter Dataset zum Beispiel nicht komprimiert werden. Wird auf einem Dataset vom Benutzer ein ungültiges Kommando abgesetzt, resultiert dies in einer gelben Fehlermeldung in der dritten Zeile des Screens (Abbildung 5.30). Durch die Fehlermeldung werden alle weiteren Kommandos blockiert, bis das falsch abgesetzte Kommando vom Benutzer durch Überschreiben mit Leerzeichen oder durch Überschreiben mit einem alternativen Kommando entfernt wird. Um dieses Problem automatisch zu beheben, wurde eine neue Screen-Anpassung *DSList-*

ProcUnavailable.evnt erstellt, welche als Aktion die fehlerhafte Benutzereingabe zurücksetzt. Da mehrere verschiedene Fehlermeldungen existieren, ist in diesem Fall der Inhalt der Fehlermeldung als Erkennungskriterium für die Screen-Anpassung nicht sinnvoll. Stattdessen wird die Farbe der Fehlermeldung zur Screen-Erkennung ausgenutzt. Falls an Position (1,3) eines Screens die Zeichenkette „DSLIST“ steht und an Position (79,3) die Textfarbe nicht cyan und nicht weiß ist, wird der Screen von der Screen-Anpassung erkannt. Um ein Ereignis zu invertieren, in diesem Fall das Ereignis cyan und weiß, wird bei dem Erkennungskriterium das Attribut *invertmatch* auf *true* gesetzt. Das Erkennungskriterium wird durch den XML-Abschnitt der Screen-Anpassung in Listing 5.4 beschrieben.

```
<description>

   <!-- Screen nicht blockiert -->
   <oia invertmatch="false" optional="false"
        status="NOTINHIBITED"/>

   <!-- Zeichenkette DSLIST im Bereich von (3,1) bis (9,1)
   <string casesense="false" col="1" ecol="9" erow="3"
          invertmatch="false" optional="false" row="3"
          value=" DSLIST -" wrap="false"/>


   <!-- Nicht cyan -->
   <attrib col="79" invertmatch="true" optional="false" ´
          plane="COLOR_PLANE" row="3" value="0x3"/>

   <!-- Nicht weiß -->
   <attrib col="79" invertmatch="true" optional="false" ´
          plane="COLOR_PLANE" row="3" value="0x7"/>

</description>
```

Listing 5.4: Erkennungskriterium für die blockierende Fehlermeldung

Da nun der Screen erfolgreich erkannt wird, kann die Fehlermeldung von der Screen-Anpassung entsprechend behandelt werden. Ziel ist es, die fehlerhafte Benutzereingabe zu löschen und dem Benutzer eine entsprechende Fehlermeldung anzuzeigen. Dazu wurde die globale Variable *errMessage* angelegt, welche die Fehlermeldung temporär zur Ausgabe zwischenspeichert. In der Screen-Anpassung wurde dazu als Aktion eine neue Aktion vom Typ „Extract a global variable“ angelegt, welche die Fehlermeldung im Screen-Bereich von Position (49,3) bis (80,3) in der globalen Variable speichert.

Um die Fehlermeldung zu beseitigen, wurde für die Screen-Anpassung eine zusätzliche Aktion vom Typ „Send key to host screen“ mit dem Wert „Send [eraseof] to the host screen at the current cursor position“ erstellt. Die Taste „[eraseof]“ löscht innerhalb einer Host-Anwendung alle Benutzereingaben auf dem Screen. Der Inhalt der globalen Variable *errMessage* wird dem

Benutzer anschließend als roter Text in der Transformation angezeigt. Die globale Variable wurde in der Entwicklungsumgebung mit dem Menüpunkt „HATS Tools/Insert Global Variable“ mit der Option „Display global variable as text“ eingefügt. Über diesen Menüpunkt wurden ebenso alle in späteren Transformationen erwähnten globalen Variablen eingefügt.

5.1.4.5. Member-Liste

Die Member-Liste kann entweder über das Dataset List Utility oder über den Edit Entry Panel beziehungsweise den View Entry Panel durch das Freilassen des Feldes „Member“ angezeigt werden. Ähnlich zur Dataset-Liste können durch vorangestellte Kommandos auf Member verschiedene Operationen wie zum Beispiel Löschen, Umbenennen oder Editieren angewandt werden. Um dem Benutzer hierbei unter die Arme zu greifen, wurden analog zur Screen-Anpassung für die Dataset-Liste weitere Screen-Anpassungen erstellt, welche die Screens mit der selbst erstellten DataSetList-Komponente und dem zugehörigen MemberList4Columns-Widget transformieren. Da sich je nach Aufruf die Member-Liste jedoch in signifikanten Merkmalen wie zum Beispiel der Länge des Eingabefeldes und somit der Breite der ersten Spalte unterscheidet, sind mehrere Screen-Anpassungen nötig.

Der Unterschied zwischen den zwei verschiedenen Versionen der Member-Liste ist in Abbildung 5.31 und Abbildung 5.32 noch einmal ersichtlich. Für die Member-Liste, welche über das Dataset List Utility erreichbar ist, wurden die Screen-Anpassungen *DSListEdit.evnt* und *DSListEdit2.evnt* erstellt. Für die sich leicht unterscheidende Member-Liste, die über den Edit- und View Entry Panel erreichbar ist, wurden die Screen-Anpassungen *Memberlist.evnt* und *Memberlist2.evnt* erstellt. Als Erkennungskriterium wurden jeweils die Spaltenüberschriften in der vierten Zeile des Screens betrachtet. So dient als Erkennungskriterium für die Screen-Anpassungen *DSListEdit.evnt* und *DSListEdit2.evnt* jeweils die Zeichenkette „Name Prompt Size Created Changed ID“ mit zwei führenden Leerzeichen und für die Screen-Anpassungen *Memberlist.evnt* und *Memberlist2.evnt* dieselbe Zeichenkette jedoch mit neun führenden Leerzeichen. Da der Benutzer analog zu der Dataset-Liste mit den Funktionstasten PF11 und PF12 nach links und rechts navigieren kann um unterschiedliche Informationen zu einem Member angezeigt zu bekommen, sind jeweils zwei Screen-Anpassungen pro Member-Liste nötig. Die Transformation ist in Abbildung 5.33 zu sehen.

5.1.4.6. ISPF-Editor

Der ISPF-Editor ist in Abbildung 5.34 dargestellt und wird geöffnet, sobald ein Member eines Datasets angezeigt oder editiert wird. Der ISPF-Editor funktioniert ähnlich wie der UNIX-Editor vi. Mit den Pfeiltasten kann innerhalb des Editors horizontal und vertikal durch das Eingabefeld navigiert werden. Der ISPF-Editor hat zusätzlich weitere besondere Eigenschaften. Um einem Dokument neue Zeilen hinzuzufügen, wird in der linken Spalte die Zeilennummer mit dem Kommando „iX“ überschrieben, wobei X die Anzahl der neu zu erstellenden Zeilen definiert.

```
  Menu  Functions  Confirm  Utilities  Help
 ------------------------------------------------------------------------------
BROWSE            PRAK097.CICS.TEST                        Row 00001 of 00003
           Name     Prompt       Size   Created          Changed          ID
_________ BMSPROG                  14  2009/04/02  2009/04/02 14:59:22  PRAK097
_________ PROG097                  15  2009/04/02  2009/04/02 17:00:32  PRAK097
_________ START01                   6  2009/04/02  2009/04/02 16:44:53  PRAK097
          **End**
```

Abbildung 5.31.: Member-Liste vom Dataset List Utility

```
  Menu  Functions  Utilities  Help
 ------------------------------------------------------------------------------
EDIT      PRAK097.CICS.TEST                                 Row 00001 of 00003
   Name      Prompt          Size    Created          Changed          ID
. BMSPROG                      14   2009/04/02  2009/04/02 14:59:22  PRAK097
. PROG097                      15   2009/04/02  2009/04/02 17:00:32  PRAK097
. START01                       6   2009/04/02  2009/04/02 16:44:53  PRAK097
  **End**
```

Abbildung 5.32.: Member-Liste vom Entry Panel

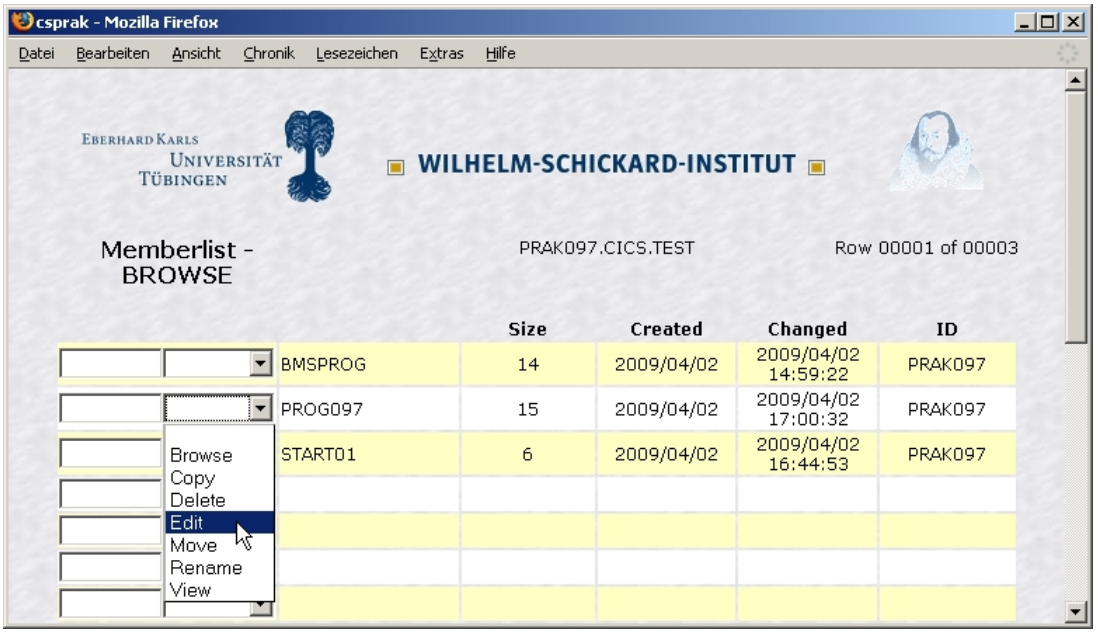

Abbildung 5.33.: Member-Liste in der HATS Web-Anwendung

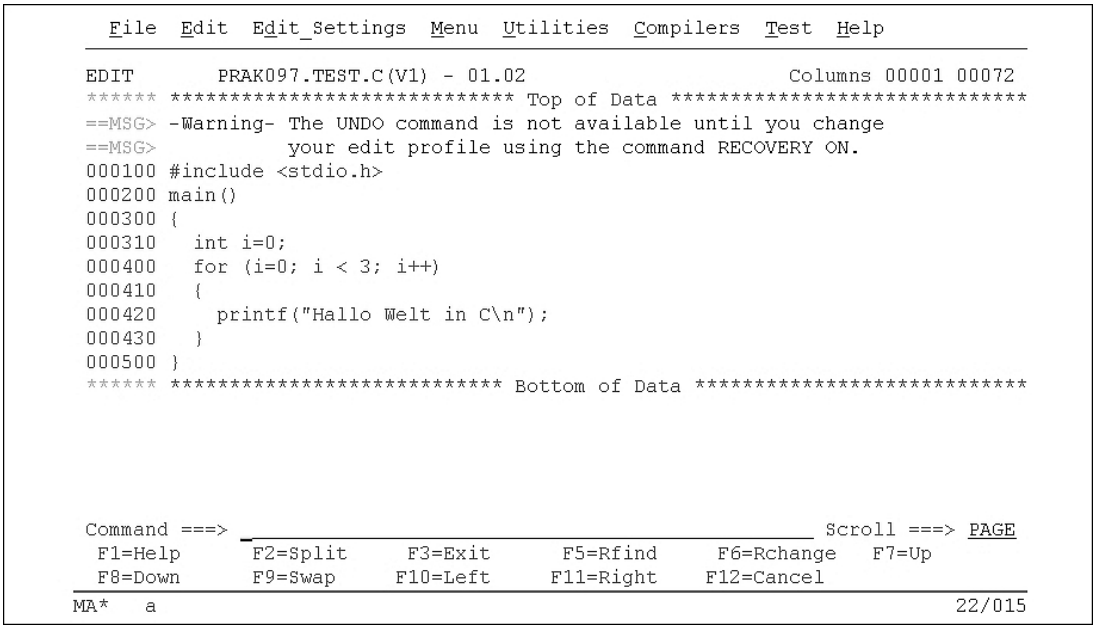

Abbildung 5.34.: ISPF Editor

Um Zeilen zu löschen wird analog das Kommando „dX“ verwendet. Um durch ein mehrseitiges Dokument zu blättern, werden die Funktionstasten PF7 und PF8 verwendet. Mit den Funktionstasten PF5 und PF6 kann innerhalb eines Dokuments nach einer Zeichenkette gesucht und diese eventuell ersetzt werden. Beim Verlassen des Editors mit der Taste PF3 wird das Dokument automatisch gespeichert.

Im ISPF-Editor können erstellte Jobs mit Hilfe von JCL-Skripten durch das Kommando „SUB“ zur Ausführung gebracht. Der Ablauf gliedert sich in mehrere Abschnitte. Zuerst wird die Meldung „Job Submitted“ angezeigt und der Job an das JES übergeben. Das Ende des Jobs wird dem Aufrufer durch Ausgabe der Statusmeldung „MAXCC=X“ auf einem neuen Screen quittiert, wobei im Fall „MAXCC=0“ die Abarbeitung erfolgreich abgelaufen ist. Andere Werte weisen auf diverse Fehler wie zum Beispiel einen Syntaxfehler im Code hin. Nähere Informationen über den Fehler erhält der Aufrufer dann meist aus der Log-Datei. Um zu überprüfen, ob eine Statusmeldung bereits vorliegt, muss der Benutzer den Screen der Host-Anwendung durch Drücken der Eingabetaste aktualisieren. Dies kann unter Umständen ein mehrmaliges Drücken der Eingabetaste zur Folge haben.

Um dem Benutzer in der HATS Web-Anwendung die für diesen Screen benötigten Funktionstasten über Links verfügbar zu machen, wurde die Screen-Anpassung *ISPFEditor.evnt* erstellt. Da der Screen des ISPF-Editors leider keine allgemeine Überschrift besitzt, musste das Erkennungskriterium für die Screen-Anpassung etwas komplexer gewählt werden. Als Kriterium wurde definiert, dass die Zeichenketten „Columns“ an Position (61,3), „Scroll ===>“ an Position (64,20) und „Command ===>“ an Position (2,22) auf einem Screen der Host-Anwendung erscheinen.

Abbildung 5.35.: ISPF-Editor in der HATS Web-Anwendung

Dies sind die einzigen Merkmale welche alle Screens des ISPF-Editors gemeinsam haben. Andere Zeichenketten sind jeweils inhaltsabhängig und können daher nicht verwendet werden.

Als Aktion wird von der Screen-Anpassung die Transformation *ISPFEditor.jsp* angewendet. Zusätzlich werden die später erwähnten globalen Variablen *subMessage1* und *subMessage2* mit einer leeren Zeichenkette initialisiert. In der Transformation, welche in Abbildung 5.35 zu sehen ist, wurde zuerst die Überschrift „ISPF Editor“ eingefügt. Darunter wurde der eigentliche Editor über die Field-Komponente und das Field-Widget eingebunden. Wichtig war hierbei, für das Widget die Monotype-Schriftart Courier zu definieren, da ansonsten die Einrückung eines Dokuments ungleichmäßig gewirkt hätte. Durch die Field-Komponente wird jede Zeile des Editors als einzeiliges HTML-Eingabefeld dargestellt. Ein mehrere Zeilen umfassender HTML-Eingabebereich (Textarea), der für den Anwender noch intuitiver und praktischer ist, wurde in der HATS Web-Anwendung nicht umgesetzt. Dies ist aufgrund der Tatsache, dass sich ein Member innerhalb des ISPF-Editors über mehrere Screens mit variabler Zeilenanzahl erstrecken kann, nur mit sehr komplexen Makros, Komponenten und Widgets möglich.

Zusätzlich wurde in der Transformation die Schaltfläche „Sub“ eingebaut, welche dem Benutzer das Absenden eines Jobs über die Maus ermöglicht. Ein Klick auf die Schaltfläche füllt die Zeichenkette „SUB“ automatisch in das Eingabefeld und bestätigt die Eingabe anschließend mit der Eingabetaste. Hierbei wird ebenfalls zwischen Eingabe und Absenden knapp eine Sekunde gewartet, wodurch der Benutzer das durch die Schaltfläche eingegebene Kommando sehen und erlernen kann.

Der vorher erwähnte umständliche Vorgang um an die Statusmeldung eines abgesendeten Jobs zu gelangen, wurde mit einer weiteren Screen-Anpassung *ISPFEditorJobSubmitted.evnt* durch Makros automatisiert. Nach dem Absenden eines Jobs erscheint dadurch in der HATS Web-Anwendung ohne Eingreifen des Benutzers unter dem Eingabefeld des ISPF-Editors die Statusmeldung des Jobs. Das Resultat ist in Abbildung 5.36 abgebildet. Aktiviert wird die Screen-Anpassung sobald als Erkennungskriterium die Zeichenketten „SUBMITTED“ im Bereich (1,21) bis (80,24) und „Columns“ beginnend an Position (61,3) auf einem Screen erkannt werden.

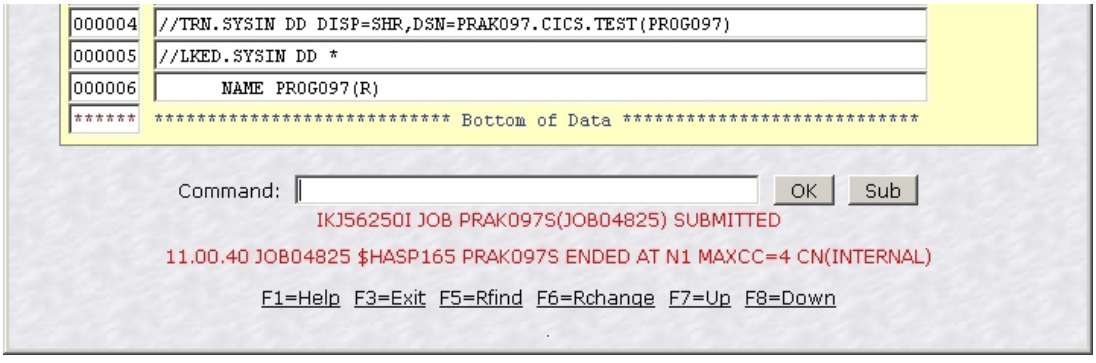

Abbildung 5.36.: Absenden eines Jobs im ISPF-Editor der HATS Web-Anwendung

Als zugehörige Aktion wird das erstellte Makro *IspfEditorSubCollectMessages.hma* ausgeführt. Der Aufbau des Makros ist in Abbildung 5.37 zu sehen. Es extrahiert den Screen-Bereich (2,21) bis (80,21) in die globale Variable *subMessage1*, wartet zwei Sekunden und emuliert in der Host-Anwendung anschließend einen Tastendruck der Eingabetaste. Falls noch keine Statusmeldung des Jobs vorliegt, wird drei Sekunden lang gewartet und erneut ein Tastendruck der Eingabetaste emuliert. Spätestens nun erscheint die Statusmeldung des Jobs im Screen-Bereich (2,1) bis (80,1), dessen Inhalt vom Makro in die globale Variable *subMessage2* extrahiert wird. Die Zeitspanne wurde in der Summe auf insgesamt fünf Sekunden gesetzt, da die Ausführung eines Jobs von der Auslastung des Systems abhängig ist und somit nicht genau hervorgesehen werden kann. Für die im Praktikum ausgeführten relativ kleinen Jobs, dürfte diese gewählte Zeitspanne ausreichend sein. Das Makro emuliert in der Host-Anwendung abschließend erneut einen Tastendruck der Eingabetaste und kehrt somit zurück zum ISPF-Editor, in welchem in der HATS Web-Anwendung unter dem Eingabefeld die globalen Variablen mit den extrahierten Statusmeldungen des Jobs ausgegeben werden. Die von der Screen-Anpassung *ISPFEditor.evnt* durchgeführte Initialisierung der globalen Variablen mit einer leeren Zeichenkette ist wichtig, da die Statusmeldung des letzten abgearbeiteten Jobs ansonsten permanent im ISPF-Editor angezeigt wird.

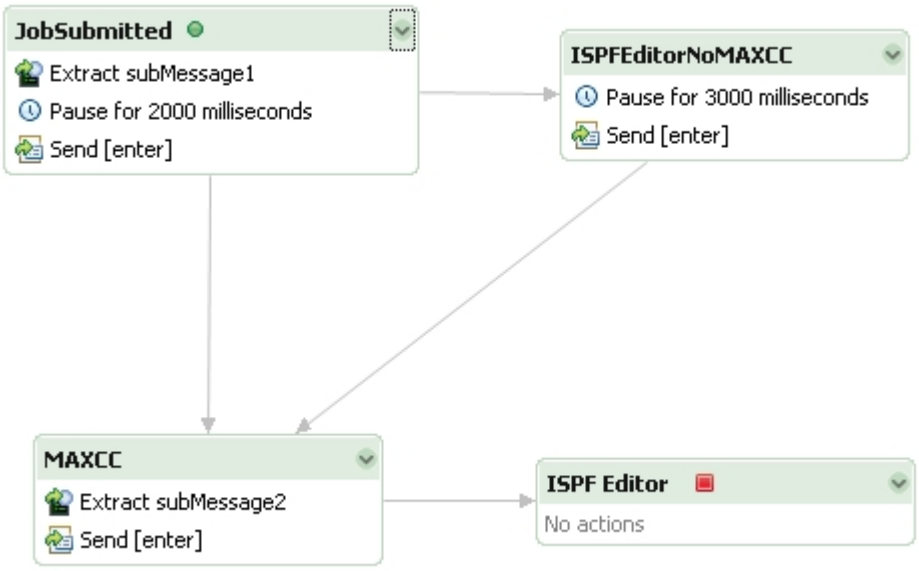

Abbildung 5.37.: Makro zum Extrahieren der Statusmeldungen eines abgearbeiteten Jobs

Sendet der Benutzer als Job einen leeren Member ab, wird die Fehlermeldung „Job not submitted“ beginnend an Position (63,3) auf dem Screen angezeigt. Da dabei die vom Benutzer eingegebene Zeichenkette „SUB“ nicht automatisch aus dem Eingabefeld verschwindet und der Screen blockiert, bis die Zeichenkette manuell mit Leerzeichen überschrieben wurde, wird dies als Sonderfall durch die dafür erstellte Screen-Anpassung *ISPFEditorJobNotSubmitted.evnt* behandelt. Als Erkennungskriterium für diese Screen-Anpassung wurde die vorher genannte Fehlermeldung

beginnend an Position (63,3) gewählt. Als Aktion wird diese Fehlermeldung in die globale Variable *subMessage1* extrahiert. Durch eine weitere Aktion vom Typ „Insert Data“ überschreibt die Screen-Anpassung den Inhalt des Eingabefeldes an Position (15,22) auf dem Screen mit Leerzeichen. Anschließend wird der Screen über die Transformation *ISPFEditor.jsp* dargestellt, damit der Benutzer ohne Einschränkung weiterarbeiten kann.

5.1.4.7. ISPF-Hilfe

Über die Funktionstaste PF1 kann der Benutzer in ISPF zu jedem Screen Hilfe erhalten. Mit den Funktionstasten PF7, PF8, PF10 und PF11 kann hierbei durch die Hilfe-Screens navigiert werden. Die Hilfe-Screens unterscheiden sich von normalen ISPF-Screens dadurch, dass die Menüleiste in der ersten Zeile nicht vorhanden ist. Um dem Benutzer die zur Navigation benötigten Funktionstasten über die Maus zugänglich zu machen, wurde die neue Screen-Anpassung *ISPFHelp.evnt* erstellt. In allen Hilfe-Screens wird beginnend an Position (1,1) die Zeichenkette „Tutorial“ angezeigt, was sich dadurch als Erkennungskriterium für die Screen-Anpassung eignet. Lediglich die Groß- und Kleinschreibung der Zeichenkette ändert sich in einzelnen Screens. Folglich ist zu beachten, dass für das Erkennungskriterium nicht zusätzlich die Option „Case Sensitive“, welche zwischen Groß- und Kleinschreibung streng unterscheidet, gewählt wird.

Als Aktion stellt die Screen-Anpassung den Screen über die erstellte Transformation *ISPFHelp.jsp* dar. Dabei wird die Überschrift mit der Text-Komponente über das Label Widget und der Inhalt des Screens mit Hilfe der Field-Komponente über das Field-Widget transformiert. Die benötigten Funktionstasten wurden als Links verfügbar gemacht.

5.1.4.8. ISPF-Abmeldung

Um sich von ISPF abzumelden, wird im ISPF Primary Option Menü die Funktionstaste PF3 verwendet oder das Kommando „X“ eingegeben. Anschließend erscheint ein Screen mit dem Titel „Specify Disposition of Log Data Set“, welcher in Abbildung 5.38 gezeigt wird. In diesem Screen legt der Benutzer fest, was mit den Log- und List-Datasets geschieht, die während der Sitzung angelegt wurden [TEUF, Kapitel 4.9]. Über die vier Menüpunkte kann der Benutzer das Log-Dataset ausdrucken, löschen, beibehalten oder bei der nächsten Anmeldung neu anlegen lassen. Darüber hinaus können noch Angaben über den zu verwendenden Drucker gemacht werden. Im Client/Server-Praktikum werden Log-Datasets nicht gedruckt. Daher wurde dem Benutzer dieses Eingabefeld in der HATS Web-Anwendung erspart.

Für diesen Screen wurde die Screen-Anpassung *SpecifyDispositionOfLogDataSet.evnt* angelegt, welche aktiv wird sobald als Erkennungskriterium die Zeichenkette „Specify Disposition of Log Data“ an Position (23,1) auf einem Screen erkannt wird. Als Aktion stellt die Screen-Anpassung den Screen über die Transformation *SpecifyDispositionOfLogDataSet.jsp* dar, welche in Abbil-

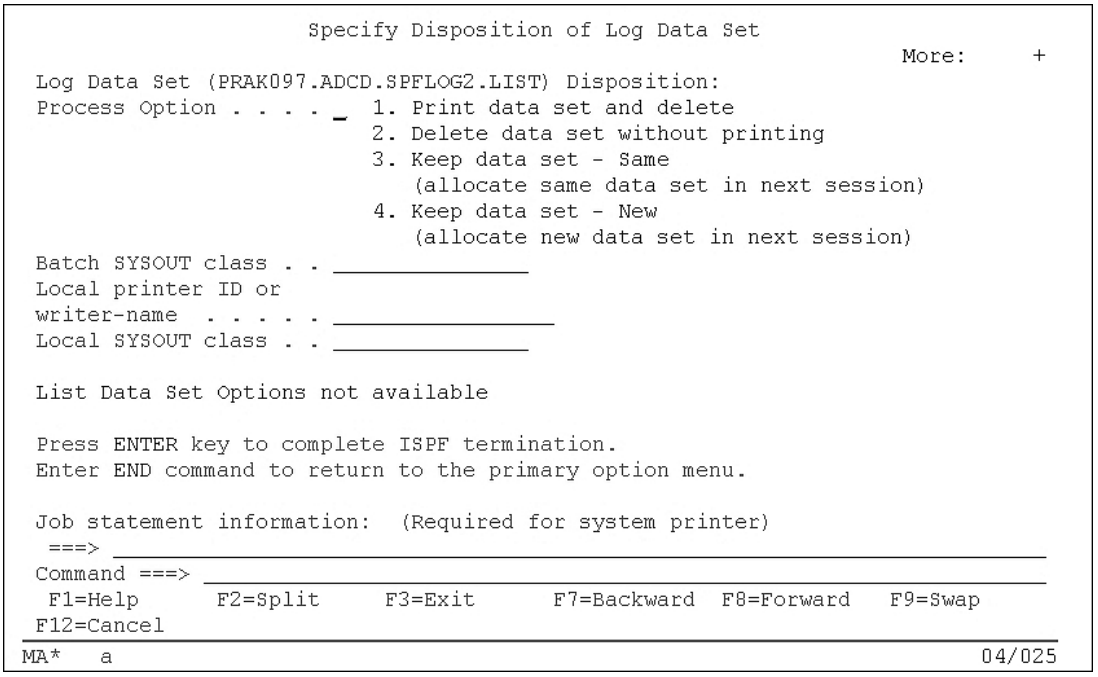

Abbildung 5.38.: Specify Disposition of Log-Dataset

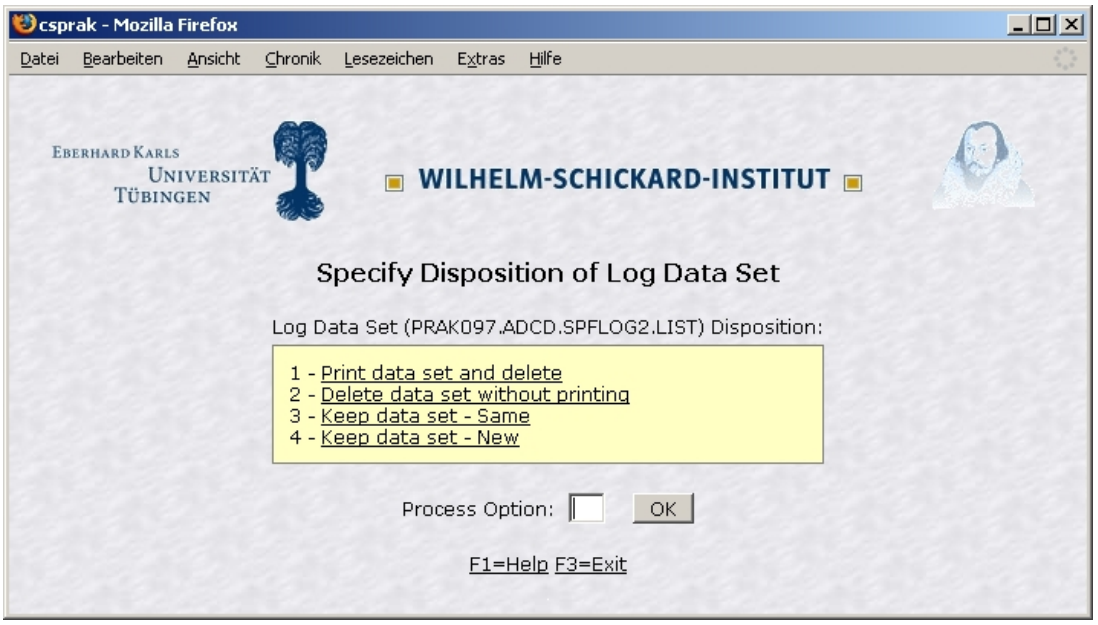

Abbildung 5.39.: Specify Disposition of Log-Dataset in der HATS Web-Anwendung

dung 5.39 gezeigt wird. Die vier Menüpunkte werden über die erstellte OptionList-Komponente mit dem OptionList-Widget dargestellt. Als zur Komponente gehörendes Eingabefeld wurde das Eingabefeld an Position (25,4) des Screens definiert. Dadurch bekommt der Benutzer die Möglichkeit, die Menüpunkte analog zum ISPF Primary Option Menü durch Links mit der Maus auszuwählen.

Das Eingabefeld wird dem Benutzer zusätzlich über die Input-Field-Komponente mit dem Text-Input-Widget angezeigt, wodurch der Benutzer die alternative Möglichkeit bekommt, den gewünschten Menüpunkt manuell einzugeben.

5.1.4.9. ISPF Standardtransformation

ISPF besteht aus zahlreichen weiteren Screens, wobei der Großteil dieser Screens für das Client/-Server-Praktikum nicht benötigt wird. Um diese ISPF Screens in der HATS Web-Anwendung jedoch trotzdem verfügbar zu machen, wurde für diese Screens die Screen-Anpassung *Default-Renderings/DefaultISPFRendering.evnt* erstellt. Somit wird gewährleistet, dass der Benutzer in der Web-Anwendung alle Inhalte der Host-Anwendung sieht und damit interagieren kann. Der Nachteil hierbei ist jedoch die primitive Umsetzung der Screens, die dem Benutzer keine automatisierten Vorgänge oder Navigationshilfen bietet. Ebenso kann es vorkommen, dass der komplette Host-Screen in der HATS Web-Anwendung unschönerweise durch viele HTML-Eingabefeldern dargestellt wird.

Als Erkennungskriterium wurde für dieses Screen-Anpassung definiert, dass sich innerhalb des Screen-Bereichs von Position (1,24) bis Position (80,24) an beliebiger Stelle die Zeichenketten „F1=Help“ und „F3=“ befinden. Dies ist das einzige Merkmal, welches alle ISPF-Screens gemeinsam haben. Da dieses Kriterium jedoch auch auf die Screens zutrifft, für die bereits Screen-Anpassungen angelegt wurden, wurde die Priorität niedriger als die Prioritäten der bereits erstellten Screen-Anpassungen gelegt.

Die Screen-Anpassung transformiert als Aktion den Screen mit der Transformation *Default-ISPFRendering.jsp*. Die Transformation stellt den Inhalt des Screens über eine Field-Komponente mit dem Field-Widget dar. Zusätzlich wird das Eingabefeld über die Input-Field-Komponente mit dem Text-Input-Widget dargestellt. Hinter dem Eingabefeld wurde eine Schaltfläche eingefügt, welche innerhalb der Host-Anwendung ein Drücken der Eingabetaste emuliert. Ebenso wurden für jeden Screen die Funktionstasten PF1 und PF3 als Links verfügbar gemacht.

Mit den im Abschnitt 5.1.4 erwähnten Screen-Anpassungen kann die Host-Anwendung ISPF nun komplett mit der HATS Web-Anwendung bedient werden.

5.1.5. CICS Umsetzung

Alle für die Umsetzung der Host-Anwendung CICS erstellten Screen-Anpassungen befinden sich innerhalb des HATS-Projekts im Verzeichnis *Screen Customizations/CICS*.

5.1.5.1. CICS-Anmeldung

Durch das Kommando „L CICS“ gelangt der Benutzer aus dem z/OS Willkommen-Screen zur CICS-Anmeldung. Es erscheint der in Abbildung 5.40 gezeigte Screen, in welchem der Benutzer seine Benutzerkennung, sein Passwort, eine Gruppenkennung sowie eine Sprache eingeben muss. Da im Client/Server-Praktikum die Felder für die Gruppenkennung und für die Sprache nicht relevant sind, werden diese Felder dem Benutzer in der HATS Web-Anwendung nicht angezeigt. Dazu wurde die Screen-Anpassung *SignonToCICS.evnt* mit der Transformation *SignonToCICS.jsp* erstellt, welche dem Benutzer lediglich ein Eingabefeld für die Benutzerkennung und für das Passwort zeigt (vgl. Abbildung 5.41). Die Screen-Anpassung wird aktiv, sobald als Erkennungskriterum die Zeichenketten „Signon to CICS“ beginnend an Position (29,1) und „WELCOME TO CICS“ beginnend an Position (2,3) auf einem Screen erscheinen.

In der Transformation wird das Eingabefeld für die Benutzerkennung und für das Passwort über die Input-Field-Komponenten mit dem Text-Input-Widget dargestellt. Zusätzlich wurde die Schaltfläche „Login“ eingefügt, welche in der Host-Anwendung ein Drücken der Eingabetaste emuliert und somit die Anmeldung veranlasst. Falls der Anmeldevorgang aufgrund einer ungültigen Benutzerkennung oder eines falschen Passworts fehlschlägt, erhält der Benutzer eine Fehlermeldung. Diese Fehlermeldung, welche standardmäßig in Zeile 23 des Screen erscheint, wird mit einer Text-Komponente über das Label-Widget angezeigt. Die Fehlermeldung wurde in der Transformation in roter Farbe unter den Eingabefeldern platziert, so dass sie für den Benutzer sofort ersichtlich ist.

Falls ein Administrator für einen Benutzer die Option gesetzt hat, dass dieser Benutzer bei der nächsten Anmeldung sein Passwort ändern soll, wird dem Benutzer in diesem Fall bei der Anmeldung ein zusätzliches Feld für das neue Passwort angezeigt. Um diesen Spezialfall zu berücksichtigen, wurde die Screen-Anpassung *SignonToCics_NewPassword.evnt* erstellt. Als Aktion zeigt die Screen-Anpassung den Screen über die Transformation *SignonToCics_NewPassword.jsp* an. Die Transformation ist analog zur Transformation *SignonToCICS.jsp* aufgebaut und zeigt zusätzlich das Feld für das neue Passwort über die Input-Field-Komponente mit dem Text-Input-Widget an.

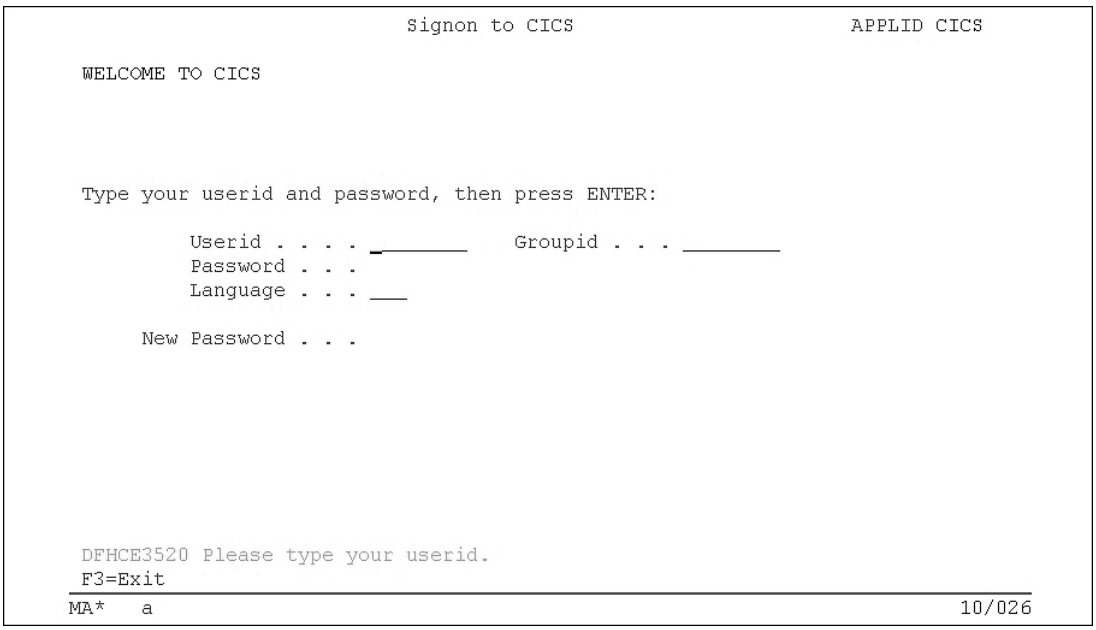

Abbildung 5.40.: CICS-Anmeldung

Abbildung 5.41.: CICS-Anmeldung in der HATS Web-Anwendung

5.1.5.2. CICS

Nach der erfolgreichen Anmeldung bei CICS wird dem Benutzer der ungewöhnliche Screen in Abbildung 5.42 angezeigt. Ein unerfahrener Anwender wird von diesem Screen irritiert, da für ihn auf den ersten Blick kein Eingabefeld erkennbar ist. Jedoch kann der komplette Screen als Eingabefeld genutzt werden. Dieser Screen stellt auch einen Fall dar, in dem die Standardtransformation für jede Zeile ein unschönes HTML-Eingabefeld generiert. Abhilfe wurde mit der Screen-Anpassung *CICSSignOnComplete.evnt* und der zugehörenden Transformation *CICSSignOnComplete.jsp* geschaffen, welche dem Benutzer lediglich ein einziges Eingabefeld und in einem getrennten Bereich eine CICS-Statusmeldung anzeigt (vgl. Abbildung 5.45). Als Statusmeldung wird der Screen-Bereich bestehend aus Zeile zwei, Zeile 23 und Zeile 24 als Text-Komponente mit dem Label-Widget in roter Farbe angezeigt. Als Eingabefeld wurde die komplette erste Zeile gewählt und über die Input-Field-Komponente mit dem Text-Input-Widget angezeigt. Für das Erkennungskriterium der Screen-Anpassung existieren aufgrund des fast leeren Screens nicht viele Möglichkeiten. Da nach der Anmeldung am unteren Ende des Screens immer eine CICS-Statusmeldung erscheint, welche standardmäßig das Präfix *DFH* besitzt [CICS1, Kapitel 1], wurde als Erkennungskriterium die Zeichenkette „DFH“ innerhalb des von der Position (1,22) bis zur Position (80,24) aufgespannten Screen-Bereichs definiert.

Durch Eingabe einer Transaktions-ID in das Eingabefeld, wird eine CICS-Transaktion aufgerufen. Wird die Transaktion anschließend durch Drücken der Funktionstaste PF3 beendet, kehrt CICS zum Haupt-Screen zurück (vgl. Abbildung 5.43). Dieser unterscheidet sich jedoch vom direkt nach der Anmeldung angezeigten Screen, weshalb eine neue Screen-Anpassung erstellt werden musste. Als Erkennungskriterium wurde die Zeichenkette „STATUS:“ an Position (2,2) auf einem Screen definiert. Als Aktion zeigt die Screen-Anpassung den Screen über die bereits erstellte Transformation *CICSSignOnComplete.jsp* an.

Im Client/Server-Praktikum werden mit Hilfe der Transaktion CEDA (Abbildung 5.44) vom Benutzer einfache Transaktionen erstellt und eingerichtet. Wird dieser Screen über die Standardtransformation von HATS mit der Field-Komponente über das Field-Widget dargestellt, wird die Zeile mit den Hinweisen für die Funktionstasten teilweise unpassend als HTML-Eingabefeld dargestellt. Aus diesem Grund wurde für diese Transaktion die Screen-Anpassung *CEDA.evnt* erstellt, welche aktiv wird, sobald als Erkennungskriterium die Zeichenkette „PF“ in Zeile 24 auf einem Screen erscheint. Diese Zeichenkette befindet sich in allen innerhalb der Transaktion CEDA auftretenden Screens, weshalb sie sich als Erkennungskriterium eignet.

Als Aktion zeigt die Screen-Anpassung den Screen über die Transformation *CEDA.jsp* an, welche den Screen, bis auf Zeile eins und Zeile 24, über die Field-Komponente mit dem Field-Widget darstellt. Das Eingabefeld in Zeile eins wird durch die Input-Field-Komponente mit dem Text-Input-Widget angezeigt. Dabei wurde zusätzlich eine Schaltfläche „OK“ eingefügt, über welche der Benutzer innerhalb der Host-Anwendung ein Drücken der Eingabetaste emulieren kann. Zeile 24 des Screens wird in der Transformation durch Links dargestellt, über welche jeweils das Drücken einer Funktionstaste emuliert werden kann.

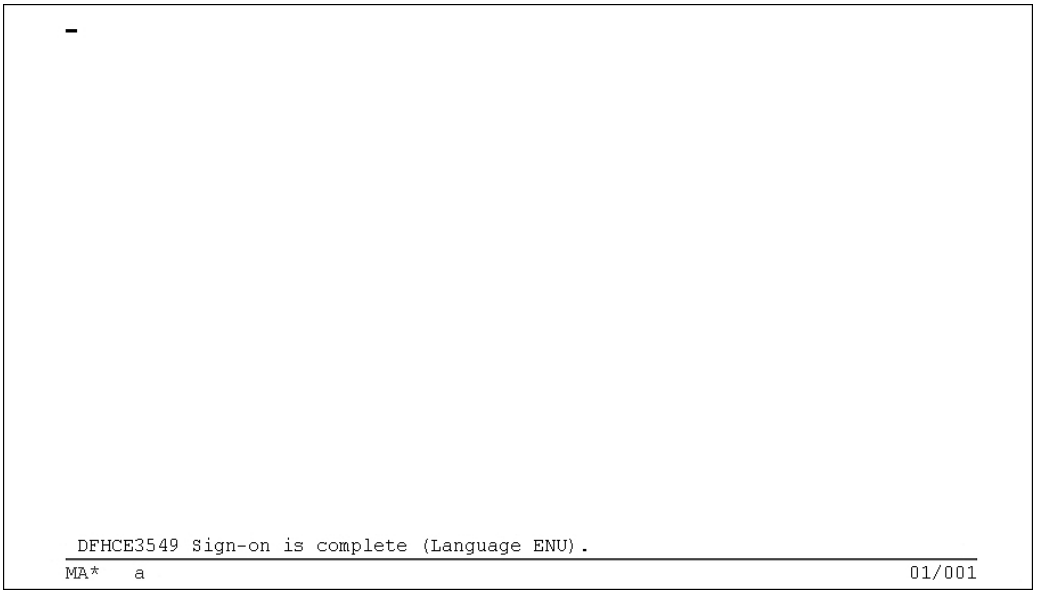

Abbildung 5.42.: CICS nach der Anmeldung

Abbildung 5.43.: CICS nach Ausführung einer Transaktion

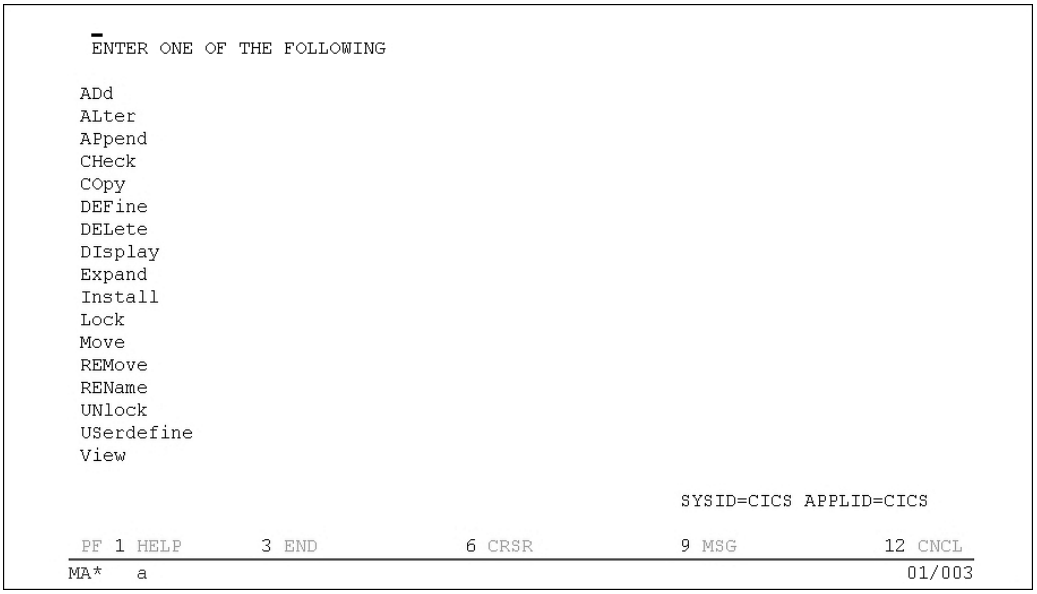

Abbildung 5.44.: CICS-Transaktion CEDA

Abbildung 5.45.: CICS in der HATS Web-Anwendung

5.1.5.3. CICS-Abmeldung

Um sich von CICS abzumelden, verwendet der Benutzer eines der folgenden Kommandos: „CESF“, „CESF LOGOFF“ oder „CESF GOODNIGHT“ [CICS2]. Die Abmeldung bestätigt CICS mit der Statusmeldung „DFHCE3590 Sign-Off is complete“ und kehrt anschließend zum Anmelde-Screen von CICS zurück. Da es jedoch mit dieser CICS-Konfiguration nicht möglich ist zum Willkommen-Screen von z/OS zurückzukehren, muss der Benutzer bei jedem Wechsel manuell die Verbindung mit dem Host unterbrechen und neu aufbauen. Folglich muss an dieser Stelle von der HATS Web-Anwendung nach einer erfolgreichen-CICS Abmeldung automatisch die Verbindung zum Host getrennt werden. Dies wurde mit der Screen-Anpassung *CICSSignOffIsComplete.evnt* realisiert. Als Erkennungskriterium wurde die gerade erwähnte Statusmeldung gewählt. Als zugehörige Aktion wird die HATS-Aktion „Disconnect“ ausgeführt, welche automatisch die Verbindung zu dem Host trennt. Um zum z/OS Willkommen-Screen zu gelangen, bietet die HATS Web-Anwendung anschließend über eine Schaltfläche „Restart“ die Möglichkeit, die Verbindung erneut herzustellen.

5.1.6. USS Umsetzung

Alle nachfolgend erwähnten USS Screen-Anpassungen befinden sich im Verzeichnis *Screen Customizations/USS* des HATS-Projekts.

5.1.6.1. USS-Shell

Die USS-Shell erreicht man in TSO durch Eingabe des Befehls „OMVS“. Das TSO-Kommando kann alternativ in ISPF über den Menüpunkt „Command“ des Primary Option Menüs abgesetzt werden. Die USS-Shell wird im Praktikum in Tutorial 7 benötigt, um über die Konsole ein Java Servlet anzulegen und dieses auf dem WAS auszuführen. Dieses Tutorial ist zum jetzigen Zeitpunkt auf dem Tübinger Mainframe noch nicht lauffähig. Die USS-Shell wurde in der HATS Web-Anwendung deshalb vorausschauend für zukünftige Tutorials umgesetzt (vgl. Abbildung 5.46). Das Zeichen „$“ gibt die jeweils aktuelle Position für die Ausführung von UNIX-Kommandos an. Die Eingabe geschieht jedoch über das Eingabefeld am unteren Ende des Screens, welche den Befehl automatisch an der aktuellen Position einfügt, ausführt und anschließend das Ergebnis in den nachfolgenden Zeilen präsentiert.

Umgesetzt wurde die USS-Shell in der HATS Web-Anwendung durch die Screen-Anpassung *OMVS.evnt*. Diese wird aktiv, sobald das für USS typische Zeichen „$“ auf einem Screen erkannt wird und in Zeile 23 und 24 die Hinweise auf Funktionstasten erscheinen, welche in Abbildung 5.46 zu sehen sind. Als Aktion stellt die Screen-Anpassung den Screen über die in Abbildung 5.47 gezeigte Transformation *OMVS.jsp* dar. Die ersten 20 Zeilen werden mit Hilfe der Field-Komponente und dem Field-Widget dargestellt. Das Eingabefeld in Zeile 21 wird über die Input-Field-Komponente mit dem Text-Input Widget angezeigt. Die Hinweise auf Funktionstasten werden wieder mit der Function-Key-Komponente über das Link-Widget transformiert.

Bei der Umsetzung der USS-Shell kam es zu folgender Schwierigkeit. Nachdem ein UNIX-Kommando abgesetzt wurde, wird es an die aktuelle Position innerhalb der USS-Shell geschrieben. Durch das Absenden des Kommandos bekommt die HATS Web-Anwendung die Aktualisierung des Screens mit und aktualisiert somit ebenfalls die Transformation mit dem neuen Inhalt. Anschließend vergeht jedoch, abhängig vom jeweiligen Befehl, eine gewisse Zeitspanne bis die Ausführung abgeschlossen ist und das Ergebnis auf dem Host-Screen angezeigt wird. Um diese asynchronen Aktualisierungen der USS-Shell in der HATS Web-Anwendung anzuzeigen, musste das in Abschnitt 3.7 erwähnte Java Applet Asynchronous Update aktiviert werden.

5.1.6.2. USS-Manpages

Manpages sind wichtige Hilfe- und Dokumentationsseiten für UNIX-Konzepte und Befehle. Durch Eingabe des Kommandos „man <*thema*>“ erhält der Benutzer umfangreiche, meist einige Seiten

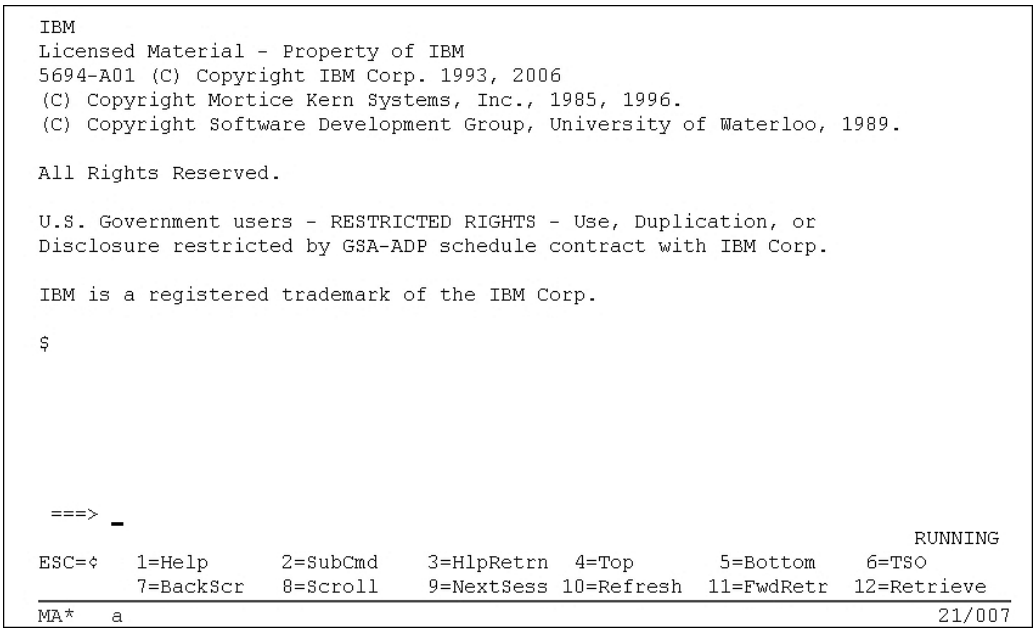

Abbildung 5.46.: UNIX System Services

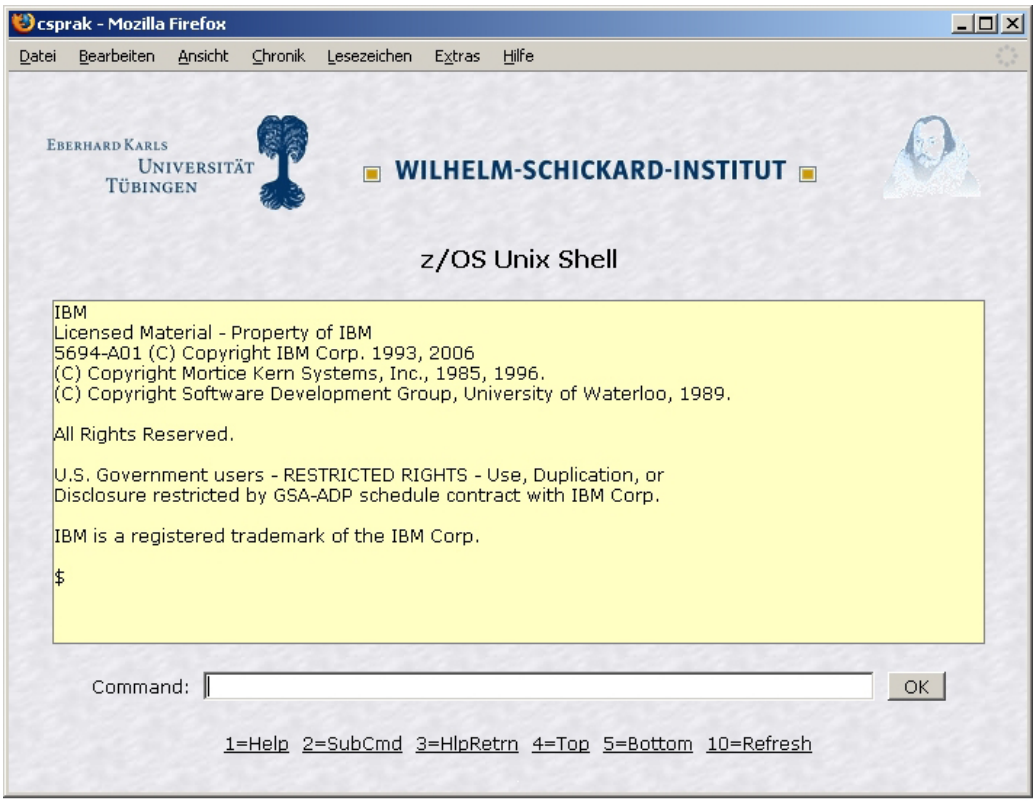

Abbildung 5.47.: UNIX System Services in der HATS Web-Anwendung

umfassende Informationen zu dem gewünschten Thema (vgl. Abbildung 5.48). Mit den Funktionstasten PF7 und PF8 kann der Benutzer durch mehrseitige Dokumentationsseiten navigieren.

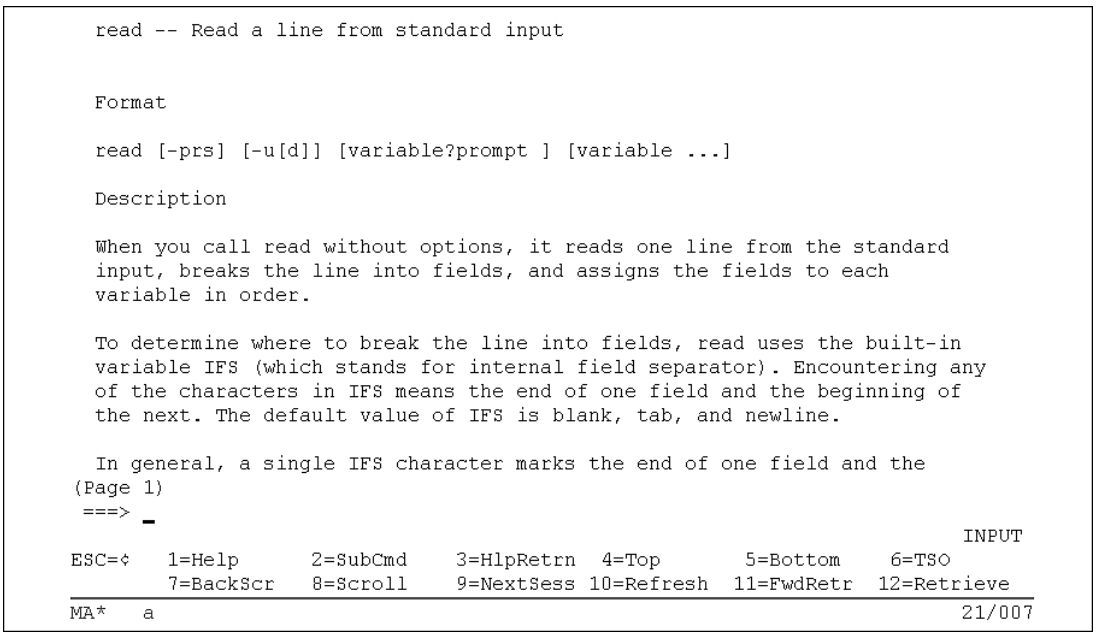

Abbildung 5.48.: USS-Manpage

Um in der HATS Web-Anwendung dem Benutzer diese umständliche Navigation zu ersparen, wurde die Screen-Anpassung *ManPageStart.evnt* und das Makro *ShellCollectManPage.hma* erstellt, welches bei Bedarf Manpages Seite für Seite automatisch in die angelegte globale Variable *manpage* extrahiert und diese dem Benutzer anschließend zusammenhängend auf einer einzigen HTML-Seite anzeigt.

Die Screen-Anpassung wird aktiv, sobald als Erkennungskriterium auf einem Screen in einer Zeile die Zeichenkette „Format“ und in einer anderen Zeile die Zeichenkette „Description“, jeweils oben und unten von Leerzeilen umgeben, erscheint. Dies ist eines der wenigen Merkmale die alle Startseiten einer USS-Manpage gemeinsam haben. Als Aktion führt die Screen-Anpassung das in Abbildung 5.49 dargestellte Makro aus. Das Makro extrahiert den kompletten Screen-Inhalt als Zeichenkette in die globale Variable *manpage*. Wichtig hierbei ist, dass die globale Variable nicht überschrieben wird, sondern die Inhalte konkateniert werden. Damit anschließend der nächste Screen angezeigt wird, emuliert das Makro in der Host-Anwendung ein Drücken der Eingabetaste. Dieser Vorgang wird in einer Schleife solange wiederholt, bis am unteren Ende eines Screens die Zeichenkette „$“ angezeigt wird, welche das Ende der Manpage markiert. Das Zeichen „$“ ist das einzige Zeichen, mit dem das Ende einer USS-Manpage bestimmbar ist. Das Makro extrahiert den letzten Screen in die globale Variable *manpage* und terminiert.

Damit die Manpage nach der Extraktion angezeigt werden kann, ist eine Screen-Anpassung *ManPageEnd.evnt* nötig. Die zusätzlich Screen-Anpassung ist notwendig, da HATS für eine Screen-

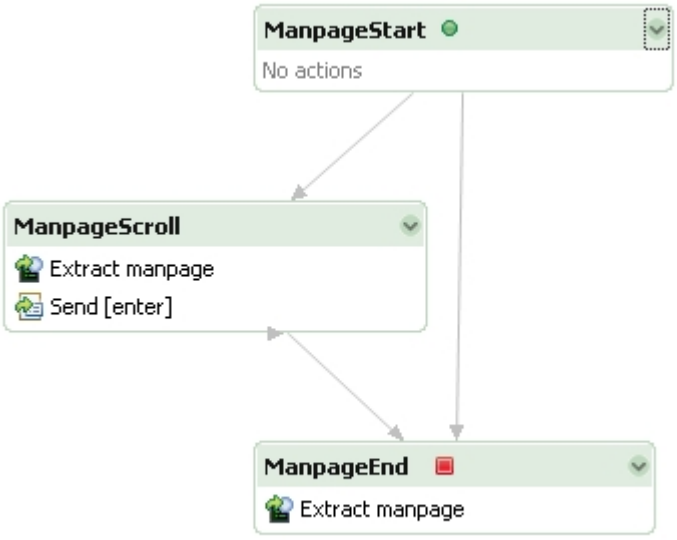

Abbildung 5.49.: Makro zum Extrahieren einer USS-Manpage in eine globale Variable

Anpassung vorschreibt, dass eine Transformation vor allen anderen HATS-Aktionen ausgeführt werden muss. Dadurch ist es mit nur einer Screen-Anpassung nicht möglich, zuerst eine globale Variable mit dem Inhalt des aktuellen Screens zu füllen und sie anschließend innerhalb einer Transformation auszugeben. Das Erkennungskriterium für die Screen-Anpassung wurde in der HATS-Entwicklungsumgebung, wie in in Abbildung 5.50 gezeigt, festgelegt.

Über dem Zeichen „$“ müssen vier Leerzeichen und darunter, um eine Position nach rechts versetzt, das Zeichen „=“ auftauchen. Dabei ist zu beachten, dass die Priorität der Screen-Anpassung höher sein muss als die der Screen-Anpassung *OMVS.evnt*, da diese ein gleiches Erkennungskriterium besitzt und sonst aktiv werden würde. Die Screen-Anpassung stellt als HATS-Aktion den Screen über die Transformation *ManPageEnd.jsp* dar und setzt durch eine weitere HATS-Aktion den Inhalt der globalen Variablen *manpage* zurück. Das Zurücksetzen ist wichtig, da sonst bei einem erneuten Aufruf einer Manpage aufgrund der Konkatenation des Inhalts in der globalen Variable die vorherig aufgerufene Manpage zusätzlich mit ausgegeben wird.

Die in der HATS Web-Anwendung durch die Transformation *ManPageEnd.jsp* dargestellte Manpage ist in Abbildung 5.51 zu sehen. Um in der Transformation die Manpage auszugeben, wurde die globale Variable *manpage* in der Entwicklungsumgebung über den Menüpunkt „HATS Tools/-Insert Global Variable“ mit der Option „Display Global Variable as static text“ eingefügt.

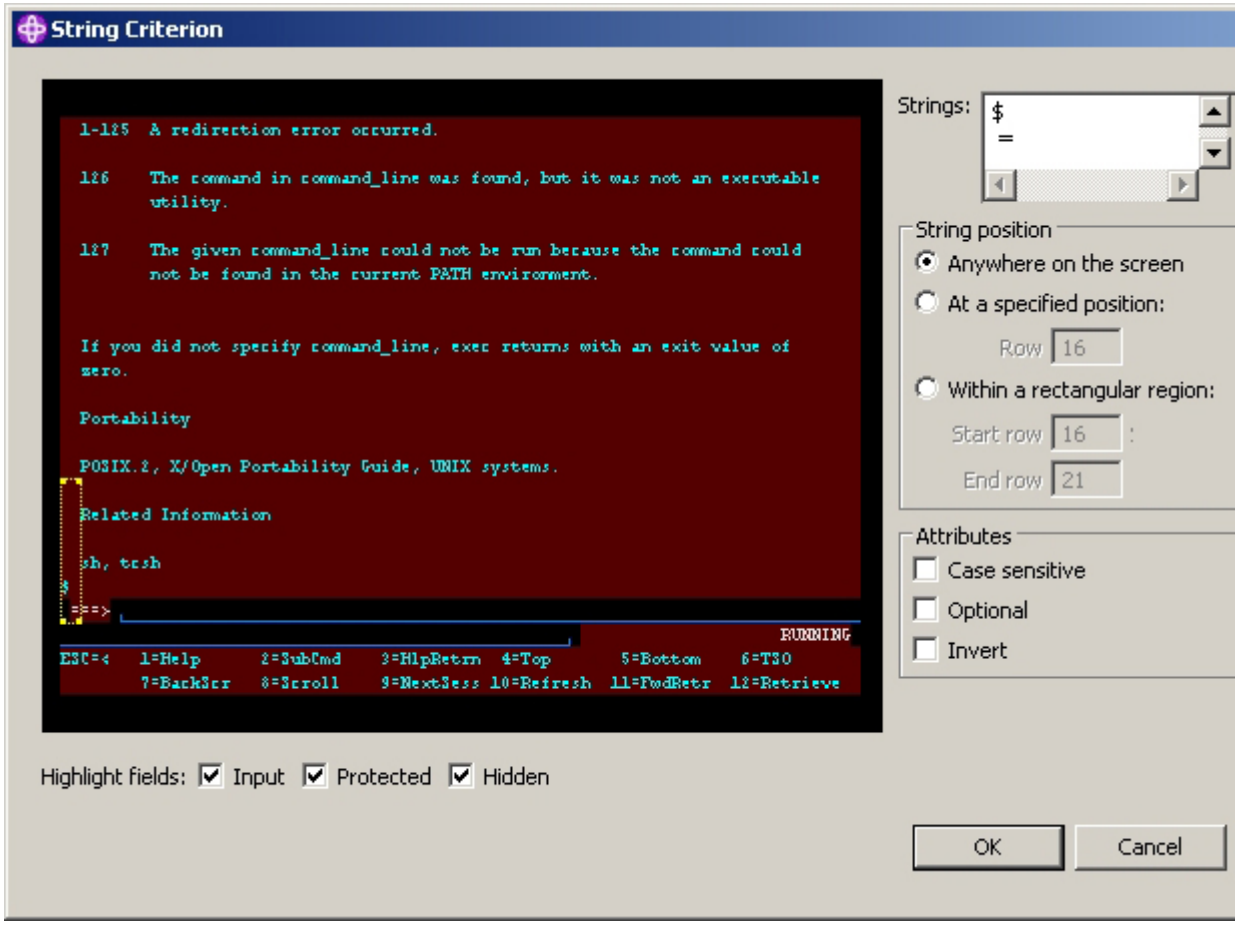

Abbildung 5.50.: Erkennungskriterium für das Ende einer Manpage

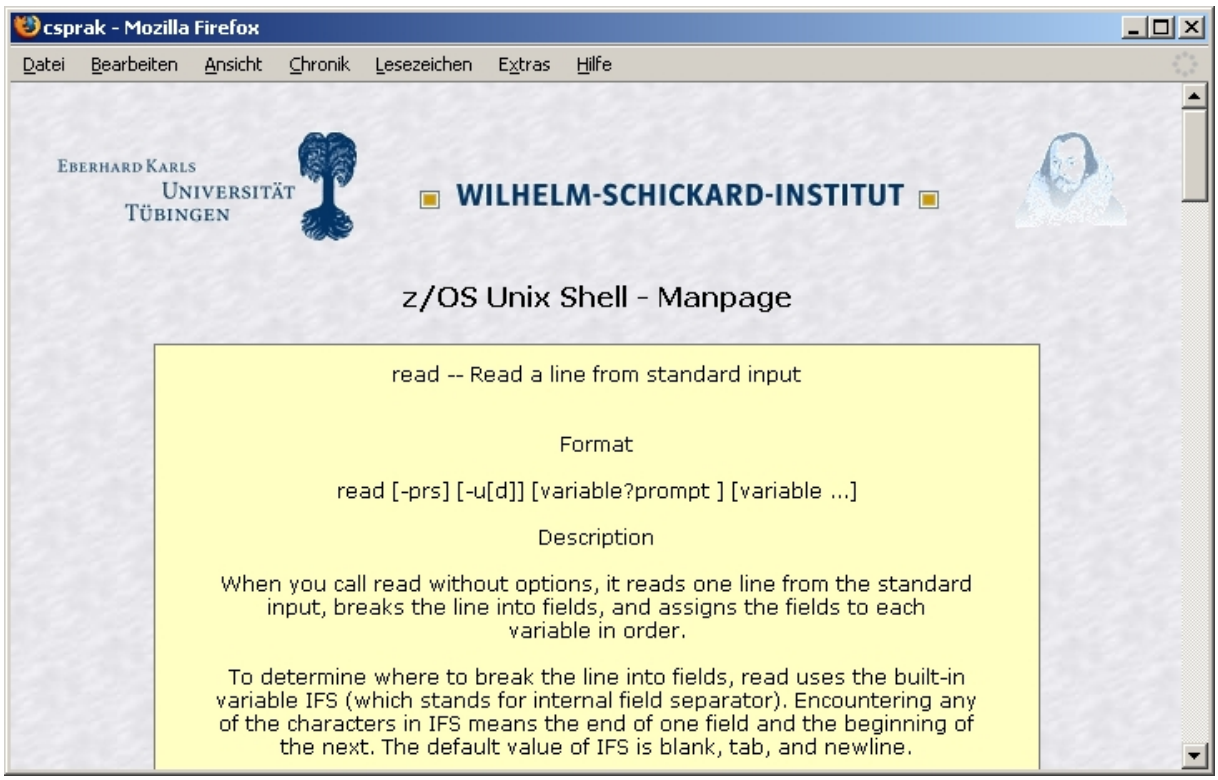

Abbildung 5.51.: USS-Manpage in der HATS Web-Anwendung

5.1.6.3. USS-Abmeldung

Wenn der Benutzer die USS-Shell beenden möchte, verwendet er das Kommando „EXIT“. Anstatt dass sich die USS-Shell nun sofort beendet, wird der Benutzer aufgefordert die Eingabetaste zu drücken (vgl. Abbildung 5.52). Dieser Schritt wird bei der Umsetzung mit der HATS Web-Anwendung mit Hilfe eines erstellten Makros automatisch erledigt. Dazu wurde das in Abbildung 5.53 dargestellte Makro *ExitOMVS.hma* erstellt, welches ein Drücken der Eingabetaste in der Host-Anwendung emuliert. Da der Benutzer nach der Abmeldung, je nachdem aus welchem Screen er USS aufgerufen hat, entweder auf einem TSO- oder einem ISPF-Screen landet, mussten für das Makro zwei verschiedene Exit-Screens definiert werden.

```
$
$
$ exit



 >>>> FSUM2331 The session has ended.  Press <Enter> to end OMVS.
                                                                    INPUT
ESC=    1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
        7=BackScr   8=Scroll    9=NextSess 10=Refresh   11=         12=
MA*   a                                                            21/002
```

Abbildung 5.52.: USS-Abmeldung

Das Makro wird durch die Screen-Anpassung *OMVS_ exit.evnt* ausgeführt. Als Erkennungskriterium dient die Zeichenkette „»»FSUM2331 The session has ended. Press <Enter> to end OMVS“ in Zeile 21 eines Screens.

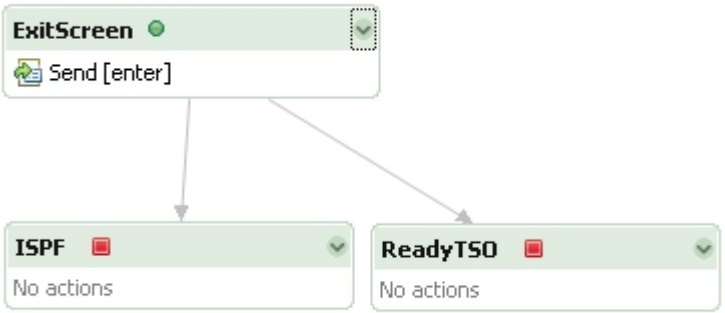

Abbildung 5.53.: Makro zum Abmelden von USS

5.1.7. DB2 Umsetzung

Im Client/Server-Praktikum wird das DB2-System für das vierte Tutorial verwendet. Dabei legt der Benutzer eine Storage Group, einen Table Space, eine Datenbank und eine Tabelle mit Hilfe der SPUFI-Funktion über SQL-Befehle an. Da in dem Tutorial lediglich das DB2 Primary Option Menü und SPUFI verwendet werden, wurden folglich nur hierfür Screen-Anpassungen erstellt. Alle nicht in das vierte Tutorial involvierten Screens werden über die Standardtransformation dargestellt, wobei die bereits erwähnten Nachteile der Standardtransformation auftreten können.

Alle nachfolgend erwähnten DB2 Screen-Anpassungen befinden sich im HATS-Projekt im Verzeichnis *Screen Customizations/DB2*.

5.1.7.1. DB2 Primary Option Menü

Das DB2 Primary Option Menü dient als Übersichtsmenü für DB2-Funktionen und ist in Abbildung 5.54 dargestellt. Zu erreichen ist es auf dem Tübinger Mainframe durch die Option „11“ innerhalb des Screens „Additional IBM Products“. Der Benutzer hat unter anderem die Möglichkeit SPUFI zu starten um SQL-Befehle abzusetzen, den DB2-Precompiler aufzurufen oder die globalen DB2-Parameter zu setzen. Das DB2 Primary Option Menü ist analog zum ISPF Primary Option Menü aufgebaut. Lediglich das Eingabefeld ist an der oberen Seite des Screens angeordnet. Um dem Benutzer die einzelnen Menüpunkte analog zum ISPF Primary Option Menü über die Maus zugänglich zu machen, wurde die Screen-Anpassung *Db2iPrimaryOptionMenu.evnt* angelegt, welche aktiv wird sobald die Zeichenkette „DB2I PRIMARY OPTION MENU“ an Position (30,1) auf einem Screen erscheint. Als Aktion wird der Screen über die zugehörende Transformation *Db2iPrimaryOptionMenu.jsp* angezeigt.

Diese Transformation ist in Abbildung 5.55 dargestellt. Sie zeigt die Menüpunkte über die OptionList-Komponente mit dem OptionListExtended-Widget an. Der OptionList-Komponente wurde das Eingabefeld an Position (14,2) des Screens zugeordnet. Um dem Benutzer die manuelle Eingabe der Menüpunkte zu ermöglichen, wurde das Eingabefeld in der Tranformation zusätzlich mit der Input-Field-Komponente über das Text-Input-Widget dargestellt. Angeordnet wurde das Eingabefeld am unteren Ende der Transformation, damit der Benutzer das Eingabefeld an der aus ISPF gewohnten Stelle vorfindet. Die Überschrift wurde mit Hilfe der Text-Komponente und dem Label-Widget aus dem Screen extrahiert und dargestellt.

5.1.7.2. DB2 SPUFI

Die Funktion SQL Processing Using File Input (SPUFI) erreicht der Benutzer durch den ersten Menüpunkt des DB2 Primary Option Menüs. Mit SPUFI ist es möglich über eine manuelle Eingabe oder eine bereits vorhandene Datei, SQL-Befehle auf dem DB2-System abzusetzen.

```
                          DB2I PRIMARY OPTION MENU          SSID: D931
COMMAND ===> _

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                  (Process SQL statements)
 2  DCLGEN                 (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4  PRECOMPILE             (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6  RUN                    (RUN an SQL program)
 7  DB2 COMMANDS           (Issue DB2 commands)
 8  UTILITIES              (Invoke DB2 utilities)
 D  DB2I DEFAULTS          (Set global parameters)
 Q  QMF                    (Query Management Facility
 X  EXIT                   (Leave DB2I)




  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
MA*   a                                                            02/015
```

Abbildung 5.54.: DB2 Primary Option Menü

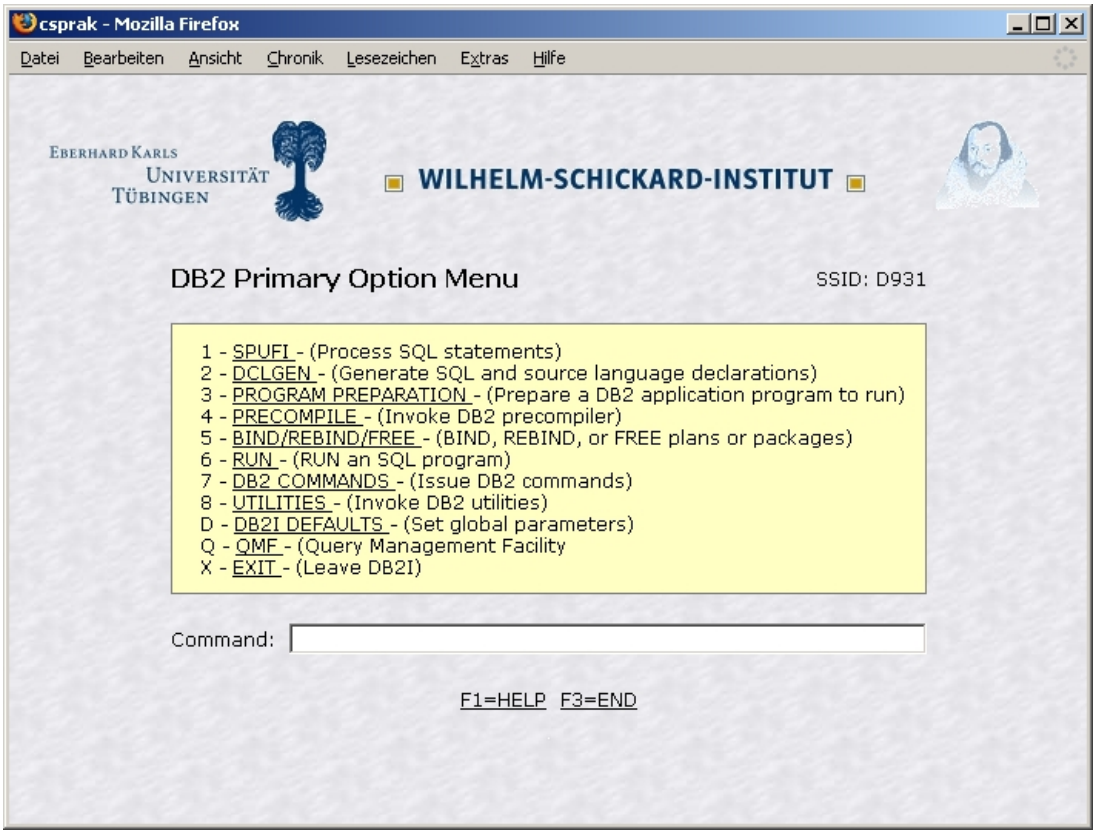

Abbildung 5.55.: DB2 Primary Option Menü in der HATS Web-Anwendung

Da sich das Eingabefeld analog zum DB2 Primary Option Menü an der oberen Seite des Screens befindet, kann die ISPF-Standardtransformation nicht verwendet werden, da diese Transformation das Eingabefeld an der unteren Seite eines Screens erwartet. Daher wurde für SPUFI die Screen-Anpassung *Spufi.evnt* erstellt, welche aktiv wird, sobald in der ersten Zeile eines Screens an beliebiger Position die Zeichenkette „SPUFI“ erkannt wird.

Als Aktion stellt die Screen-Anpassung den Screen über die Transformation *SPUFI.jsp* dar. Dabei wird der Screen-Bereich von Zeile vier bis Zeile 22 mit der Field-Komponente über das Field-Widget und eine Auswahl der Funktionstasten über die Function-Key-Komponente mit dem Link-Widget dargestellt. Da für diesen Screen im Client/Server-Praktikum das Eingabefeld nicht benötigt wird, wurde aus Gründen der Übersichtlichkeit auf dessen Umsetzung verzichtet. Stattdessen wurde die Schaltfläche „OK“ eingefügt, welche zur Bestätigung ein Drücken der Eingabetaste in der Host-Anwendung emuliert.

Da SPUFI nach dem Editieren eines SQL-Befehls den Hinweis auf die Funktionstasten in den Zeilen 21 bis 23 teilweise mit einer SPUFI-Meldung überschreibt (vgl. Abbildung 5.56), musste eine weitere Screen-Anpassung *SpufiExecuted.evnt* erstellt werden. Die Screen-Anpassung wird aktiv, sobald in der ersten Zeile die Zeichenkette „SPUFI“ und in Zeile 21 und Zeile 23 eine gestrichelte Linie erscheint, welche aus einer Anreihung des Zeichens „-“ besteht.

Als zugehörige Aktion wird die Transformation *SpufiExectuted.jsp* angezeigt, welche analog zur Transformation *Spufi.jsp* aufgebaut ist. Lediglich die SPUFI-Meldung wird dem Benutzer mit Hilfe der Text-Komponente und des Label-Widgets in roter Farbe am unteren Ende der Transformation angezeigt. Da die SPUFI-Meldung auf dem Screen die Hinweise auf die Funktionstasten verdeckt, können diese nicht über die Function-Key-Komponente dargestellt werden. Um die Funktionstasten in der HATS Web-Anwendung trotzdem über Links zugänglich zu machen, wurden in der Entwicklungsumgebung über den Menüpunkt „HATS Tools/Insert Host Keypad/-Individual Key“ die Funktionstasten PF1 und PF3 als Links eingefügt. Das Stylesheet und die Beschriftung der Links wurden anschließend im Quelltext noch angepasst, damit die Links analog zu den Funktionstasten der anderen Transformationen dargestellt werden. Die fertige Transformation ist in Abbildung 5.57 zu sehen. Wird der Hinweis von SPUFI vom Benutzer mit der Eingabetaste quittiert, führt SPUFI den abgesetzten SQL-Befehl aus und zeigt auf einem neuen Screen Informationen beziehungsweise Fehlermeldungen zum SQL-Befehl an. Falls die Informationen umfangreich genug sind, kann der Benutzer mit den Funktionstasten PF7 und PF8 durch sie navigieren. Um dem Benutzer diese Funktionstasten in der HATS Web-Anwendung als Links anzuzeigen, wurde die Screen-Anpassung *SpufiOut.evnt* mit der Transformation *SpufiOut.jsp* erstellt. Die Screen-Anpassung wird aktiv, sobald auf einem Screen beginnend an Position (11,3) die Zeichenkette „.SPUFI.OUT“ angezeigt wird. Der Screen-Bereich von Zeile drei bis Zeile 21 wird von der Transformation über die Field-Komponente mit dem Field-Widget dargestellt. Die Funktionstasten werden über die Function-Key-Komponente über das Link-Widget als Links angezeigt. Da der Benutzer im Client/Server-Praktikum auf diesem Screen keine Kommandos absetzt, wurde anstatt eines Eingabefeldes eine Schaltfläche „OK“ eingefügt, welche in der Host-Anwendung ein Drücken der Eingabetaste emuliert.

```
  7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
  8  AUTOCOMMIT ...... ===> YES        (Y/N - Commit after successful run?)
  9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>

        DSNE803A INPUT FILE WAS NOT CHANGED. PRESS ENTER TO CONTINUE
  F1=H                                                          HANGE
  F7=UP         F8=DOWN       F9=SWAP      F10=LEFT     F11=RIGHT     F12=RETRIEVE
MA*   a                                                          06/029
```

Abbildung 5.56.: SPUFI-Meldung

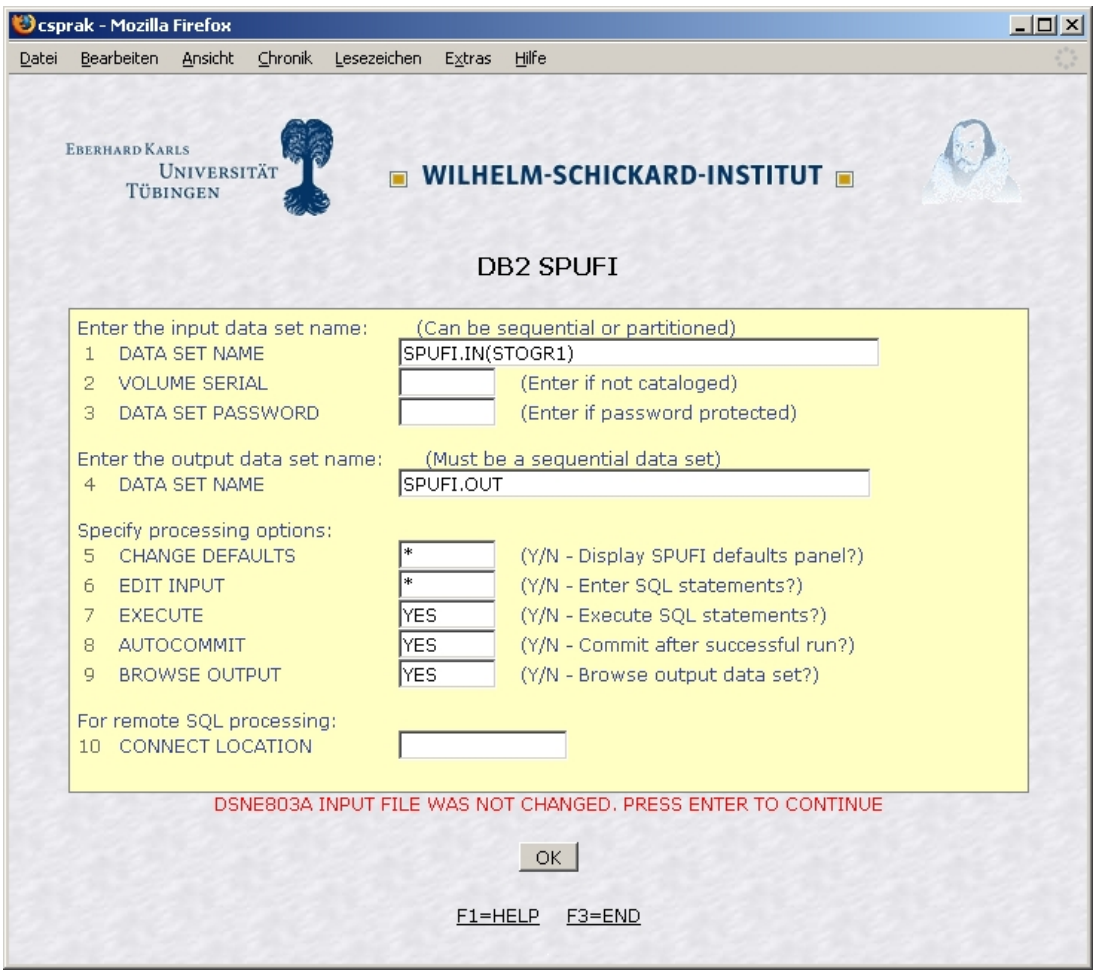

Abbildung 5.57.: SPUFI-Meldung in der HATS Web-Anwendung

5.1.8. Umsetzung sonstiger Screens

Mit den erstellten Screen-Anpassungen werden alle im Client/Server-Praktikum benötigten Screens in der HATS Web-Anwendung ansprechend dargestellt. Sonstige Screens, auf die kein Erkennungskriterium der erstellten Screen-Anpassungen zutrifft, werden über die HATS-Standardtransformation *default.jsp* dargestellt (vgl. Abbildung 5.58).

Die Transformation stellt den kompletten Screen über die Field-Komponente mit dem Field-Widget dar. Zusätzlich wurde eine Schaltfläche „OK / Enter“ eingefügt, mit welcher der Benutzer in der Host-Anwendung ein Drücken der Eingabetaste emulieren kann. Am Ende der Transformation wurden zwei Links für die Funktionstasten PF1 und PF3 eingefügt, damit die HATS Web-Anwendung mit großer Wahrscheinlichkeit über die Maus bedienbar bleibt.

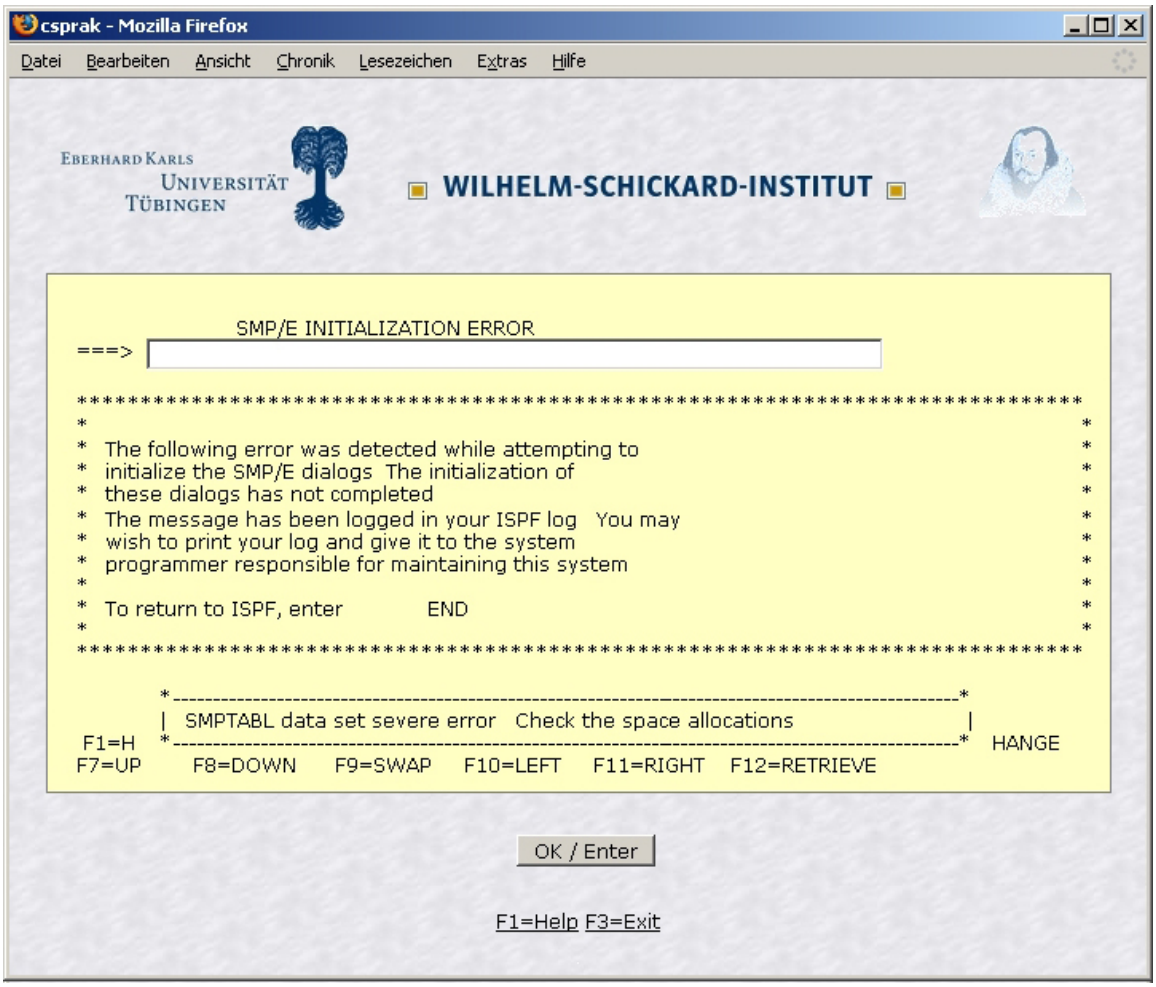

Abbildung 5.58.: Standardtransformation der HATS Web-Anwendung

5.1.8.1. Blockierte Screens

Während der Interaktion mit der Host-Anwendung können gelegentlich blockierte Screens auftreten. Blockierte Screens werden nur vorübergehend angezeigt und erlauben keine Benutzereingaben [HATS3, Kapitel 7]). Der Screen deblockiert meistens innerhalb weniger Millisekunden. Blockierte Screens treten zum Beispiel beim Abmelden von TSO auf. In der HATS Web-Anwendung werden diese blockierten Screens standardmäßig mit der Standardtransformation dargestellt.

Da der Benutzer durch blockierte Screens keine wichtigen Informationen erhält, wurde in der erstellten HATS Web-Anwendung die Darstellung von blockierten Screens deaktiviert. Dazu wurde die Screen-Anpassung *Inhibited/Inhibited.evnt* erstellt, welche aktiv wird sobald ein blockierter Screen erkannt wird. Das dazu nötige Erkennungskriterium kann nicht über einen Menüpunkt der Entwicklungsumgebung gesetzt werden. Stattdessen musste der in Listing 5.5 abgebildete Code-Ausschnitt im Quelltext der Screen-Anpassung eingefügt werden. Nicht blockierte Screens kennzeichnet HATS im Element *oia* standardmäßig durch das Attribut *status* mit dem Wert *NOTINHIBITED*. Für dieses Erkennungskriterium musste das Attribut *status* durch das Attribut *invertmatch=true* invertiert werden. Folglich wird die Screen-Anpassung aktiv sobald der Status eines Screens nicht *NOTINHIBITED* ist. Als Aktion wurde für die Screen-Anpassung eine Aktion vom Typ „Pause normal processing“ mit einer Zeitangabe von 1000 Millisekunden festgelegt, wodurch die HATS Web-Anwendung im Fall eines blockierten Screens eine Sekunde lang wartet und anschließend überprüft ob ein neuer Screen vorliegt. Liegt immer noch ein blockierter Screen vor, wiederholt sich der Vorgang.

```
1  <event defaultHostKey="" description="" type="screenRecognize">
2
3    ...
4
5    <description>
6      <oia invertmatch="true" optional="false" status="NOTINHIBITED"/>
7    </description>
8
9  </event>
```

Listing 5.5: Erkennungskriterium für blockierte Screens

5.2. Entwickelte Komponenten und Widgets

Um die Funktionalität der HATS Web-Anwendung zu erweitern, wurden mit der HATS-Entwicklungsumgebung zusätzliche Komponenten und Widgets entwickelt, welche im Nachfolgenden vorgestellt werden. Der Code dieser Komponenten und Widgets befindet sich im HATS-Projektverzeichnis *Source/csprak*.

5.2.0.2. OptionList-Komponente

Die OptionList-Komponente, welche zum Beispiel bei der Umsetzung des ISPF Primary Option Menüs zum Einsatz kommt, besitzt die Funktionalität drei Spalten eines Screens anhand von festgelegten Koordinaten zu unterscheiden und dabei den Inhalt einer definierten Spalte jeweils zeilenweise als Kommando für ein Eingabefeld zu interpretieren. Diese Kommandos werden von dem zugehörigen OptionList-Widget als Links dargestellt und können dadurch vom Benutzer über einen Mausklick ausgeführt werden. Die restlichen Spalten werden innerhalb einer HTML-Tabelle in Textform ausgegeben.

In der HATS-Entwicklungsumgebung werden die Spalten und das zugehörige Eingabefeld in den Einstellungen der Komponente definiert. Für das ISPF Primary Option Menü wurde über die Einstellungen in Abbildung 5.59 festgelegt, dass sich im Screen-Bereich von Position (14,22) bis (24,22) das Eingabefeld befindet. Zusätzlich wurde festgelegt, dass sich jeweils zwischen den Spalten zwei bis vier die Abkürzung, zwischen den Spalten fünf bis 17 die Kurzbeschreibung und zwischen den Spalten 19 bis 55 die normale Beschreibung eines Menüpunktes befinden. Die Spalte Kurzbeschreibung wird über das im folgenden Abschnitt 5.2.0.3 vorgestellte Widget als Link dargestellt, welcher die Spalte mit der Abkürzung als Kommando interpretiert und dieses automatisch in das zugewiesene Eingabefeld überträgt und nach knapp einer Sekunde mit der Eingabetaste bestätigt. Die Abkürzung und die Beschreibung werden vom Widget in Textform ausgegeben.

Der Code der Komponente ist im Anhang in Listing A.2 zu finden. Die Klasse *OptionList* erbt von der Klasse *Component* und überschreibt die Methoden *recognize()* und *getCustomProperties()*. In der Methode *recognize()* werden in den Zeilen 40 bis 50 die in der Entwicklungsumgebung für die Komponente definierten Einstellungen über ein *Properties*-Objekt *settings* ausgelesen und in lokalen Variablen gespeichert. Die verwendete Methode *getProperty(String key, String defaultValue)* gibt dabei den Wert für die durch den Parameter *key* definierte Einstellung zurück. Wurde in den Einstellungen kein Wert angegeben, wird der im Parameter *defaultValue* angegebene Standardwert verwendet. Die Standardwerte werden in den Zeilen 21-30 definiert und sind so gewählt, dass für das ISPF Primary Option Menü keine Einstellungen mehr geändert werden müssen.

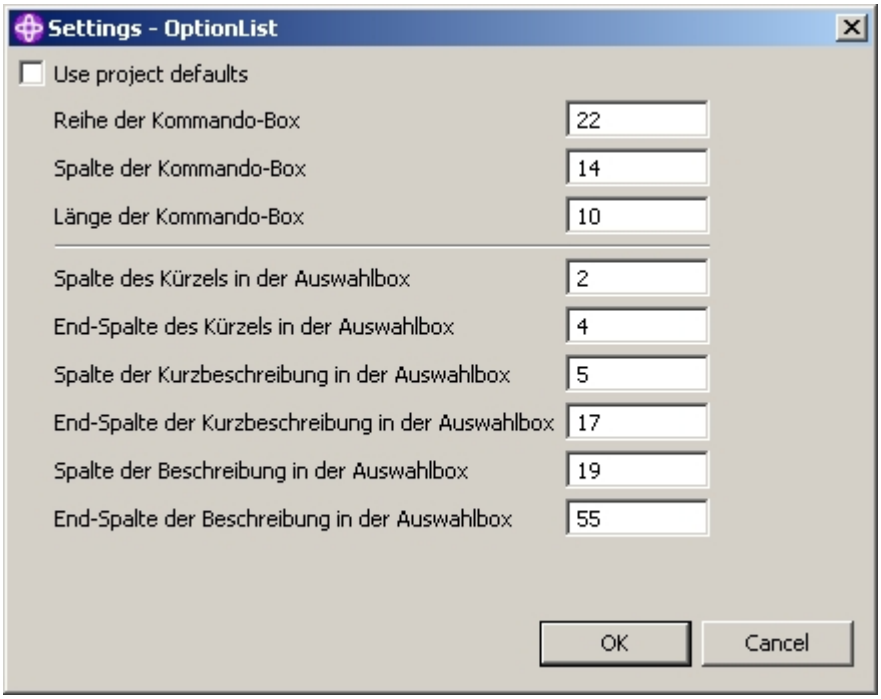

Abbildung 5.59.: Einstellungen der OptionList-Komponente

Anschließend werden in den Zeilen 51-57 die Koordinaten des für die Komponente ausgewählten Screen-Bereichs aus einem *BlockScreenRegion*-Objekt *region* mit Hilfe der Methoden *startRow()*, *startCol()*, *endRow()* und *endCol()* ausgelesen und in lokalen Variablen gespeichert. Die Zeichenketten innerhalb dieses Screen-Bereichs werden in den Zeilen 60-62 mit der Methode *getScreenRect()* des *HsrScreen*-Objekts *sc* in das Charakter-Array *buffer* extrahiert. Der Inhalt des Screen-Bereichs wird im Folgenden über das Array *buffer* zeilenweise durchlaufen, wobei in den Zeilen 85-101 der aktuelle Inhalt der Spalten Zeichen für Zeichen in die Charakter-Arrays *citem*, *cdescription* und *cfullcaption* kopiert wird. In diesen Arrays befinden sich nun Abkürzung, Kurzbeschreibung und Beschreibung des Menüpunktes der aktuellen Zeile. Nach jedem Zeilendurchlauf werden in den Zeilen 103-105 die Charakter-Arrays in den Java Typ String umgewandelt und in Zeile 108 als Parameter für das Anlegen eines neuen *ListItemComponentElement*-Objekts benutzt, welches somit jeweils einen Menüpunkt des Screens beinhaltet.

Dem *ListItemComponentElement*-Objekt werden in Zeile 112 mit der Methode *setOnSelectActions()* über einen Java Vektor vom Typ *ScriptAction* zwei Aktionen zugewiesen. Der Vektor beinhaltet als erste Aktion ein Objekt vom Typ *SetFormValue*, welches das in den Komponenteneinstellungen definierte Eingabefeld mit der Abkürzung des Kommandos füllt. Die zweite Aktion ist ein Objekt vom Typ *SubmitFormAction*, welches zum Absenden des übergebenen Kommandos ein Drücken der Eingabetaste emuliert. In Zeile 121-123 werden alle erstellten *ListItemComponentElement*-Objekte in ein *ComponentElement*-Array geschrieben, welches von der Methode *recognize()* zurückgegeben wird. Dies ist möglich, da die Klasse *ComponentElement*

die Superklasse der Klasse *ListItemComponentElement* ist. Das zurückgegebene Array gibt die HATS-Anwendung nach dem Ablauf der Methode intern an das der Komponente zugewiesene Widget weiter, wodurch dieses Zugriff auf die *ComponentElement*-Objekte des Screens erhält.

Die Methode *getCustomProperties()* in Zeile 138 wird verwendet, um in der HATS-Entwicklungsumgebung der Komponente, über das in Abbildung 5.59 dargestellte Formular, Einstellungen zuweisen zu können. Die Einstellungen werden in einem Java Vektor gespeichert, welcher Objekte vom Typ *HCustomProperty* aufnimmt. Ein *HCustomProperty*-Objekt repräsentiert in den Einstellungen ein Eingabefeld, welchem eine Bezeichnung, ein Typ, ein Standardwert und andere Eigenschaften zugewiesen werden können. Der Java Vektor wird in Zeile 150 von der Methode zurückgegeben.

Aus Gründen der Übersichtlichkeit wurde die Behandlung von Sonderfällen, wie zum Beispiel die Umwandlung des Kommandos „blank“ in ein Leerzeichen zum Anzeigen von Informationen über ein Dataset (vgl. Abbildung 5.24), im Listing nicht abgedruckt.

5.2.0.3. OptionListExtended-Widget

Das OptionListExtended-Widget, welches die OptionList-Komponente darstellen kann, befindet sich im Anhang in Listing A.3. Die Klasse *OptionListExtended* erbt von der Klasse *Widget* und implementiert das Interface *HTMLRenderer* mit der Methode *drawHTML()*. Die Methode *drawHTML()* erstellt aus den von der Komponente übergebenen *ComponentElement*-Objekten eine HTML-Ausgabe und speichert diese in dem StringBuffer *buffer*. Dazu wird das *ComponentElement*-Array, das in Zeile 16 dem Widget übergeben wird, durchlaufen und dabei in Zeile 29 jedes enthaltene *ComponentElement*-Objekt zu einem *ListItemComponentElement*-Objekt gecasted. Über die Methoden *getItem()* und *getFullCaption()* gibt dieses Objekt die von der Komponente gespeicherte Abkürzung beziehungsweise die Beschreibung des Menüpunktes zurück, welche in der Variablen *buffer* formatiert gespeichert werden. Um dem *ListItemComponent*-Objekt die Funktionalitäten zuzuweisen, wird in Zeile 34 jeweils ein Objekt der Klasse *LinkElement* über das Objekt *HTMLElementFactory* mit der Methode *createLink()* erstellt. Dieses Objekt repräsentiert den Link für das auszuführende Kommando. In Zeile 35 wird dem Link über die Methode *setClassName()* noch ein Stylesheet zugewiesen. Anschließend wird in Zeile 36 und 37 das Objekt vom Typ *LinkElement* mit der Methode *render()* und *renderEndTag()* als HTML-Code in den StringBuffer geschrieben, welcher als Rückgabewert der Methode *drawHTML()* in eine Transformation eingebettet wird.

Die zusätzlich erstellten Widgets *OptionList* und *OptionListSingle* unterscheiden sich in der Ausgabe der Informationen. Das OptionList-Widget verzichtet auf die Ausgabe der Beschreibung und das OptionListSingle-Widget zusätzlich auf die Ausgabe der Kurzbeschreibung. Da in den zwei Widgets nur bereits bekannte Techniken benutzt wurden, wird an dieser Stelle lediglich auf den Java Code der Widgets im HATS-Projekt verwiesen.

5.2.0.4. DatasetList-Komponente

Die DataSetList-Komponente, welche zu der Umsetzung des ISPF Dataset List Utility verwendet wird, bietet für Eingabefelder die Möglichkeit an, Kommandos über eine Auswahlliste abzusetzen (vgl. Abbildung 5.29). Dabei wird im Feld „Options“ in der Komponenteneinstellung festgelegt, welche Kommandos in der Auswahlliste verfügbar sein sollen. Die Kommandos werden darin in der Form „Kommando=Beschreibung“ durch Semikolon separiert aufgelistet. Um ebenso die auf dem Screen angezeigten Informationen über ein Dataset zusammen mit der Auswahlliste schön formatiert in einer HTML-Tabelle ausgeben zu können, müssen in den Komponenteneinstellungen die einzelnen Spalten des Screens analog zu Abbildung 5.60 definiert werden. Hierbei kann angegeben werden, ob ein Screen aus zwei oder aus vier Spalten besteht. Die Standardwerte wurden so gewählt, dass die Komponente ohne Änderung der Einstellungen mit dem ISPF Dataset List Utility verwendet werden kann.

Der Code der Komponente ist im Anhang in Listing A.4 zu finden. Die Klasse erbt analog zur OptionList-Komponente von der Klasse *Component* und überschreibt die Methoden *recognize()* und *getCustomProperties()*. In der Methode *recognize()* werden in den Zeilen 31-45 die in der HATS-Entwicklungsumgebung für die Komponente definierten Einstellungen über ein *Properties*-Objekt *settings* ausgelesen und in lokalen Variablen gespeichert. Der Inhalt des Feldes „Options“ der Komponenteneinstellung wird in den Zeilen 61-65 in Kommandos und die dazu gehörenden Beschreibungen aufgetrennt. Daraus wird jeweils ein *ListItemComponentElement*-Objekt erstellt, welches dem *ComponentListElement*-Array *cel* hinzugefügt wird. Dieses Array repräsentiert den Inhalt der Auswahlliste, welche später durch das zugehörige Widget dargestellt wird. In den Zeilen 68-73 werden anschließend die Koordinaten des für die Komponente ausgewählten Screen-Bereichs aus einem *BlockScreenRegion*-Objekt *region* mit Hilfe der Methoden *startRow()*, *startCol()*, *endRow()* und *endCol()* ausgelesen und in lokalen Variablen gespeichert. Die Zeichenketten innerhalb dieses Screen-Bereichs werden mit der Methode *getScreenRect()* des *HsrScreen*-Objekts *sc* aus dem Screen in das Charakter-Array *buffer* extrahiert. Das Array *buffer* wird in den Zeilen 83-127 durchlaufen, wobei der aktuelle Inhalt der Spalten zeilenweise in die Charakter-Arrays *dsName*, *col1*, *col2* und falls gewünscht in *col3* und *col4* kopiert wird. Nach jedem Zeilendurchlauf werden in den Zeilen 107-109 die Charakter-Arrays in den Java Typ String umgewandelt und als Parameter für das Anlegen eines neuen *TextComponentElement*-Objekts benutzt. Ein *TextComponentElement* dient als Container für aus dem Host-Screen extrahierten Text.

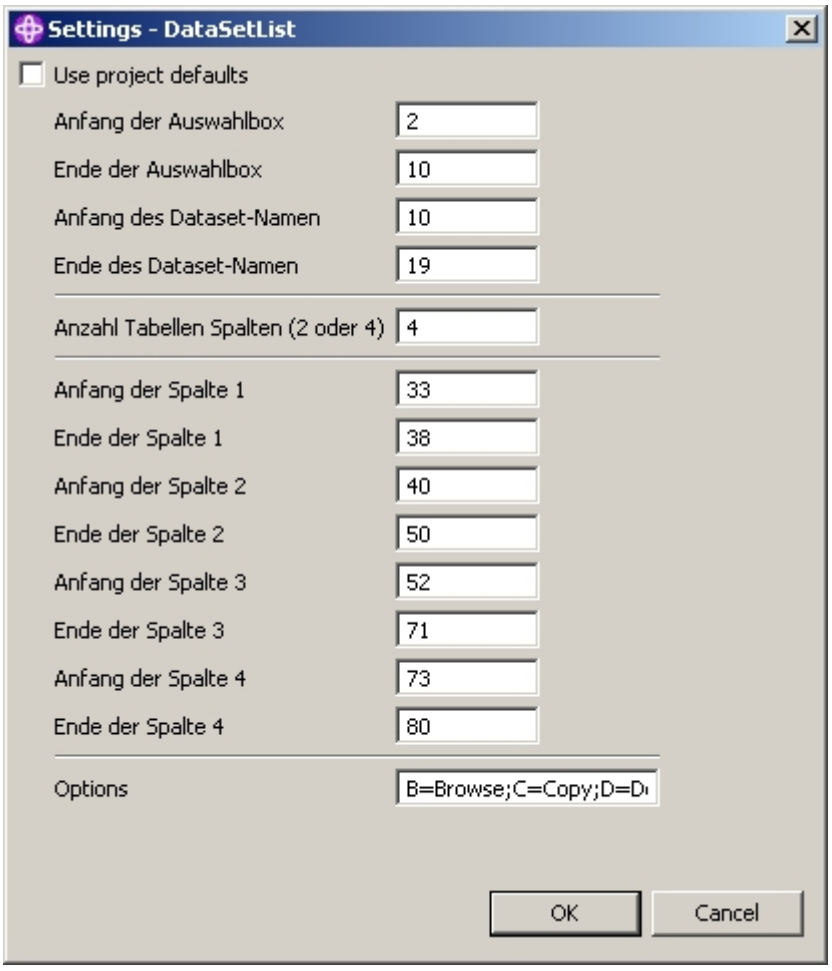

Abbildung 5.60.: Einstellungen der DatasetList-Komponente

In jedem Zeilendurchlauf wird zusätzlich in Zeile 103 ein Objekt vom Typ *InputComponentElement* erstellt, welches das Eingabefeld des Host-Screens repräsentiert. Mit Hilfe dieses Objekts und dem in Zeile 61 angelegten *ComponentElementList*-Objekt wird in Zeile 104 ein Objekt vom Typ *SelectionComponentElement* angelegt, wodurch die Beziehung zwischen dem Eingabefeld und der durch das Widget dargestellten Auswahlliste hergestellt wird. In den Zeilen 107-109 wird das *SelectionComponentElement*-Objekt und die *TextComponentElement*-Objekte dem *ComponentElement*-Array hinzugefügt. in Zeile 136 gibt die *recognize()*-Methode das *ComponentElement*-Array zurück, welches die HATS-Anwendung nach dem Ablauf der Methode intern an das der Komponente zugewiesene Widget weiterreicht.

Die Methode *getCustomProperties()* in Zeile 139 wurde analog zu der Methode in der OptionList-Komponente implementiert, weshalb hier nicht näher darauf eingegangen wird.

5.2.0.5. DataSetList4Columns-Widget

Die DataSetList-Komponente kann über die erstellten Widgets *DataSetList2Columns*, *DataSetList4Columns* und *MemberList4Columns* dargestellt werden. Da das Konzept dieser Widgets ähnlich ist, wird an dieser Stelle nur das umfangreichste Widget DataSetList4Columns erklärt. Der Code der nicht erwähnten Widgets findet der Leser im Verzeichnis *Source/Widgets* des HATS-Projekts.

Das DataSetList4Columns-Widget, welches die DataSetList-Komponente darstellen kann, befindet sich im Anhang in Listing A.5. Wie jedes Widget erbt die Klasse *DataSetList4Columns* von der Klasse *Widget* und implementiert das Interface *HTMLRenderer* mit der Methode *drawHTML()*. In den Zeilen 20-25 werden darin die in der Entwicklungsumgebung vorgenommenen Einstellungen des Widgets über das *Properties*-Objekt *settings* ausgelesen. Das Formular für die Einstellungen ist in Abbildung 5.61 dargestellt. Darin wird vom Entwickler die Überschrift für einzelne Spalten sowie die sich abwechselnde Hintergrundfarbe für die HTML-Tabelle festgelegt.

In den Zeilen 36-40 wird das Grundgerüst für die HTML-Tabelle in den StringBuffer *buffer* geschrieben. Die Spaltenüberschriften werden dabei den Einstellungen des Widgets entsprechend gesetzt. Um nun das im Konstruktoraufruf übergebene *ComponentElement*-Array mit den beinhalteten *SelectionComponentElement*- und *TextComponentElement*-Objekten darzustellen, wird eine Fallunterscheidung benötigt. Pro Zeile des Screens speichert die DataSetList-Komponente in dieses Array ein *SelectionComponentElement*-Objekt und vier *TextComponentElement*-Objekte. Folglich sind das erste und jeweils sechste Element des Arrays *SelectionComponentElement*-Objekte und alle weiteren Elemente *TextComponentElement*-Objekte. Die Zeilen 43-52 und 59-75 sind für die Darstellung der *SelectionComponentElement*-Objekte zuständig. Aus einem *SelectionComponentElement*-Objekt wird über die Methode *getField()* das Eingabefeld ausgelesen. Dieses wird über ein *HTMLElementFactory*-Objekt mit der Methode *createTextInput()* in das HTML-Format umgewandelt und zur Ausgabe in den StringBuffer geschrieben. Die zugehö-

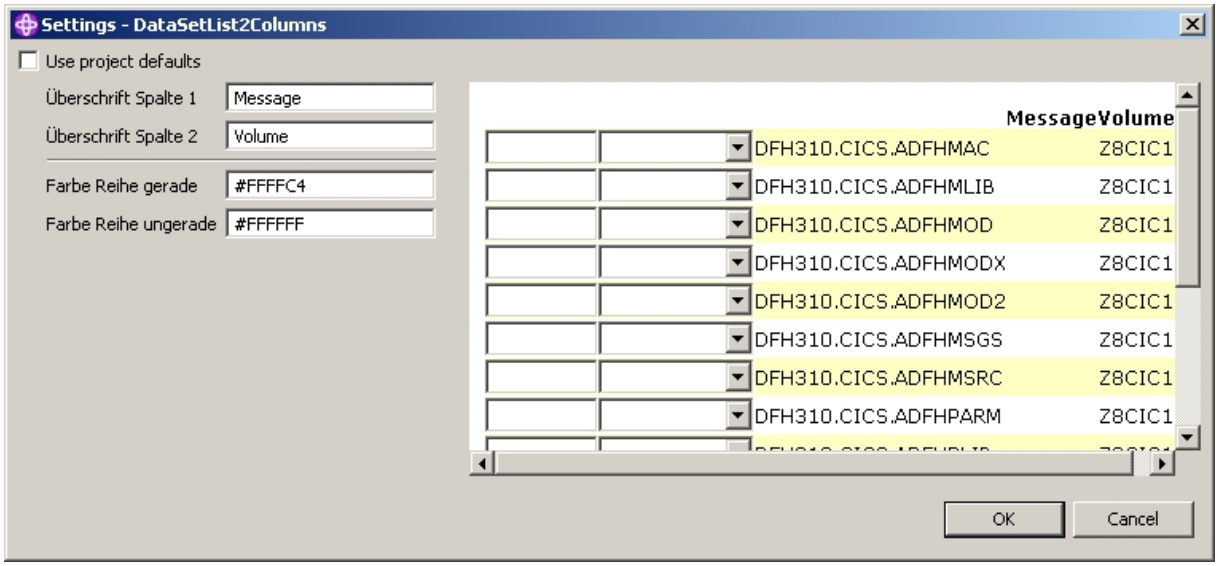

Abbildung 5.61.: Einstellungen des DataSetList-Widgets

rige Auswahlliste wird über die Methode *createDropdown()* des *HTMLElementFactory*-Objekts mit dem jeweiligen *SelectionComponentElement*-Objekt als Parameter in das HTML-Format umgewandelt und ebenfalls dem StringBuffer übergeben. Die Zeilen 53-58 und 76-81 sind für die Darstellung der *TextComponentElement*-Objekte zuständig. Der Inhalt dieses Objekts wird mit der Methode *getText()* ausgelesen und dem StringBuffer an geeigneter Stelle hinzugefügt.

Um die HTML-Tabelle übersichtlicher zu gestalten, wurde zusätzlich in den Zeilen 61-65 eine Fallunterscheidung eingebaut, welche Zeilen der Tabelle abwechselnd mit den Hintergrundfarben *colorRowEven* und *colorRowOdd* darstellt. Diese Farben sind dabei in den Einstellungen des Widgets festlegbar. Zuletzt gibt die *drawHTML()*-Methode den StringBuffer mit dem HTML-Inhalt zurück. Dieser HTML-Inhalt repräsentiert innerhalb einer Transformation die DataSetList-Komponente und das DataSetList4Columns-Widget.

5.3. Erstellung eines SOAP Web Service

Zusätzlich zu der im vorherigen Abschnitt erstellten HATS Web-Anwendung wurde im Rahmen dieser Diplomarbeit ein Teil einer CICS-Transaktion als SOAP Web Service verfügbar gemacht. Als CICS-Transaktion wurde dabei die Transaktion NACT verwendet, mit welcher in Tutorial 14 des Client/Server-Praktikums gearbeitet wird. Um den Web Service nutzen zu können, wurde zusätzlich ein Web Service Client in Form von JavaServer Pages erstellt.

Der erstellte Web Service und der Web Service Client befinden sich auf der beigelegten virtuellen Maschine im HATS-Projekt *NactWebservice*. Installiert wurde der Web Service auf dem unter z/Linux eingerichteten Websphere Application Server. Der Web Service ist dort über den Web Service Client über die URL `http://galadriel.cs.uni-tuebingen.de:9080/NactWebservice/TestClient/TestClient.jsp` zu erreichen. Zu beachten ist, dass der Web Service für die NACT-Transaktion des Mainframes der Universität Leipzig erstellt wurde und deshalb für den Aufruf ein Account für die LPAR Lucas benötigt wird.

5.3.0.6. NACT CICS-Transaktion

Die NACT-Transaktion ist eine CICS-Anwendung, mit welcher Kundendaten der fiktiven Firma KanDoIT verwaltet werden können. Mit dieser Anwendung ist es möglich Kundendaten zu suchen, zu erstellen, zu löschen und zu ändern. Für weitergehende Informationen zum Aufbau dieser CICS-Anwendung sei auf [HORS] verwiesen.

5.3.1. Vorgehensweise zur Erstellung des Web Service

Es wurde die Suche nach Kundendaten in der NACT-Transaktion über einen Web Service zur Verfügung gestellt. Der Aufrufer übergibt dem Web Service seine CICS-Benutzerdaten und eine gültige Kundennummer. Als Antwort liefert der Web Service den zur Kundennummer passenden Namen und die Anschrift. Ist die Kundennummer ungültig, liefert der Web Service eine Fehlermeldung. Um den Web Service aufrufen zu können wurde ein kleiner Test-Client in Form von JavaServer Pages erstellt, über welchen die Ansteuerung des Web Service mit einem HTML-Formular möglich ist.

Um mit Hilfe von HATS einen Web Service zu erstellen wird ein Makro benötigt, welches mit Hilfe von Variablen und Makro-Aktionen vom Typ „Prompt“ die Benutzerinteraktion mit der Host-Anwendung beschreibt. Diese Benutzerinteraktion wird anschließend als Web Service verfügbar gemacht. Dazu muss aus dem Makro ein Integration Object erstellt werden, aus welchem man in der HATS-Entwicklungsumgebung über mehrere Dialoge einen Web Service erstellen lassen kann. Dabei hat man die Wahl, den Web Service basierend auf JAX-RPC oder JAX-WS zu generieren. Damit ein Web Service basierend auf JAX-WS auf dem Websphere Application Server 6.1 läuft,

wird das Feature Pack for Web Services for Websphere Application Server V6.1 [WAS1] benötigt, welches unter anderem JAX-WS in der Version 2.0 unterstützt. Für diese Arbeit wurde ein Web Service auf Basis von JAX-RPC erstellt, wodurch eine Installation des Feature Packs nicht nötig war.

5.3.2. Zugrundeliegendes Makro

Das für den Web Service erstellte Makro *NACTdisplayAcc.hma* ist in Abbildung 5.62 dargestellt. Als Variablen für das Makro wurden die CICS-Benutzerkennung, das CICS-Passwort und eine Kundennummer gewählt, welche später als Eingabe für den Web Service dienen sollen. Diese Variablen müssen bereits vor Ablauf des Makros durch den Benutzer entsprechend gefüllt werden.

Das Makro navigiert zuerst durch das Kommando „L C001“ zur CICS-Anmeldung. Dort füllt es mit Hilfe von Prompt-Aktionen die Felder für die CICS-Benutzerkennung und das Passwort mit den in den Variablen gespeicherten Werten. Um zwischen den Eingabefeldern zu navigieren, werden Input-Aktionen verwendet, welche innerhalb der Host-Anwendung ein Drücken der Tabulator-Taste emulieren. Nachdem die Eingabefelder gefüllt worden sind, wird über eine Input-Aktion ein Drücken der Eingabetaste emuliert. Wird die Benutzerkennung und das Passwort von der Host-Anwendung akzeptiert, startet das Makro durch eine Input-Aktion mit dem Kommando „NACT“ die NACT-Transaktion. Innerhalb der NACT-Transaktion navigiert das Makro über die Tabulator-Taste und das Kommando „D“ zum Eingabefeld für die Kundennummer. Die Kundennummer wird anschließend über eine Prompt-Aktion aus der Variable *account* an die Host-Anwendung übergeben. Falls die Kundennummer gültig ist, werden im folgenden Schritt aus dem Screen die Kundendaten Vorname, Nachname, Telefonnummer und Adresse als Zeichenketten mit Extract-Aktionen in Variablen extrahiert. Als letzte Makro-Aktion muss der CICS-Benutzer noch ordnungsgemäß abgemeldet werden. Zum Abmelden wird in der Host-Anwendung ein Drücken der Funktionstaste PF3 emuliert und mit einer Input-Aktion das Kommando „CESF LOGOFF“ eingegeben.

Für den Fehlerfall wird die Variable *errMessage* gehalten. Ein Fehlerfall kann auftreten, falls die eingegebene Kundennummer außerhalb des gültigen Bereichs für Kundennummern liegt. Für diesen Fehlerfall muss im Makro ein Pfad eingebaut werden, bei welchem die Fehlermeldung extrahiert wird. Wird eine Kundennummer nicht akzeptiert, wird die Verbindung mit CICS nach dem Extrahieren der Fehlermeldung vom Makro beendet. Die Abmeldung wurde durch die Verbindung des Zustands *MustBeNumericAndInRange* mit dem Zustand *Logoff* gelöst.

5.3.3. Erstellung des JAX-RPC Web Service

Mit einem Rechtsklick auf das Makro *NACTdisplayAcc.hma* wurde in der Entwicklungsumgebung über den Menüpunkt „Create Integration Object“ das Integration Object *NACT_ DisplayAcc.java* im Verzeichnis *Source/IntegrationObject* erstellt. Das nun erstellte Integration Object kapselt die

Abbildung 5.62.: Makro für die Suche nach Kundendaten

Interaktion mit der Host-Anwendung. Um daraus einen Web Service zu erstellen, werden zuerst Web Service Support-Dateien benötigt. Die Web Service Support-Dateien sind Java-Klassen welche die Ein- und Ausgabeparameter für den Web Service beschreiben. Diese wurden mit einem Rechtsklick auf das Integration Object über den Menüpunkt „Create Web Service Support Files“ des aufklappenden Menüs im Projektverzeichnis *Source/webserviceclasses* generiert. In diesem Schritt muss ebenfalls der Name des zukünftigen Web Service festgelegt werden. In dieser Arbeit wurde der Web Service *NactWS* genannt. Die erstellten Web Service Support-Dateien sind nachfolgend aufgelistet.

Dateien 1 Web Service Support

```
source/
  webserviceclasses/
    NACT_DisplayAcc_Input_Properties.java
    NACT_DisplayAcc_Output_Properties.java
    NactWS.java
```

Um den Web Service mit Hilfe dieser Dateien zu erstellen wurde durch einen Rechtsklick auf die Datei *NactWS.java* mit dem Menüpunkt „Web Services/Create Web Service“ des aufklappenden Menüs die Konfigurationsoberfläche in Abbildung 5.63 aufgerufen. Das erste Auswahlfeld beschreibt, ob es sich um einen Bottom-Up oder einen Top-Down Web Service handelt. Bei einem Bottom-Up Web Service wird der Web Service aus bereits vorhandenen Java-Klassen generiert, während bei einem Top-Down Web Service das Grundgerüst des Web Service allein aus einer WSDL-Datei generiert wird. In den weiteren Auswahlfeldern hat der Entwickler die Möglichkeit festzulegen ob es sich um einen JAX-RPC oder JAX-WS Web Service handelt und für welchen Application Server der Web Service erstellt werden soll. Ebenfalls muss angegeben werden, welche Projekte und Java-Klassen involviert sind. Zusätzlich kann der Entwickler automatisch ein Grundgerüst für einen Test-Client erstellen lassen. Nach Auswahl der gewünschten Optionen und einem Klick auf die Schaltfläche „Finish“ wird der Web Service erstellt. Für den erstellten Web Service wurden die in Abbildung 5.63 gezeigten Einstellungen verwendet.

Nach erfolgreicher Erstellung befindet sich im Projektverzeichnis *Web Content/WEB-INF/Web Service Definitions* die WSDL-Datei *NactWS.wsdl*, welche den Web Service beschreibt. Auszüge aus der WSDL-Datei befinden sich im Anhang in Listing A.6. Eine Übersicht und Erklärung zu den in der WSDL-Datei verwendeten XML-Datentypen findet man in [W3C3] und [W3C2].

Interessant an der WSDL-Datei ist zu sehen, wie die HATS-Entwicklungsumgebung die Ein- und Ausgabeparameter von dem Web Service in die WSDL-Datei integriert hat. Zuerst werden die Eingabeparameter *NACT_DisplayAcc_Input_Properties* und die Ausgabeparameter *NACT_DisplayAcc_Output_Properties* innerhalb des Elements *wsdl:types* beginnend in der vierten Zeile der WSDL-Datei als XML-Elemente vom Datentyp *complexType* deklariert. Diese bestehen wiederum aus einem XML-Element von Datentyp *sequence*. Über die Elemente *wsdl:message* werden beginnend in den Zeilen 36 und 41 die Nachrichten für die

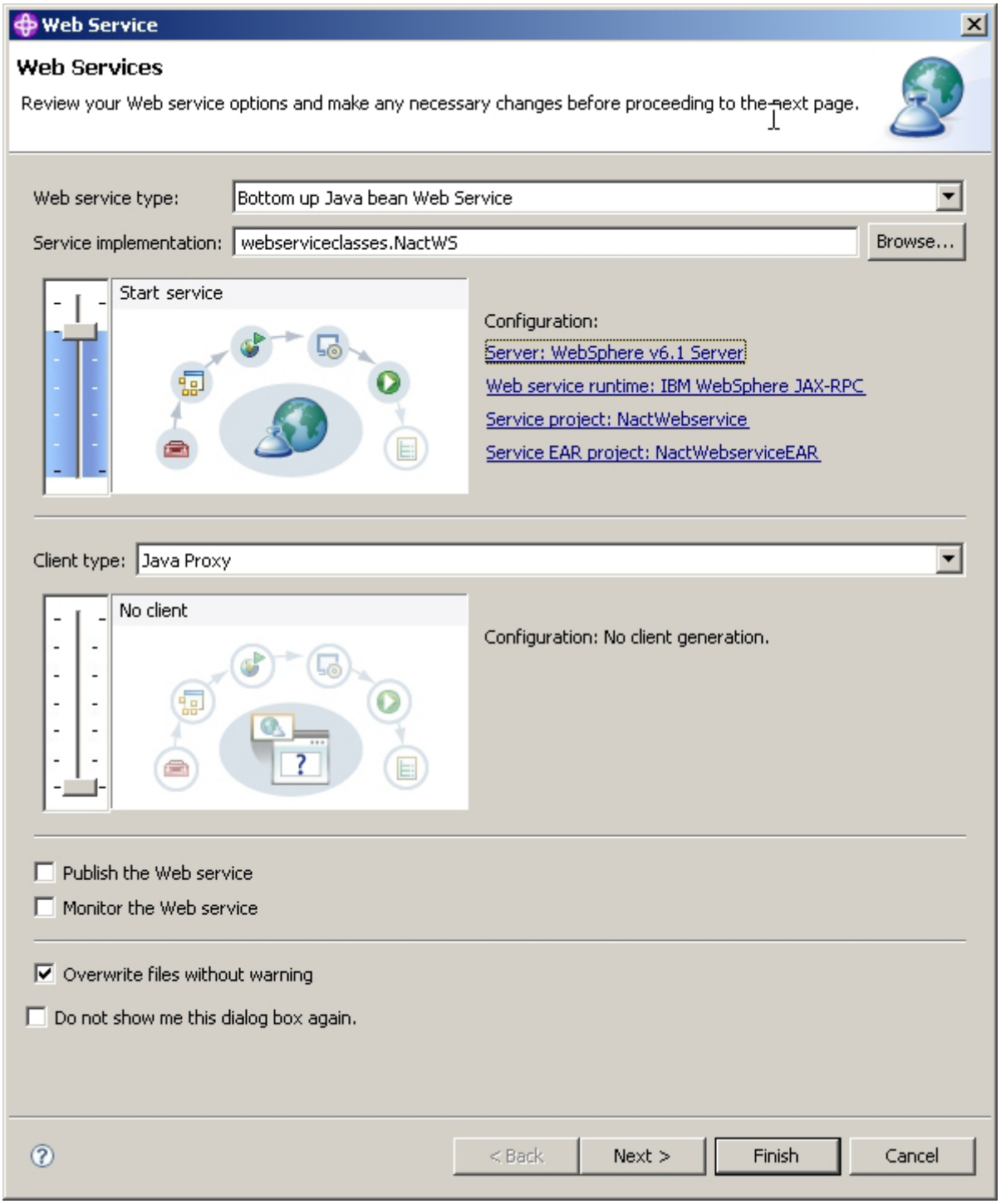

Abbildung 5.63.: HATS-Dialog für die Erstellung eines Web Service

Anfrage (Request) und für die Antwort (Response) beschrieben. Innerhalb des Elements *wsdl:-portType* in Zeile 50 werden die Operationen des Web Service durch das Element *wsdl:operation* definiert und über die Elemente *wsdl:input* und *wsdl:output* mit den vorher definierten Nachrichten in Verbindung gebracht. Da dieser Web Service lediglich aus der durch das Makro bereitgestellten Operation *nACT_DisplayAccProcessWS* besteht, wird in Zeile 51 auch nur ein Element vom Typ *wsdl:operation* benötigt.

Über das nachfolgende Element *wsdl:binding* in Zeile 59 wird das Protokoll und das Nachrichtenformat ausgewählt. Als Protokoll wurde hier SOAP über HTTP und als Nachrichtenformat die im Feld *wsdl:message* definierten Nachrichten verwendet. In Zeile 80 wird im Element *wsdl:service* zuletzt noch die Adresse des Web-Service aufgeführt, damit Clients anhand dieser WSDL-Datei den Web Service finden und benutzen können.

5.3.4. Testen mit dem Web Service Explorer

Um den erstellten Web Service zu testen gibt es die Möglichkeit einen Web Service Client zu erstellen oder den Web Service Explorer der HATS-Entwicklungsumgebung zu benutzen. Der Web Service Explorer kann in der Entwicklungsumgebung über den Menüpunkt „Run/Launch the Web Service Explorer“ oder direkt über einen Rechtsklick auf die gewünschte WSDL-Datei und dem Menüpunkt „Web Services/Test with Web Service Explorer“ aus dem aufklappenden Menü gestartet werden. Wird der Web Service Explorer über die zuerst beschriebene Möglichkeit unabhängig von einer WSDL-Datei gestartet, ist es innerhalb des Web Service Explorers nötig die gewünschte WSDL-Datei über die Auswahl „Open WSDL“ zu öffnen. Bei einem direkten Aufruf über die WSDL-Datei entfällt dieser Schritt. Wird die erstellte WSDL-Datei mit dem Web Service Explorer geöffnet, erscheint innerhalb des Webbrowsers die in Abbildung 5.64 dargestellte Seite.

Die Seite unterteilt sich in die Bereiche „Navigator“, „Actions“ und „Status“. Im „Navigator“-Bereich der Seite sieht der Entwickler alle verfügbaren Operationen des Web Service. Nach der Auswahl einer Operation ändert sich der Bereich „Actions“ und zeigt ein zur Operation des Web Service passendes Formular für die Eingabeparameter an. In der Abbildung 5.64 wird das Eingabeformular für die Eingabeparameter der Operation *nACT_DisplayAccProcessWS* angezeigt. Der Entwickler kann zum Testen Eingabeparameter eingeben und durch einen Mausklick auf die Schaltfläche „Go“ die Anfrage an den Web Service stellen. Im Bereich „Status“ bekommt der Entwickler die gesendete XML-Nachricht im Quelltext angezeigt (vgl. Abbildung 5.65). Die Antwort des Web Service wird ebenfalls als XML-Nachricht im Bereich „Status“ angezeigt (vgl. Abbildung 5.66). Somit hat der Entwickler volle Kontrolle über den Inhalt der Anfrage an den Web Service und dessen Antwort. Dies ist vor allem bei der Fehlersuche von enormem Vorteil.

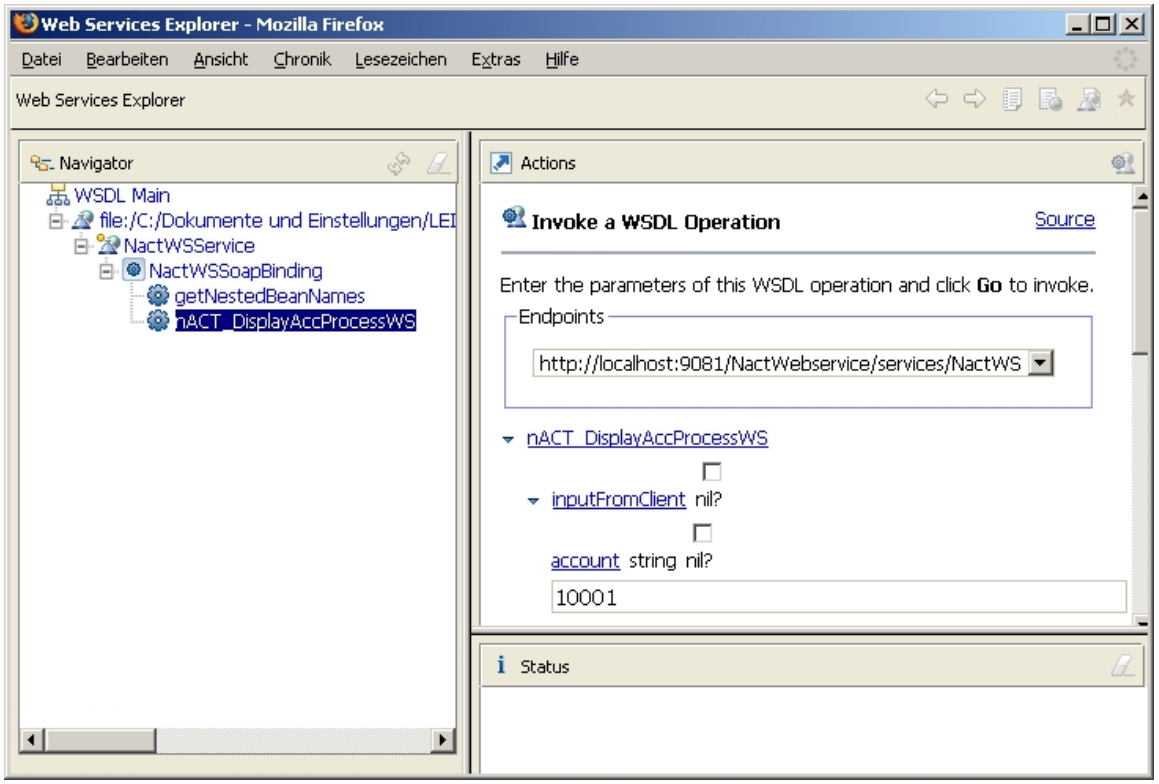

Abbildung 5.64.: HATS Web Service Explorer

Abbildung 5.65.: Anfrage an den Web Service im HATS Web Service Explorer

Abbildung 5.66.: Antwort des Web Service im HATS Web Service Explorer

5.3.5. Erstellung eines Web Service Client

Wurde der Web Service erfolgreich erstellt und mit dem Web Service Explorer getestet, kann der Entwickler zur Benutzung des Web Service einen eigenen Client erstellen. Hierbei kann sich der Entwickler bei der Erstellung des Clients von der Entwicklungsumgebung unterstützen lassen. Dazu existieren verschiedene Möglichkeiten. Über einen Rechtsklick auf die WSDL-Datei *NactWS.wsdl* des Web Service und anschließender Auswahl des Menüpunktes „Web Services/-Generate Client“ im aufklappenden Menü kann sich der Entwickler für den Web Service einen Java Proxy in Form von mehreren Java-Klassen erstellen lassen. Diesen Proxy kann der Entwickler in eigenen Java Anwendungen für den Zugriff auf den Web Service integrieren.

Für diese Arbeit wurde eine Lösung mit JavaServer Pages gewählt, da der Benutzer auf diesem Weg lediglich ein Webbrowser für die Kommunikation mit dem Web Service benötigt. Dazu wird über einen Rechtsklick auf die Datei *NactWS.java*, welche zu den erstellten Web Service Support-Dateien gehört, über den Menüpunkt „Web Services/Generate Samples JSPs“ des aufklappenden Menüs die Oberfläche in Abbildung 5.67 aufgerufen. Damit kann sich der Entwickler durch Auswahl des Feldes „Web service samples JSPs“ in einem wählbaren Ordner im Projektverzeichnis *Web Content* JavaServer Pages erstellen lassen, welche den Zugriff auf den Web Service über ein HTML-Formular ermöglichen. Im Feld „Methods“ legt der Entwickler fest, welche Operationen des Web Service durch den Client aufrufbar sein sollen. Dieses Verfahren erstellt die JSPs *Input.jsp*, *Method.jsp*, *Result.jsp* und *TestClient.jsp*, die das Grundgerüst für einen Web Service Client darstellen und anschließend beliebig im Quelltext angepasst werden können.

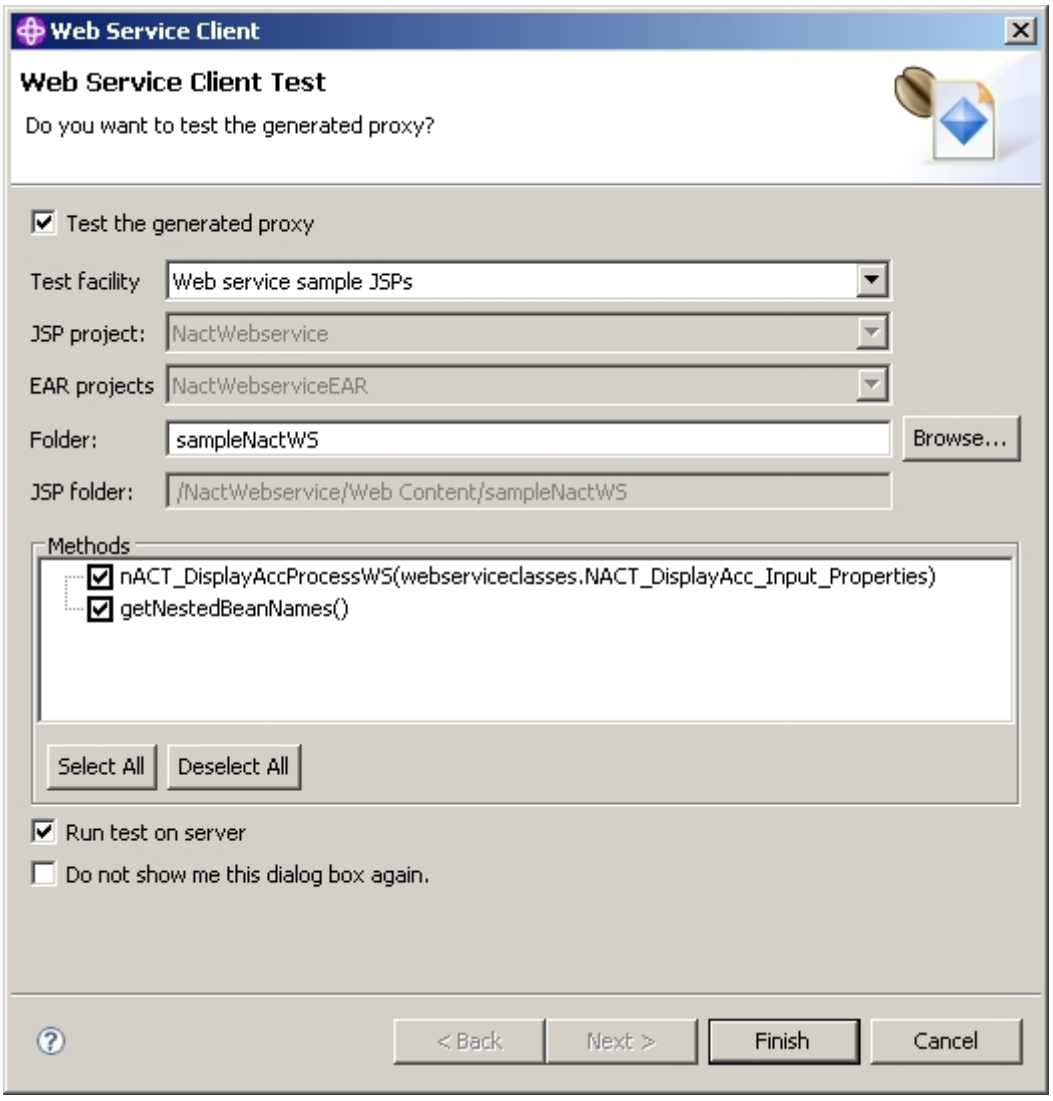

Abbildung 5.67.: HATS-Dialog für die Erstellung eines Web Service Client

Die JSP *Input.jsp* beinhaltet das HTML-Formular für die Eingabeparameter des Web Service. Über die JSP *Method.jsp* werden HTML-Links dargestellt, welche eine Auswahl der auszuführenden Web Service Operation ermöglichen. Je nach Auswahl der Web Service Operation wird das HTML-Formular der JSP *Input.jsp* den Eingabeparametern entsprechend angepasst. Da in dem erstellten Web Service *NactWS* lediglich eine Operation implementiert wurde, wird die JSP *Method.jsp* für den Web Service Client nicht benötigt. Stattdessen wurde in der JSP *Input.jsp* die Variable *method*, welche die momentan durch die JSP *Method.jsp* selektierte Operation enthält, fest auf die Operation *nACT_DisplayAccProcessWS* gesetzt. Während die JSP *Result.jsp* die Antwort des Web Service ausgibt, verbindet die JSP *TestClient.jsp* die JSPs *Input.jsp* und *Result.jsp* über ein HTML-Frameset. Die Einbindung der nicht benötigten JSP *Method.jsp* wurde dabei auskommentiert.

Der erstellte Web Service Client ist in Abbildung 5.68 zu sehen. Nach der Eingabe der CICS-Benutzerkennung, dem zugehörenden CICS-Passwort und der gewünschten Kundennummer genügt ein Mausklick auf die Schaltfläche „Invoke“ um die Anfrage an den Web Service zu stellen. Nach einer kurzen Bearbeitungszeit erscheint das Ergebnis des Web Service im HTML-Format (vgl. Abbildung 5.69). Tritt ein Fehler auf, wird die Fehlermeldung dem Benutzer angezeigt, was in Abbildung 5.70 zu sehen ist. Das Stellen einer erneuten Anfrage ist möglich, sobald eine Antwort erhalten wurde.

In diesem Abschnitt wurde demonstriert, wie sich mit HATS eine Host-Anwendung in einen Web Service kapseln lässt. Alle durch ein Makro beschreibbaren Interaktionen lassen sich mit diesem Verfahren durch einen Web Service umsetzen. Die Komplexität des Makros und damit auch der Erstellung des Web Service steigt proportional mit dem Umfang dieser Interaktionen.

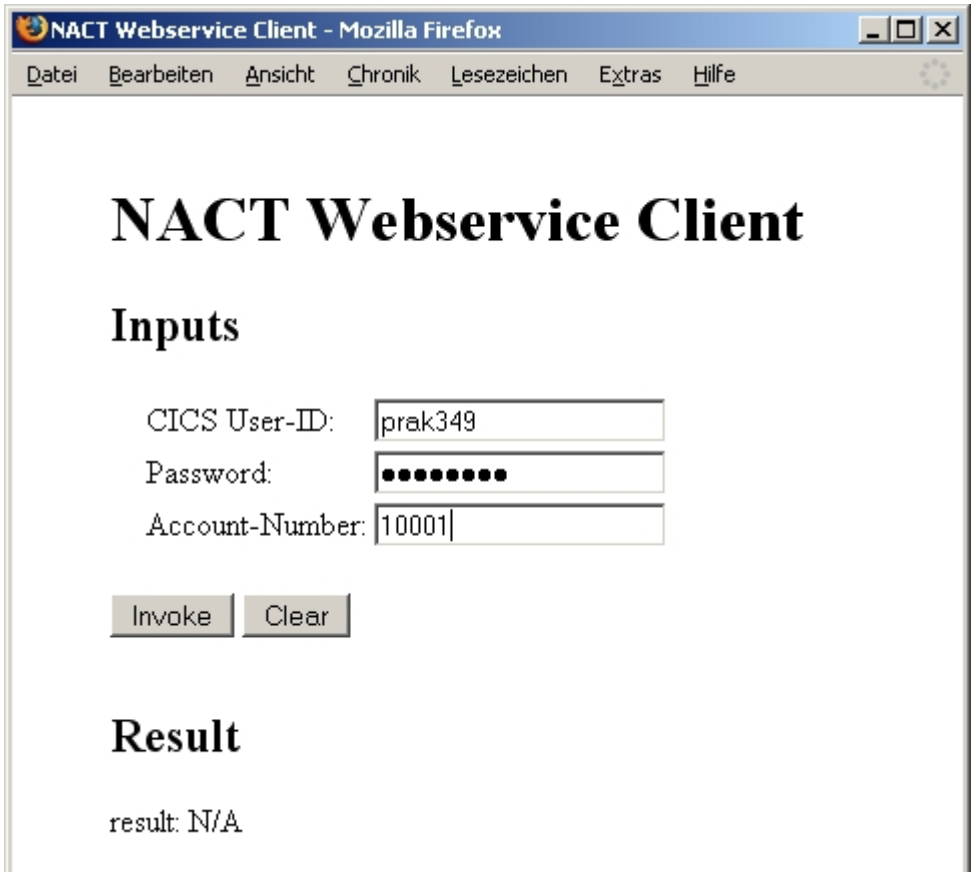

Abbildung 5.68.: Stellen einer Anfrage im Web Service Client

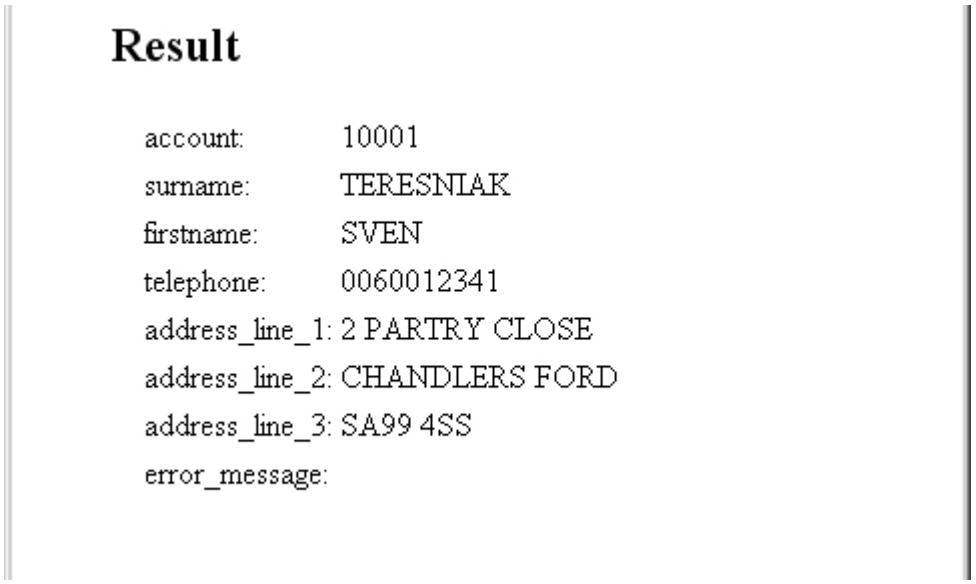

Abbildung 5.69.: Antwort des Web Service im Web Service Client

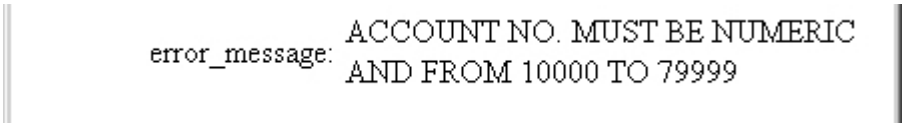

Abbildung 5.70.: Fehlermeldung des Web Service im Web Service Client

6. Zusammenfassung und Ausblick

Mit HATS ist es möglich moderne Oberflächen für 3270- oder 5250-Host-Anwendungen zu erstellen. Um jedoch die Oberfläche optisch ansprechend zu gestalten und dem Anwender die Bedienung zu vereinfachen, sind Anpassungen nötig, wobei die aufzubringende Arbeit von Größe und Struktur der Host-Anwendung abhängig ist. Folgen alle Screens der Host-Anwendung einer einheitlichen Struktur, kann mit wenigen Umsetzungsregeln mit einer Standardtransformation die Oberfläche optisch ansprechend erstellt werden. Ist dies jedoch nicht der Fall, müssen Screens nach der Struktur gruppiert werden, wobei für jede Gruppe eine eigene Screen-Anpassung erstellt werden muss. Wie komplex eine HATS Web-Anwendung werden kann, wurde durch die in Kapitel 5 dokumentierte Erstellung einer Anwendung ersichtlich.

Die erstellte HATS Web-Anwendung ist bezüglich der Performance geringfügig schlechter als die Host-Anwendungen. Datenpakete müssen vom Client zur HATS Web-Anwendung auf dem Websphere Application Server und von diesem weiter zu den Host-Anwendungen gesendet werden. Da die Pakete eine längere Strecke zurücklegen, bedeutet dies auch eine höhere Latenz. Zusätzlich zu dieser Latenz muss die Ausführungszeit der HATS Web-Anwendung auf dem Websphere Application Server gerechnet werden. Da die Gesamtzeit jedoch immer noch im Millisekundenbereich liegt, ist dieser Performanceunterschied für den Benutzer kaum spürbar. Folglich hat er für einen möglichen Einsatz in der Praxis vermutlich keine Bedeutung.

Durch die Erstellung von modernen Oberflächen mit HATS existiert eine attraktive Alternative zur 3270-Terminalemulation. Da sich moderne Oberflächen bei entsprechender Gestaltung intuitiv bedienen lassen, können sich Unternehmen für bestimmte Mitarbeitergruppen, wie zum Beispiel Aushilfskräfte, ausgedehnte Schulungen und Weiterbildungen für die Bedienung von Host-Anwendungen ersparen. Durch zusätzliche Hilfestellung wie zum Beispiel einer Formularvervollständigung kann die Bedienung einer Host-Anwendung sogar beschleunigt werden, wodurch ein effizienteres Arbeiten möglich ist. Ein weiteres wichtiges Argument für moderne Oberflächen ist im immer mobiler werdenden Zeitalter die Möglichkeit, mit dem Mobilfunktelefon oder einem PDA von überall Zugriff auf wichtige Host-Anwendungen und ihre Daten zu bekommen. Wird eine HATS Web-Anwendung als Oberfläche für die Host-Anwendung verwendet, ist clientseitig lediglich ein Webbrowser nötig. Dieser gehört heute schon zur Grundausstattung von vielen mobilen Geräten. Meiner Meinung nach werden mit immer schnelleren und billigeren Zugängen mobile Geräte in Zukunft noch viel mehr genutzt werden.

Mit HATS ist es ebenfalls möglich Web Services zu erstellen, die einen Zugriff auf Host-Anwendungen realisieren. Dies ist besonders für Unternehmen interessant, welche zunehmend auf die Integration und Wiederverwendung bestehender Host-Anwendungen setzen. Neben dem Vorteil der Plattformunabhängigkeit sparen sich Unternehmen durch Web Services Kosten für teure Neuentwicklungen.

Die Verwendung von HATS hat aber nicht nur Vorteile. Da die erstellte HATS Web-Anwendung sehr stark von den zugrundeliegenden Host-Anwendungen abhängig ist, kann es bei Softwareupdates der Host-Anwendungen passieren, dass die Web-Anwendung nicht mehr reibungslos funktioniert. Falls zum Beispiel der Copyright-Hinweis von ISPF geändert wird, wird der Screen von der erstellten Screen-Anpassung nicht mehr korrekt erkannt. Folglich müssen dann die Erkennungskriterien der entsprechenden Screen-Anpassung ebenfalls aktualisiert werden, da ansonsten der Screen über die Standardtransformation dargestellt wird. Anzumerken ist allerdings, dass diese Host-Anwendungen seit Jahren in dieser Form existieren und Änderungen durch Softwareupdates unwahrscheinlich sind.

Auf dieser Arbeit aufbauend, sind weitere Studien- oder Diplomarbeiten denkbar. Der für die NACT-Transaktion erstellte Web Service kann durch zusätzliche Makros auf die komplette NACT-Transaktion ausgeweitet und graphisch aufbereitet werden. Somit wäre über den Web Service ein komplettes Steuern der Transaktion, also auch das Löschen oder Hinzufügen von Datensätzen, möglich.

Es wäre ebenfalls denkbar, in der HATS Web-Anwendung die Transformation für den ISPF-Editor zu verbessern, wodurch dieser mit neu erstellten Komponenten und Widgets als zusammenhängendes Textfeld dargestellt werden könnte. Dieses Textfeld könnte man zusätzlich durch eine auf JavaScript basierende Syntaxhervorhebung für verschiedene Programmiersprachen aufbessern.

Vorstellbar wäre ebenso HATS Web-Anwendungen für immer bedeutender werdendes E-Learning zu verwenden. Für jedes Tutorial des Client/Server-Praktikums könnte eine alleinstehende HATS Web-Anwendung entwickelt werden. Diese könnte auf allen am jeweiligen Tutorial beteiligten Screens Hilfestellungen geben und somit interaktiv durch das Tutorial führen.

Literaturverzeichnis

[CICS1] IBM. CICS Transaction Server for z/OS, CICS Messages and Codes. Version 3. Release 2. http://publib.boulder.ibm.com/infocenter/cicsts/v3r2/topic/com.ibm.cics.ts.messages.doc/pdf/dfhg4c00.pdf

[CICS2] IBM. CICS Transaction Server for z/OS, CICS Supplied Transactions. Version 3. Release 2. http://publib.boulder.ibm.com/infocenter/cicsts/v3r2/topic/com.ibm.cics.ts.doc/pdf/dfha7c00.pdf

[HATS1] IBM. Host Access Transformation Services (HATS), V7 Information Center. http://publib.boulder.ibm.com/infocenter/hatshelp/v71/index.jsp (Stand: Mai 2009)

[HATS2] IBM. HATS Produktbeschreibung. http://www-142.ibm.com/software/dre/ecatalog/detail.wss?locale=de_DE&synkey=C107844U41571U58 (Stand: Mai 2009)

[HATS3] IBM. HATS User's and Administrator's Guide, Version 7.1 http://publib.boulder.ibm.com/infocenter/hatshelp/v71/topic/com.ibm.hats.doc/doc/PDFs/en/uguide.pdf

[HATS4] IBM. HATS Web Application Programmer's Guide, Version 7.1 http://publib.boulder.ibm.com/infocenter/hatshelp/v71/topic/com.ibm.hats.doc/doc/PDFs/en/proggd.pdf

[HATS5] IBM. HATS Advanced Macro Guide, Version 7.1 http://publib.boulder.ibm.com/infocenter/hatshelp/v71/topic/com.ibm.hats.doc/doc/PDFs/en/macro.pdf

[HATS6] IBM. HATS System Requirements. http://www-01.ibm.com/software/awdtools/hats/sysreqs/index4.html (Stand: Mai 2009)

[HATS7] IBM. HATS 7.1 API. http://publib.boulder.ibm.com/infocenter/hatshelp/v71/topic/com.ibm.hats.doc/doc/javadoc/index.html (Stand: Mai 2009)

[JSR1] Java Specification Request. JSR315: Java Servlet Specification 3.0. http://jcp.org/en/jsr/detail?id=315 (Stand: Mai 2009)

[JSR2] Java Specification Request. JSR220: Enterprise JavaBeans 3.0.
http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html
(Stand: Mai 2009)

[JSR3] Java Specification Request. JSR224: Java API for XML-Based Web Services 2.1.
http://jcp.org/aboutJava/communityprocess/maintenance/jsr224/
(Stand: Mai 2009)

[JSR4] Java Specification Request. JSR101: Java APIs for XML based RPC.
http://jcp.org/aboutJava/communityprocess/final/jsr101/index2.html
(Stand: Mai 2009)

[JSR5] Java Specification Request. Frequently asked Questions.
http://www.jcp.org/en/introduction/faq (Stand: Mai 2009)

[HORS] John Horswill. Designing and Programming CICS Applications.
O'Reilly Media. 2000

[IBM1] IBM. System Z.
http://www-03.ibm.com/systems/de/z/ (Stand: Mai 2009)

[IBM2] IBM. Personal Communications.
http://www-01.ibm.com/software/network/pcomm/ (Stand: Mai 2009)

[IBM3] IBM. System i Access.
http://www-03.ibm.com/systems/i/software/access/ (Stand: Mai 2009)

[IBM4] IBM. z/OS Operating System.
http://www-03.ibm.com/systems/z/os/zos/ (Stand: Mai 2009)

[IBM5] IBM. z/VM Operating System. http://www.vm.ibm.com/ (Stand: Mai 2009)

[IBM6] IBM. Linux on IBM System Z.
http://www-03.ibm.com/systems/z/os/linux/ (Stand: Mai 2009)

[IBM7] IBM. z/OS V1R5.0 UNIX System Services User's Guide.
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/bpxza440/
CCONTENTS (Stand: Mai 2009)

[IBM8] IBM. Rational Software Architect Produktseite.
http://www-01.ibm.com/software/awdtools/architect/swarchitect/
(Stand: Mai 2009)

[IBM9] IBM. Russell Butek, Nicholas Gallardo. JAX-RPC versus JAX-WS.
http://www.ibm.com/developerworks/webservices/library/
ws-tip-jaxwsrpc.html (Stand: Mai 2009)

[RED1] IBM. Web Services Handbook for WebSphere Application Server 6.1.
http://www.redbooks.ibm.com/abstracts/sg247257.html
(Stand: Mai 2009)

[RED2] IBM. WebSphere Application Server V6.1 Security Handbook.
http://www.redbooks.ibm.com/redpieces/abstracts/sg246316.html
(Stand: Mai 2009)

[RFC1] IEEE. Request for Comment 1576. TN3270 Current Practices.
http://tools.ietf.org/html/rfc1576 (Stand: Mai 2009)

[SLES] SUSE. SUSE Linux Enterprise Server Partner Products.
http://www.novell.com/partnerguide/section/481.html
(Stand: Mai 2009)

[SPRU] Wilhelm Spruth. Vorlesungsskript: Client/Server Systeme. Teil 10.
Universität Tübingen. 2006

[SUN1] Sun Microsystems.
Java Technology Java 2 Platform Enterprise Edition (J2EE) Overview.
http://java.sun.com/j2ee/overview.html (Stand: Mai 2009)

[SUN2] Sun Microsystems. The J2EE 1.4 Tutorial.
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Overview5.html
(Stand: Mai 2009)

[SUN3] Sun Microsystems. JavaServer Pages Technology FAQ.
http://java.sun.com/products/jsp/faq.html (Stand: Mai 2009)

[STAR1] Thomas Stark. J2EE. Markt und Technik Verlag. 2005

[STAR2] Karsten Samaschke, Thomas Stark. Das J2EE Codebook.
Addison-Wesley Verlag. 2007

[TEUF] Michael Teuffel. TSO, Time Sharing Option im Betriebssystem z/OS MVS.
7. Auflage. Oldenbourg. 2001

[W3C1] W3C. SOAP Version 1.2.
http://www.w3.org/TR/soap12-part1/ (Stand: Mai 2009)

[W3C2] W3C. Web Services Description Language 1.1.
http://www.w3.org/TR/wsdl (Stand: Mai 2009)

[W3C3] W3C. XML Schema Part 1: Structures Second Edition.
http://www.w3.org/TR/xmlschema-1/ (Stand: Mai 2009)

[WAS1] IBM. Feature Pack for Web Services for WebSphere Application Server V6.1.
http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg21264563
(Stand: Mai 2009)

[WAS2] IBM. WebSphere Application Server Produktlinie.
http://www-01.ibm.com/software/webservers/appserv/wasproductline/
(Stand: Mai 2009)

[WAS3] IBM. System Requirements for IBM WebSphere Application Server V6.1 for z/Linux.
http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27007673
(Stand: Mai 2009)

[WAS4] IBM. WebSphere Application Server Technote #1251265.
http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg21251265
(Stand: Mai 2009)

[WAS5] IBM. Application Server Network Deployment for Distributed Platforms
Version 6.1 (wird mit dem Produkt ausgeliefert)

[WIK1] Wikipedia, die freie Enzyklopädie. Model 1.
http://en.wikipedia.org/wiki/Model_1 (Stand: März 2009)

[WIK2] Wikipedia, die freie Enzyklopädie. SOAP Protokoll.
http://de.wikipedia.org/wiki/SOAP (Stand: März 2009)

[WIK3] Wikipedia, die freie Enzyklopädie. Browserweiche.
http://de.wikipedia.org/wiki/Browserweiche (Stand: April 2009)

[X3270] Jeff Sparkes, Paul Mattes. X3270, Open-Source 3270 Terminalemulator.
http://x3270.bgp.nu/

A. Code

A.1. ChangePassword.java

```
1  import com.ibm.HostPublisher.IntegrationObject.*;
2  import com.ibm.HostPublisher.Server.Ras;
3
4  public class ChangePassword extends HPubHODCommon
5  {
6    ...
7
8    // Generated Objects
9    HAOVariable username = new HAOVariable(...);
10   HAOVariable old_password = new HAOVariable(...);
11   HAOVariable new_password = new HAOVariable(...);
12   HAOVariable new_password2 = new HAOVariable(...);
13   HAOVariable errMessage = new HAOVariable(...);
14
15   // Constructor
16   public ChangePassword()
17   { // Generated vector population
18     vHAOVariables.addElement(username);
19     vHAOVariables.addElement(old_password);
20     vHAOVariables.addElement(new_password);
21     vHAOVariables.addElement(new_password2);
22     vHAOVariables.addElement(errMessage);
23     hPubStartPoolName = new String("main");
24     hPubStartChainName = null;
25     hPubStartType = BEAN_START_FROM_POOL;
26     hPubEndChainName = null;
27     hPubEndType = BEAN_END_TO_POOL;
28     stringHODMacroFilename = "ChangePassword.hma";
29     stringBeanName = new String("ChangePassword");
30     hPubBeanName = new String("ChangePassword");
31   }
32
33   /* Generated Setters */
34   public void setUsername( String stringToBe)
35   {
36     username.stringValue = stringToBe;
37   }
38   public void setOld_password( String stringToBe)
39   {
40     old_password.stringValue = stringToBe;
41   }
```

```
42   public void setNew_password( String stringToBe)
43   {
44     new_password.stringValue = stringToBe;
45   }
46   public void setNew_password2( String stringToBe)
47   {
48     new_password2.stringValue = stringToBe;
49   }
50   public void setErrMessage( String stringToBe)
51   {
52   }
53
54   /* Generated Getters */
55   public String getUsername()
56   {
57      return (username.stringValue);
58   }
59   public String getOld_password()
60   {
61      return (old_password.stringValue);
62   }
63   public String getNew_password()
64   {
65      return (new_password.stringValue);
66   }
67   public String getNew_password2()
68   {
69      return (new_password2.stringValue);
70   }
71   public String getErrMessage()
72   {
73      return (errMessage.stringValue);
74   }
75
76   public void DoTheWork()
77   {
78     DoTheWork(this);
79   }
80
81 }
```

Listing A.1: ChangePassword.java

A.2. Komponente OptionList.java

```
1  package csprak.components;
2
3  import java.util.Properties;
4  import java.util.ResourceBundle;
5  import java.util.Vector;
6
7  import com.ibm.hats.common.HCustomProperty;
8  import com.ibm.hats.transform.actions.ScriptAction;
9  import com.ibm.hats.transform.actions.SetFormValue;
10 import com.ibm.hats.transform.actions.SubmitFormAction;
11 import com.ibm.hats.transform.components.SelectionListComponent;
12 import com.ibm.hats.transform.elements.ComponentElement;
13 import com.ibm.hats.transform.elements.ListItemComponentElement;
14 import com.ibm.hats.transform.regions.BlockScreenRegion;
15 import com.ibm.hsr.screen.HsrScreen;
16
17 public class OptionList extends SelectionListComponent {
18
19   private HsrScreen sc;
20
21   private int dflt_row_input = 22;
22   private int dflt_col_input = 14;
23   private int dflt_length_input = 2;
24
25   private int dflt_item_start = 2;
26   private int dflt_item_end = 4;
27   private int dflt_descr_start = 5;
28   private int dflt_descr_end = 17;
29   private int dflt_fullcap_start = 19;
30   private int dflt_fullcap_end = 55;
31
32   public OptionList(HsrScreen arg0) {
33     super(arg0);
34     this.sc = arg0;
35   }
36
37   @Override
38   public ComponentElement[] recognize(BlockScreenRegion region, Properties settings) {
39
40     // Einstellung der Komponente auslesen
41     int row_input = Integer.parseInt(settings.getProperty("row_input", String.valueOf(
         dflt_row_input)));
42     int col_input = Integer.parseInt(settings.getProperty("col_input", String.valueOf(
         dflt_col_input)));
43     int length_input = Integer.parseInt(settings.getProperty("length_input", String.
         valueOf(dflt_length_input)));
44     int item_start = Integer.parseInt(settings.getProperty("item_start", String.valueOf
```

```
        (dflt_item_start)));
45      int item_end = Integer.parseInt(settings.getProperty("item_end", String.valueOf(
            dflt_item_end)));
46      int descr_start = Integer.parseInt(settings.getProperty("descr_start", String.
            valueOf(dflt_descr_start)));
47      int descr_end = Integer.parseInt(settings.getProperty("descr_end", String.valueOf(
            dflt_descr_end)));
48      int fullcap_start = Integer.parseInt(settings.getProperty("fullcap_start", String.
            valueOf(dflt_fullcap_start)));
49      int fullcap_end = Integer.parseInt(settings.getProperty("fullcap_end", String.
            valueOf(dflt_fullcap_end)));
50
51      // Auswahlbereich des Screens auslesen
52      int startRow = region.startRow();
53      int endRow = region.endRow();
54      int startCol = region.startCol();
55      int endCol = region.endCol();
56      int numCols = endCol - startCol + 1;
57      int numRows = endRow - startRow + 1;
58
59      // Zeichenketten aus Screen im Auswahlbereich in die Variable buffer extrahieren
60      int bufferLength = numRows * numCols;
61      char[] buffer = new char[bufferLength];
62      sc.getScreenRect(buffer, bufferLength, startRow, startCol, endRow, endCol, com.ibm.
            hsr.screen.HsrScreen.VISIBLE_TEXT_ONLY);
63
64      // Komponente soll immer erkannt werden
65      boolean recognized = true;
66
67      // Elemente einer Zeile im Auswahlbereich
68      char[] citem = new char[item_end - item_start + 1];
69      char[] cdescription = new char[descr_end - descr_start + 1];
70      char[] cfullcaption = new char[fullcap_end -
71                                     fullcap_start + 1];
72      char[] tmp = null;
73      String item = "";
74      String description = "";
75      String fullcaption = "";
76
77      ComponentElement[] ce = new ComponentElement[numRows];
78      ListItemComponentElement lice;
79      int component_counter = 0;
80
81      int i=0;
82      while (i < bufferLength) {
83
84
85        // Abkürzung aus Buffer auslesen
86        for (int j=0; j < citem.length; j++) {
```

```
87          if (i+j+item_start < bufferLength)
88            citem[j] = buffer[i + j + item_start-2];
89        }
90
91        // Kurzbeschreibung aus Buffer auslesen
92        for (int j=0; j < cdescription.length; j++) {
93          if (i+j+descr_start < bufferLength)
94            cdescription[j] = buffer[i + j + descr_start-2];
95        }
96
97        // Beschreibung aus Buffer auslesen
98        for (int j=0; j < cfullcaption.length; j++) {
99          if (i+j+fullcap_start < bufferLength)
100           cfullcaption[j] = buffer[i + j + fullcap_start-2];
101       }
102
103       item = new String(citem);
104       description = new String(cdescription);
105       fullcaption = new String(cfullcaption);
106
107       // Komponenten erstellen und einem Vektor hinzufügen
108       lice = new ListItemComponentElement(item, description, fullcaption);
109       Vector<ScriptAction> v = new Vector<ScriptAction>();
110       v.add(new SetFormValue(lice.getItem(), convertRowColToPos(row_input, col_input,
              sc.getSizeCols()), length_input));
111       v.add(new SubmitFormAction());
112       lice.setOnSelectActions(v);
113       ce[component_counter++] = lice;
114
115       // Springe in nächste Zeile
116       i += numCols;
117     }
118
119     // Vektor in Array umwandeln
120     if (recognized) {
121       ComponentElement[] res = new ComponentElement[component_counter];
122       for (int q=0; q < component_counter; q++) {
123         res[q] = ce[q];
124       }
125       return res;
126     }
127     else {
128       return null;
129     }
130
131   }
132
133   public int getPropertyPageCount() {
134     return (1);
```

```
135   }
136
137   // Komponenteneinstellungen verfügbar machen
138   public Vector getCustomProperties(int iPageNumber, Properties properties,
          ResourceBundle bundle) {
139     Vector<HCustomProperty> v = new Vector<HCustomProperty>();
140     v.add(new HCustomProperty("row_input", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, " Reihe der Kommando-Box", true, null, null, String.
            valueOf(dflt_row_input), null, null));
141     v.add(new HCustomProperty("col_input", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "Spalte der Kommando-Box", true, null, null, String.
            valueOf(dflt_col_input), null, null));
142     v.add(new HCustomProperty("length_input", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "Länge der Kommando-Box", true, null, null, String.
            valueOf(dflt_length_input), null, null));
143     v.add(new HCustomProperty("seperator1", com.ibm.hats.common.HCustomProperty.
            TYPE_SEPARATOR, "", false, null, null, "", null, null));
144     v.add(new HCustomProperty("item_start", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "Spalte des Kürzels in der Auswahlbox", true, null, null
            , String.valueOf(dflt_item_start), null, null));
145     v.add(new HCustomProperty("item_end", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "End-Spalte des Kürzels in der Auswahlbox", true, null,
            null, String.valueOf(dflt_item_end), null, null));
146     v.add(new HCustomProperty("descr_start", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "Spalte der Kurzbeschreibung in der Auswahlbox", true,
            null, null, String.valueOf(dflt_descr_start), null, null));
147     v.add(new HCustomProperty("descr_end", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "End-Spalte der Kurzbeschreibung in der Auswahlbox",
            true, null, null, String.valueOf(dflt_descr_end), null, null));
148     v.add(new HCustomProperty("fullcap_start", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "Spalte der Beschreibung in der Auswahlbox", true, null,
             null, String.valueOf(dflt_fullcap_start), null, null));
149     v.add(new HCustomProperty("fullcap_end", com.ibm.hats.common.HCustomProperty.
            TYPE_INT_NOT_NEGATIVE, "End-Spalte der Beschreibung in der Auswahlbox", true,
            null, null, String.valueOf(dflt_fullcap_end), null, null));
150     return (v);
151   }
152
153   public Properties getDefaultValues(int iPageNumber) {
154     return (super.getDefaultValues(iPageNumber));
155   }
156 }
```

Listing A.2: Komponente OptionList.java

A.3. Widget OptionListExtended.java

```
1  package csprak.widgets;
2
3  import java.util.Properties;
4
5  import com.ibm.hats.transform.elements.ComponentElement;
6  import com.ibm.hats.transform.elements.ListItemComponentElement;
7  import com.ibm.hats.transform.html.HTMLElementFactory;
8  import com.ibm.hats.transform.html.LinkElement;
9  import com.ibm.hats.transform.renderers.HTMLRenderer;
10 import com.ibm.hats.transform.widgets.Widget;
11
12 public class OptionListExtended extends Widget implements HTMLRenderer {
13
14   private ComponentElement[] ce;
15
16   public OptionListExtended(ComponentElement[] arg0, Properties arg1) {
17     super(arg0, arg1);
18     this.ce = arg0;
19   }
20
21   public StringBuffer drawHTML() {
22     StringBuffer buffer = new StringBuffer(256);
23     HTMLElementFactory factory = HTMLElementFactory.newInstance(
24         contextAttributes, settings);
25
26     ListItemComponentElement lice;
27     LinkElement le;
28     for (int i=0; i < ce.length; i++) {
29       lice = (ListItemComponentElement) ce[i];
30       if (lice.getItem().length() == 1)
31         buffer.append(lice.getItem() + " - ");
32       else
33         buffer.append(" " + lice.getItem() + " - ");
34       le = factory.createLink(lice);
35       le.setClassName("HATSLINK");
36       le.render(buffer);
37       le.renderEndTag(buffer);
38       buffer.append(" - " + lice.getFullCaption() + "<br />");
39     }
40
41     return (buffer);
42   }
43
44 }
```

Listing A.3: Widget OptionListExtended.java

A.4. Komponente DataSetList.java

```
1  package csprak.components;
2
3  public class DataSetList extends Component {
4
5    private HsrScreen sc;
6    private int dsInputStart = 2;
7    private int dsInputEnd = 8;
8    private int dsNameStart = 9;
9    private int dsNameEnd = 49;
10   private int numTableCols = 2;
11   private int col1Start = 51;
12   private int col1End = 71;
13   private int col2Start = 73;
14   private int col2End = 80;
15   private int col3Start = 0;
16   private int col3End = 0;
17   private int col4Start = 0;
18   private int col4End = 0;
19   private String optionsEnum = "B=Browse;CO=Copy;D=Delete;E=Edit;
20                               F=Free;I=Information;M=Member List; MO=Move;
21                               R=Rename;S=Short Info;V=View;Z=Compress";
22
23   public DataSetList(HsrScreen arg0) {
24     super(arg0);
25     this.sc = arg0;
26   }
27
28   @Override
29   public ComponentElement[] recognize(BlockScreenRegion region, Properties settings) {
30
31     // Einstellungen der Komponente auslesen
32     this.dsInputStart = Integer.parseInt(settings.getProperty("dsInputStart", String.
         valueOf(dsInputStart)));
33     this.dsInputEnd = Integer.parseInt(settings.getProperty("dsInputEnd", String.
         valueOf(dsInputEnd)));
34     this.dsNameStart = Integer.parseInt(settings.getProperty("dsNameStart", String.
         valueOf(dsNameStart)));
35     this.dsNameEnd = Integer.parseInt(settings.getProperty("dsNameEnd", String.valueOf(
         dsNameEnd)));
36     this.numTableCols = Integer.parseInt(settings.getProperty("numTableCols", String.
         valueOf(numTableCols)));
37     this.col1Start = Integer.parseInt(settings.getProperty("col1Start", String.valueOf(
         col1Start)));
38     this.col1End = Integer.parseInt(settings.getProperty("col1End", String.valueOf(
         col1End)));
39     this.col2Start = Integer.parseInt(settings.getProperty("col2Start", String.valueOf(
         col2Start)));
```

```
40      this.col2End = Integer.parseInt(settings.getProperty("col2End", String.valueOf(
            col2End)));
41      this.col3Start = Integer.parseInt(settings.getProperty("col3Start", String.valueOf(
            col3Start)));
42      this.col3End = Integer.parseInt(settings.getProperty("col3End", String.valueOf(
            col3End)));
43      this.col4Start = Integer.parseInt(settings.getProperty("col4Start", String.valueOf(
            col4Start)));
44      this.col4End = Integer.parseInt(settings.getProperty("col4End", String.valueOf(
            col4End)));
45      this.optionsEnum = settings.getProperty("optionsEnum", optionsEnum);
46
47      char[] dsName;
48      char[] col1;
49      char[] col2;
50      char[] col3;
51      char[] col4;
52
53      Vector<ComponentElement> ces = new Vector<ComponentElement>();
54      InputComponentElement ice;
55      SelectionComponentElement sce;
56
57      // Options-Feld auftrennen
58      String[] options = optionsEnum.split(";");
59      String[] option;
60
61      ComponentElementList cel = new ComponentElementList();
62      for (int i=0; i < options.length; i++) {
63        option = options[i].split("=");
64        cel.addElement(new ListItemComponentElement(option[0], option[1], option[1]));
65      }
66
67      // Koordinaten des Screen-Bereichs auslesen
68      int startRow = region.startRow();
69      int endRow = region.endRow();
70      int startCol = region.startCol();
71      int endCol = region.endCol();
72      int numCols = endCol - startCol + 1;
73      int numRows = endRow - startRow + 1;
74
75      // Screen in buffer extrahieren
76      int bufferLength = numRows * numCols;
77      char[] buffer = new char[bufferLength];
78      sc.getScreenRect(buffer, bufferLength, startRow, startCol, endRow, endCol, com.ibm.
            hsr.screen.HsrScreen.VISIBLE_TEXT_ONLY);
79
80
81      // Screen zeilenweise parsen nach dem Schema:
82      // [input] [data set name] [column 1] [column 2]
```

```
83      for (int i=0; i < bufferLength; i += numCols) {
84
85        // Char-Arrays mit passender Größe initalisieren
86        dsName = new char[dsNameEnd - dsNameStart];
87        col1 = new char[col1End - col1Start];
88        col2 = new char[col2End - col2Start];
89
90        for (int j=dsNameStart; j < dsNameEnd; j++) {
91          dsName[j-dsNameStart] = buffer[i+j];
92        }
93
94        for (int j=col1Start; j < col1End; j++) {
95          col1[j-col1Start] = buffer[i+j];
96        }
97
98        for (int j=col2Start; j < col2End; j++) {
99          col2[j-col2Start] = buffer[i+j];
100       }
101
102       // Eingabefeld mit Auswahlliste verbinden
103       ice = new InputComponentElement("", convertRowColToPos(i/numCols+startRow,
              dsInputStart, sc.getSizeCols()), dsInputEnd-dsInputStart);
104       sce = new SelectionComponentElement(ice, cel);
105
106       ces.add(sce);
107       ces.add(new TextComponentElement(new String(dsName)));
108       ces.add(new TextComponentElement(new String(col1)));
109       ces.add(new TextComponentElement(new String(col2)));
110
111       // optional zusätzliche Spalten lesen
112       if (numTableCols == 4) {
113         col3 = new char[col3End - col3Start];
114         col4 = new char[col4End - col4Start];
115
116         for (int j=col3Start; j < col3End; j++) {
117           col3[j-col3Start] = buffer[i+j];
118         }
119
120         for (int j=col4Start; j < col4End; j++) {
121           col4[j-col4Start] = buffer[i+j];
122         }
123
124         ces.add(new TextComponentElement(new String(col3)));
125         ces.add(new TextComponentElement(new String(col4)));
126       }
127     }
128
129
130     // Vector in Array umwandeln
```

```
131     ComponentElement[] res = new ComponentElement[ces.size()];
132     for (int i=0; i < ces.size(); i++) {
133       res[i] = (ComponentElement) ces.elementAt(i);
134     }
135 
136     return res;
137   }
138 
139   public Vector getCustomProperties(int iPageNumber, Properties properties,
          ResourceBundle bundle) {
140     Vector<HCustomProperty> v = new Vector<HCustomProperty>();
141     // ...
142     return (v);
143   }
144 
145   @Override
146   public ComponentElement[] recognize(LinearScreenRegion region,
147       Properties settings) {
148     return null;
149   }
150 
151   public int getPropertyPageCount() {
152     return (1);
153   }
154 
155   public Properties getDefaultValues(int iPageNumber) {
156     return (super.getDefaultValues(iPageNumber));
157   }
158 
159 }
```

Listing A.4: Komponente DataSetList.java

A.5. Widget DataSetList4Columns.java

```
1  package csprak.widgets;
2
3  public class DataSetList4Columns extends Widget implements HTMLRenderer {
4
5    private ComponentElement[] ce;
6    private String headCol1;
7    private String headCol2;
8    private String headCol3;
9    private String headCol4;
10   private String colorRowEven = "#FF0000";
11   private String colorRowOdd = "#00FF00";
12
13   public DataSetList4Columns(ComponentElement[] arg0, Properties arg1) {
14     super(arg0, arg1);
15     this.ce = arg0;
16   }
17
18   public StringBuffer drawHTML() {
19
20     this.headCol1 = settings.getProperty("headCol1", headCol1);
21     this.headCol2 = settings.getProperty("headCol2", headCol2);
22     this.headCol3 = settings.getProperty("headCol3", headCol3);
23     this.headCol4 = settings.getProperty("headCol4", headCol4);
24     this.colorRowEven = settings.getProperty("colorRowEven", colorRowEven);
25     this.colorRowOdd = settings.getProperty("colorRowOdd", colorRowOdd);
26
27     StringBuffer buffer = new StringBuffer();
28     HTMLElementFactory factory = HTMLElementFactory.newInstance(contextAttributes,
           settings);
29
30     SelectionComponentElement sce;
31     SelectElement se;
32     TextComponentElement tce;
33     InputElement ie;
34     FieldComponentElement fce;
35
36     buffer.append("<table width=100% class=\"styled\"><tr>");
37     buffer.append("<th width=20%></th><th width=40%></th>");
38     buffer.append("<th width=10%>" + headCol1 + "</th><th width=10%>" + headCol2 + "</
           th>");
39     buffer.append("<th width=10%>" + headCol3 + "</th><th width=10%>" + headCol4 + "</
           th>");
40     buffer.append("</tr><tr style=\"background-color:" + colorRowEven + ";\">");
41
42     for (int i=0; i < ce.length; i++) {
43       if (i == 0) {
44         buffer.append("<td width=20% align=\"right\">");
```

```
45          sce = (SelectionComponentElement) ce[i];
46          fce = sce.getField();
47          ie = factory.createTextInput(fce);
48          ie.render(buffer);
49          se = factory.createDropdown(sce, "");
50          se.render(buffer);
51          buffer.append("</td>");
52        }
53        else if (i % 6 == 1) {
54          buffer.append("<td width=40%>");
55          tce = (TextComponentElement) ce[i];
56          buffer.append(tce.getText());
57          buffer.append("</td>");
58        }
59        else if (i % 6 == 0) {
60
61          // Abwechselnde Hintergrundfarbe
62          if (i % 12 == 0)
63            buffer.append("</tr><tr style=\"background-color: " + colorRowEven + ";\">");
64          else
65            buffer.append("</tr><tr style=\"background-color: " + colorRowOdd + ";\">");
66
67          buffer.append("<td width=20% align=\"right\">");
68          sce = (SelectionComponentElement) ce[i];
69          fce = sce.getField();
70          ie = factory.createTextInput(fce);
71          ie.render(buffer);
72          se = factory.createDropdown(sce, "");
73          se.render(buffer);
74          buffer.append("</td>");
75        }
76        else {
77          buffer.append("<td width=10% style=\"text-align: center;\">");
78          tce = (TextComponentElement) ce[i];
79          buffer.append(tce.getText());
80          buffer.append("</td>");
81        }
82
83      }
84      buffer.append("</tr></table>");
85
86      return (buffer);
87    }
88
89    public int getPropertyPageCount() {
90      return (1);
91    }
92
93    public Vector getCustomProperties(int iPageNumber, Properties properties,
```

```
      ResourceBundle bundle) {
94    Vector<HCustomProperty> v = new Vector<HCustomProperty>();
95    v.add(new HCustomProperty("headCol1", com.ibm.hats.common.HCustomProperty.
          TYPE_STRING, "Überschrift Spalte 1", true, null, null, headCol1, null, null));
96    v.add(new HCustomProperty("headCol2", com.ibm.hats.common.HCustomProperty.
          TYPE_STRING, "Überschrift Spalte 2", true, null, null, headCol2, null, null));
97    v.add(new HCustomProperty("headCol3", com.ibm.hats.common.HCustomProperty.
          TYPE_STRING, "Überschrift Spalte 3", true, null, null, headCol3, null, null));
98    v.add(new HCustomProperty("headCol4", com.ibm.hats.common.HCustomProperty.
          TYPE_STRING, "Überschrift Spalte 4", true, null, null, headCol4, null, null));
99    v.add(new HCustomProperty("seperator", com.ibm.hats.common.HCustomProperty.
          TYPE_SEPARATOR, "", false, null, null, "", null, null));
100   v.add(new HCustomProperty("colorRowEven", com.ibm.hats.common.HCustomProperty.
          TYPE_STRING, "Farbe Reihe gerade", true, null, null, colorRowEven, null, null))
          ;
101   v.add(new HCustomProperty("colorRowOdd", com.ibm.hats.common.HCustomProperty.
          TYPE_STRING, "Farbe Reihe ungerade", true, null, null, colorRowOdd, null, null)
          );
102   return v;
103 }
104
105 public Properties getDefaultValues(int iPageNumber) {
106   return (super.getDefaultValues(iPageNumber));
107 }
108
109 }
```

Listing A.5: Widget DataSetList4Columns.java

A.6. NactWS.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ... >

 <wsdl:types>
  <schema ... >

   <complexType name="NACT_DisplayAcc_Input_Properties">
    <sequence>
     <element name="account" nillable="true" type="xsd:string"/>
     <element name="password" nillable="true" type="xsd:string"/>
     <element name="userid" nillable="true" type="xsd:string"/>
     ...
    </sequence>
   </complexType>

   ...

   <complexType name="NACT_DisplayAcc_Output_Properties">
    <sequence>
     <element name="telephone" nillable="true" type="xsd:string"/>
     <element name="account" nillable="true" type="xsd:string"/>
     <element name="firstname" nillable="true" type="xsd:string"/>
     <element name="error_message" nillable="true" type="xsd:string"/>
     <element name="address_line_2" nillable="true" type="xsd:string"/>
     <element name="address_line_3" nillable="true" type="xsd:string"/>
     <element name="address_line_1" nillable="true" type="xsd:string"/>
     <element name="password" nillable="true" type="xsd:string"/>
     <element name="surname" nillable="true" type="xsd:string"/>
     <element name="userid" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>

  </schema>
 </wsdl:types>

 <wsdl:message name="nACT_DisplayAccProcessWSResponse">
   <wsdl:part element="intf:nACT_DisplayAccProcessWSResponse"
              name="parameters"/>
 </wsdl:message>

 <wsdl:message name="nACT_DisplayAccProcessWSRequest">
   <wsdl:part element="intf:nACT_DisplayAccProcessWS"
              name="parameters"/>
 </wsdl:message>


```

```
48
49
50  <wsdl:portType name="NactWS">
51   <wsdl:operation name="nACT_DisplayAccProcessWS">
52    <wsdl:input message="intf:nACT_DisplayAccProcessWSRequest"
53                name="nACT_DisplayAccProcessWSRequest"/>
54    <wsdl:output message="intf:nACT_DisplayAccProcessWSResponse"
55                 name="nACT_DisplayAccProcessWSResponse"/>
56   </wsdl:operation>
57  </wsdl:portType>
58
59  <wsdl:binding name="NactWSSoapBinding" type="intf:NactWS">
60  <wsaw:UsingAddressing xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
        wsdl:required="false"/>
61
62    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/
        >
63
64    <wsdl:operation name="nACT_DisplayAccProcessWS">
65     <wsdlsoap:operation soapAction="nACT_DisplayAccProcessWS"/>
66
67     <wsdl:input name="nACT_DisplayAccProcessWSRequest">
68        <wsdlsoap:body use="literal"/>
69
70     </wsdl:input>
71
72     <wsdl:output name="nACT_DisplayAccProcessWSResponse">
73        <wsdlsoap:body use="literal"/>
74
75     </wsdl:output>
76   </wsdl:operation>
77
78  </wsdl:binding>
79
80  <wsdl:service name="NactWSService">
81    <wsdl:port binding="intf:NactWSSoapBinding" name="NactWS">
82      <wsdlsoap:address location="http://localhost:9081/NactWebservice/services/NactWS"/
          >
83    </wsdl:port>
84  </wsdl:service>
85
86 </wsdl:definitions>
```

Listing A.6: NactWS.wsdl

A.7. Websphere Applications Server Start Script

```
#!/bin/bash
#
# apache
#
# chkconfig: 5 90 10
# description: Start up the WebSphere Application Server.

RETVAL=$?
WAS_HOME="/opt/IBM/WebSphere/AppServer"
case "$1" in
start)
if [ -f $WAS_HOME/bin/start_server1.sh ]; then
echo $"Starting IBM WebSphere Application Server"
$WAS_HOME/bin/start_server1.sh
fi
;;
stop)
if [ -f $WAS_HOME/bin/stopServer.sh ]; then
echo $"Stop IBM WebSphere Application Server"
$WAS_HOME/bin/stopServer.sh server1
fi
;;
status)
if [ -f $WAS_HOME/bin/serverStatus.sh ]; then
echo $"Show status of IBM WebSphere Application Server"
$WAS_HOME/bin/serverStatus.sh server1
fi
;;
*)
echo $"Usage: $0 {start|stop|status}"
exit 1
;;
esac
exit $RETVAL
```

Listing A.7: WAS start_server.sh

B. Übersetzte Fachbegriffe

Ausnahme	Exception
Benutzerkennung	User ID
Benutzerkonten	Accounts
Erkennungskriterium	Screen Recognition Criteria
Schaltfläche	Button
Funktionstasten	Function Keys
Geschäftslogik	Business Logic
Geteilte globale Variablen	Shared Global Variable
Menüleiste	Action Bar
Screen-Anpassung	Screen Customization
Screen-Kombination	Screen Combination
Standardanpassung	Default Rendering

C. Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CICS	Customer Information Control System
CSS	Cascaded Style Sheet
EAR	Enterprise Application Archive
EBCDIC	Enterprise Binary Coded Decimals Interchange Code
EJB	Enterprise Java Bean
GUI	Graphical User Interface
HATS	Host Access Transformation Service
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ISPF	Interactive System Productivity Facility
J2SE	Java 2 Platform Standard Edition
Java EE	Java Platform Enterprise Edition
JCL	Job Control Language
JAR	Java Archive
JCL	Job Control Language

JAR Java Archive

JDBC Java Database Connectivity

JES Job Entry System

JMS Java Messaging Service

JSF JavaServer Faces

JSP JavaServer Page

LPAR Logical Partitions

MVC Model View Controller

RCP Rich Client Platform

RAR Ressource Adapter Archive

SLES SUSE Linux Enterprise Server

SPUFI SQL Processing Using File Input

TLD Tag Library Descriptor

TSO Time Sharing Option

UDDI Universal Description, Discovery and Integration

URL Uniform Resource Locator

USS Unix System Services

WAR Web Application Archive

WAS Websphere Application Server

WSDL Web Service Description Language

XML Extensible Markup Language

D. Inhalt der beigefügten DVDs

Im Anhang der Diplomarbeit befinden sich die drei DVDs mit den Beschriftungen DVD1, DVD2 und DVD3.

In den Verzeichnissen VM1 auf DVD1, VM2 auf DVD2 und VM3 auf DVD3 befindet sich verteilt auf die drei DVDs eine virtuelle Maschine mit der für diese Arbeit verwendeten HATS-Entwicklungsumgebung unter Windows XP Professional. Um die virtuelle Maschine benutzen zu können, müssen alle Dateien aus den genannten Ordnern in einen Ordner auf dem lokalen PC kopiert werden. Anschließend kann die virtuelle Maschine mit dem VMWare Player gestartet werden, welcher sich im Ordner *Software/VMWare Player 2.5.2* auf DVD1 befindet. Beim Starten der virtuellen Maschine werden Sie eventuell gefragt, ob die virtuelle Maschine kopiert oder verschoben wurde. Wählen Sie in diesem Fall die Option „I copied it“ aus.

Um die virtuelle Maschine kompakt zu halten, wurde die Nutzung von virtuellem Speicher deaktiviert. Für eine effektive Arbeit sollte der virtuelle Speicher im Gastsystem aktiviert werden. Öffnen Sie dazu innerhalb der virtuellen Maschine die Systemsteuerung und klicken auf das Icon „System“. Wechseln Sie nun in den Reiter „Erweitert“ und klicken im Abschnitt „Systemleistung“ auf „Einstellungen“. Wählen Sie in dem neuen Fenster den Reiter „Erweitert“ und klicken Sie im Bereich „Virtueller Arbeitsspeicher“ auf die Schaltfläche „Ändern“. Aktivieren Sie die Option „Größe wird vom System verwaltet“ und bestätigen Sie die Auswahl durch die Schaltfläche „Festlegen“ und einem anschließendem Klick auf „OK“. Nach einem Neustart der virtuellen Maschine ist der virtuelle Speicher aktiviert.

In dem Verzeichnis *Software* auf der DVD1 befinden sich der VMWare Player, die Installationsdateien des WebSphere Application Servers ND 6.1 für z/Linux, sowie die Installationsdateien der HATS-Enwicklungsumgebung inklusive Runtime-Lizenz.

Im Verzeichnis *Quellen* auf der DVD2 befinden sich die für diese Diplomarbeit verwendeten und zitierten Online-Quellen. Im Verzeichnis *Tutorial* auf derselben DVD befindet sich ein Tutorial, welches eine kurze Einführung in die HATS-Entwicklungsumgebung geben soll. Das Tutorial beschreibt, wie ein einfacher Screen einer Host-Anwendung über eine Screen-Anpassung mit einer HATS Web-Anwendung dargestellt werden kann.