

# Transfermonitor

**Automatisierung**

**des**

**AOXS basierten Datentransfers**

Dokumenttyp	Autor	Revision	Datum
Anwenderhandbuch	Horst Fiedler	1.6	März 2002

### **Abstrakt**

Der Transfermonitor ergänzt die im AOXS-Handbuch festgehaltenen Datentransfermöglichkeiten um Automatisierung und Visualisierung.

### **Versionsgeschichte**

0.1	9.7.2001	Erstversion
1.0	15.11.2001	Als Teil von Transfer 1.1
1.1	8.1.2002	Hinweise zu Fehlersuche
1.2	31.1.2002	Parametrierung in Schritten
...		

### **Weitere Literatur**

- [1] AOXS 1.1 ASAM-ODS Datenaustausch mit XML, XSLT und ECMAScript
- [2] AopWeb 2.7 Die Web-Anbindung for ASAM-ODS V3.x Server

### **Über den Verfasser**

Hr. Dipl.-Ing. H. Fiedler ist erreichbar per EMail [horst.fiedler@tiff.com](mailto:horst.fiedler@tiff.com). Firmenanschrift, Telefon/Fax, usw. siehe <http://www.tiff.com/>.

# Inhaltsverzeichnis

<b>1</b>	<b>Überblick</b>	<b>1</b>
<b>2</b>	<b>Bedienung des Monitors</b>	<b>3</b>
2.1	Aktivierung mit WebStart . . . . .	3
2.2	Aktivierung in Consolefenster . . . . .	5
2.3	Verwendung . . . . .	5
2.3.1	Statustabellenanzeige . . . . .	5
2.3.2	Startdialog . . . . .	6
<b>3</b>	<b>Administration</b>	<b>9</b>
3.1	Stoppen und Wiederaanlauf des Servers . . . . .	9
3.2	Sichern von Laufzeitdaten . . . . .	10
3.3	Fehlersuche . . . . .	11
3.4	Weitere Tipps . . . . .	13
<b>4</b>	<b>Installation</b>	<b>15</b>
<b>5</b>	<b>Parametrierung</b>	<b>17</b>
5.1	Automatisierungsdienst . . . . .	17
5.2	Transferdienst . . . . .	17
5.2.1	Abbildungs-Template . . . . .	18
5.2.2	Transfer-Skript . . . . .	18
<b>A</b>	<b>Glossar</b>	<b>23</b>
<b>B</b>	<b>Beispiele</b>	<b>27</b>
B.1	Transferdienst (als Webservice) . . . . .	27
B.2	Automatisierungsdienst . . . . .	35



# Kapitel 1

## Überblick

Der automatisierte Datentransfer ist eine in mehrere Komponenten unterteilte, verteilt implementierte Anwendung. Jede Komponente (als Blase dargestellt) kann auf einem anderen Host ablaufen.

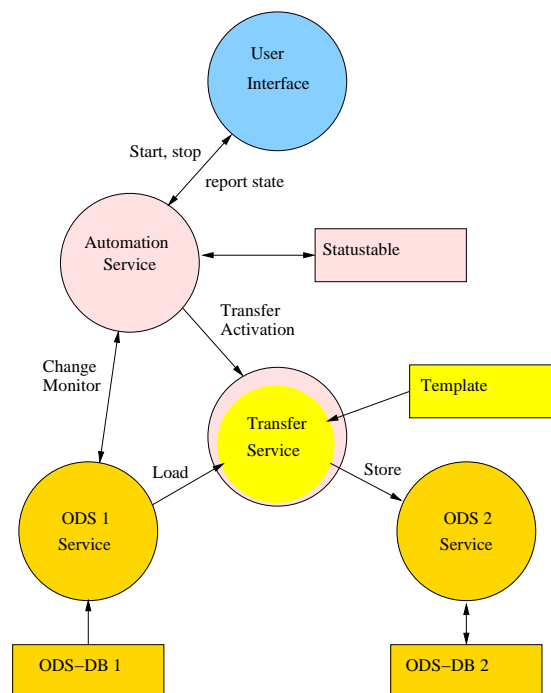


Abbildung 1.1: Gesamtanwendung

Die einzelnen Komponenten einer Konfiguration der Datentransferanwendung sind, von unten nach oben:

- zwei ASAM-ODS Dienste, bereitgestellt von 3'ter Seite, z.B. von AVL. Der die Datenquelle bedienende Dienst muß nur lesen können, der die Datensinke bedienende Dienst muß alle ODS 3.2 Operationen unterstützen.

- ein Transferdienst, der die Abbildung der Datenstruktur und prozedurale Umformungen vornimmt. Dieser Dienst kapselt den in JavaScript parametrisierten Datentransfer, der seinerseits ein XML-Templat für die auf ODS 2 definierte Sicht nutzt.
- ein Automatisierungsdienst, der Statusänderungen beobachtet (Datum/Zeitfeld in ODS 1) und im Bedarfsfall den Transferdienst aktiviert. Der Status der Transfers wird in einer persistenten Statustabelle gepflegt.
- Anzeige- und Bedienstelle(n) erlauben Anzeige und Bedienung von entfernten Arbeitsplätzen. Falls der Datentransfer aktiviert ist, läuft er auch ohne Bedienstelle weiter. Bedienstellen können sich jederzeit einhängen.

Es können mehrere Automatisierungsdienste (mit unterschiedlichem Namen) zur Bedienung mehrerer Quelldatenserver installiert sein, jeder Automatisierungsdienst kann zu einer Zeit nur einen Transferdienst nutzen, es kann jedoch zur Laufzeit festgelegt werden welcher von mehreren vorbereiteten Diensten für die jeweils nachfolgenden Datentransfers zu verwenden ist.

Üblicherweise sind aber nur 3 Rechner beteiligt: Automatisierung, Transfer und Zieldatenserver laufen am selben Host (Betriebssystem frei wählbar, Voraussetzung ist, daß Java zur Verfügung steht) ab, nur die Quelldaten kommen von einem anderen Rechner (z.B. einer VAX) und die Bedienung erfolgt von einem Desktoprechner (z.B. einem Windows-PC) aus.

Falls die Automatisierung zur Verwendung der `exec()` - Method, um den Transfer zu aktivieren, parametrisiert wird, wird Automatisierung und Transfer immer am selben Host ausgeführt.

# Kapitel 2

## Bedienung des Monitors

### 2.1 Aktivierung mit WebStart

Der Monitor wird am Arbeitsplatz des Anwenders aus dem Internetexplorer aktiviert. Im Adressenfeld des IE (oder Netscape Navigator) wird der URL des mitgelieferten AopWeb angegeben, z.B.

`http://s10mva01:8080/aopweb`

und auf der Seite, auf der die bekannten Datendienste angeboten werden, der Knopf "Sonstig" gedrückt. Auf der nachfolgenden Seite findet sich dann der Transfermonitor.



Abbildung 2.1: Anwendung starten

Durch Anwahl (Mausklick) wird die Anwendung aktiviert. Falls die Anwendung noch nie auf den Rechner des Anwenders geladen wurde oder eine neue Version am Server vorhanden ist, erscheint



Abbildung 2.2: Download

und

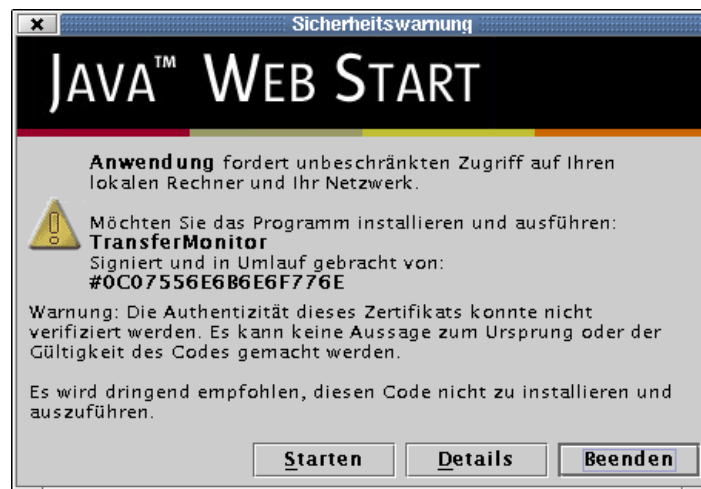


Abbildung 2.3: Sicherheitscheck

Je nach Einstellung des WebStart - Packets wird auch nachgefragt, ob für diese Anwendung ein Desktop- und/oder Windows-Startmenü-Eintrag erzeugt werden soll, um nachfolgende Aktivierungen noch einfacher und ohne Aop-Web durchführen zu können. Es erfolgt dabei immer eine Überprüfung, ob am Desktop des Anwenders die neueste Version vorhanden ist, falls nicht, wird nachgeladen.

## 2.2 Aktivierung in Consolefenster

Falls die Verwendung von WebStart nicht gewünscht ist, kann der TransferMonitor auch aus einem Consolefenster aktiviert werden. Dies ist abhängig vom verwendeten Betriebssystem. Zum Start auf HP-UX: Siehe `./bin/transfermonitor`, auf Windows PC's kann z.B.

```
java -Dautomation.router=http://aserver:8080/aopweb/servlet/rpcrouter
-Dautomation.service=axs-automation
-Dtransfer.service=transfer-04
-jar transfermonitor.jar
```

verwendet werden. Dabei ist `aserver` der Host, auf dem die Automatisierung läuft, `axs-automation` ist der Name des Automatisierungsdienstes, `transfer-04` ist der Name des Transferdienstes (kann im Startdialog geändert werden), die Datei `transfermonitor.jar` sollte aus dem gelieferten Transferpaket auf die Festplatte kopiert worden sein.

## 2.3 Verwendung

Zur Bedienung steht ein GUI zur Verfügung, das sowohl lokal auf der Maschine, auf dem auch das Automatisierungsdienst abläuft, als auch remote, z.B. auf einem Desktop-PC, der Java installiert hat, ausgeführt werden kann.

### 2.3.1 Statustabellenanzeige

TRR Datei	Datum Datei	Transferme.	Fertig um	Statustext	Status
ST3064_031	1020318105538	transfer-05	20020318143141	457ST3064 031_FU	fertig
ST3064_030	1020318103037	transfer-05	20020318143053	457ST3064 030_FU	fertig
ST3064_029	1020318095258	transfer-05	20020318142957	457ST3064 029_FU	fertig
ST3064_028	1020318095123	transfer-05	20020318142913	NO DATA	fertig
ST3064_027	1020318083304	transfer-05	20020318142846	457ST3064 027_FU	fertig
ST3064_026	1020315095352	transfer-05	20020318142630	457ST3064 026_FU	fertig
ST3064_025	1020315084704	transfer-05	20020318142541	457ST3064 025_FU	fertig
ST3064_024	1020315074550	transfer-05	20020318140238	457ST3064 024_FU	fertig
ST3064_023	1020314143329	transfer-05	20020318140155	457ST3064 023_FU	fertig
ST3064_022	1020314135229	transfer-05	20020318140112	457ST3064 022_FU	fertig
ST3064_021	1020314134936	transfer-05	20020318140025	NO DATA	fertig
ST3064_020	1020314131103	transfer-05	20020318135949	457ST3064 020_FU	fertig
ST3064_019	1020314130536	transfer-05	20020318135905	NO DATA	fertig
ST3064_018	1020314112526	transfer-05	20020318135839	NO DATA	fertig
ST3064_017	1020314105740	transfer-05	20020318135814	457ST3064 017_FU	fertig
ST3064_016	1020314095345	transfer-05	20020318135729	457ST3064 016_FU	fertig
ST3064_015	1020313134729	transfer-05	20020318135646	457ST3064 015_FU	fertig

Abbildung 2.4: Bedienung Automatisierung

Bedienelemente:

- Der erste Knopf in der Knopfleiste erlaubt die Aktualisierung der Anzeige, wofür die Statustabelle vom Automatisierungsdienst abgeholt wird,
- das Diskettensymbol erlaubt das lokale Speichern der Statustabelle,
- der Startknopf erlaubt die Aktivierung der Automatik,
- der Stopknopf deaktiviert die Automatik, falls ein Datentransfer aktiv ist, erscheint eine Warnmeldung und der Stop-Versuch kann abgebrochen, wiederholt oder forciert (der laufende Transfer wird abgebrochen) werden.
- In der Tabelle ist nur das Statusfeld änderbar, um z.B. einen Neutransfer zu initiieren,
- Click auf Status- bzw. Fehlertextspalte zeigt den gesamten Text, falls das Feld in der Tabelle zu schmal ist,
- Click in andere Spalten sortiert die Tabelle nach dieser Spalte, ein weitere Click in dieselbe Spalte dreht die Sortierreihenfolge um,
- das ampelfarbige Feld links unten zeigt den Status der Automatisierung (aktiviert = grün, inaktiv = rot, undefiniert = orange) und im Feld entweder das aktuelle Intervall im grünen Feld oder die letzte Stop-Ursache im roten Feld: beendet durch Anwender = 1, QuellDatenserver nicht mehr erreichbar = 2, durch wiederholte Transferfehlversuche = 3, durch Systemfehler = 4, kill-Aktion = 9,
- der Schieberegler erlaubt es, die Wartezeit zwischen ergebnislosen Abfragen an den Quellserver, ob sich Daten änderten, anzupassen,
- das Datumfenster zeigt, wann zuletzt die Statusdatei geändert wurde.

### 2.3.2 Startdialog

Nach Aktivierung des Startknopfes erscheint ein Dialog:



Abbildung 2.5: Start dialog

Diese Aktivierung/Deaktivierung kann für GUI-losen Betrieb, z.B. zur automatischen Reaktivierung nach Systemneustart oder um nur zu bestimmten Tageszeiten Transfers durchzuführen, auch aus der Kommandozeile (z.B. auch im Boot-Skript des Rechners) erfolgen.

Die Frage der Security (Authentizierung) ist noch nicht behandelt. Da der im Hintergrund ablaufende Automatisierungsdienst keine Zugriffsaauthentizierung erfordert, sollte der Zugriff auf die Automatisierung vorerst durch die Bereitstellung des Zugriff auf die die Automatisierung kontrollierende Software geregelt werden. Z.B. kann das AopWeb-Passwort geändert werden (verhindert Download des Monitors) oder die File-Permissions auf dem Server werden so gesetzt, daß nur autorisierte Anwender Zugriff auf die Datei `transfermonitor.jar` haben.

Die im Startdialog abgefragte Userauthentizierung dient nur dazu, die Zugriffsberechtigung auf den Zielsystem (z.B. Santorin) sicherzustellen. Diese Information wird zur Speicherung der Daten benötigt, sofern der Zielsystem eine Authentizierung verlangt, verhindert aber nicht, daß Datentransfers eingeleitet werden oder der Status von Statustabelleneinträgen geändert werden kann.



# Kapitel 3

## Administration

Die Administration der Anwendung erfolgt am Server, d.h. alle relevanten Konfigurationsparameter und sonstige Daten sind am Server im Anwendungsverzeichnis.

Für wiederkehrende Aufgaben ist ein einfaches Text Menü vorhanden. Dieses Menü ist für UNIX - Server als Shellskript in `./bin/menu` hinterlegt. Falls der Server unter Windows betrieben wird, sollten die Aktionen in Desktop- im Start-Menü untergebracht werden (Anpassung durch Administrator).

Nach Login am Server und Eingabe von "menu" erscheint:

```
Actions available:
santorin          s
tomcat            t
statustable view  v
statustable erase e
logfiles list     f
list end of most recent l
delete old logfiles g
transfermonitor (X11) m
netscape aopweb (X11) n
automation start  a
automation stop   b
check for java    j
readme           r
Choice:
```

"santorin" und "tomcat" führen in Untermenü's, mit "(X11)" gekennzeichnete sind nur zulässig, wenn ein X-Display (X - Windows ist das übliche Fenstersystem auf allen Betriebssystemen außer denen von Microsoft, dort nur über Zusatzsoftware, z.B. Reflection) zur Verfügung steht.

### 3.1 Stoppen und Wiederanlauf des Servers

ist via Tomcat - Untermenü möglich:

```
Tomcat Actions -----
start                s
stop                 x
process status      p
kill                 k
log console (X11)   c
last 200 log lines  l
services deploy     d
services dds remove u
                    Choice:
```

Achtung: Nicht alle Aktionen sind in jedem Zustand zulässig:

x, d tomcat muß laufen,

u tomcat soll gestoppt sein. u ist NICHT undeploy sondern löscht die Datei in der die Dienste persistent registriert sind.

x Funktioniert z.Z. nicht auf HP-UX. Falls Prozess nicht terminiert muß kill -TERM (k) verwendet werden.

k Sollte nur als Nothilfe verwendet werden. Die Automatisierung sollte vorher abgedreht werden (STOP-Button im transfermonitor), da anderenfalls die Statustabelle nicht geschrieben wird und daher der Zustand der Transfers nicht gesichert würde,

d ist nur dann nötig, wenn Dateien im Verzeichnis ./dds geändert wurde oder neu installiert oder ein "services remove" durchgeführt wurde.

D.h. im Normalfall ist nur "start" z.B. nach Booten der Maschine, nötig, sofern kein Start aus dem Bootskript (init.d/rc-Konfiguration) erfolgt.

### 3.2 Sichern von Laufzeitdaten

Die anwendungsspezifischen Dateien sind vor einer Neuinstallation zu sichern und nach der Installation wiederherzustellen. Das gilt insbesondere für die persistente (zur Laufzeit entstehende) Statustabelle.

Wesentliche Dateien, die ev. über Neuinstallation gerettet werden sollten, sind:

- apache/jakarta-tomcat/conf/tomcat-users.xml  
Access control für Servlet-Zugriffe
- apache/jakarta-tomcat/webapps/aopweb/WEB-INF/web.xml  
Servlet Deployment Descriptor
- apache/jakarta-tomcat/webapps/aopweb/aoxs-automation.txt  
Statustabelle (persistent)
- apache/jakarta-tomcat/webapps/aopweb/DeployedServices.ds  
Registrierte Webservices

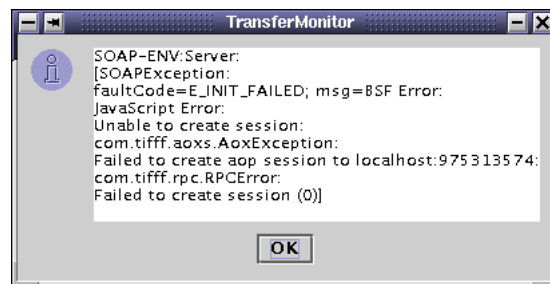
- apache/jakarta-tomcat/webapps/aopweb/WEB-INF/\*.xml (exkl. web.xml), Benutzerspezifische Views und Subschemas
- scripts (alle)
- templates
- dds  
Spezifische Transferkonfigurationen

Zum Sichern dieser Dateien kann das Skript `backup` verwendet werden.

### 3.3 Fehlersuche

Der Transfermonitor besteht aus 3 Teilen: Einem Bedienteil (GUI) und einem Automatisierungsteil (Automatisierungsdienst) und einem Datentransferprogramm (AOXS). Zumeist wird nur der GUI-Anteil am Desktop des Anwenders laufen, während Automatisierung und Transfer am Server durchgeführt werden. Folgende Quellen stehen für eine Problemsuche in den jeweiligen Komponenten zur Verfügung:

- Bildschirmausgabe, z.B.



**Abbildung 3.1:** Fehler beim Herstellen der Verbindung

kann bei falscher Eingabe im Startdialog erscheinen. In all diesen Fällen wird der Fehler in die Standardausgabe der Anwendung geschrieben bzw. bei JNLP-Verwendung in das optionale Konsolenfenster. Die meisten Probleme im Bereich des GUI sind einfache Fälle, z.B. Java (ist Voraussetzung) nicht vorhanden. Der hier gezeigte Fall ist allerdings ein Fehler, der den Automatisierungsteil betrifft.

Die Meldung "Automatisierungsdienst nicht verfügbar" erscheint, falls der Dienst nicht installiert wurde (WebService nicht "deployed") oder nicht starten konnte, z.B. Statusdatei nicht les-/anlegbar. Letztere Ursache ist oft durch relative Dateiengabe zusammen mit unvorhergesehenem "Working Directory" des Webdienstes (Tomcat) verursacht.

- Die Automatisierung verwendet die Log-Datei der Servlet-Engine (Tomcat) zur Protokollierung. Diese findet sich im Verzeichnis "logs". Der oben gezeigte Fehler ist dort ebenfalls sichtbar, z.B.

```

ERROR AoxException: Failed to create aop session to localhost:975313574:
com.tiff.rpc.RPCError: Failed to create session

ERROR com.tiff.js.Aop.jsConstructor():
com.tiff.aoxs.AoxException: Failed to create aop session to localhost:975313574:
com.tiff.rpc.RPCError: Failed to create session (0)
com.tiff.aoxs.AoxException: Failed to create aop session to localhost:975313574:
com.tiff.rpc.RPCError: Failed to create session (0)
  at com.tiff.aoxs.aop.AopService.createSession(AopService.java:120)
  at com.tiff.js.Aop.jsConstructor(Aop.java:91)

  at java.lang.reflect.Method.invoke(Native Method)
  at org.mozilla.javascript.FunctionObject.callVarargs(FunctionObject.java:593)
  at org.mozilla.javascript.FunctionObject.construct(FunctionObject.java:539)
...

```

- Transferfehler  
sind immer bezogen auf eine TRR-Datei und sollten unter dem Namen der Trr-Datei im Verzeichnis logs zu finden sein. Falls ein Transfer mit einem Fehler beendet wird, wird statt "uptodate" ein Fehlercode in das Statusfeld geschrieben und der Text der Fehlermeldung (Exception) wird durch Pointclick ins zugehörigen Textfeld angezeigt, z.B.

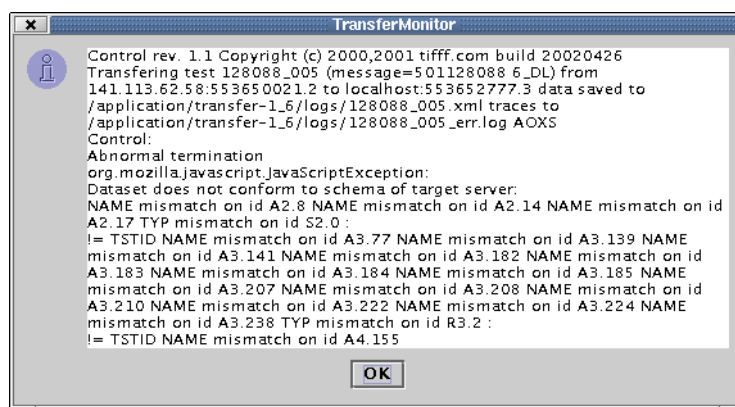


Abbildung 3.2: Fehler beim Transfer

wenn das Template für den Transfer mehr als die erlaubten Unterschiede zum Schema des Zielsystems aufweist.

Falls die Ursache nicht ermittelbar ist, empfiehlt es sich, den Transfer im Batch-Modus zu wiederholen und dabei erweiterte Traces einzuschalten.

Die Aktivierung eines einzelnen Transfers erfolgt mittels Kommando, z.B. in Terminalfenster der HP:

```
> control scripts/transfer-04.js debug=true SM3152_020
```

Vorteil: Direkte Terminalausgabe der Traces, rascher turnaround beim Testen, es werden keine WeBServices benötigt

Nachteil: Keine Statustabellenupdates, d.h. das Gesamtsystem weiß nichts von der Durchführung.

Falls nur das Speichern des Transferdokuments nicht möglich war, kann auch das in diesem Fall als XML Datei gesicherte Dokument in den Zielsever importiert werden, z.B. mit

```
> control scripts/aoxs-import.js trrname=SM3306_019a debug=true
```

aber ACHTUNG: Das gesicherte XML-Dokument enthält keine Processing-Instructions, d.h. diese sind vorher händisch nachzutragen, falls der Default (update existing, insert new) nicht verwendet werden soll.

### 3.4 Weitere Tipps

Falls die Ursache nicht ermittelbar ist, empfiehlt es sich, den Transfer im Batch-Modus (siehe auch Abschnitt 5.2.2) zu wiederholen und dabei erweiterte Traces einzuschalten.

Falls das Webservice insgesamt mit einem Fehler (z.B. "outOfMemory", siehe Webservices Logfile) endete, gibt es nur zumeist nur die Möglichkeit, den Dienst neu zu starten:

- Servlet Engine neu starten (aber immer am festgelegten Arbeitsverzeichnis bzw. mit der vorgesehenen Prozedur),
- Prüfen, ob die AopWeb-Anwendung geht:  
`http://ihrServer:8080/aopweb`
- Prüfen, ob der SOAP-Dienst verfügbar:  
`http://ihrServer:8080/aopweb/admin`
- Die Datei tomcat.log ansehen, insbesondere das Ende.



## Kapitel 4

# Installation

TransferMonitor wird nur zusammen mit den Komponenten AopWeb und AOXS und Apache-Tomcat als zusammengestelltes und vorkonfiguriertes System geliefert. Betreffend aktuelle Hinweise, siehe Datei ReadMe in Unterverzeichnis doc des Laufzeitsystems.

Dort ist auch festgehalten, mit welchen Produktkomponenten (Releases) das Gesamtsystem aufgebaut ist.

Das Packet wird als komprimiertes TAR-Archiv (UNIX) bzw. als .zip-Datei (Windows) geliefert. Das Entpacken kann, z.B. unter HP-UX, mit

```
cd /application
gzip -d transfer-1_5.tar.gz
tar -xf transfer-1_5.tar
```

erfolgen. Dadurch wird ein Verzeichnis `./transfer` angelegt. In diesem Verzeichnis befindet sich eine Datei `DoFirst`, die diverse Systemvariable und Funktionen definiert.

Mit `cd transfer; . DoFirst` erfolgt der Einstieg ins Laufzeitsystem. Diese Anweisungen können auch in der `.profile` Datei des Anwenders (bzw. Administrators) automatisch ausgeführt werden.

Betreffend Zugriffsrechte usw: Anpassungen entsprechend AopWeb Handbuch durchführen. Scriptanpassungen entsprechend AOXS Handbuch.

Es sind zwei Webservices registriert: Automation und Transfer, die auch auf getrennten Rechnern laufen können. Nur der Host des Transferdienstes kann im Startdialog geändert werden, der Host des Automatisierungsdienstes ist nur durch Startparameter änderbar. Siehe Variable `SERVLETHOST`, definiert in der Datei `DoFirst`.

SOAP-Services werden mittels des SOAP-Servicemanagers installiert, z.B. aus dem DOS-Fenster mit

```
java org.apache.soap.server.ServiceManagerClient
http://galois:8080/aopwebs/servlet/rpcrouter
deploy dds/aoxs-transfer.xml
```

wobei der URL des RPC-Routers anzupassen ist und der classpath die div. Libraries (SOAP, XML) enthalten muß (dds ist das Verzeichnis, in dem bei mir lokal die Deskriptoren sind). Siehe auch Abschnitt 3.4.

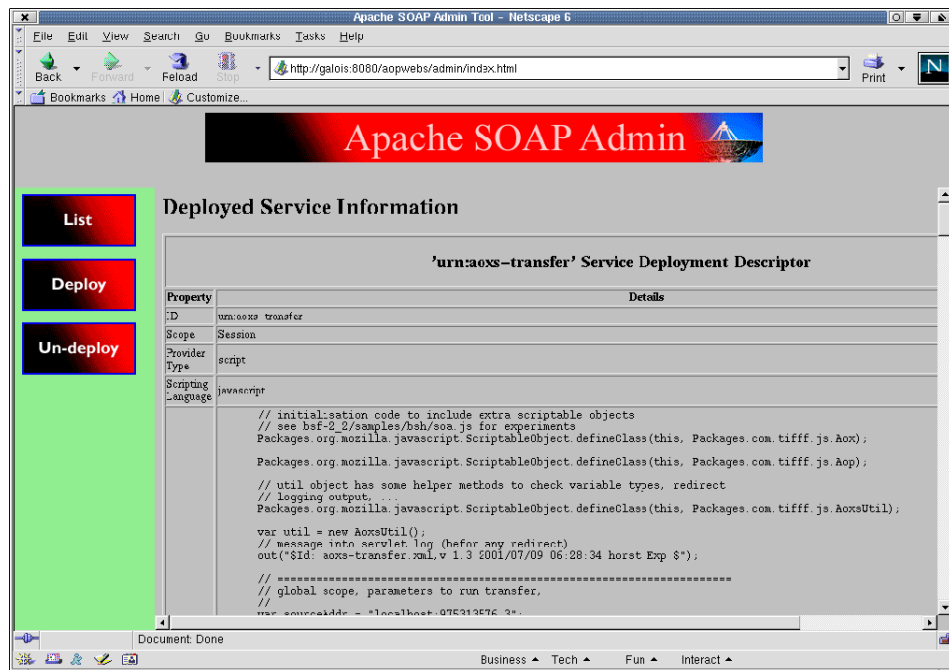


Abbildung 4.1: Servicemanager

Der Servicemanager sollte sich ca. so wie hier gezeigt präsentieren.

Falls der Transferdienst Zwischenergebnisse und Logfiles schreibt, ist sicherzustellen, daß die entsprechenden (im Skript genannten) Verzeichnisse am Server existieren.

Auf jedem PC, auf dem der Transfermonitor durchgeführt werden soll, sind folgende Komponenten (je nach Variante) nötig:

1. Die Anwendung läuft als Webservice am Server und wird via Transfermonitor vom Desktop des Anwenders aus bedient. Benötigt wird am Desktop Java Runtime 1.3+ und Java WebStart (Download von <http://www.sun.com/java>). Der Anwender startet seinen User Agent (z.B. IE) und startet den Transfermonitor auf der AopWeb - Registrierseite (<http://s10mva01:8080/aopweb>, siehe Abbildung 2.1).

Dieses Verfahren garantiert, daß auch bei Softwareupdates die jeweils gültige Version zum Einsatz kommt.

Falls JNLP nicht zur Verfügung steht, kann der Transfermonitor (Datei `aopweb/apps/transfermonitor.jar` auch lokal am Desktop (oder via Shares) zu installiert und gestartet werden (siehe Abschnitt 2.2).

2. Alles wird auf Server durchgeführt, nur GUI des Monitors am PC: X-Software, z.B. Reflection, ...(nicht Teil des Lieferumfangs). Achtung: Bei limitierter Bandbreite der Netzwerkverbindung kann es zu sehr langsamen Bildaufbau kommen.

# Kapitel 5

## Parametrierung

### 5.1 Automatisierungsdienst

Alle Konfigurationsparameter für die Automatisierung sind im Deploymentdescriptor (siehe Anhang B.2) des Automatisierungsdienstes hinterlegt.

- **transferstatus**  
gibt den Pfadnamen (relativ zum Arbeitsverzeichnis der Servletengine) der Statusdatei an,
- **transferservice**  
bezeichnet den zu aktivierenden Transferdienst. Falls der Dienstname mit **urn:** beginnt, wird der Dienst als **WebService** aktiviert, anderenfalls wird die **exec()** - Methode verwendet, um einen externen Prozess zu aktivieren. Dieser Name kann im Start-Dialog des Transfermonitors z.B. um testweise einen anderen Dienst zu verwenden, geändert werden.
- **sourcerpc**  
gibt den zu überwachenden Datenserver an. Diese Angabe wird auch den aktivierten Transferdiensten als **execute()**-Argument mitgegeben,
- **element, query und report-Optionen**  
erlauben, die Abfrage festzulegen, mit der eine Änderung am Quelldatenserver erkannt werden soll.

### 5.2 Transferdienst

Der Transferdienst ist als Skript implementiert und daher sehr weitreichend änderbar. Als Referenz zur Erstellung des Skripts und des damit zusammenhängenden Templates dient das AOXS-Handbuch ([1]) und die Applicationnote "TRR Datentransfer".

Diese Parametrierung kann durch den Anwender (er sollte Javascript lesen/schreiben können) erfolgen oder beauftragt werden. Auf jeden Fall sollten vor Beginn dieser Parametrierung die zu transferierenden Daten analysiert worden sein, die Datenbank des Zielservers modelliert sein und eine grobe Abbildung informell festgelegt sein. Falls der Quelldatenserver entsprechend AVL's

TRR Struktur gestaltet ist, werden i.A. Versuchsattribute (Name, Start, Ende, usw. ) aus den entsprechenden xy-Dialog Datensätzen stammen. Die Ausgestaltung kann dann inkrementell erfolgen. Je mehr Attribute einfach abgebildet werden können, um so einfacher gestaltet sich die Parametrierung, wenn viele Attribute erst errechnet und/oder zusammengesetzt werden müssen, und auch die Ergebnisse vorangegangener Transfers (anderer TRR-Dateien) mitberücksichtigt werden müssen, wird die Parametrierung dementsprechend komplexer.

### 5.2.1 Abbildungs-Template

Das "Template" stellt ist ein in XML (entsprechend AOXS-DTD) formuliertes Teil-Datenmodell des Zielsystems, das mittels eingebetteter XML-Processing Instructions die Quelle der Daten angibt. Es sind nur diejenigen Entitäten und Attribute enthalten, die im Zuge des Datentransfers "befüllt" werden sollen. Kreise im Datenmodell sollten vermieden werden. Details betreffend zulässiger Konstrukte innerhalb der Processing Instructions: Siehe AOXS-Handbuch ([1]).

### 5.2.2 Transfer-Skript

Das AOXS-Handbuch beschreibt auch die Erstellung von Skripten zur Behandlung von Datentransfers. Die dort behandelten Beispiele beziehen sich auf nicht automatisierten, d.h. in einem Console-Fenster (oder mittels User-Agent - Fenster) aktivierten Transfer im "Batch"-Modus.

Um ein erstelltes Skript "automatisiert" betreiben zu können, muß das Skript Zusatzanforderung erfüllen, d.h. es muß in einen SOAP-Deskriptor eingebettet werden und es muß die Methode `execute()` implementieren.

Da die Erstellung nicht trivial ist und i.A. das erstellte Skript getestet werden muß und/oder bei später auftretenden Problem eine Fehlersuche gestartet werden muß, hier ein paar Empfehlungen betreffend Erstellung eines Transfer-Skripts:

1. Mit einem einfachen, nicht automatisierbaren, Skript beginnen, z.B. Quellserver öffnen, Template füllen und als XML-Dokument ausgeben:

```
var aop = new Aop("myServer:975313579.3");
out("aop object for", rpcadr, "access created");

var aox = new Aox("templates/myTemplate.xml");
out("template load ok");

var ie = aop.getIEId("I1.1", "A2", "SM3309_22");
aox.process("p1", aop, ie);

aox.serialize("logs/myFirstResult.xml");
```

Alle nötigen Informationen dazu finden sich im AOXS-Handbuch.

2. Skript so umstrukturieren, daß es die Methode `execute()` intern verwendet, z.B. wird obiges Skript zu:

```

var template = "templates/myTemplate.xml";

// the work data
var aox;

// the ODS service
var aop;

// the execute() method tells test (by name) to be processed
function execute (sourceAddr, test, message, user, passwd) {
    aop = new Aop(sourceAddr);

    var ie = aops.getIEId("I1.1", "A2", test);
    aox.process("p1", aop, ie);

    aox.serialize("logs/myFirstResult.xml");

    return("No transfer yet, just testing data fetch");
}

// the main section becomes obsolete when above methods become called
// externally by automation service

// usually first argument passed when calling script from shell
var tnam = namedvalue("trrname");

var sourceAddr = namedvalue("sourceaddr");
var message = "";

execute(sourceAddr, tnam, message, namedvalue("user"), namedvalue("passwd"));

```

Dazu ein paar Hinweise:

- Alle Variablen und Funktionen, die im Template und im Controlskript übergreifend vorkommen, sind global zu halten.
- Als Ergebnis (Script return value) sollte die execute()-methode einen Text retournieren oder eine Exception anziehen. Falls ein Text retourniert wird, schreibt die Automatisierung diesen Text zur Anwenderinformation in das Feld "Versuch" der Statustabelle (gedacht ist hier an den Namen des im Zielservers erzeugten Versuchs o.Ä.) und setzt den Transferstatus OK. Falls eine Exception erzeugt wird, sollte diese einen Hinweis zur Fehlerursache enthalten (wird in der Spalte "Fehler" der Statustabelle hinterlegt). Der Status wird auf APPERROR ("transfer failed") gesetzt.
- out() - Anweisungen können gesammelt werden, dazu gibt es (siehe AOXS-Handbuch) eine Methode "outbuf", um am Ende die Geschichte des Fehlers nur im Fehlerfall auszugeben.
- Die Ausgabe von Zwischenergebnissen sollte durch eine Variable ein- bzw. ausgeschaltet werden können, um später den Normalbetrieb nicht durch "Debug"-Kode zu belasten und dennoch das Koding auch zum Debuggen verwenden zu können.

3. Skript testen und erweitern, bis es die Transferaufgabe erfüllt, und daran denken: "Automatisiert erstellter Garbage bleibt Garbage!". Das fertige Skript kann dann automatisiert im `exec()`-Modus betrieben werden. Falls des Script als Webservice betrieben werden soll, ist ein weiterer Schritt erforderlich:
4. Skript in einen SOAP-Deployment Deskriptor einbetten und mittels SOAP-Manager am Webserver als Dienst einrichten. Dazu wird das Skript mittels Texteditor um folgende Zeilen ergänzt:

```
<?xml version="1.0"?>
<!--
  Hier können Anmerkungen (betreffend Besonderheiten, Änderungen usw.
  als XML-Kommentar untergebracht werden
  ...
-->
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:aoxs-transfer">
  <isd:provider type="script"
                scope="Session"
                methods="execute">
    <isd:script language="javascript"><![CDATA[
// Skriptpräambel (Klassendefinitionen)

// Skript (unmodifiziert):

// Skript Ende
    ]]>
  </isd:script>
</isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

**Achtung::** Der WebDienst hat einen Namen (in obigen Beispiel "aoxs-transfer") als "unique resource name" (URN). Der Name ist frei wählbar und kann bei Starten der Automatisierung angegeben werden, es empfiehlt sich allerdings, diesen Namen auch als Dateinamen des Deployment Descriptors (im Beispiel wäre dies: `aoxs-transfer.xml` zu verwenden).

Damit das mit SOAP aktivierte Skript die speziellen Klassen für den ASAM-ODS - Zugriff aktiviert, sind die verwendeten Klassen als Präambel im Skript anzugeben, z.B.:

```
Packages.org.mozilla.javascript.ScriptableObject.defineClass(
  this, Packages.com.tiff.js.Aox
);
Packages.org.mozilla.javascript.ScriptableObject.defineClass(
  this, Packages.com.tiff.js.Aop
);
Packages.org.mozilla.javascript.ScriptableObject.defineClass(
  this, Packages.com.tiff.js.AoxsUtil
```

```
);  
var util = new AoxsUtil();
```

Um auch später einzelne Transfers testen zu können, sollten Batch-Skript und das im Deploymentdeskriptor eingebettete Skript Quellcode-identisch bleiben. Das Skript kann auch im “Batch”-Modus die `defineClass`-Präambel enthalten.

Tests im nicht automatisieren Modus bieten mehrere Vorteile: Einfacherer Testzyklus (kein “Deploy”) und damit kürzere Turnaround Zeiten, bessere Isolation und bessere Interpretierbarkeit von Exceptions, da diese nicht durch die SOAP-Schale abgefangen und als “SOAP-Exception” retourniert werden.

Ein Skriptbeispiel findet sich im Anhang B.1 (und im Softwarepaket).



# Anhang A

## Glossar

Achtung: Einige Ausdrücke aus dem technischen “Slang” sind nicht erläutert (“Resultset”, “OpenSource”, “Container”, ...).

**AopWeb** ist ein *Servlet*, das Anfragen nach ASAM-ODS-Daten in HTML - Output umgewandelt. Die Rolle, die die Komponente hier im Zusammenhang mit WeBservices spielt ist die eines Containers für die verwendeten WeBservices. Die Transferanwendung kann auch ohne AopWeb betrieben werden, ein anderer Container (z.B. der Original Apache SOAP-Container) kann die genannten Dienstimplementierungen beherbergen.

**Apache** Organisation ([www.apache.org](http://www.apache.org)) aus dem “OpenSource” - Bereich, bekannt durch die Bereitstellung des gleichnamigen WebServer.

**AOXS** ist ein Softwarepaket, das die Erstellung von Anwendungen, die ASAM-ODS und XML verbinden, erlaubt. AOXS-Anwendungen können sehr flexibel konfiguriert werden, z.B. auch als Gateway <sup>1</sup>, um z.B. das ASAM-ODS Applikationsmodell zur Laufzeit (beim Datenzugriff) zu ändern.

**HTTP** (HyperText Transfer Protocol) ist das vom W3C spezifizierte Protokoll, mit dem insbesondere im Internet Informationen (z.B. *HTML*-Seiten) übertragen werden. HTTPS ist die um Sicherheitsmerkmale (Encryption) erweiterte Variante.

**HTML** (HyperText Markup Language) ist als Seitenbeschreibungssprache zusammen mit *HTTP* die Basis des WWW (WorldWideWeb).

**ECMAScript** erlaubt Teile von Anwendungen in Skriptsprache handzuhaben und damit die Flexibilität (Konfigurierbarkeit) zu erhöhen, da das Verhalten mit einfachen, auch vom Kunden anpassbaren Skripts, beeinflusst werden kann. ECMAScript-Implementierungen sind auch, unabhängig davon, ob sie in C++ oder Java implementiert sind, unter der Bezeichnung *JavaScript* bekannt.

**Java** Von Sun entwickelte Programmiersprache, die besonders bei Enterprise-Anwendungen (Enterprise Beans) und teilweise eingebettet in HTML-Seiten (Applets) Verbreitung fand. Vorteil: Wesentlich höhere Portabi-

---

<sup>1</sup>von ASAM-ODS als Mapper bezeichnet

lität (“Write once, run anywhere”) als alle anderen aktuellen Programmiersprachen.

**JNLP** (Java Network Lauch Protocol) ist ein von Sun spezifiziertes Protokoll zur Verteilung und Aktivierung von Desktop-Komponenten.

**Servlet** ist eine innerhalb einer *Servlet Engine* ablaufende Software. Anfragen eines *User Agent* (GET) werden via Servlet Engine (z.B. *tomcat*) an eine spezifische Implementierung (z.B. *AopWeb*) weitergeleitet, die eine Antwort (i.A. eine HTML-Seite) dynamisch erzeugt.

**Servlet Engine** Eine Servlet-Engine erlaubt es, Web-Inhalte dynamisch zu erzeugen. Durch abfangen der HTTP-Requests und spezifischer, aus dem Abfragekontext sich ergebender Bedingungen wird der retournierte Inhalt dynamisch erzeugt. Das funktioniert auch mit CGI-Scripts, PHP o.Ä., Servletengines erlauben aber ein direkteres Binden an die Hintergrundobjekte, die über EJB, CORBA oder DCOM erreichbar sind. Im hier dargestellten Fall sind sie über AOP (ODS 3.2) oder über SOAP erreichbar.

**SOAP** Das Simple Object Adapter Protocol reduziert, wie im Namen ausgedrückt, den nötigen Infrastrukturaufwand, indem keine neuen Protokolle (RMI, IIOP DCOM usw.) nötig werden, sondern bereits vorhandene Internet Protokolle (*HTTP* oder *SMTP*) verwendet werden.

**Thin-Client** Anwendungen, die ihre gesamte Anwender-Interaktion über einen *User Agent* abwickeln können, z.B. Formular ausfüllen und “Transmit”, Antwort als HTML-Seite.

**Tomcat** ist die von *Apache* stammende Implementierung einer *ServletEngine*.

**User-Agent** Hinter diesem Term versteckt sich nichts anderes als der klassische Web-Browser (Explorer, Navigator, ...). Erweiterungen werden mittels Pluggins (z.B. als Applets) ermöglicht, die Gestaltungsmöglichkeiten bleiben aber dennoch oft eingeschränkt (bzw. führen zu langen Ladezeiten) und die Einbettung von leichtgewichtigen Anwendungen führt sehr oft zu Inkompatibilitäten, sodaß je nach User-Agent einiges spezifisches Koding erzeugt werden muß. Eine Abhilfe kann da *WebStart* schaffen. Als Einstiegspunkt in Webanwendungen bleibt der User-Agent allerdings unentbehrlich.

**WebStart** ist ein vom *User Agent* aktivierbare Anwendung zur Handhabung des *JNLP* und damit zur Administration von Web-Start-Anwendungen. Der Transfermonitor kann mittels WebStart am Desktop des Anwenders aktiviert werden.

**W3C** (WorldWideWeb Council) ist die Organisation, die die Internet-Standardisierung betreibt.

**XML** (eXtensible Markup Language) ist eine vom W3C spezifizierte Sprache mit denselben Wurzeln wie *HTML*, aber von Beginn an strenger definiert und eingeschränkt auf die Beschreibung strukturierter Datenmengen, d.h. die Präsentation wird nicht festgelegt. Daher auch die Verbreitung am Gebiet des “Datenaustausches” (EDI). Es gibt einige erweiternde Spezifikation, z.B. XML-Schema zur erweiterten (über den Dokumenttyp/DTD

hinausgehende) Beschreibung der Datenstrukturen, XSL/T zur Beschreibung von Transformationen usw.



# Anhang B

## Beispiele

### B.1 Transferdienst (als Webservice)

Der Transferdienst ist ein Webservice, in das das Transferkontrolskript eingebettet ist:

```
<?xml version="1.0"?>
<!-- This SOAP deploymentdescriptor offers a scriptable webservice at
      session level, e.g. to execute a transfer.
      There may be multiple different webservices defined using
      different URN's or the requestor may pass different templates
      when submitting request (adapt transfer method to include
      templateName into call arguments).

ATTENTION:
- Ensure namespace aware XML parser (otherwise no SOAP methods recognized)
- Special care when using relative filenames, e.g. for template, logging:
  Ensure servlet engine (tomcat) using appropriate working directory.
  See var workdir and absolute setting alternative.
- Dont redirect stderr (will effect all webservices), use outbuf instead.
- Keep source compatible to batch script version (if any)
-->

<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
            id="urn:aoxs-transfer">
  <isd:provider type="script"
                scope="Session"
                methods="execute">
    <isd:script language="javascript"><![CDATA[
// initialisation code to include extra scriptable objects
Packages.org.mozilla.javascript.ScriptableObject.defineClass(
  this, Packages.com.tiff.js.Aox
);
Packages.org.mozilla.javascript.ScriptableObject.defineClass(
  this, Packages.com.tiff.js.Aop
);
// util object has some helper methods to check variable types, redirect
// logging output, ...
Packages.org.mozilla.javascript.ScriptableObject.defineClass(
```

```

    this, Packages.com.tiff.js.AoxsUtil
  );
  // run thru constructor to perform registrations
  var util = new AoxsUtil();

  // =====
  // parameters to run transfer,
  // =====
  var destAddr = "localhost:975313574";

  // root directory (any file operations, e.g. logging, are relative)
  // Using users directory (e.g. where tomcat has been startet):
  var workdir = namedvalue("user.dir") + "/";
  // Absolute (example)
  //var workdir = "/home/horst/dev/applications/aoxs/test/";
  out (" Working directory", workdir);

  // template to be used (relative to workdir)
  var template = "templates/aoxs-transfer.xml";

  // Increased verbosity
  var debug = (namedvalue("debug") == "true");

  // =====
  // global variables (visible in template and script)
  // =====

  // the work data
  var aox;

  // the ODS services
  var aops;
  var aopt;

  // =====
  // Execute Transfer
  // @param source address is host:rpcaddress of data source service
  // @param test is name of test to be selected. See below about
  //   adaption required when test pathname is used by caller.
  // @param message holds message (return value of preceding execution
  //   on same TRR).
  // @param user and @param passwd may be used to authenticate with
  //   data destination service
  // =====
  function execute (sourceAddr, test, message, user, passwd) {
    // log to global logfile (= servlet log)
    java.lang.System.out.println("Transferring test " + test + " (message=" + message + ") " + " 1

    // make output buffering available
    outbuf("outbuffer");
    out("Transferring test",test,"from",sourceAddr,"to",destAddr,"using",template);
    out (" Last transfer message", message);
  }

```

```
// check resources
// open source, may raise exceptions
aops = new Aop(sourceAddr);

// open target using login info, may raise exception
aopt = new Aop(destAddr,"USER=" + user, "PASSWORD=" + passwd);

// RPC interface verbosity (RPC timing)
if (debug) {
    aops.setDebug(1);
    aopt.setDebug(1);
}

// using testname (without path) get source data local instance id:
var testId = aops.getIEId("I1.1", "A2", test);
// using pathname this might look like that:
//var testId = aops.toIEId(test);
out ("test " + test + " has ID " + testId);

// for testing one might set to true to avoid calls changing
// destination server database
readonly = false;

// read in template holding data model and PI's
aox = new Aox("file:" + workdir + template);

// add a user (return value used only for succeeding out)
// The default user is given in template, name is set explicit here
var uie = aox.createIE("A15");
aox.processIE("add", uie);
aox.setIEName(uie, "a.krawat");

// for testing, create another (at end we will join them)
// this will be the one getting the references
var uie2 = aox.createIE("A15");

// get test according template definition
out ("processing p_test instructions (filling down to test element level)");
try {
    aox.process("p_test", aops, testId);
    if (debug) {
        aox.serialize("file:" + workdir + "logs/" + test + "_p_test.xml");
    }
}

// initiale recorder numbering (might be fetched from target using relate
// after to level process() or using direct fetch), we use a constant
recNr = 0;
// find out which measurements from source shall be put into
// the output document.
// mds is an array of id's (children of testId)
var mds = aops.getIEIds(testId, "A3");
for (var i = 0; i < mds.length; i++) {
    // getting by id is mandatory due to some measurements not unique
```

```

// accessing name as available in source service (befor process(),
// referenced by process(), therefor global access
meaName = aops.getIENAME(mds[i]);
// meaId is set by .process(), therefor global
meaId = null;
// meaSum is (conditionally) created durind D data processing (local)
var meaSum = null;

var doStore = !readonly;
out(" processing source data id: " + mds[i] + " name: " + meaName);
if (meaName == "D") {
// ommit some columns (if included), limited amount (999)
  aox.process("p_mea", aops, mds[i], "P31L,P31R,P32L,P32R,PAN1,PAN2,T32L,T32R", 999);
// show documents measurement Id
out("D-data loaded, new meaId=", meaId);
// some additional filtering of those data
var messTyp = aox.getIEAttr(meaId, "MessTyp");
// accessing aox local name now (after process())
out("Messung " + aox.getIENAME(meaId) + " is of type " + messTyp);
if (messTyp == "SERIE") {
  // sample code only, not very meanaingfull
  out("filtering data of messtype " + messTyp);
  var meqIds = aox.getIEIds(meaId, 0);
  var expIEId = null;
  var hasPntCol = false;
  out("#Quantities: " + meqIds.length);
  // see which columns are there and check for actions
  for (var k = 0; k < meqIds.length; k++) {
    var mqName = aox.getIENAME(meqIds[k]);
    out(" ", k, mqName);
    // just for demo get and set some measured data values
    if (mqName == "ZEIT") {
      var vs = aox.getVS(meaId, 0, meqIds[k]);
      for (var l = 0; l < vs.length; l++)
out("vs",l,vs[l]);
      if (vs.length > 2) {
// replace a value
vs[2] = "99:00:77"; // an invalid date
aox.setVS(meaId, 0, meqIds[k], vs);
      }
    }
    // just to have a column for deleteIE test
    if (mqName == "BEFFW")
      expIEId = meqIds[k];
    // to filter out some rows
    if (mqName == "STS_STNR") {
      var vs = aox.getVS(meaId, 0, meqIds[k]);
      filterV = new Array();
      for (var l = 0; l < vs.length; l++)
if (vs[l] == "1")
      filterV[l] = 1;
    else
      filterV[l] = 0;
    }
  }
}
}

```

```

// PNTNR already there ?
if (mqName == "PNTNR")
    hasPntCol = true;
}
if (! hasPntCol) {
    // add a new implicit setvalue columns (Properties as String!)
    // Note: Using compiled mode (optimization >= 0) numbers do
    // not need to be quoted (array is String-array)
    // Dont use null (for empty) here:
    var attrs = new Array("0", "0", "0", "PNTNR", "", "6");
    // here null's are valid, note tstId (aox, not aops related)
    var refs = new Array( null, null, meaId, tstId, motId);
    var meqId = aox.createIE("A5", attrs, refs);

    // create the local column as an independend implicit columns
    // (Start = 1, Increment = 1)
    // using Strings, otherwise double is used represented later as "1.0"
    // when running in "interpreted mode"
    //var vnew = new Array("1", "1");
    // or (alternative):
    var vnew = new Array(new java.lang.Integer(1), new java.lang.Integer(1));
    // any value "on", "yes", "true", "1" "is..." means true, any
    // other means false.
    aox.setVS(meaId, 0, meqId, vnew, "isIndep", "isImpl");

    // test last=new column
    // call uses index (meqIds.length) instead of meaId to identify column
    var mcattrs = aox.getMCAttrs(meaId, 0, meqIds.length);
    if (mcattrs != null)
        out("MC attrs: " + mcattrs[0] + " " + mcattrs[1] + " " + mcattrs[2]);
}
// if filter column got used, create subset for summary measurement
// holding just first point of each cycle
if (filterV != null) {
    out("Filtering using", filterV);
    var msnew = aox.filterMS(meaId, 0, filterV);
    if (msnew >= 0) {
        if (debug) {
            aox.serialize("file:" + workdir + "logs/test_f.xml");
            out("data saved to file:" + workdir + "logs/test_f.xml (submatrix " + msnew + " created)");
        }
        // no need to check for existance, if exists, the store will map to one
        // see template for attribute list order
        var attrs = new Array("0", messTyp, "Summary", "", "Summary data");
        // use references from original test
        var refs = new Array(aox.getIEId(tstId, 0), aox.getIEId(tstId, 1));
        tstSumId = aox.createIE("A3", attrs, refs);
        // create a measurement and attach to tstSum (different test)
        meaSum = aox.createIE("A4", new Array(),
            new Array(tstSumId, aox.getIEId(tstSumId, 0), aox.getIEId(tstSumId, 1)
                ));
        // to fill lot of attributs aox.processIE(PIName, meaSum) is usefull
        aox.setIENAME(meaSum, "History");
        // and move submatrix to that new measurement

```

```

        aox.moveMS(meaid, msnew, meaSum);
        // and set to "append" mode
        aox.setMSHint(measum, 0, 0);
    } else {
        out("Filter result: none");
    }
}
}

    } else if (meaname.substring(0,1) == "!") {
// create a measurement with name derived
var RecNumber=parseInt(meaName.substring(1,2));
if ((RecNumber>0)&(RecNumber<=9)) {
    // global recordernumber, used in p_
    recNr++;
    aox.process("p_mea", aops, mds[i], "");
} else {
    doStore = false;
}

    } else {
doStore = false;
// others data (e.g. !E, ...) are already consumed by "p_mea"
out ("No threatment for " + meaname + " data");
    }

        if (doStore) {
// join duplicates (only fist time)
if (uie2 != null) {
    aox.joinIE(uie, uie2); // keep first inserted
    //aox.joinIE(uie2, uie); // keeps last inserted
    uie2 = null;
}

// remove duplicates
aox.unify();

// and store
out ("Checking for consistency befor stripping:");
out (aox.check());

aox.stripMI();
out ("Garbage collection done");

// some values may change depending on content found in target
aox.relate(aopt, "s");
out("Relating data to target ok, stack:", stack);

// conditions set using aox_c decide on update/ins
var x = aox.status(aopt, "c");
out("Data status:");
out(" error   :", x[0]);
out(" ignore  :", x[1]);
out(" continue:", x[2]);
out(" insert  :", x[3]);

```

```

out(" update :", x[4]);

if (x[0] == 0){
  if (aox.store(aopt))
    out("Data stored to target session");
  else {
    throw "Server failed to store data";
  }
} else {
  throw "No store attempt made due to " + x[0] + " errors";
}
out ("Transfer of", meaName,"ok");
  } else {
out ("Measurement", meaName," not stored");
  }

  // drop measurements, MEQ's and, if created, summary data
  while (meaId != null) {
while (aox.getMSCnt(meaId) > 0)
  aox.deleteMS(meaId, 0);
aox.stripMI();
aox.deleteIE(meaId);
if (meaSum != null) {
  if (meaId == meaSum) {
    // after 2'nd pass
    aox.deleteIE(tstSumId);
    tstSumId = null;
    meaSum = null;
  }
}
meaId = meaSum;
  }
} // over all source data records (measurements)

// get data but dont setup new buffer and write to file
// sometime we might add a "getLog" method
var logfile = workdir + "logs/" + test + "_ok.log";
output(logfile, outbuf());

} catch (process) {
  // capture exceptions
  out("failed to process " + test + ": " + process);
  var logfile = workdir + "logs/" + test;
  output(logfile + "_err.log", outbuf());
  aox.serialize("file:" + logfile + ".xml");
  out(" data saved to", logfile + ".xml");
  out(" traces saved to", logfile + "_err.log");
  throw process;
}

// provide result name (as list, first entry is motid)
var ret="unknown";
moties = aox.getIEIds("I1.1", 0);
for (i=0; i < moties.length; i++) {

```

```

    ret = aox.getIEName(moties[i]);
    tsties = aox.getIEIds(moties[i], 0);
    for (j = 0; j < tsties.length; j++) {
        ret = ret + " " + aox.getIEName(tsties[j]);
    }
}
return ret;
}

//=====
// functions called from scriptlets (embedded in XML-PI's)
//=====

// The number is taken from "related" instance in target server.
// incrTestSeq() shall be after NAME attribute to have a proper IE-id
// for the to be incremented attribute within stack array.
// Access to var seqNr is usefull for attributes in same element but
// in sequence behind the incrementing one
// Attention: Needs stack, aox, aopt and seqNr to be global
var seqNr;
function incrTestSeq() {
    out("incrTestSeq()");
    // stack is defined in template
    if (typeof(stack[2]) != "undefined") {
        //if (typeof(stack[2]) == "string" && stack[2].charAt(0) == 'I') {
        seqNr = Number(aopt.getIEValue(stack[2], 'VersuchsName')) + 1;
        } else {
            seqNr = 1;
            out(" Test sequence reset, stack:", stack);
        }
    }
    // as "relate" is not using functions we have to set it explicitly in body
    aox.setIEValue(seqNr);
    out('Test sequence number:', seqNr);
}

]]>
</isd:script>
</isd:provider>
<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>

```

## B.2 Automatisierungsdienst

Da die Implementierung des Dienstes, unterschiedlich zum Transferdienst, in Java und nicht als ECMAScript vorliegt, sind nur einzelne Parameter (Options) im Deploymentdescriptor änderbar.

```
<?xml version="1.0"?>
<!-- Automatization service for transfer scheduling -->
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
            id="urn:aoxs-automation-soap">
  <!-- Using this provider to allow argument substitutions -->
  <isd:provider type="com.tiff.aopweb.AopWebProvider"
              scope="Application"
              methods="init start stop term getServiceInfo setIntervall
setStatus getStatusEntries">
    <isd:java class="com.tiff.service.aoxs.Automation" static="false"/>

    <!-- options, see also transfermonitor launch arguments -->
    <!-- the external statustable name -->
    <!-- multiple automations may use same but not concurrently -->
    <isd:option key="transferstatus"      value="status/aoxs-automation.txt" />

    <!-- transferrouter: only if automation and transfer use different host -->

    <!-- the transfer service (exec or soapservice if starting with urn:) -->
    <isd:option key="transferservice"    value="urn:aoxs-transfer" />

    <!-- sourcerpc is address of source data service -->
    <isd:option key="sourcerpc"         value="localhost:975313576" />

    <!-- element specifies by name the application element to be queried -->
    <isd:option key="element"          value="Versuche" />

    <!-- the query text, the %-var is runtime replaced by highest date -->
    <!-- substitution is optional, one may force fill with old by manually
         setting to ancient value and restarting in manual mode -->
    <!-- if omitted, no source queries are made -->
    <isd:option key="query"             value="Filedate>=%filedate%" />

    <!-- the attributes to be reported from source data service on query -->
    <isd:option key="report0"           value="TestId" />
    <isd:option key="report1"           value="TestName" />
    <isd:option key="report2"           value="Filedate" />
  </isd:provider>

  <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>

  <!-- no mappings yet -->
</isd:service>
```