TRex

Transaction Remote Executer

API Handbuch

Version 1.8

TRex Programmierhandbuch

Copyright

Copyright © 1986-2003 XPS Software GmbH

Alle Rechte vorbehalten.

Warenzeichen

Java ist ein Warenzeichen von Sun Microsystems, Inc.

Windows ist ein Warenzeichen der Microsoft Corporation.

MVS, OS/390, VSE, VSE/ESA, VM/CMS, OS/400, TSO, CICS und IMS sind Warenzeichen der IBM Corporation.

Andere in diesem Handbuch erwähnten Marken- und Produktnamen sind Warenzeichen der jeweiligen Rechtsinhaber und werden hiermit anerkannt.

Inhaltsverzeichnis

Inhaltsverzeichnis ... 3

Abbildungen ... 4

Einführung ... 5
- Architektur ... 5
- Services ... 6
- Vorbereitung ... 6

XPSDaemon ... 7
- Allgemein ... 7
- Das TRex Beispielprogramm ... 7
  - Quelltext zu TREXUP01 - Cobol ... 8
  - Programmlogik ... 10
- Definition der benötigten Resourcen ... 11
  - Programmdefinitionen im CICS ... 11
  - APPC Definitionen im CICS ... 12
  - APPC Definitionen in XPSDaemon ... 12
  - Definition eines TRex Service ... 14
  - APPC Redirector ... 16
  - TCP/IP Redirector ... 16
  - Resourcedefinitionen für APPC/MVS - Batch ... 18
  - Resourcedefinitionen für APPC/MVS - IMS ... 19
  - Resourcedefinitionen für AS/400 ... 20

Java Client ... 21
- Allgemein ... 21
- Das TRex Beispielprogramm ... 21
  - Quelltext zu TRexTester.java ... 21
  - Programmlogik ... 27

Tools ... 31
- Verwaltung von Zertifikaten ... 31
  - Benutzerzertifikate ... 31
  - Vertrauenswürdige Aussteller ... 32
  - Vertrauenswürdige Server ... 33

Abbildungen

Abb. 1: Datenaustausch über TRex 5
Abb. 2: XPSDaemon - APPC Definitionen 12
Abb. 3: XPSDaemon - Bearbeiten einer APPC Definition 13
Abb. 4: XPSDaemon - Aktive APPC Verbindungen 14
Abb. 5: CICS - Aktive Verbindungen 14
Abb. 6: XPSDaemon - Redirector Definitionen 15
Abb. 7: XPSDaemon - Bearbeiten eines Service 15
Abb. 8: Übersicht der CICS Definitionen 17
Abb. 9: Übersicht der APPC/MVS Batch Definitionen 18
Abb. 10: Übersicht der IMS Definitionen 19
Abb. 11: XPSDaemon/400 – Redirector Definitionen 20
Abb. 12: Übersicht der AS/400 Definitionen 20
Abb. 13: CICS TS-QUEUE TREXOUT 30
Abb. 14: Verwaltung von Benutzerzertifikaten 31
Abb. 15: Vertrauenswürdige Aussteller 32
Abb. 16: Vertrauenswürdige Server 33

Kapitel

1

Einführung

Architektur

TRex, der Transaction Remote Executer von XPS, ist ein programmierbarer Tunnel über TCP/IP, der zur Abwicklung von sicheren Transaktionen über Plattformgrenzen hinweg verwendet werden kann.

Der Einsatz von TRex erfolgt in Verbindung mit XPSDaemon, dem TCP/IP Server von XPS für die Hostsysteme z/OS, OS/390, MVS/ESA, VSE/ESA und OS/400.

Die TRex Client Programmierschnittstelle kann auf jedem Betriebssystem genutzt werden, für das eine Java Virtual Machine ab der Version 1.2 verfügbar ist.

Die nachfolgende Abbildung zeigt beispielhaft, wie der Austausch von Daten unter Verwendung von TRex abläuft.

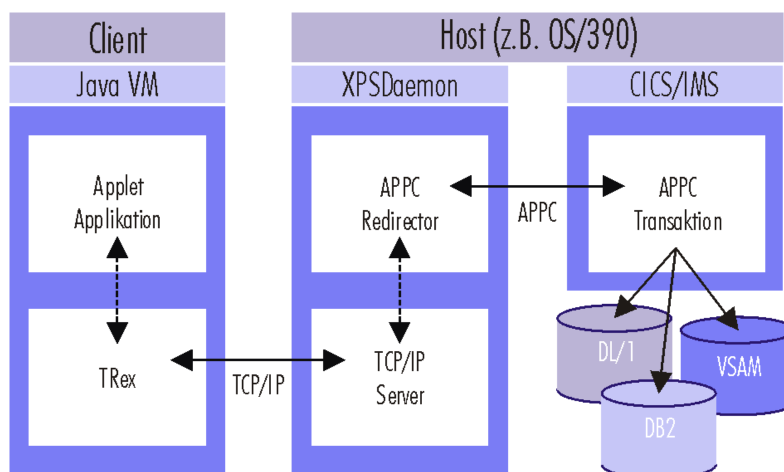

Abb. 1: Datenaustausch über TRex

Neben der eigentlichen Datenübermittlung ermöglicht TRex die Nutzung weiterer interessanter Features. So besteht die Möglichkeit, dass sich die beiden Kommunikationspartner (Client/XPSDaemon) im Rahmen des Verbingungsaufbaus durch X.509 V3 Zertifikate authentifzieren. Die übermittelten Daten können 128-Bit Blowfish verschlüsselt werden. Es ist möglich, die Integrität der übertragenen Daten durch die Verwendung einer Prüfsumme (Hashwert) sicherzustellen und die Datenströme wahlweise in nur eine oder in beide Übertragungsrichtungen zu komprimieren.

Unter Verwendung von XPSDaemon/Fireproof, einem Java Servlet von XPS, kann der gesamte Datenaustausch mit einem Hostrechner über das HTTP Protokoll erfolgen. Damit ist der Einsatz des Tunnels im Zusammenspiel mit einer Firewall möglich.

XPSDaemon kann im Verbund mit TRex entweder wie oben abgebildet als APPC Redirector oder auch als TCP/IP Redirector eingesetzt werden. Die Wahl ist situationsbedingt für jedes Projekt zu treffen.

Als TCP/IP Redirector funktioniert XPSDaemon ähnlich wie oben abgebildet. Allerdings erfolgt die Kommunikation zwischen XPSDaemon und dem "Datenbeschaffungsprogramm" nicht über APPC(LU6.2) sondern über TCP/IP.

Die Wahl des XPSDaemon Funktionsmodus hat auf die Entwicklung des Java Clientprogramms nur eine Auswirkung hinsichtlich des Kommunikationsmodus, der entweder Half-Duplex (APPC) oder Full-Duplex (TCP/IP) ist.

Services

Die Identifikation des Dienstes, den TRex hostseitig verwenden möchte, erfolgt über den sogenannten 'Service'. Der Service ist das eindeutige Bindeglied zwischen dem Java Clientprogramm und dem Hostprogramm, das für die Verarbeitung zuständig ist.

Im Rahmen der XPSDaemon Onlineverwaltung können verschiedene Services definiert werden. Neben einem eindeutigen Namen sind dort alle weiteren servicespezifischen Parameter festzulegen.

Clientseitig ist bei der Erzeugung einer TRex Instanz jeweils der Name des Service anzugeben, der zum Einsatz kommen soll. Technisch gesehen ist der Service für das Clientprogramm neben den TCP/IP Parametern, die bekannt sein müssen, um eine Verbindung mit XPSDaemon aufzubauen, die einzig benötigte Information.

Vorbereitung

Vor der ersten Nutzung von TRex ist zunächst die Installation von XPSDaemon auf dem gewünschten Hostsystem erforderlich. Die Vorgehensweise hierzu ist im entsprechenden XPSDaemon Handbuch beschrieben.

In dieser Dokumentation wird davon ausgegangen, dass XPSDaemon korrekt auf dem Hostsystem installiert wurde, und dass eine TCP/IP Verbindung zwischen Client und XPSDaemon aufgebaut werden kann.

Clientseitig besteht TRex aus einem Java Archiv, das sowohl bei der Programmentwicklung als auch zum Ausführungszeitpunkt verfügbar sein muss. Der Name des Programmarchivs lautet 'trex.jar' und es enthält alle Java Klassen, die für den Einsatz von TRex benötigt werden.

Kapitel 2

XPSDaemon

Allgemein

Dieses Kapitel enthält eine Beschreibung der administrativen Schritte, die auf dem Host durchzuführen sind, um TRex einsetzen zu können.

Im Allgemeinen werden die benötigten XPSDaemon Onlinetransaktionen aus Sicherheitsgründen nicht für jeden Anwender aufrufbar sein. Es ist daher notwendig, dass der Anwender, der die Definitionen durchführt, über die entsprechenden Administratorrechte verfügt.

Das TRex Beispielprogramm

Im Installationsumfang von XPSDaemon ist ein Beispielprogramm für die Kommunikation mit einem TRex Client enthalten. Nach erfolgter Installation befindet sich der Quelltext des Programms in der XPSDaemon Maclib.

Das Beispielprogramm liegt als Cobolquelltext mit dem Programmnamen 'TREXUP01' vor.

Das Beispiel zeigt, wie die Kommunikation eines CICS Programms über eine APPC Verbindung mit einem TRex Java Client erfolgen kann. Dies entspricht in etwa dem in Abb. 1 auf Seite 5 dargestellten System.

Im nachfolgend abgedruckten Quelltext des Beispielprogramms ist zu erkennen, dass das Programm keinerlei APPC Logik enthält. Der Grund hierfür liegt in der Verwendung des CICS APPC Adapterprogramms 'TREXADTC' von XPS.

'TREXADTC' ist ebenfalls im Installationsumfang von XPSDaemon enthalten. Es liegt als Quelltext und als Phase vor, so dass es ohne weitere Vorbereitungen verwendet werden kann. Es ist lediglich in eine Loadlib zu kopieren, von der es in das gewählte CICS geladen werden kann.

Das Adapterprogramm übernimmt das gesamte APPC Handling und übergibt die Programmsteuerung durch einen Linkaufruf an entsprechenden Ausführungspunkten an ein Verarbeitungsprogramm, in unserem Fall an 'TREXUP01'.

Damit ist die Implementation einer eigenen APPC Logik nicht notwendig. Bei Bedarf kann die APPC Logik jedoch auch selbst implementiert werden. In diesem Fall ist die Verwendung des Adapterprogramms hinfällig

Quelltext zu TREXUP01 - Cobol

```
000001 CBL APOST
000002       *-------------------------------------------------------------*
000003       *      TREX CICS COBOL SAMPLE PROGRAM                         *
000004       *-------------------------------------------------------------*
000005        ID DIVISION.
000006        PROGRAM-ID.
000007            TREXUP01.
000008        AUTHOR.
000009              -- XPS SOFTWARE --   19.01.2001
000010        ENVIRONMENT DIVISION.
000011       *
000012        DATA DIVISION.
000013        WORKING-STORAGE SECTION.
000014        01  FELDER.
000015            05 INI-LOW-VALUES           PIC X          VALUE LOW-VALUES.
000016            05 STATE-RECEIVE            PIC S9(8) COMP VALUE 1.
000017            05 STATE-SEND               PIC S9(8) COMP VALUE 2.
000018            05 STATE-END                PIC S9(8) COMP VALUE 3.
000019            05 REQUEST-TO-SEND          PIC S9(8) COMP VALUE 1.
000020            05 SEND-LENGTH              PIC S9(2) COMP VALUE 14.
000021            05 TS-LENGTH                PIC S9(4) COMP.
000022       *
000023        LINKAGE SECTION.
000024        01  DFHCOMMAREA.
000025            05  STATE-FUNCTION          PIC S9(8) COMP.
000026            05  DATA-ADDR               POINTER.
000027            05  DATA-LENGTH             PIC S9(8) COMP.
000028            05  USER-FIELD              PIC S9(8) COMP.
000029            05  SERVICE-NAME            POINTER.
000030            05  USER-NAME               POINTER.
000031            05  USER-PASSWORD           POINTER.
000032            05  SEND-OPTION             PIC S9(8) COMP.
000033       *
000034        01  APPCDATA.
000035            05 APPC-LENGTH              PIC S9(2) COMP.
000036            05 APPC-DATA                PIC X(12).
000037       *
000038        01  DATA01.
000039            05 LENGTH01                 PIC S9(2) COMP.
000040            05 TEXT01                   PIC X(12).
000041       *
000042        EJECT
000043       ****************************************************************
000044       ****************************************************************
000045       **                                                            **
000046       **             PROCEDURE DIVISION                             **
000047       **                                                            **
000048       ****************************************************************
000049       ****************************************************************
000050        PROCEDURE DIVISION.
000051        MAIN SECTION.
000052            EXEC CICS IGNORE CONDITION ERROR
000053            END-EXEC.
000054       *
000055            IF USER-FIELD = ZERO
000056               EXEC CICS DELETEQ TS QUEUE('TREXOUT')
000057               END-EXEC
000058               MOVE 1 TO USER-FIELD
000059            END-IF.
000060       *
000061            IF STATE-FUNCTION = STATE-RECEIVE
000062               PERFORM RECEIVE-STATE
000063       *
000064            ELSE IF STATE-FUNCTION = STATE-SEND
000065               PERFORM SEND-STATE
000066       *
000067            ELSE IF STATE-FUNCTION = STATE-END
000068               PERFORM END-STATE
000069            END-IF.
000070       *
000071            EXEC CICS RETURN
000072            END-EXEC.
000073            GOBACK.
000074       *
000075       *-------------------------------------------------------------*
000076       *  RECEIVE STATE: PROGRAM CANNOT SEND                         *
000077       *                 PROGRAM CANNOT CHANGE STATE                 *
000078       *-------------------------------------------------------------*
000079        RECEIVE-STATE.
000080            PERFORM GET-DATA.
000081            EXIT.
000082       *
000083       *-------------------------------------------------------------*
000084       *  SEND STATE:    PROGRAM CAN    SEND                         *
000085       *                 PROGRAM CAN    CHANGE STATE                 *
000086       *-------------------------------------------------------------*
```

```
000087        SEND-STATE.
000088            PERFORM GET-DATA.
000089            IF USER-FIELD = 1
000090               EXEC CICS GETMAIN
000091                         SET       (DATA-ADDR)
000092                         LENGTH    (SEND-LENGTH)
000093                         INITIMG   (INI-LOW-VALUES)
000094               END-EXEC
000095               SET ADDRESS OF DATA01 TO DATA-ADDR
000096               MOVE SEND-LENGTH      TO LENGTH01
000097               MOVE 'TEST SEND 01'   TO TEXT01
000098               MOVE SEND-LENGTH      TO DATA-LENGTH
000099               MOVE 2                TO USER-FIELD
000100               MOVE STATE-SEND       TO STATE-FUNCTION
000101            ELSE
000102               EXEC CICS GETMAIN
000103                         SET       (DATA-ADDR)
000104                         LENGTH    (SEND-LENGTH)
000105                         INITIMG   (INI-LOW-VALUES)
000106               END-EXEC
000107               SET ADDRESS OF DATA01 TO DATA-ADDR
000108               MOVE SEND-LENGTH      TO LENGTH01
000109               MOVE 'TEST SEND 02'   TO TEXT01
000110               MOVE SEND-LENGTH      TO DATA-LENGTH
000111               MOVE STATE-RECEIVE    TO STATE-FUNCTION
000112            END-IF.
000113            EXIT.
000114       *
000115       *--------------------------------------------------------------*
000116       *  END-CONNECTION STATE: CONNECTION IS CLOSED FROM PARTNER LU   *
000117       *                 PROGRAM CAN CLEANUP                          *
000118       *--------------------------------------------------------------*
000119        END-STATE.
000120            PERFORM GET-DATA.
000121            EXIT.
000122       *
000123        GET-DATA.
000124            IF DATA-ADDR NOT = NULL
000125               MOVE DATA-LENGTH        TO TS-LENGTH
000126               SET ADDRESS OF APPCDATA TO DATA-ADDR
000127               EXEC CICS WRITEQ TS
000128                         QUEUE     ('TREXOUT')
000129                         FROM      (APPCDATA)
000130                         LENGTH    (TS-LENGTH)
000131                         MAIN
000132               END-EXEC
000133            END-IF.
000134            EXIT.
```

Programmlogik

Die Logik des Beispielprogramms beschränkt sich, wie bereits erwähnt, auf ein Minimum.

Der Austausch von Informationen mit dem APPC Adapterprogramm erfolgt über eine Communication Area, die in den Zeilen 23 bis 32 definiert wird. Die Commarea enthält die folgenden Felder:

Feldname	Länge	Verwendung
STATE-FUNCTION	4 Byte	In diesem Feld wird der aktuelle Status der Verbindung bzw. die auszuführende Funktion ausgetauscht. Es wird darüber informiert, ob gelesene Daten zur Verarbeitung bereitstehen, ob Daten zu senden sind, oder ob die Verbindung beendet wurde.
DATA-ADDR	4 Byte	Dieses Feld dient zur Übermittlung der Adresse der Übergabedaten. Es ist im Falle des Einlesens von Daten auszuwerten und im Falle des Versendens von Daten z u füllen.
DATA-LENGTH	4 Byte	Dieses Feld wird zur Übermittlung der Datenlänge verwendet. Es ist analog zum Vorgängerfeld im Falle des Einlesens von Daten auszuwerten und im Falle des Versendens von Daten zu füllen.
USER_FIELD	4 Byte	Dieses Feld dient zur Speicherung einer beliebigen Information durch das Programm. Beispielsweise kann es als transaktionsübergreifender Sicherungsbereich für eine Speicheradresse verwendet werden, an der Statusinformationen des Programms abgespeichert werden.
SERVICE -NAME	4 Byte	Mit Hilfe dieses Feldes wird die Bezeichnung des angeforderten Service an das Programm übergeben. Damit besteht die Möglichkeit, mit einem Programm verschiedene Services zu bearbeiten.
USER-NAME	4 Byte	Dieses Feld wird zur Übermittlung des Benutzernamens verwendet, der vom Java Clientprogramm an die TRex Instanz übergeben werden kann.
USER-PASSWORD	4 Byte	Dieses Feld wird zur Übermittlung des Passworts verwendet, das vom Java Clientprogramm an die TRex Instanz übergeben werden kann.
SEND-OPTION	4 Byte	In diesem Feld wird die APPC Request To Send Option ausgetauscht. Damit besteht für den Kommunikationspartner, der momentan das Recht zum Senden gemäß dem Half-Duplex Protokoll nicht besitzt, die Gegenseite um die Erteilung des Senderechts zu bitten.

Die Steuerung des Programms befindet sich in den Zeilen 55 bis 73. Das Programm liest solange Daten ein und speichert sie in einer CICS TS-Queue namens 'TREXOUT', bis es von der Gegenseite zum Senden von Informationen aufgefordert wird.

Die Vorgehensweise der gegenseitigen Zuteilung das Senderechts liegt darin begründet, dass das APPC Protokoll ein Half-Duplex Protokoll ist. Das bedeutet, dass sich immer nur eine Seite der Verbindung im Sendestatus befindet.

Die Gegenseite befindet sich solange im Empfangstatus, bis ihr explizit das Senderecht erteilt wird. Das Versenden von Daten im Empfangstatus führt zu einer Protokollverletzung. Das erste Senderecht liegt beim Clientprogramm.

Sobald das Programm von der Gegenseite zum Senden von Daten aufgefordert wird, was durch den Wert von 'STATE-SEND' im Feld 'STATE-FUNCTION' der Commarea angezeigt wird, versendet es zunächst die Zeichenkette 'TEST SEND 01' an die Gegenseite.

Es ist zu beachten, dass das Beispielprogramm die Gegenseite **nicht** in den Sendestatus versetzt, da es selbst im Sendestatus verbleibt, was in Zeile 100 ersichtlich ist. Damit wird das Adapterprogramm angewiesen, die übergebenen Daten zunächst an die Gegenseite zu senden und dann die Kontrolle wieder an das Testprogramm zurückzugeben.

Nach erneuter Aktivierung des Programms wird diesmal die Zeichenkette 'TEST SEND 02' versendet und die Gegenseite erneut zum Senden berechtigt. Dies wird in Zeile 111 durch Füllen des Statusfeldes 'STATE-FUNCTION' mit dem Wert 'STATE-RECEIVE erreicht.

Um die Verbindung zu beenden, ist der Wert 'STATE-END' in 'STATE-FUNCTION' zu übertragen.

Das APPC Protokoll erfordert von den Kommunikationspartnern, dass die übertragenen Daten mit einem Feld in der Länge von 2 Byte beginnen, das die Gesamtlänge der Daten enthält. Das Längenfeld selbst ist somit bei der Ermittlung der Datenlänge zu berücksichtigen. Im Beispielprogramm erfolgt dies in den Zeilen 98 und 110.

Hinweis: Bei Verwendung des Adapterprogramms sollte das Anwendungsprogramm keinerlei Speicherbereiche freigeben, die zur Übermittlung von Daten verwendet werden, da die Freigabe immer vom Adapterprogramm durchgeführt wird. Das gilt auch für Speicherbereiche, die explizit vom Anwendungsprogramm zur Übermittlung von Daten an die Gegenseite reserviert wurden wie es in den Zeilen 90 und 102 zu sehen ist.

Definition der benötigten Resourcen

Um die Dienste des XPSDaemon APPC Redirectors zu nutzen, sind neben der Entwicklung der Anwendungsprogramme auch die notwendigen Resourcendefinitionen durchzuführen.

Die Definitionen der Resourcen erfolgt an verschiedenen Stellen. Neben der Definition der Anwendungsprogramme im CICS sind auch Definitionen notwendig, die die APPC Kommunikation zwischen XPSDaemon und CICS ermöglichen.

Programmdefinitionen im CICS

Damit das Beispielprogramm und das Adapterprogramm im CICS verwendet werden können, sind die entsprechenden Resourcedefinitionen im CICS durchzuführen.

Nach der Installation von XPSDaemon befindet sich eine Datei namens 'CICSDEF' in der XPSDaemon Maclib. Diese Datei enthält die notwendigen Resourcedefinitionen, die mit Hilfe des CICS Batchprogramms 'DFHCSDUP' geladen werden können.

Die folgenden Definitionen - ein Auszug aus der Datei 'CICSDEF' - sind für die Ausführung des Beispielprogramms notwendig:

```
000014 *
000015 * PROGRAM
000016 *
000017  DEFINE PROGRAM(TREXADTC)  GROUP(xpstrex) LANGUAGE(ASSEMBLER)
```

```
000018  DEFINE PROGRAM(TREXUP01)  GROUP(xpstrex)
000019 *
000020 * TRANSACTION
000021 *
000022  DEFINE TRANSACTION(trex) PROGRAM(TREXADTC) GROUP(xpstrex)
000023 *
```

Der Gruppenname und der Transaktionscode sind wahlfrei. Die benötigten Einträge können alternativ auch unter Verwendung der CICS RDO Transaktion 'CEDA' erfolgen.

APPC Definitionen im CICS

Die nachfolgende Liste enthält die benötigten APPC Definitionen, die im CICS durchzuführen sind, um die Kommunikation mit XPSDaemon zu ermöglichen. Die Liste ist ein weiterer Auszug aus der Datei 'CICSDEF' auf der XPSDaemon Maclib, die mit angepassten Parametern an 'DFHCSDUP' übergeben werden kann.

```
000002 *
000003 * TREX PARTNER CONNECTION
000004 *
000005  DEFINE CONNECTION(trex) GROUP(xpstrex) NETNAME(xpsdaem)
000006         ACCESSMETHOD(VTAM) PROTOCOL(APPC) SINGLESESS(NO)
000007         DATASTREAM(USER) RECORDFORMAT(U) AUTOCONNECT(YES)
000008         INSERVICE(YES)
000009 *
000010 * TREX SESSIONS
000011 *
000012  DEFINE SESSIONS(xpsdaem) GROUP(xpstrex) CONNECTION(trex)
000013         MODENAME(appclogm) PROTOCOL(APPC) MAXIMUM(n,n)
```

Bei der Definition der Verbindung ist es erforderlich, dass der VTAM Netname von XPSDaemon bekannt ist. Der Verbindungs- und Gruppenname sind frei wählbar.

APPC Definitionen in XPSDaemon

Analog zu den notwendigen Definitionen im CICS ist die APPC Verbindung auch in XPSDaemon zu definieren. Dazu stellt der XPSDaemon Transaktionsmonitor verschiedene Eingabemasken zur Verfügung.

Nach dem Starten der XPSDaemon Verwaltungstransaktion 'XDAD' ist der Menüpunkt 'H APPC Definitionen' auszuwählen, was zur Anzeige der folgenden Bildschirmmaske führt:

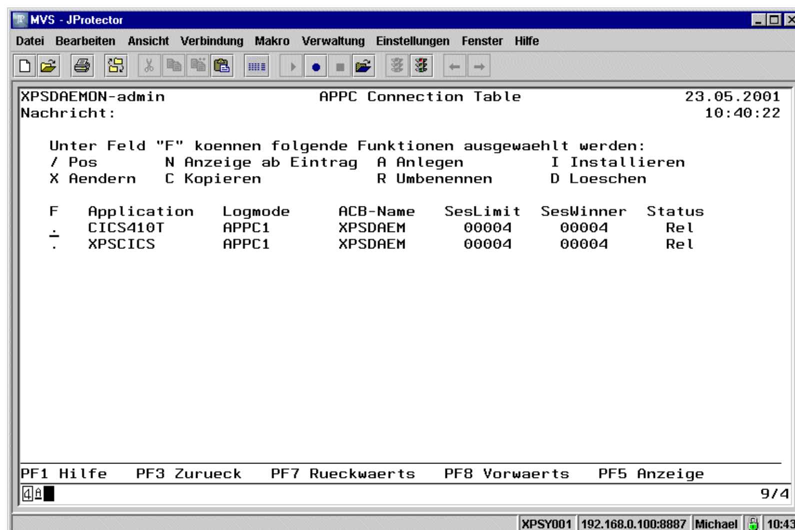

Abb. 2:XPSDaemon - APPC Definitionen

Die dargestellte Übersicht zeigt die bereits definierten APPC Verbindungen für XPSDaemon. Durch Eingabe von 'X' im Funktionsfeld vor einer Definition kann diese bearbeitet werden, durch Eingabe von 'A' eine neue Definition erstellt werden.

Zur Bearbeitung oder Neuerstellung einer APPC Definition wird die folgende Eingabemaske verwendet:

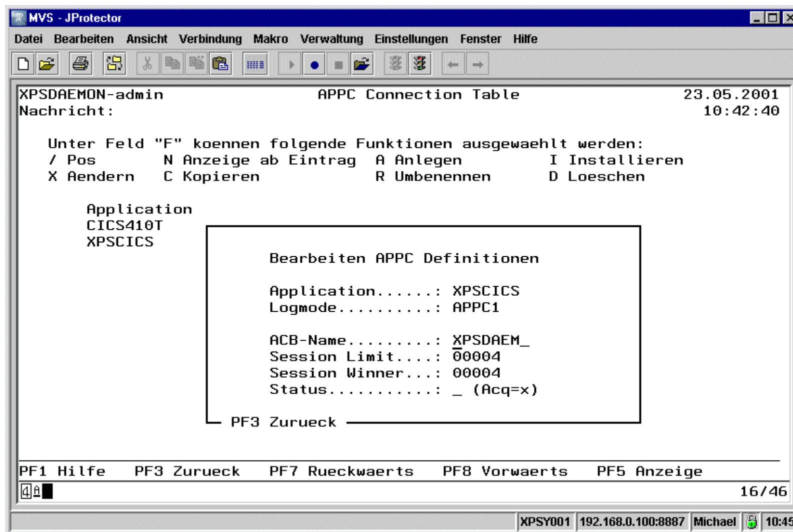

Abb. 3: XPSDaemon - Bearbeiten einer APPC Definition

Die Eingabefelder haben die folgende Bedeutung:

Application Der VTAM Netname der Applikation, mit der eine APPC Verbindung aufgebaut werden soll.

Logmode Der VTAM Logmode für die APPC Verbindung.

ACB-Name Der Name des XPSDaemon VTAM ACBs, der zum Verbindungsaufbau verwendet werden soll. Dieser Parameter ist erforderlich, da XPSDaemon mehrere VTAM ACBs eröffnen kann.

Session Limit Die maximale Anzahl der Sessions, die über die APPC Verbindung parallel aufgebaut werden können.

Session Winner Die Anzahl der Session Winner sollte mit dem gewählten Session Limit übereinstimmen, da der Aufbau der APPC Verbindung immer von XPSDaemon angestoßen wird.

Status Durch Eingabe von 'X' in diesem Feld wird XPSDaemon dazu veranlasst, die APPC Verbindung bereits beim Startup aufzubauen. Andernfalls erfolgt der Aufbau sobald die Verbindung erstmals benötigt wird.

Nachdem die APPC Verbindung definiert und gesichert wurde (drücken der Taste 'PF3') kann sie durch die Eingabe von 'I' im Funktionsfeld aktiviert werden.

Durch Drücken der Taste 'PF5' kann eine Übersicht der momentan aktiven APPC Verbindungen, die für XPSDaemon bestehen, angezeigt werden. Dazu dient die nachfolgend abgebildete Bildschirmmaske:

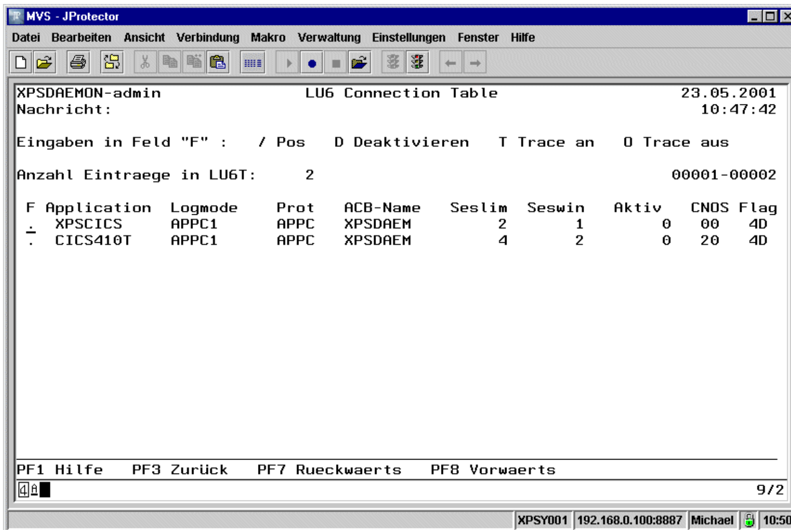

Abb. 4: XPSDaemon - Aktive APPC Verbindungen

Die Übersicht zeigt die Parameter für jede aktive Verbindung und ermöglicht durch Eingabe von 'D' in das Funktionsfeld die Deaktivierung einer gewählten APPC Verbindung.

Nach erfolgreichem Verbindungsaufbau muss die APPC Verbindung auch in der Partnerapplikation als aktiv erscheinen. Die folgende Abbildung zeigt die Ausgabe des CICS Kommandos 'CEMT I CONN', mit dem eine Übersicht der momentan aktiven Verbindungen im CICS angezeigt werden kann:

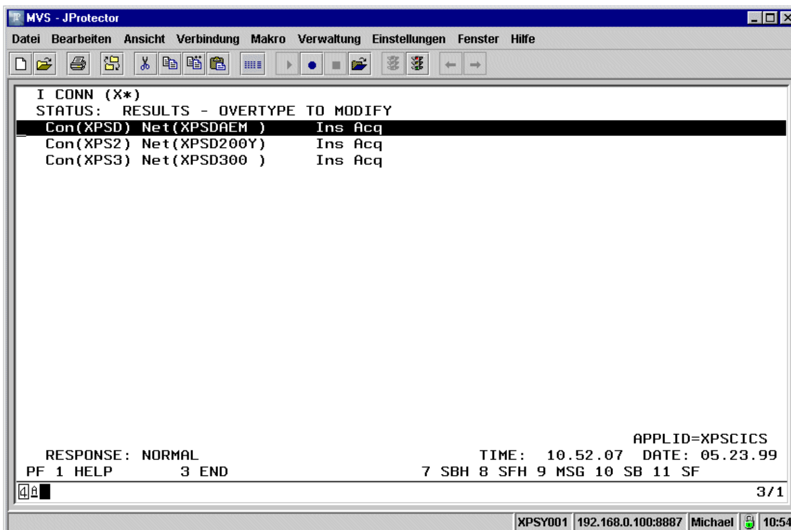

Abb. 5: CICS - Aktive Verbindungen

Der farblich hervorgehobenen Zeile ist zu entnehmen, dass das XPSCICS eine aktive (Acquired) APPC Verbindung zur VTAM Applikation 'XPSDAEM' besitzt.

Mit diesem Schritt sind alle notwendigen Vorbereitungen für die Ausführung von TRex getroffen. Was noch fehlt ist ein Java Programm, das das Beispielprogramm im CICS zur Ausführung bringt. Wie bereits erwähnt, ist dazu die Definition eines Service nötig.

Definition eines TRex Service

Das Bindeglied zwischen einem Java Anwendungsprogramm und der zugehörigen Hosttransaktion ist der TRex Service. TRex Services machen es überflüssig, dass das Java Anwendungsprogramm genauere Kenntnis von den am Host ablaufenden Prozessen haben muß. Im Java Programm wird lediglich

bestimmt, welcher Service angefordert wird. Alle weiteren Einstellungen sind ausschließlich im Host durchzuführen.

TRex Services können im Rahmen der XPSDaemon Onlineverwaltung nach Aufruf der Transaktion 'XDAD' durch Auswahl des Menüpunkts 'I Redirector Definitionen' verwaltet werden. Dazu wird die nachfolgend abgebildete Bildschirmmaske verwendet:

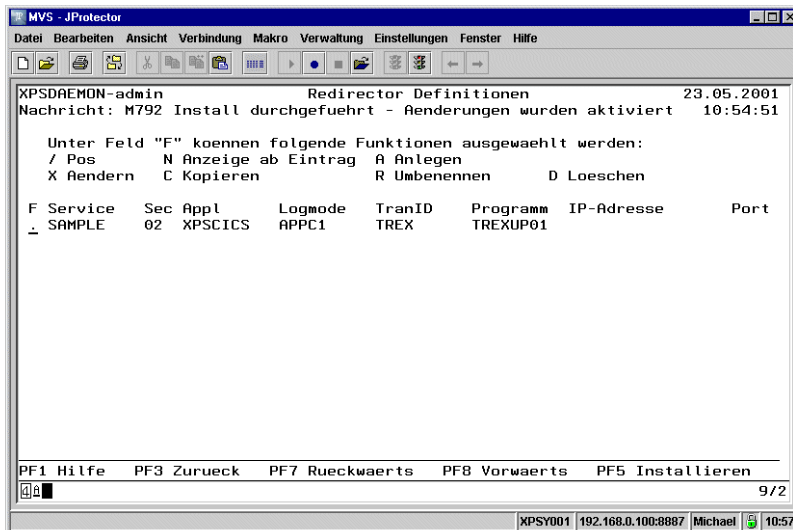

Abb. 6: XPSDaemon - Redirector Definitionen

In der Übersicht sind bestehende Services aufgeführt. Neben dem Servicenamen sind die Serviceparameter aufgeführt. Durch Eingabe von 'A' im Funktionsfeld kann ein neuer Service angelegt und durch Eingabe von 'X ein bestehender Service bearbeitet werden.

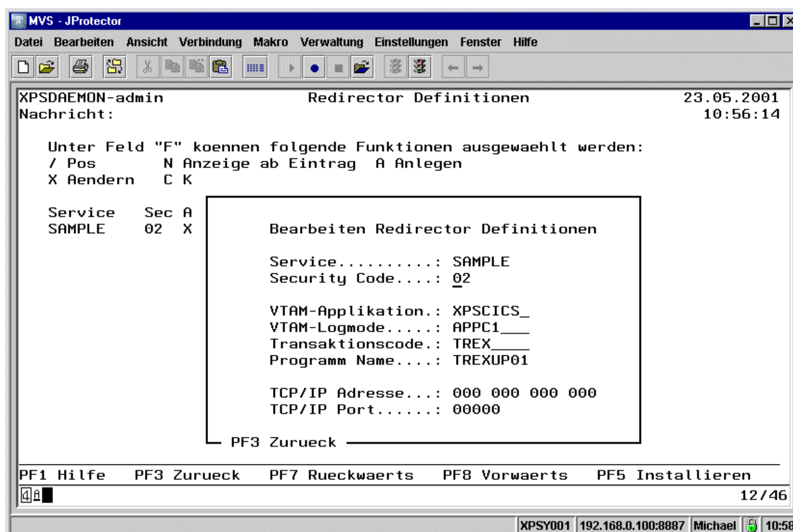

Abb. 7: XPSDaemon - Bearbeiten eines Service

Die Eingabefelder haben die folgende Bedeutung:

Service In diesem Eingabefeld ist ein eindeutiger Name für den Service festzulegen. Dieser Name wird vom Java Anwendungsprogramm bei der Erzeugung einer TRex Instanz verwendet, um XPSDaemon darüber zu informieren, welcher Service angefordert wird.

Security Code Mit Hilfe des Security Codes kann der Zugriff auf den Service beliebig eingeschränkt werden. Nähere Informationen zum internen XPSDaemon Sicherheitskonzept sind den entsprechenden Handbüchern zu entnehmen.

APPC Redirector

Die nächste Gruppe von Eingabefeldern ist zu füllen, wenn der Service mit der Ausführung einer APPC Transaktion verbunden sein soll.

VTAM-Applikation Mit dieser Eingabe wird der VTAM Netname der Applikation festgelegt, in der die APPC Transaktion gestartet werden soll.

VTAM-Logmode An dieser Stelle sollte derselbe VTAM Logmode angegeben werden wie bei der Definition der zugehörigen APPC Verbindung.

Transaktionscode In diesem Feld ist der Code der Transaktion zu bestimmen, die von XPSDaemon im Hostsubsystem gestartet werden soll. An den Transaktionscode ist die Ausführung eines Applikationsprogramms geknüpft. Für das Beispielprogramm ist das von XPS gelieferte APPC Adapterprogramm im CICS mit diesem Transaktionscode zu verknüpfen (siehe Abschnitt 'Programmdefinitionen im CICS' auf Seite 11).

Programm Name Eine Eingabe in dieses Feld ist nur notwendig, wenn das XPS APPC Adapterprogramm verwendet wird. In diesem Fall ist in diesem Feld der Name des Programms anzugeben, das per Link-Aufruf vom Adapterprogramm zu bestimmten Ausführungszeitpunkten die Kontrolle erhalten soll, wie oben im besprochenen Beispielprogramm erläutert.

TCP/IP Redirector

Die sich anschließende Gruppe von Eingabefeldern ist alternativ zu füllen, wenn der Service zur Umleitung von TCP/IP Datenströmen eingesetzt werden soll.

Die Rahmenbedingungen sind beim Einsatz als TCP/IP Redirector vergleichbar mit denen beim Einsatz als APPC Redirector. Der einzige Unterschied besteht darin, dass XPSDaemon nicht über APPC mit einer Transaktion in einem Hostsubsystem kommuniziert, sondern über TCP/IP mit einem Listenerprogramm (Serverprogramm).

Dieses Listenerprogramm muss nicht notwendigerweise auf dem selben Host verfügbar sein wie XPSDaemon. Entscheidend ist, dass eine TCP/IP Verbindung zwischen den beiden Programmen aufgebaut werden kann.

Die Wahl des TCP/IP Redirectors kann sich als sinnvoll erweisen, wenn z.B. der Datentransfer eines bereits bestehenden TCP/IP Programms über einen sicheren Tunnel umgeleitet werden soll.

Es ist zu berücksichtigen, dass TCP/IP, im Gegensatz zu APPC, ein statusfreies Übertragungsprotokoll ist. Das bedeutet, dass keine Rücksicht auf Sende- bzw. auf Empfangstatus zu nehmen ist. In verschiedenen Situationen kann dies vorteilhaft sein.

Für den Einsatz als TCP/IP Redirector muss XPSDaemon die TCP/IP Parameter des Listenerprogramms kennen, mit dem eine Verbindung aufgebaut werden soll.

Diese Angaben sind in den beiden Feldern für TCP/IP Adresse und TCP/IP Port zu machen.

Hinweis: Damit Veränderungen sofort aktiv werden, sind die Redirector Defintionen durch Drücken der PF5-Taste neu zu installieren. Andernfalls werden Veränderungen erst nach einem Neustart von XPSDaemon aktiv.

Mit der Definition des Service sind hostseitig alle benötigten Einstellungen und Definitionen für den Einsatz von TRex vorgenommen.

Das folgende Schaubild verdeutlicht noch einmal in einer Übersicht, wie die einzelnen Definitionen und Parameter in den verschiedenen Systemen zusammenspielen und aufeinander Bezug nehmen.

Abb. 8: Übersicht der CICS Definitionen

Resourcedefinitionen für APPC/MVS - Batch

Es besteht die Möglichkeit, dass ein Javaprogramm über TRex mit einem APPC/MVS Batchprogramm kommuniziert. Dazu sind die im Folgenden aufgeführten Definitionen notwendig.

XPSDaemon

Die Redirector Definitionen für APPC/MVS Batch sind prinzipiell analog zu CICS. Als Applikation ist 'MVSLU01' anzugeben, sofern die von IBM vorgegebenen Standards bei der Installation von MVS, OS/390 oder z/OS eingehalten wurden. Als Beispielprogramm wird 'TREXUP03' im Rahmen der Installation auf die XPSDaemon Maclib kopiert.

APPC/MVS - Batch

Die XPSDaemon Maclib enthält nach der Installation ein Member mit Namen 'ADDTP'. In diesem Member befindet sich eine Reihe von Job Control Statements, mit deren Hilfe das von XPS gelieferte APPC Adapterprogramm 'TREXADTB' in der APPC/MVS Umgebung bekannt gemacht werden kann. Falls ein eigenes APPC Programm zum Einsatz kommen soll, ist die JCL entsprechend anzupassen.

Des weiteren ist sicherzustellen, dass auf SYS1.PARMLIB ein Member mit Namen 'APPCPMxx' vorhanden ist, in dem sich ein 'LUADD ACBNAME(MVSLU01)' Statement befindet.

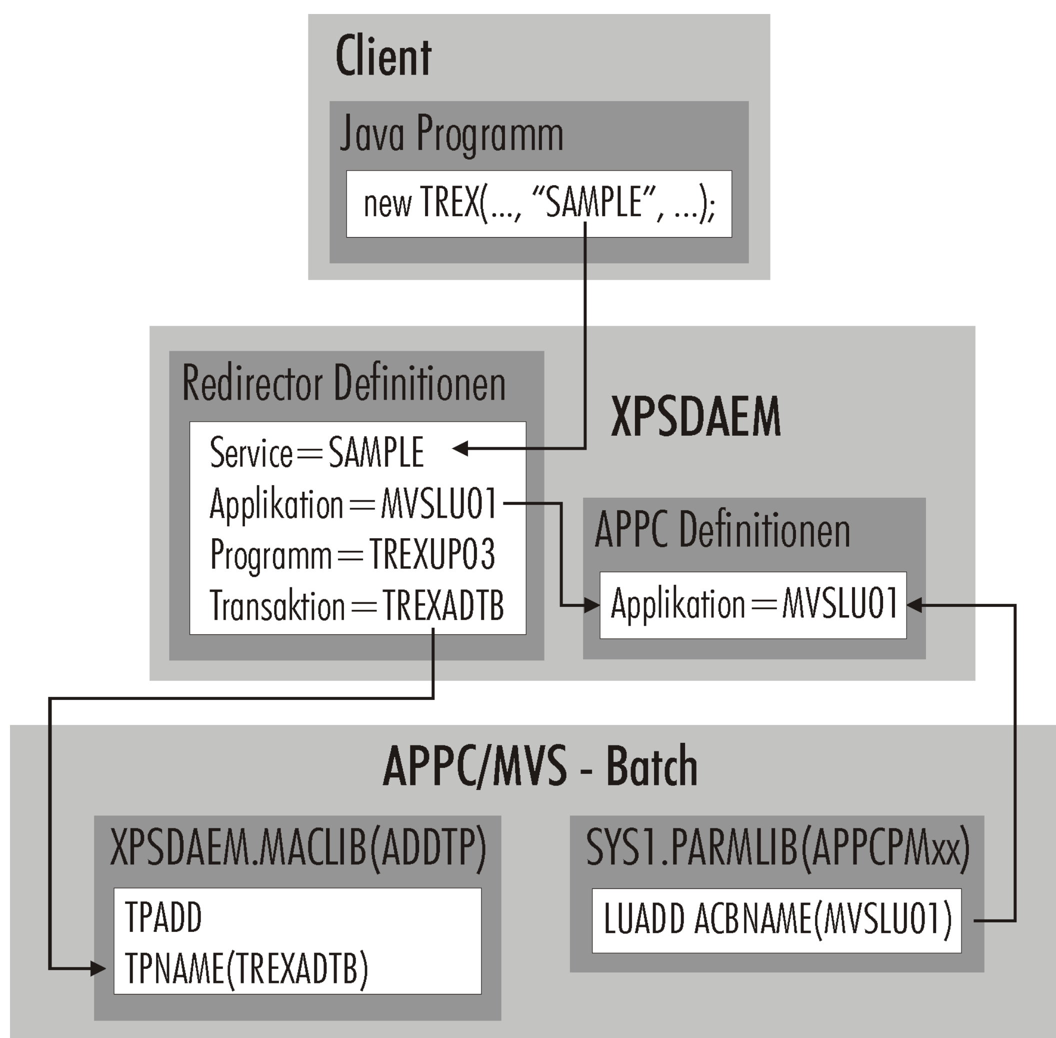

Abb. 9: Übersicht der APPC/MVS Batch Definitionen

Resourcedefinitionen für APPC/MVS - IMS

TRex bietet weiterhin auch die Möglichkeit, über ein Javaprogramm mit einem IMS Programm zu kommunizieren. Dazu sind die im Folgenden aufgeführten Definitionen notwendig.

XPSDaemon

Die Redirector Definitionen für IMS sind prinzipiell analog zu APPC/MVS. Als Applikation ist 'IMSLU62' anzugeben, sofern die von IBM vorgegebenen Standards bei der Installation von MVS, OS/390 oder z/OS eingehalten wurden. Als Beispielprogramm wird 'TREXUP04' im Rahmen der Installation auf die XPSDaemon Maclib kopiert.

APPC/MVS - IMS

Die XPSDaemon Maclib enthält nach der Installation ein Member mit Namen 'ADDTPIMS'. In diesem Member befindet sich eine Reihe von Job Control Statements, mit deren Hilfe das von XPS gelieferte APPC Adapterprogramm 'TREXADTI' in der APPC/MVS Umgebung bekannt gemacht werden kann. Falls ein eigenes APPC Programm zum Einsatz kommen soll, ist die JCL entsprechend anzupassen.

Des weiteren ist sicherzustellen, dass auf SYS1.PARMLIB ein Member mit Namen 'APPCPMxx' vorhanden ist, in dem sich ein 'LUADD ACBNAME(IMSLU62)' Statement befindet.

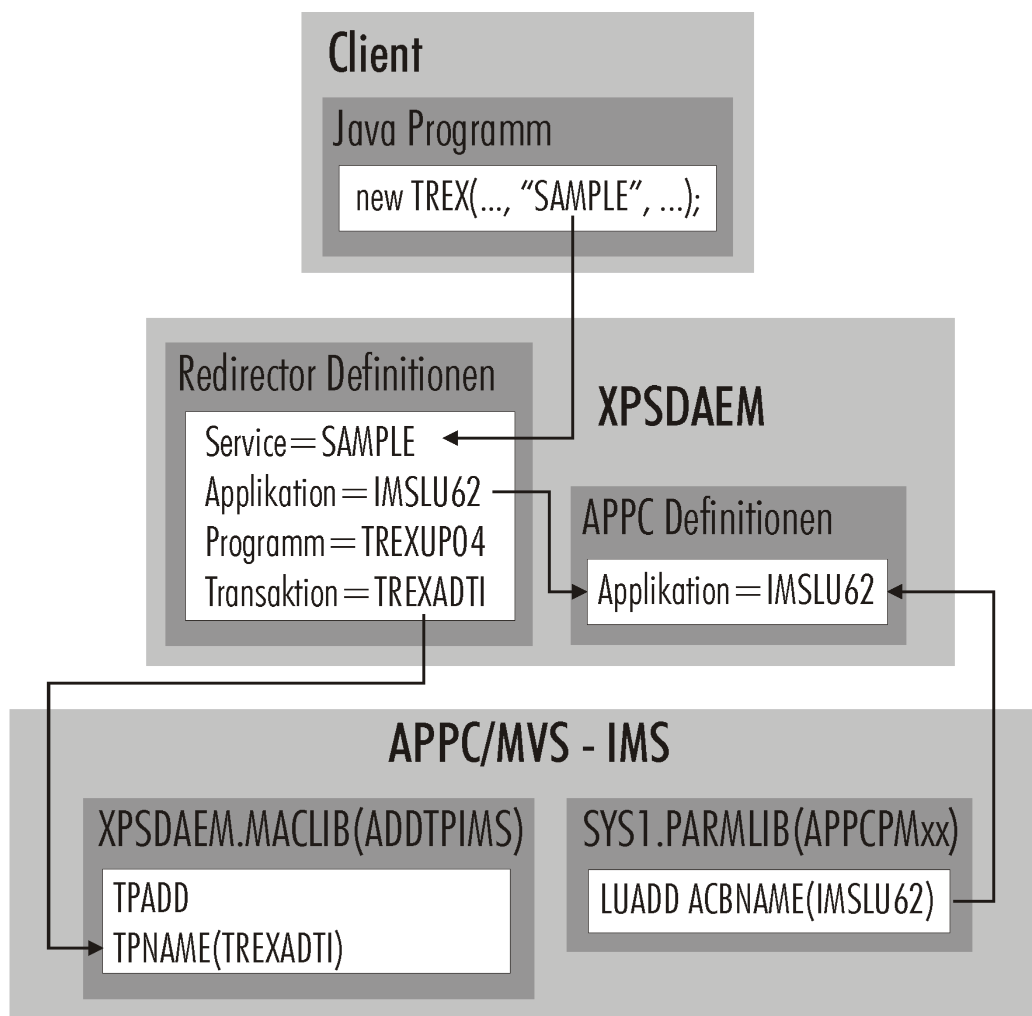

Abb. 10: Übersicht der IMS Definitionen

Resourcedefinitionen für AS/400

Schließlich bietet TRex auch die Möglichkeit, über ein Javaprogramm mit einem AS/400 APPC Programm zu kommunizieren. Dazu sind die im Folgenden aufgeführten Definitionen notwendig.

XPSDaemon

Die Redirector Definitionen erfolgen mit Hilfe der nachfolgend abgebildeten Maske im Rahmen der XPSDaemon/400 Online Verwaltung:

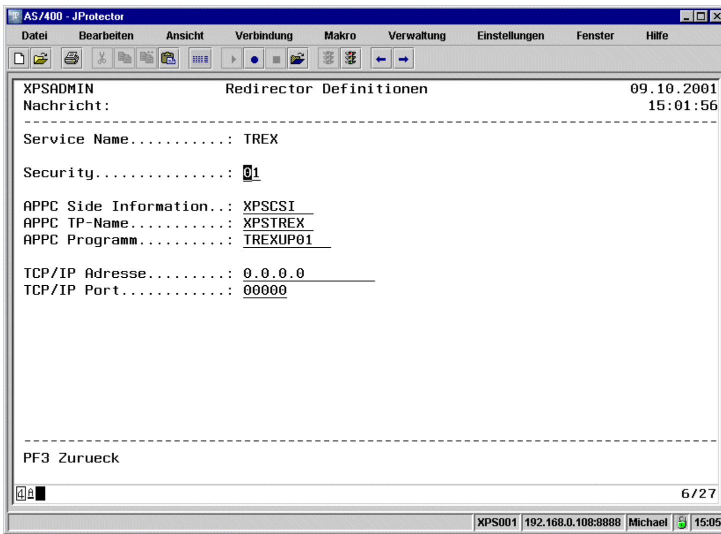

Abb. 11: XPSDaemon/400 – Redirector Definitionen

Das APPC Adapterprogramm 'XPSTREX' wird von XPS zur Verfügung gestellt und befindet sich nach der Installation von XPSDaemon/400 auf der XPS Source/Object Bibliothek. Um die dynamische Verwendung von Unterprogrammen zu ermöglichen, müssen diese als Serviceprogramme kompiliert werden und die Funktion "APPC_Conversation" exportieren. Als Vorlage wird hierfür das Beispielprogramm 'TREXUP01' mitgeliefert.

AS/400

Die Side Information 'XPSCSI' definiert die APPC Umgebung für XPSDaemon/400. Im Rahmen der Installation von XPSDaemon/400 wird das QCSRC-Bibliotheksmember 'XPS_APPC_C' kopiert, das eine Beispielprozedur zur Erstellung der XPSCSI Side Information enthält.

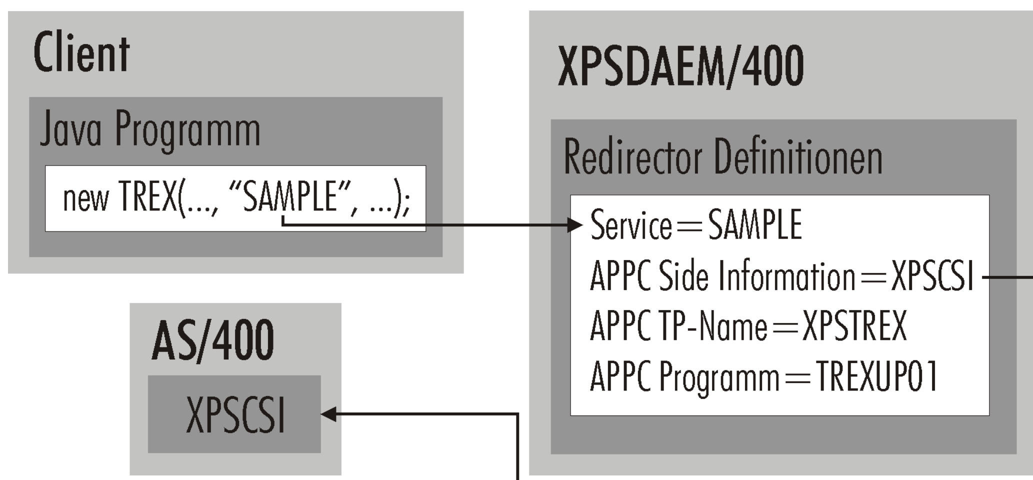

Abb. 12: Übersicht der AS/400 Definitionen

Kapitel 3

Java Client

Allgemein

Dieses Kapitel enthält eine Beschreibung des Java Clients. Unter einem Java Client is ein Java Anwendungsprogramm zu verstehen, das unter Verwendung des TRex Application Programming Interface (TRex API) eine Verbindung mit einem Hostrechner aufbaut und dort die Ausführung von Transaktionen veranlasst.

Zur Ausführung eines TRex Anwendungsprogramms wird die Verwendung einer Java Virtual Machine der Version 1.2 oder höher vorausgesetzt.

Des Weiteren muss das TRex Programmarchiv 'trex.jar' sowohl während der Entwicklungsphase als auch zum Ausführungszeitpunkt verfügbar sein.

Das TRex Beispielprogramm

Zur Einführung des TRex API wird ein Beispielprogramm analysiert. Dieses Beispielprogramm kann mit dem Host Beispielprogramm, das im vorigen Kapitel beschrieben wurde, kommunizieren und somit den Einsatz der verschiedenen Techniken demonstrieren.

Quelltext zu TRexTester.java

```
// --- package information ------------------------------------------------- //
package xps.trex;

// --- import information -------------------------------------------------- //
import xps.common.XPSByteBuffer;
import xps.util.XPSTransTab;
import java.util.*;
import java.io.*;

// --- class description --------------------------------------------------- //
/**
 * TRexTester is a simple sample program to show the usage of the TRex API. The
 * {@link xps.trex.TRexEventListener} interface is implemented to receive
 * notifications about events originated from the host transaction program.
 *
 * @author  Michael Knoll - MiknoSoft, XPS Software GmbH
 * @version 1.7
 */
public class
TRexTester
implements TRexEventListener
{
  // --- command line option flags ----------------------------------------- //
  /**
   * Don't use any security protocol.
   */
  final static int NO_SECURITY_PROTOCOL = 0x0000001;
  /**
   * Force TRex to trace a number of events to file.
   */
  final static int PRINT_TRACE = 0x0000002;
  /**
   * Write notification about some events to System.out.
```

```
 */
final static int WRITE_LOG = 0x0000004;
/**
 * Quotation mark
 */
final static char QUOTATION_MARK = '"';

// --- common definitions ---------------------------------------------- //
final static int    SIZE_OF_SHORT = 2;
final static String VERSION       = "TRex API test program: Version 1.8";

// -- workfields -------------------------------------------------------- //
/**
 * default session name
 */
static String m_sessionName = "TRexTester";
/**
 * default IP-address
 */
static String m_ipAddress = "192.168.0.100";
/**
 * default IP-port
 */
static int m_iPort = 8885;
/**
 * default service to use
 */
static String m_service = "SAMPLE";
/**
 * user name
 */
static String m_userName;
/**
 * user password
 */
static String m_password;
/**
 * PKCS#12 instance
 */
static String m_pkcs12Instance;
/**
 * password for PKCS#12 instance
 */
static String m_pkcs12Password;
/**
 * name of tracefile
 */
static String m_traceFile = "TRexTester.tra";
/**
 * default flags
 */
static int m_iFlags = NO_SECURITY_PROTOCOL;
/**
 * byte buffer to process data to write
 */
XPSByteBuffer m_writeBuffer = new XPSByteBuffer(32);
/**
 * byte buffer to process data to read
 */
XPSByteBuffer m_readBuffer = new XPSByteBuffer(32);
/**
 * flag to indicate that processing is over
 */
boolean m_fTerminate = false;
/**
 * TRex instance to be used
 */
TRex m_trex;
/**
 * instance for EBCDIC to unicode and unicode to EBCDIC translation
 */
XPSTransTab m_transTab;

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * The main method scans any parms transmitted to the program using command
 * line arguments and creates a new instance of the TRexTester. Calling
 * java -cp trex.jar xps.trex.TRexTester -?
 * will show the available command line arguments.
 */
public static
void
main
// ---------------------------------------------------------------------- //
(String[] args)
// ---------------------------------------------------------------------- //
{
  System.out.println(VERSION);

  String parm = null;
```

```
  try
  {
    // process command line arguments
    for (int i = 0; i < args.length; i++)
    {
      parm = args[i].toUpperCase();
      // check for help information
      if ("-?".equals(parm))
      {
        printHelp();
        System.exit(0);
      } // if
      // check for TCP/IP address
      else if (parm.startsWith("-A"))
        m_ipAddress = parm.substring(2);
      // check for activation of security protocol
      else if ("-E".equals(parm))
        m_iFlags &= (0xffffffff - NO_SECURITY_PROTOCOL);
      // check for PKCS#12 instance => don't use uppercase translated value
      else if (parm.startsWith("-I"))
      {
        String work = args[i].substring(2);
        // check quotation marks
        if (QUOTATION_MARK == work.charAt(0))
          m_pkcs12Instance = work.substring(1, work.length() - 2);
        else
          m_pkcs12Instance = new String(work);
      } // else
      else if ("-L".equals(parm))
        m_iFlags |= WRITE_LOG;
      // check for PKCS#12 password => don't use uppercase translated value
      else if (parm.startsWith("-O"))
        m_pkcs12Password = args[i].substring(2);
      // check for TCP/IP port
      else if (parm.startsWith("-P"))
        m_iPort = Integer.parseInt(parm.substring(2));
      // check for requested service
      else if (parm.startsWith("-S"))
        m_service = parm.substring(2);
      // check for trace
      else if ("-T".equals(parm))
        m_iFlags |= PRINT_TRACE;
      // check for user to logon
      else if (parm.startsWith("-U"))
        m_userName = parm.substring(2);
      // check for user password to logon
      else if (parm.startsWith("-W"))
        m_password = parm.substring(2);
    } // for
    // time to begin
    new TRexTester();
  } // try
  catch (Exception e)
  {
    System.out.println("Exception thrown for parm["+parm+"]");
    e.printStackTrace();
  } // catch
} // main

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * The TRexTester constructor creates the TRex instance to be used for
 * conversation with the remote host transaction program. If the use of
 * security options is desired a call to specifySecurityOptions is made to
 * inform TRex about the desired security options.
 */
public
TRexTester                            // constructor
// ---------------------------------------------------------------------- //
()
// ---------------------------------------------------------------------- //
{
  // create common options to use
  int iCommonOptions = TRex.COM_HALF_DUPLEX_MODE;
  // activate all traces if desired
  if (0 != (PRINT_TRACE & m_iFlags))
    iCommonOptions |=   TRex.COM_BASE_EVENT_TRACE
                      | TRex.COM_CONNECTION_TRACE
                      | TRex.COM_DATASTREAM_TRACE
                      | TRex.COM_EXTENDED_DATA_TRACE
                      | TRex.COM_HOST_TRACE;

  if (0 != (WRITE_LOG & m_iFlags))
    System.out.println("service=["+m_service+"], address=["+m_ipAddress
                      +"], port=["+m_iPort+"]");

  // create a new TRex instance
  m_trex = new TRex(this,                         // TRexEventListener
                   m_ipAddress,                   // IP address of XPSDaemon
                   m_iPort,                       // XPSDaemon port
```

```
                    null,                            // no servlet path
                    m_service,                       // requested service
                    m_sessionName,                   // name of our session
                    new Locale("de", "DE"),          // locale to use
                    XPSTransTab.TT_GERMANY_AUSTRIA, // host code page
                    iCommonOptions,                  // our common options
                    m_traceFile);                    // name of trace file

  // specify user data if present
  if (   (null != m_userName)
      || (null != m_password))
    m_trex.specifyUserData(m_userName,      // user name to use for logon to remote system
                           m_password);     // password to use for user logon
  // specify security options if to use
  if (0 == (NO_SECURITY_PROTOCOL & m_iFlags))
    m_trex.specifySecurityOptions(TRex.SEC_USE_SECURITY_PROTOCOL,
                                  1024,              // RSA keylength
                                  null,              // use standard trex db
                                  m_pkcs12Instance,  // PKCS#12 instance
                                  m_pkcs12Password); // PKCS#12 password
  // create a translate table for Unicode/EBCDIC translation
  m_transTab = new XPSTransTab(XPSTransTab.TT_GERMANY_AUSTRIA);
  // open tunnel for conversation with remote host transaction
  m_trex.openTunnel();
} // TRexTester

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// ==> implementation of TRexEventListener interface                     //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * TRex informs the program about the successfull opening of the tunnel
 * using this callback method. In response to that the test program sends
 * 2 strings to the host application program. With the 2nd send the receive
 * mode is entered by specifying TRex.TSW_CHANGE_DIRECTION.
 */
public
void
TRexTunnelReady                      // tunnel is ready for processing
// ---------------------------------------------------------------------- //
(int    iTunnelStatusWord,
 byte[] bData,
 int    iLength)
// ---------------------------------------------------------------------- //
{
  // inform user
  if (0 != (WRITE_LOG & m_iFlags))
    System.out.println("tunnelReady: TSW=["
                       +Integer.toHexString(iTunnelStatusWord)
                       +"] dataLength="+iLength);
  // be friendly
  write("Hello", TRex.TSW_KEEP_DIRECTION);
  // send sample text and switch to receive mode
  write("This is a TRex sample", TRex.TSW_CHANGE_DIRECTION);
} // TRexTunnelReady

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * TRex informs the program about the fact that data has been received from
 * the connected host transaction program using this callback method.
 * <p>
 * First of all a string is created from the received data. The 2 byte length
 * field demanded by the APPC protocol needs to be respected. In a real world
 * program the received data has to be analyzed due to the negotiated format
 * of the exchanged data.
 * <p>
 * Next the string is translated to unicode because the host transaction
 * program sends data encoded in EBCDIC.
 * <p>
 * If the TSW_CHANGE_DIRECTION flag is set, the m_fTerminate flag is turned
 * on to force termination after the last data has been sent. The
 * m_fTerminate flag will be checked in TRexOnEvent if an EV_DATA_WRITTEN
 * event has been detected.
 */
public
void
TRexDataRead                         // data has been received over tunnel
// ---------------------------------------------------------------------- //
(int    iTunnelStatusWord,
 byte[] bData,
 int    iLength)
// ---------------------------------------------------------------------- //
{
  // extract readable data if possible
  if (null != bData)
  {
    m_readBuffer.setLength(0);
    m_readBuffer.append(bData, SIZE_OF_SHORT, iLength - SIZE_OF_SHORT);
    String data = m_readBuffer.toString();
    // translate to unicode
```

```
    data = m_transTab.EB2UC(data);
    // inform user
    if (0 != (WRITE_LOG & m_iFlags))
      System.out.println("dataRead: data=["+data+"] TSW=["
                         +Integer.toHexString(iTunnelStatusWord)
                         +"] length="+data.length());
    // if change direction is set, send last data and terminate
    if (0 != (TRex.TSW_CHANGE_DIRECTION & iTunnelStatusWord))
    {
      // inform user
      System.out.println("dataRead: about to force termination");
      // indicate to TRexOnEvent(TRexEventListener.EV_DATA_WRITTEN)
      // that the tunnel shall be closed
      m_fTerminate = true;
      // that's it
      write("Goodbye", TRex.TSW_KEEP_DIRECTION);
    } // if
  } // if
} // TRexDataRead

// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * TRex informs the program about most of the events using this callback
 * method. Depending on the type of event the data parameter will contain
 * additional event specific information.
 */
public
boolean
TRexOnEvent                          // there's been an event on tunnel
// ----------------------------------------------------------------------- //
(int    iEvent,
 Object data)
// ----------------------------------------------------------------------- //
{
  String  event   = null;
  boolean fExit   = false;
  boolean fResult = true;
  // decide which event has been transmitted
  switch (iEvent)
  {
    case TRexEventListener.EV_TUNNEL_CLOSED:
      event = "EV_TUNNEL_CLOSED";
      fExit = true;
      break;
    case TRexEventListener.EV_USER_NAME_FROM_CERT:
      event = new String("EV_USER_NAME_FROM_CERT ["+(String)data+"]");
      break;
    case TRexEventListener.EV_KEY_GEN_STARTED:
      event = new String("EV_KEY_GEN_STARTED: keylength="+(Integer)data);
      break;
    case TRexEventListener.EV_KEY_GEN_ENDED:
      event = "EV_KEY_GEN_ENDED";
      break;
    case TRexEventListener.EV_VERSION_UPDATE_STARTED:
      event = "EV_VERSION_UPDATE_STARTED";
      break;
    case TRexEventListener.EV_ABOUT_TO_BACKUP_FILE:
      event = "EV_ABOUT_TO_BACKUP_FILE: info="+(String)data;
      break;
    case TRexEventListener.EV_ABOUT_TO_DOWNLOAD_FILE:
      event = new String("EV_ABOUT_TO_DOWNLOAD_FILE: fileinfo="+(String)data);
      break;
    case TRexEventListener.EV_VERSION_UPDATE_ENDED:
      event = "EV_VERSION_UPDATE_ENDED";
      // this event demands a program termination
      fExit = true;
      break;
    case TRexEventListener.EV_DATA_WRITTEN:
      event = "EV_DATA_WRITTEN";
      // check if connection is to terminate
      if (true == m_fTerminate)
        m_trex.closeTunnel();
      break;
  } // switch

  // display info
  if (0 != (WRITE_LOG & m_iFlags))
    System.out.println("onEvent: event="+event);
  // exit if necessary
  if (true == fExit)
    System.exit(0);
  return fResult;
} // TRexOnEvent

// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * TRex informs the program about the occurrence of an exception using this
 * callback method. Returning true lets TRex process the exception.
 */
public
```

```
boolean
TRexOnException                      // there's been an event on tunnel
// -------------------------------------------------------------------- //
(Exception e,
 String    info,
 String    sessionName)
// -------------------------------------------------------------------- //
{
  // let TRex do what's appropriate
  return true;
} // TRexOnException

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * TRex asks the program if a certificate sent from the host for the purpose
 * of server authentication shall be trusted. Returning ELRC_SHOW_DIALOG will
 * force TRex to display a dialog to a terminal user to decide if the
 * certificate shall be trusted.
 */
public
int
TRexShallCertBeTrusted               // decide if you'll trust certificate
// -------------------------------------------------------------------- //
(TRexX509Cert cert)
// -------------------------------------------------------------------- //
{
  // let TRex display a dialog to the user
  return TRex.ELRC_SHOW_DIALOG;
} // TRexShallCertBeTrusted

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * TRex asks the program to create a vector including version information for
 * a number of archives to provide a live update feature. This feature isn't
 * available for standard TRex applications.
 */
public
Vector
TRexGetJarVersions                  // nothing to do for standard TRex client
// -------------------------------------------------------------------- //
()
// -------------------------------------------------------------------- //
{
  return null;
} // TRexGetJarVersions
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
// <== implementation of TRexEventListener interface                    //
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
/**
 * Send data to the host transaction program. The first 2 bytes of the byte
 * buffer to be sent have to include the length of the buffer over all.
 * <p>
 * Before the data to send is appended to the buffer it is translated from
 * unicode to EBCDIC because the host transaction program expects EBCDIC
 * data to be sent.
 * <p>
 * After backpatching the APPC demanded length field (the first 2 bytes) the
 * data is extracted from the XPSByteBuffer and sent to the host transaction
 * program using TRex's writeData() method.
 */
public
void
write                                // write data to host
// -------------------------------------------------------------------- //
(String data,
 int    iFlag)
// -------------------------------------------------------------------- //
{
  // reset buffer
  m_writeBuffer.setLength(0);
  // mark position for length backpatcher
  m_writeBuffer.markPosition(SIZE_OF_SHORT);
  // inform user
  if (0 != (WRITE_LOG & m_iFlags))
    System.out.println("write: data=["+data+"] TSW=["+iFlag+"]");
  // translate to EBCDIC
  data = m_transTab.UC2EB(data);
  // insert data to send
  m_writeBuffer.append(data);
  // complete the length field
  m_writeBuffer.insertAtMarkedPosition((short)m_writeBuffer.length());
  // get hold of buffer
  byte[] bDataOut = m_writeBuffer.toBuffer();
  // send buffer to host
  m_trex.writeData(iFlag, bDataOut, bDataOut.length);
} // printHelp

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ //
```

```
    /**
     * Display information about program usage.
     */
    static
    void
    printHelp                        // print some help information
    // ------------------------------------------------------------------- //
    ()
    // ------------------------------------------------------------------- //
    {
      System.out.println("*****************************************************************");
      System.out.println("* TRexTester accepts the following arguments:                   *");
      System.out.println("*                                                               *");
      System.out.println("* -?            : prints this help information                  *");
      System.out.println("* -A<ipAddress>: connects to specified IP address               *");
      System.out.println("* -E            : activates use of security protocol            *");
      System.out.println("* -I<PKCS12>    : PKCS#12 instance stored in TRex database      *");
      System.out.println("* -L            : writes console log                            *");
      System.out.println("* -O<password> : password for PKCS#12 instance access          *");
      System.out.println("* -P<ipPort>    : connects to specified port                    *");
      System.out.println("* -S<service>   : request the specified TRex service            *");
      System.out.println("* -T            : writes trace to file TRexTester.tra           *");
      System.out.println("* -U<username> : specifies username for remote system logon     *");
      System.out.println("* -W<password> : specifies password for remote system logon     *");
      System.out.println("*                                                               *");
      System.out.println("* Example: -L -Awww.anyhost.com -P8885 -SSAMPLE -I\"Axel F\" -Ozappa *");
      System.out.println("*****************************************************************");
    } // printHelp
  } // class TRexTester
```

Die Programmausführung ist unter Windows Betriebssystemen durch folgende Kommandozeileneingabe zu starten:

```
java -cp trex.jar xps.trex.TRexTester <Argumente>
```

Das Archiv trex.jar enthält die Java Klasse TRexTester.class, deren Quelltext oben abgebildet ist. Mit Hilfe der Kommandozeilenargumente '-A', '-P' und '-S' ist das Testprogramm darüber zu informieren, welcher TRex Service unter welcher IP-Adresse angefordert werden soll.

Durch Aufruf des Programms mit dem Kommandozeilenargument "-?" können die verfügbaren Programmstartoptionen aufgelistet werden. Dazu wir die Methode "printHelp" aufgerufen.

Programmlogik

main

Die Methode ***main*** analysiert zunächst die übergebenen Kommandozeilenargumente und erzeugt anschließend eine neue Instanz von ***TRexTester***, sofern das Kommandozeilenargument '-?' nicht gefunden wurde.

Konstruktor

Im ***TRexTester*** Konstruktor wird zunächst das Integer Feld ***iCommonOptions*** gefüllt, das Flags zur Angabe der allgemeinen Optionen enthält. Von besonderer Bedeutung ist die Angabe des Flags ***TRex.COM_HALF_DUPLEX_MODE,*** mit dem festgelegt wird, dass die aufzubauende Verbindung im Half-Duplex Modus betrieben werden soll. Wie bereits oben erwähnt, müssen sich die beiden Kommunikationspartner im Half-Duplex Modus gegenseitig das Senderecht einräumen. Bei Verwendung einer APPC-Verbindung ist die Wahl dieses Modus verpflichtend.

Danch wird eine neue Instanz eines TRex Objekts erzeugt. Dabei ist der erste Parameter - hier ***this*** - sehr wichtig. Mit diesem Parameter wird TRex darüber informiert, welches Objekt als ***TRexEventListener*** fungieren soll. In diesem Fall ist das ***TRexTester***, was durch das ***implements TRexEventListener*** Statement erreicht wird. Das hier übermittelte Objekt wird von TRex zu verschiedenen Zeitpunkten über das Eintreten bestimmter Ereignisse informiert.

Falls über die Kommandozeile ein Benutzername oder ein Passwort angegeben wurde, werden diese Angaben durch Aufruf der TRex Methode ***specifyUserData()*** an TRex übergeben,

Sollte die Verwendung von Security-Optionen gefordert worden sein, werden diese durch einen Aufruf der TRex Methode ***specifySecurityOptions()*** gesetzt.

Danach erzeugt das Testprogramm eine Instanz der Klasse ***XPSTransTab***. Mit Hilfe dieser Instanz können lesbare Daten von EBCDIC nach Unicode und umgekehrt übersetzt werden.

Als letztes wird im Konstruktor die Öffnung des Kommunikationstunnels zu XPSDaemon veranlasst. Dies geschieht durch Aufruf der TRex Methode ***openTunnel()***, womit sämtliche vorbereitende Schritte ausgeführt sind.

TRex baut nun eine TCP/IP Verbindung zu XPSDaemon mit den gegebenen Sessionparametern auf. Sobald die Verbindung erfolgreich aufgebaut wurde, wird der ***TRexEventListener*** durch Aufruf der Methode ***TRexTunnelReady()*** darüber informiert.

TRexTunnelReady

In dieser Methode führt das Clientprogramm zwei direkt aufeinanderfolgende Sendbefehle aus. Dies wird durch Aufruf der lokalen Methode ***write()*** erreicht. Dabei ist zu beachten, dass beim ersten Aufruf für das Flag ein Wert von ***TRex.TSW_KEEP_DIRECTION*** gesetzt wird, womit das Recht zum Senden bei ***TRexTester*** verbleibt. Beim zweiten Aufruf wird das Flag ***TRex.TSW_CHANGE_DIRECTION*** gesetzt. Das hat zur Folge, dass das Beispielprogramm beim zweiten Aufruf das Sendrecht abgibt und in den Empfangmodus wechselt.

write

Die ***write()*** Methode erwartet als Übergabeparameter eine Zeichenkette, die an das Hostprogramm gesendet werden soll, sowie ein Flag, mit dem der Status, den das Testprogramm nach Ausführung des Versendens der Daten einnehmen soll, festgelegt wird.

Zunächst werden in dieser Methode die Daten für das Versenden vorbereitet. Dazu wird eine Instanz von ***XPSByteBuffer*** verwendet. Nach Initialisierung des Puffers werden zunächst 2 Byte für die Länge der nachfolgenden Daten reserviert. Dann werden die Daten nach EBCDIC übersetzt, da das Hostprogramm die gesendeten Daten in EBCDIC erwartet. Dann werden die übersetzten Daten an den Puffer angehängt und das Längenfeld gefüllt. Das extrahierte Bytefeld wird schließlich an TRex zum Versenden übergeben. Dies erfolgt durch Aufruf der TRex Methode ***writeData()***.

TRexDataRead

Nachdem mit dem zweiten Sendbefehl das Senderecht an das Hostprogramm übergeben wurde, ist das nächste Ereignis, über das das Testprogramm in Kenntnis gesetzt wird, das Eintreffen von Daten, die vom Hostprogramm gesendet werden.

Dazu wird die ***TRexEventListener*** Methode ***TRexDataRead*** von TRex unter Übergabe der erhaltenen Daten aufgerufen. Zunächst werden die erhaltenen Daten protokolliert. In diesem Rahmen findet eine Übersetzung der Daten von EBDIC nach Unicode statt. Dies erfolgt unter Verwendung einer Instanz von ***XPSByteBuffer***. Bei Einhaltung der gezeigten Vorgehensweise wird sichergestellt, dass die gelesenen Daten korrekt übersetzt werden. Schließlich wird getestet, ob das Testprogramm wieder Daten senden soll. Dies erfolgt durch Überprüfung des Schalters ***TRex.TSW_CHANGE_DIRECTION*** im übergebenen Tunnel Status Wort. Wenn dies der Fall ist, wird eine letzte Zeichenkette 'Goodbye' an das Hostprogramm gesendet und das Terminierungsflag ***m_fTerminate*** auf 'true' gesetzt.

TRexOnEvent

Mit Hilfe dieser Methode informiert TRex einen ***TRexEventListener*** über das Eintreten verschiedener Ereignisse. Die einzelnen Ereignisse werden unter Verwendung eines 'case-Statements' unterschieden.

Von besonderem Interesse ist im Testprogramm das Eintreten des Ereignisses ***TRexEventListener.EV_DATA_WRITTEN***. Mit diesem Ereignis informiert TRex den Listener jedes Mal über den Umstand, dass Daten versendet wurden. Wenn dieses Ereignis eintritt, wird überprüft, ob das Terminierungsflag ***m_fTerminate*** gesetzt ist. Ist dies der Fall, veranlasst das Testprogramm die Schließung des Tunnels unter Verwendung des Aufrufs der TRex Methode ***closeTunnel()***. Das Flag sollte vor und nicht nach dem letzten Aufruf der ***write()*** Methode gesetzt werden, da durch den asynchronen Ablauf der Kommunikation der Fall eintreten könnte, dass der ***TRexEventListener*** über das Ereignis ***EV_DATA_WRITTEN*** früher informiert wird, als das Flag nach Rückkehr aus der ***write()*** Routine gesetzt werden kann.

Schließlich wird der ***TRexEventListener*** über die erfolgte Schließung des Tunnels mit Hilfe des Ereignisses ***TRexEventListener.EV_TUNNEL_CLOSED*** informiert. Das Eintreten dieses Ereignisses wird zum Anlaß für die Programmbeendigung genommen.

Konsolenausgabe

Falls der Kommandozeilenparameter '-L' angegeben wird, werden während der Ausführung des Testprogramms z.B. folgende Informationen auf System.out protokolliert:

```
TRex API test program: Version 1.8
service=[SAMPLE], address=[192.168.0.200], port=[8885]
XPS TRex Version 1.8.0 - 10.10.2002
onEvent: event=EV_DATA_WRITTEN
tunnelReady: TSW=[10000001] dataLength=0
write: data=[Hello] TSW=[0]
onEvent: event=EV_DATA_WRITTEN
write: data=[This is a TRex sample] TSW=[1]
onEvent: event=EV_DATA_WRITTEN
dataRead: data=[TEST SEND 02] TSW=[10000000] length=12
dataRead: data=[TEST SEND 03] TSW=[10000001] length=12
dataRead: about to force termination
write: data=[Goodbye] TSW=[0]
onEvent: event=EV_TUNNEL_CLOSED
```

Nach erfolgreicher Ausführung des Testprogramms müssen sich im CICS die gesendeten Daten in der Temporary Storage Queue 'TREXOUT' befinden. Die Anzeige des TS Queue Inhalts kann durch Starten der Transaktion 'CEBR TREXOUT' erfolgen. Die ersten zwei Zeichen jedes Eintrags enthalten das vom APPC Protokoll geforderte Längenfeld.

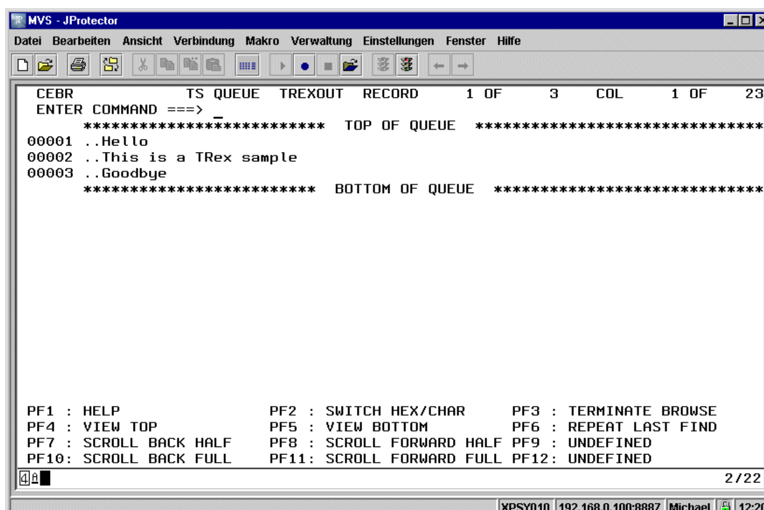

Abb. 13: CICS TS-QUEUE TREXOUT

Anhand dieses Beispiels kann die prinzipelle Vorgehensweise bei der Entwicklung von Porgrammen, die TRex Dienste benutzen, nachvollzogen werden. Die einfache Struktur des TRex APIs und die Bereitstellung von Host APPC Adapterprogrammen durch XPS erlaubt es, dass das Hauptaugenmerk bei der Entwicklung auf die Implementierung der Geschäftslogik gerichtet werden kann.

Die vollständige Beschreibung des TRex APIs erfolgt in der im Installationsumfang enthaltenen HTML basierten Onlinedokumentation auf Basis von Javadoc.

Kapitel

4

Tools

Verwaltung von Zertifikaten

Verschiedene Einstellungen zur Konfiguration von TRex und auch hostseitig in XPSDaemon erfordern es, dass TRex digitale X.509 V3 Zertifikate verarbeitet. Clientseitig werden diese Zertifikate in einer XPS Datenbasis gespeichert. Der TRex Client enthält eine Klasse namens ***TRexCertMan***, die zur Verwaltung der Zertifikate, die in der Standarddatenbasis trex.xpsdb gespeichert sind, verwendet werden kann.

Das Programm ist unter Windows folgendermassen zu starten:

```
java -cp trex.jar xps.trex.TRexCertMan
```

Das Programm verwendet die nachfolgend beschriebenen Dialoge zur Verwaltung der Zertifikate.

Benutzerzertifikate

Benutzerzertifikate sind notwendig, um einen Client gegenüber XPSDaemon zu authentifizieren. Falls der XPSDaemon Administrator die Clientauthentifizierung verlangt, muss die XPS Datenbasis mindestens ein gültiges Benutzerzertifikat enthalten. Die Verwendung des Zertifikats erfolgt durch Angaben für die Parameter ***pkcs12Instance*** und ***pkcs12Password*** der Methode ***specifySecurityOptions()*** des TRex APIs.

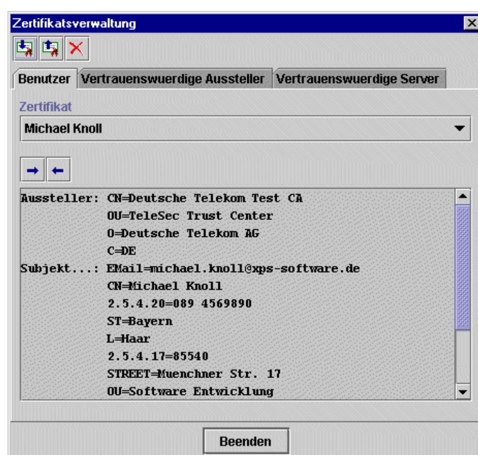

Abb. 14: Verwaltung von Benutzerzertifikaten

Der Dialog stellt die folgenden Funktionen zur Zertifikatsverwaltung bereit:

→, ← Nächstes in Kette/Vorheriges in Kette

Sollte die ausgewählte PKCS#12 Datei mehr als ein Zertifikat enthalten, kann mit diesen Schaltflächen zwischen den verschiedenen Zertifikaten geblättert werden. Wenn eine PKCS#12 Datei mehrere Zertifikate enthält, befinden sich darin neben dem Benutzerzertifikat auch noch ein oder mehrere Ausstellerzertifikate.

Importieren

Mit dieser Funktion können neue Zertifikate in die JProtector Datenbasis importiert werden. Dazu wird ein Dateiauswahldialog eingeblendet, in dem die Lokation der PKCS#12 Datei auszuwählen ist, die importiert werden soll.

Nach der Auswahl der Datei erfragt JProtector das Passwort, das zur Verschlüsselung der Daten in der PKCS#12 Datei verwendet wurde. Nach dem Importieren steht das Zertifikat zur Verwendung in Sessions zur Verfügung.

Hinweis: JProtector kann PKCS#12 Dateien importieren, die von den neueren Browserversionen von Microsoft (*.pfx) und Netscape (*.p12) exportiert werden.

Exportieren

Zuvor importierte PKCS#12 Dateien können unter Verwendung dieser Funktion wieder exportiert werden. Die exportierte Datei ist eine 1:1 Kopie der zuvor importierten PKCS#12 Datei.

Löschen

Mit dieser Funktion können Zertifikate, die nicht verwendet werden, aus der JProtector Datenbasis gelöscht werden.

Vertrauenswürdige Aussteller

Zertifikate vertrauenswürdiger Aussteller werden zur Prüfung der Gültigkeit von Serverzertifikaten herangezogen. Das bedeutet, dass das Zertifikat des Ausstellers eines XPSDaemon Serverzertifikats in der XPS Datenbasis als vertrauenswürdiger Aussteller geladen sein muß. Die Verwaltung erfolgt unter Verwendung der nachfolgend abgebildeten Registerkarte:

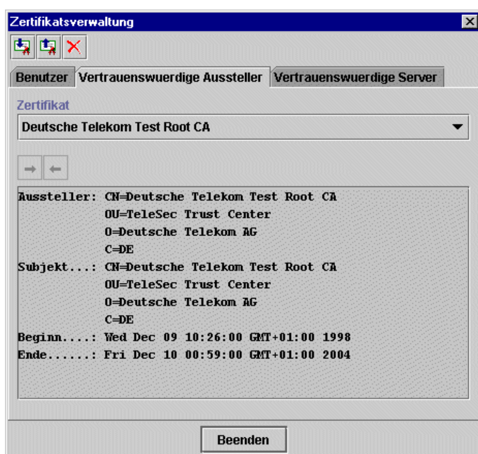

Abb. 15: Vertrauenswürdige Aussteller

Die Funktionen zur Verwaltung entsprechen den oben genannten. Es ist jedoch zu beachten, dass Zertifikate vertrauenswürdiger Aussteller nicht als PKCS#12 Dateien gespeichert werden.

Vertrauenswürdige Server

TRex speichert Zertifikate von XPSDaemon Servern, die als vertrauenswürdig gekennzeichnet wurden, in der XPS Datenbasis. Mit der entsprechenden Registerkarte können die momentan gespeicherten Zertifikate verwaltet werden. Dazu wird die nachfolgend abgebildete Registerkarte verwendet:

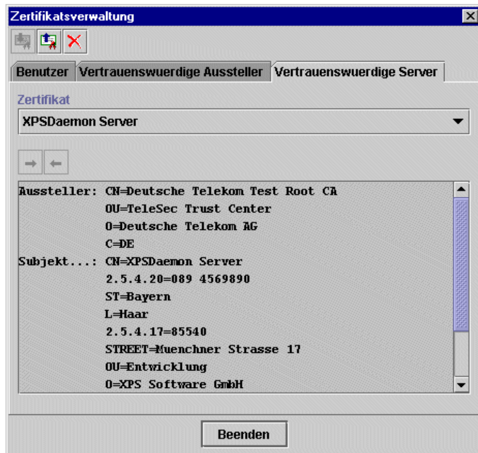

Abb. 16: Vertrauenswürdige Server

Die Funktionen zur Verwaltung entsprechen den oben genannten. Allerdings ist das Importieren vertrauenswürdiger Serverzertifikate nicht erlaubt. Damit ein Serverzertifikat in die XPS Datenbasis gelangt, muß es von einem XPSDaemon Server im Rahmen der Serverauthentifizierung gesendet, und vom Benutzer als vertrauenswürdig akzeptiert werden.

Index

A

Adapterprogramm ... 16
Administrator... 7
APPC... 6, 7, 10, 11, 16, 29

B

Blowfish... 5

C

CEBR ... 29
CEDA... 12
CICS...7, 11, 14
Communication Area... 10

D

DFHCSDUP... 11

F

Firewall... 5

H

Half-Duplex...10, 27
Host ... 7

I

Integritätsprüfung... 5

J

Java Archive... 6
Java Virtual Machine... 5

K

Komprimierung... 5

P

PKCS#12... 32

S

Senderecht... 10
Service...6, 14

T

TCP/IP ... 5
Transaktion... 5
Tunnel... 16
Tunnel Status Wort... 28

V

Verschlüsselung... 5
VTAM ... 13

X

XPSDaemon...5, 7

Z

Zertifikat ...5, 31