

AOXS

Application Gateway for ASAM-ODS 3.2 Services

Document Type	Author	Revision	Date
Project documentation	Horst Fiedler	0.17	October 2001

Abstract

Enterprises running multiple ASAM-ODS services are looking for solutions where application structures including namings are automatically mapped in order to get a “confederated” data view. The requested application shall process client data request the same way as a real ASAM-ODS server (e.g. AVL’s Santorin) but shall provide a different view. The component is called “application gateway” by it’s nature. Physical protocol (ODS RPC based protocol) is not changed.

Revision History

Currently not provided at this place, see **ReadMe** files provided as part of distributed aoxs packages.

Caveats

This paper currently acts as a project documentation.

As a project documentation this paper may hold outdated informations (e.g. texts included in early concepts at project setup time), notes about design decisions made, ...

Starting with approx. version 0.5, usage related parts of this document will be merged into AOXS core manual.

Currently, the AOXS base documentation is available as a separate document. Readers not familiar with AOXS shall take notice of AOXS documentation as AOXS-GW is an extension to AOXS implementing just one additional object: **Aos**, an ASAM-ODS RPC service implementation.

About author

Mr. H. Fiedler may be reached using EMail horst.fiedler@tiff.com. or using URL <http://www.tiff.com/>.

Contents

1	Development Notes	1
1.1	Requirements	1
1.2	Base Concept	2
1.2.1	System components and protocols	2
1.2.2	Basic operation	3
1.2.3	Setup and initialization	3
1.2.4	Content translation	4
1.2.5	Rescaling	5
1.3	Implementation	5
1.3.1	Kernel	6
1.3.2	Restrictions	8
2	User Manual	9
2.1	Software installation	9
2.2	Building Map definition	9
2.2.1	Renaming attributes	10
2.2.2	Altering structure	10
2.2.3	Constant content	10
2.2.4	Dictionaries	11
2.2.5	Using alternate ID values	12
2.2.6	Virtual attributes	12
2.3	Service setup	13
2.3.1	Server Script	13
2.3.2	Activating Control	14
2.4	Content translation	15
2.4.1	Example	15
2.5	Common mistakes	18
3	Extensions	21
3.1	Performance features	21
3.1.1	Caching	21
3.1.2	Additional indices	21
3.1.3	Dynamic reloads	21
3.2	Additional interfaces (beyond ONCRPC)	21
3.2.1	Running as servlet	22
3.2.2	Firewall handling	22
3.2.3	Registry handling	23

Chapter 1

Development Notes

This chapter may include outdated informations. The only reason to keep them: To be able to look back for initial requirements discussed with customer.

1.1 Requirements

The primary task is to make an national language translation of any language dependent metadata (e.g. attribute names, element names) and other static data (e.g. unit names, dimension names). Additional requirements are:

- Operational even with not yet released interface definition version 3.2 which will include Security features, Inheritance, extended Query.
- Structural differences may include attributes changing to different elements, even number of elements may differ,
- Able to handle rescaling of both, attributes using fixed attribute information ¹ and measured data values using proper units within get/put - requests.
- Able to reload mapping information during runtime (option)
- Detailed information on map validity during startup and map change.
- Map information shall be parametrized using XML.
- An extra monitor interface (beside parametrization interface), e.g. using servlet based approach with map parameter upload (http put) and monitoring by http get requests would be accepted (option).

No special actions (name translation) are required for instance attributes, just units shall be converted as above.

Protocol layer version transformation (e.g. to access a version 2 service using version 3 requests) is not planned.

¹currently not returned by AOSJ, Santorin ?

1.2 Base Concept

The component intercepts messages from client to server by acting like a real ASAM-ODS server and translates the request based on internally managed informations (setup) before requesting “equivalent” information from another pre-set ASAM-ODS server. Same way result messages are transformed back.

As this happens on ASAM-ODS protocol layer (Version3.x) this interceptor is called a mapper. But there are other keywords in use within this paper depending on aspects: Intermediate-, transparent-, gateway- or proxy-server. The backend server is sometime called “real” server. The product name is AOXS-AOS as it is an optional addon to AOXS, e.g. a scriptable object implementation to handle ASAM-ODS service.

1.2.1 System components and protocols

Following types of active components may appear. Each of them may run on different host and talk using predefined protocols to each other.

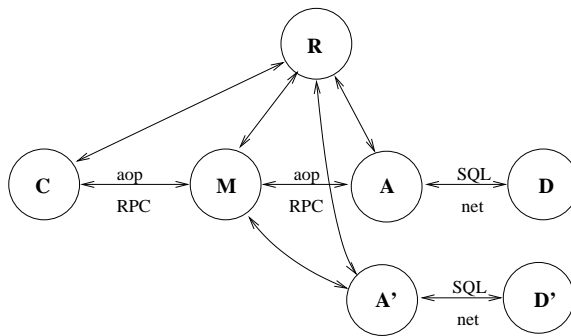


Figure 1.1: Protocols

The processing nodes in figure 1.1 are:

- C: Any client ODS client application,
- R: An AVL “Registry” where (beside portmapper) available ODS services are registered,
- A: Any ASAM-ODS 3.x service (e.g. AVL Santorin),
- D: A database behind the services (e.g. Oracle), and
- M: An ASAM-ODS 3.x service (gateway) mapping names and structures, e.g. providing different view compared to original service A.

Not included in that overview are infrastructure components like nameservice, httpservice (e.g. to get DTD’s, Map definitions using URI’s), dictionary services (LDAP), ...

The protocols occurring in that figure are

- AOP (ASAM-ODS protocol version 3) over ONC-RPC,
- SQL/SQLnet over JDBC or database server specific protocols and
- HTTP/HTTPS/LDAP when connections thru firewalls apply (see extensions section).

There may be an other implementation component supplied by AVL: A registry server where ASM-ODS servers “comming up” shall register to be able to retrieve RPC addresses in client programs. As this is a nonstandard feature it will be threated in a later phase. As the instantiation of AOXS-Objects is easy (scriptable) there shall be no problem to setup a facility allowing clients to “know” the relevant service (using LDAP would be the standard way). The same applies to backend side: AOXS gets the address to be used for backend connection befor beeing activated (e.g. in script) and currently does not query a registry for “known services”.

Open issues:

- Security: There is only one map held within AOS describing visible ASAM-ODS objects and there relationship to objects in backend. Therefor the class informations (ASAM-ODS elements and attributes available) has to be readable. If there are security parameters (user, password, project, ...) to access that information in backend service, they have to be available a startup time. Session related security arguments are simply passed to backend service as one client session maps to one “primary” backend session. If data from different servers shall be combined, no individual sessions are used to “secondary” backend services.
- Are there network connection where a protocol able to cross firewalls or using encryption shall be used?
There is a strong need for information infrastructure supporting SSL and (tunneling) proxies.

1.2.2 Basic operation

After startup and initialization (see below), the service **M** waits for requests by clients **C**. Any request to create an ASAM-ODS session (`aop_openenv`) is threated as within any real (nontransparent) server. But instead using a database connectionm, a session to a backend ASAM-ODS server is created and, if successfull, is registered within transparent server **M**.

Any `get/put` operation on an open session is analyzed about included names, units, ... (items to be transformed) and one or more requests using request parameters created by transformation are made to the backend server. The replie(s) are accumulated and returned.

The “static” elements (units, quantities, physical dimensions) are always returned out of a separate ASAM-ODS service session assuming “global ID uniqueness”, e.g. its only necessary to divert the request to the appropriate server.

1.2.3 Setup and initialization

Assuming the global ID space for dictionary entites, following inforamtion has to be passed to map service on startup:

- Location of backend service **A** for normal data elements (tests, measurements, measured quantities, submatrices, ...),
- Location of backend service **A'** for dictionary data elements (readonly),
- Map definition for attributes. This shall be the URI identifying a XML file which holds attribute associations. For format (DTD) see below.
- Location of registry (preferable LDAP accessible) to register availability of this ASAM (map) service.

When no mapping information is passed, the service creates a default mapping, where all attributes/elements ... map to identity.

Elements and normal attributes, for which transformation information is marked “hidden” are regarded to be “not existing” (this might be extended to thread any measured quantity which refers to a quantity without match within client quantities as invisible). This way it is possible to expose only subsets.

Same happens on store request. Attributes not found in transformation are passed thru unmodified (except unit information) to allow usage of instance attributes. Due to instance attributes, any known attribute shall appear in map information to be able to distinguish.

At end of initialization process, the map server registers (optionally, startup property driven) with registry (provided by other supplier).

1.2.4 Content translation

Units, quantities and dimensions

For some informations it is not sufficient to transform metainformation (like attribute names), but content needs translations too. E.g. the NAME and DESCRIPTION fields of dictionary data (Units. quantities) need to be made available in different language. Following variants to retrieve the appropriate values may occur:

- Both (two sets of national language specific instances) are on backend-server, e.g. english units/dimensions/quantities intermixed with german ones. This makes “ID translation” necessary, as a german “Leistung” will have an ID different to english “Power”: Wherever ID values for unit/quantities, ... occur in requests they become changed before request to backend data server is made, and are changed back in reply. This requires administration of ID translation tables within mapper.
- the sets are on same backend database tables accessed by two ASAM servers using different ASAM-ODS basename to database column mapping or on two different servers but same ID value refers to instances having same “meaning” (tables extracted from “global” ones). In that (preferred) situation (global ID space) no translation is necessary but access has to be diverted to the server **A'** holding the appropriate “language” (usually not the backend server **A** as that one would show content in “wrong” language).
- they are on a second source ODS server, which reflects the database “seen by client”. In that case translation AND diverting of request is necessary,

- and/or they are to be maintained locally within transparent server. In that case the mapper maintains the dictionary entities which may be good for performance but creates additional (to be maintained) database tables and makes usage of direct database access mandatory to perform mapping,
- and/or one may add “dictionaries” able to translate attribute values in both directions.

And (no matter from where they are fetched, but far away access makes this more important): Is it allowed/suggested to hold referenced instances within transparent servers cache to speed up semiconstant dictionary data access.

The initial implementations behave as follows: The map definition may hold optional instances. Requests fetching instances get data merged from original source with map defined source. There is no restriction that it is just name or description. Any attribute can become overwritten by map supplied data. The instances can be easily set up by just exporting data from appropriate ASAM-ODS server.

Later revisions may fill that instance container using requests diverted to “secondary” servers. In that case the container may get caching functionality (on write a “write thru” shall occur thus avoiding unnecessary restarts when dictionary entity contents changes).

In any case: A “global ID” mechanism is assumed although the initial implementation allows to manually overwrite ID’s if that precondition is not satisfied².

Be aware: Associating base name different to assignment made in backend is always available, e.g. mapping the NAME attribute to application attribute “enginename” when backend has “Motorbezeichnung” mapped to NAME. But in that case “enginename” has to be a column of same table on backend as “Motorbezeichnung”.

1.2.5 Rescaling

Any rescaling (from stored/default unit to requested unit) is done by backend server.

If requests are made using 0-Unit (“default unit”) the mapper may be extended to explicitly add the default unit to the request using referenced measured quantity instance name. The later operation makes a lookup for default unit in backend server, if found, that unit is lookup in default unit translation table, if found, the values are requested in that unit.

1.3 Implementation

The implementation is a two sided interface implementation. The passive (frontend) side acts as ASAM-ODS server waiting for client requests, the backend side accesses an ASAM-ODS (might be real or again transparent) service. The implementation is multithreaded, e.g. processing of one request shall not depend on completion of another one (but quite sure overall behaviour depends on behaviour of backend server too).

²this way one may find in practice which way to proceed with that topic avoiding dead end implementation

Any session creation request creates a new mapping session. It is allowed to pass additional properties within create session - request, e.g. to establish one of potentially multiple different mappings. Those extra properties are filtered out before the request is forwarded to backend to create the data session.

If sessions are “readonly” (no transactions used) the server might pool backend connections, e.g. all frontend sessions are using same backend session (does not conflict with multithreading of requests).

Any data request is expanded (unmarshalled), translations are performed, and reserialized request is forwarded to backend.

On startup, the transforming service tries to detect inconsistencies, e.g. units belonging to one dimension must match the equivalent dimension after translation.

1.3.1 Kernel

To implement the kernel there have been three design alternatives:

1. Similar too or adaption of AOSJ to just include an XML/SAX Parser firing events on Elements and Attributes to parse an XML file instead using SVCxxx tables/files.
2. Adaption of AOSJ, an ODS 3.1 service implementation to get another “backend” implementation. Beside FF (flat file) and DB (SQL database) a proxy implementation might be added.
3. Adaption of AOXS, an ODS 3.1 client implementation to handle XML data, gets an extension (a service stub) to be able to act as an RPC server.

Implementation of variant (1), mentioned within original concept (see section ??), was stopped quite soon after evaluation of some aspects in favor of variant (2), where the complete backend connection part is reimplemented: An XML equivalent of SVCATTR, SVCREF, ... within MetaInfGW is used and shall allow implementations referring to other ODS services. AppElemGW, MetaInfGW, AttrGW, RefGW, ... (subclasses of RlxXXX and implementing the respective interfaces) have been added.

The benefit of that approach is: It allows to keep existing backend implementation without any change, e.g. allows to split implementation of one ODS service across heterogenous physical storage implementations. As first detail designs and partial implementation of variant 2 showed, that a total replacement of AOSJ kernel would allow to create an XML centric service, third variant has been chosen.

Variant (3) is derived from AOXS and as AOXS combines ECMAScript, XML and ASAM-ODS, the new server is implemented as a scriptable object (e.g. one may create ODS service using ECMAScript). Any benefits from variant 2 can be made available although most of the implementations allowing local persistence need some efforts to be reimplemented. The metadata used by actual service implementations (SVCENT, SVCATTR, SVCREF) are replaced by AOXS kind of XML document. The mapping procedures, which are application (gateway functionality) specific and not ASAM-ODS specific are again saved in XML processing instructions.

The first delivery will include a service implementation able to handle most of the ODS 3.1 requests with 1:1 (just different naming) to backend services. The next step will be to use the global ID scope and to offer access to different implementations.

Design change (compared to initial offer): The mapping information is held within XML document but instead of being included in XML elements (MAP), the information is held in attributes of elements identified by common ID and in XML processing instructions. This change is due to experiences made with AOXS project itself where usage of XML methods (e.g. XSL/T) have been changed to scripting by performance reasons and flexibility. It was found out, that scripting facilities are mandatory when “mapping” structural informations. Using XML PI allows to use validating parsers without the need to add non ASAM-ODS elements (MAP) to AOXS DTD.

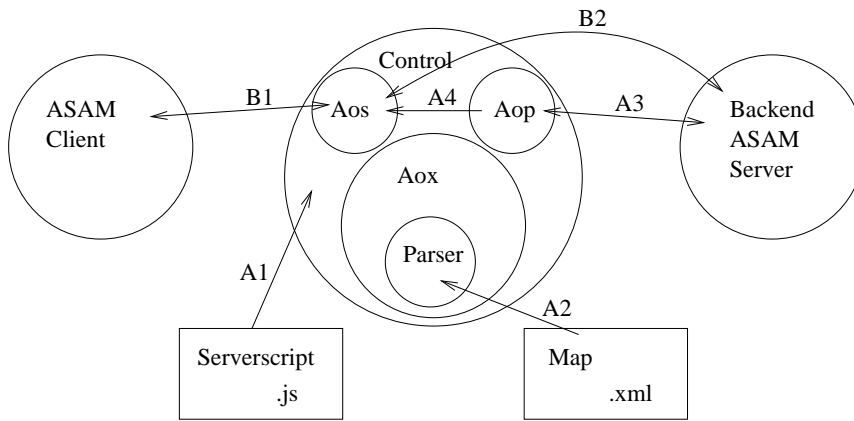


Figure 1.2: AOXS-AOS

A Service startup phase

1. The Control (main) is created and reads a script which tells which objects shall be created,
2. the instantiated Aox object loads using XML parser the map definition,
3. the instantiated Aop object checks availability of given backend service,
4. the instantiated Aos object takes backend address from Aop and uses Aox for mapping to offer RPC service.

B Client connection

1. a client connects to Aos which in turn
2. connects to backend to establish gateway.

Practically two AOXS schemes are used: The one read in at startup holding the schema to be presented to clients and the schema the backend server uses which can be fetched e.g. using AopWeb. AOXS-AOS associates the things within those schemes by using the XML-ID attribute which shall be in common for all common entities and attributes. As (see below) defining a map starts by editing the backend scheme, this rule should be easy to be followed.

Only Aox implementation is new (it's a "huge" object), Aox and Aop are reused (and slightly adapted in a way not conflicting with usage in different project) from AOXS base.

1.3.2 Restrictions

Changes to data model of backend servers and/or changes in transformation definition shall be followed by a map server restart. Optionally a re-init (keeping sessions but rereading and verifying map definition and cleaning up cached informations) can be made available but this may be usefull only if clients are stateless.

Chapter 2

User Manual

Shall become core part of document when design notes and other reminders will become removed from final document.

2.1 Software installation

Unpack as indicated in ReadMe and basic documentation of AOXS (included as ManualG.pdf in doc directory as it is currently german only). It is strongly suggested to have base AOXS package running before attempting to setup mapper¹.

2.2 Building Map definition

To set up an ASAM-ODS gateway following steps have to be made:

1. Get an AOXS schema describing backend server. E.g. AopWeb schema download may be used or a scripted fetch using AOXS may be made. See AOXS main document about how to setup scripts. A sample script (to be executed by Control script interpreter) is available in `test/scripts/getschema.js` of distribution package. This document will be the starting point for any mappings. Without applying any changes, a 1:1 mapping is defined.
2. Create a view by consistently removing elements and attributes from schema (projection),
3. Replace names by translated names (ID's shall not be changed preserved) on remaining Ax elements,
4. Add new attributes and entities using nonoverlapping XML-ID's and put construction rules into "Map"-processing instructions.

The general format of "map" XML-processing instruction is.

```
<?map id script?>
```

¹If AopWeb is available one may perform mandatory scheme and data fetches using AopWeb

where “id” is the ID of backend element. That element has to exist. If id is omitted, an identity map is assumed (same ID used on frontend as at backend side).

“script” is either an ECMA-Script function body (enclosed with braces) or a dictionary name referring to a dictionary created in startup script. If script is omitted, no transformations occur. If only the script is provided the attribute is “pure” virtual (recommended). Currently scripts are **NOT** allowed for AE (Entity) elements. See section 2.2.6 regarding use of scripting.

If both arguments are provided, they shall be separated by at least one space character. Can be used for “semivirtual”, e.g. writethru to update database. On read the script provides the value, on write the value is physically written.

All variables used within script are either local (preceded by keyword “var”) or global. The global scope includes the tol level script used to setup the gateway (e.g. `scripts/server.js`). Be aware that when accessing a server concurrently by multiple users, the use of global variables (except when used readonly) is discouraged.

WARNING: parsing of mapping instruction is “lowlevel”. Avoid leading and trailing whitespace characters and separate script (if any) by single space from id. The script itself (everything including the outer braces) can be written using any style including line and formfeeds.

2.2.1 Renaming attributes

Assuming the original scheme includes an attribute

```
<AA ID="A4.12" NAME="Auspuff" DTYP="STRING"/>
```

one may create a different representation by replacing the NAME, e.g.

```
<AA ID="A4.12" NAME="Exhaust" DTYP="STRING"/>
```

Same way element names and reference names may be adapted.

2.2.2 Altering structure

Only removal of relationships is possible. Always ensure, that if e.g.

```
<AR ID="R4.1" NAME="UserSchutz(UsersID)" REF="S16.6"/>
```

is removed, the associated

```
<AS ID="S16.6" NAME="Messung(UsersID)" REF="R4.1"/>
```

is removed too. All relationships are represented by pairs (either AR-AS or AS-AS for n:m).

2.2.3 Constant content

The need for replacing content usually occurs for rather constant instances, e.g. names of units or quantities. One has to add the instance to the model and keep the to replaced attribute value nonempty, e.g. appending unit instances (partially filled) to unit definition

```

<AE ID="A8" NAME="Einheit" TYP="DUN">
  <AA ID="A8.0" NAME="EinhID" TYP="ID" DTYP="INT"/>
  <AA ID="A8.1" NAME="EinhName" TYP="NAME" DTYP="STRING"/>
  <AA ID="A8.2" NAME="EinhFaktor" TYP="GAIN" DTYP="DOUBLE"/>
  <AA ID="A8.3" NAME="EinhOffset" TYP="OFFSET" DTYP="DOUBLE"/>
  <AA ID="A8.4" NAME="EinhInfo" TYP="DESCRIPTION" DTYP="STRING"/>
  <AS ID="S8.16" NAME="Einheitgruppen(BASE)" REF="S10.17"/>
  <AS ID="S8.12" NAME="Quantities(EinhID)" TYP="DUNID" REF="R7.0"/>
  <AS ID="S8.11" NAME="Messgroesse(EinheitsID)" TYP="DUNID" REF="R5.2"/>
  <AR ID="R8.0" NAME="Phys. Dimension(PhysDimID)" TYP="DIMID" REF="S9.15"/>
  <IE ID="I8.231"><IV/><IV>deg kw</IV></IE>
  <IE ID="I8.264"><IV/><IV>g/kwh</IV><IV></IV><IV></IV>
    <IV>specific consumption</IV></IE>
  <IE ID="I8.266"><IV/><IV>deg F</IV><IV></IV>
    <IV>Temperatur in Fahrenheit</IV></IE>
  <IE ID="I8.267"><IV/><IV>deg K</IV><IV></IV>
    <IV>Temperatur in Kelvin</IV></IE>
</AE>

```

will replace (only) the given values if included in a reply. Trailing attributes may be omitted. The IV elements have to be exactly in same order within IE element as AA elements (attributes) appear within AE elements.

The gatewayserver always adds ID to the request list before fetching from backend server to be able to perform that substitution².

Attention: Setup for content translation (additional to metadata map) is done after initial service setup as one shall first get the basic transformation running. See section 2.4 for additional hints.

2.2.4 Dictionaries

For some attributes translating to a constant is not sufficient. To use dictionaries, one may read in (at startup time) one or more named dictionaries. If instead of script a dictionary is supplied, the content is transformed both on read and on update/insert.

Following restrictions apply:

- The dictionary has to be bidirectionally unique (to make the operation reversible),
- Values which cannot be translated, stay unchanged. Therefore uniqueness shall apply to whole set of words (of both sides).

To create dictionaries: Within startup script, add the proper equivalent pairs of verbs to a named (1'st argument) dictionary e.g.

```

dd("e2d", "Test", "Versuch");
dd("e2d", "Endurance", "Dauerlauf");
dd("e2d", "Mike", "Michael");

```

where lefthand side (2'nd argument) is the clients/frontend used verb and righthand side (3'rd argument) is verb as used at backend side.

To use a dictionary just put the name of the dictionary into the attribute map (instead of script), e.g.

²forcing attributes to empty is not possible this way

```
<AA ID="A2.10" NAME="MOT2" DTYP="STRING"><?map e2d?></AA>
```

Verbs, which are not recognized, are routed thru unchanged.

2.2.5 Using alternate ID values

If element or attribute ID's shall be exchanged (mapped) within gateway, XML-PI (processing instructions) named “map” can be used, e.g.

```
<AA ID=A12.5 NAME= ...><?map A27.4?></AA>
```

associates attribute A12.5 from gateways model to A27.4 in backend model.

Attention: No automatic adaption of lower levels, e.g. if an element map (on AE element) is NOT mapping to identity, all attributes shall have a map PI attached. The datatypes shall match as up till now no datatype conversion is implementet. The names usually will differ.

2.2.6 Virtual attributes

Attributes may be “created” within GW server using scripts. If the content of a map-PI attached to an attribute does not match a backend attribute ID, an attempt is made to interpreted the content as an ECMA-script which shall be used to “calculate” the attribute value when reading (fetching) data. Any methods available within ECMA-script specification and all all methods found in aop object (see AOXS base documentation) can be used for that purpose. At the end a **return** statement shall provide the value to be returned to client.

Practically the script defined is the body of a script function. The header (not! to be specified in map-PI) is automatically created, e.g. assuming, the script appears on attribute with ID A19.91, the complete function compiled on startup when processing the map definition XML-file is prefixed with

```
function getA19_91(snr, aid, iid)
```

When the evaluation is called due to user fetching instances holding the virtual attribute, the compiled function is called using the 3 integer arguments

- **snr** which is the sequence number (result set enumeration) of actual query (numbering starts with 1). This information is usefull e.g. to set/reset accumulators (for summing up something).
- **aid** and
- **iid** as current instance ID. This information usually is required to fetch “related” things.

The “hidden” value of “semivirtual” attributes (e.g. when the attribute maps to an existing backend attribute) is not yet made available as argument.

Be aware: “scripted” attributes usually shall not be written into backend on update/insert operations as the process to calculate them usually cannot be reversed. Therefor one shall create attribute ID's NOT matching an ID existing within model of backend server. Update/Insert operations for “virtual” attributes mapped to an existing attribute will be executed as if the attribute would be a real one, although the client cannot verify the update as on next

fetch the attributes value is again calculated. To make the change “visible”, the client could access the real server or define another attribute which is not scripted and which maps to the same backend attribute as the scripted implicitly did on write.

Scripts can be used to “pull” lower level up (e.g. creating a mean value) or to concatenate multiple string value attributes into one attribute.

2.3 Service setup

In the most simple way one need 3 informations:

1. Which backend server to use (usually the one from which the schema originated), e.g. host anme where backend is installed and RPC program number under which it is registered,
2. the address the new server (gateway) shall get, and
3. the schema (map) created before

Then one may startup the gateway implementation, e.g. assuming control.bat is available on current working directory:

```
control scripts/server.js "MODEL=aopmap.xml"
```

On UNIX (where control is a shell script) the quotes around name value pairs can be omitted.

2.3.1 Server Script

An example for scripts/server.js (operating system independent):

```
// Logfile
redirect("server.log");

// build a service (datamodel passed as call argument)
var input = namedvalue("MODEL");
var aox = new Aox(sch);
out("Model\" + input + "\" loaded");

// client (backend) RPC (host:prognr.version)
var aop = new Aop("saturn:975313576.3");
out("Aop object for backend server access created");

// service rpc (only prognr, host=localhost, version=3)
var aos = new Aos(aox, aop, "975313579", 180000);
out("Aos object to serve client requests created");

// periodically check backend availability (instead aos.run());
// the delay shall be less than timeout value (keep alive).
var testvar = "";
while (testvar != null) {
  // dont use getIENName("I1.1") as those are cached in session
  testvar = aop.getIEValue("I1.1", aop.getAAName("A1", "DESCRIPTION"));
  out(testvar);
}
```

```

    sleep(100000);
}

out("Exiting");
quit();

```

2.3.2 Activating Control

An example Windows .bat file (control.bat) activating the Java VM and processing the Control object which in turn interprets the script above:

```

@echo off
rem Execute control.
rem As = - characters are "special" in .bat, name value pair's shall be quoted

rem script is minimum
if A%1 == A goto err1

rem Where the application is found (here on parent directory)
set TFHOME=..

rem Working directory
rem where tranfer shall execute and where relativ files are located
cd %TFHOME%\test

rem setup Java (explicit to avoid conflicts)
set JAVAHOME=c:\opt\jdk1.3

rem setup classpath
set LIBS=%TFHOME%\lib
set CP=%LIBS%\control.jar
set CP=%CP%;%LIBS%\aop.jar
set CP=%CP%;%LIBS%\xerces.jar
set CP=%CP%;%LIBS%\xalan.jar
set CP=%CP%;%LIBS%\bsf.jar
set CP=%CP%;%LIBS%\js.jar
@echo on
%JAVAHOME%\bin\java -Xmx40m -cp %CP% -Drpc.authstyle=UNIX -Drpc.uid=777 -Drpc.gid=220 -Drpc.ti
@echo off
goto end

:err1
    echo Usage: control scriptfile ["name=value"] [...]
    echo    Name value pairs shall be quoted
:end

```

If RPC is set up without authentication (using only application internal weak authentication) the uid/gid settings can be omitted from java call line. Note that uid/gid are have no native support on windows operating systems.

2.4 Content translation

The map definition can be extended to hold data which will replace content (e.g. unit name or quantity description) in replies returned to client.

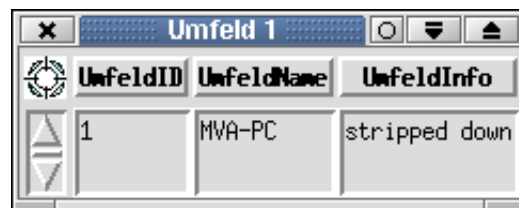
Attention: A more simple solution is to have foreign language content in same application element and to just reassign the ODS base attributes NAME, DESCRIPTION to other application attribute (column), but this makes back-end server table extension necessary.

1. Make a request (using AopWeb or a scripted fetch) to map server requesting all instances of e.g. unit table and save that XML file,
2. Edit that XML file by setting all values which shall be passed thru (not replaced) to zero length strings.
3. Replace the text strings remaining by content wanted to appear at client side (Dont change XML ID's !).
4. Merge data into the map definition (e.g. using text editor) by adding the IE XML-Nodes to the appropriate AE (application element) node.
5. Restart map server

Entities, which are set to “content translation” may not be updated, e.g. *aop_putval* requests are denied even if the instance attributes are not part of existing IE's. This situation makes content translations different to cache handling mentioned below.

2.4.1 Example

Original presentation of entity:



UmfeldID	UmfeldName	UmfeldInfo
1	MVA-PC	stripped down

Figure 2.1: original

By renaming attribute, using constants and adding a virtual field which is calculated by getting min and max from child instances, the entity looks like

SystemID	SystemName	SystemInfo	Testrange
1	AOXS	Mapped server using MVA-PC backend	19950126 - 19970121

Figure 2.2: mapped

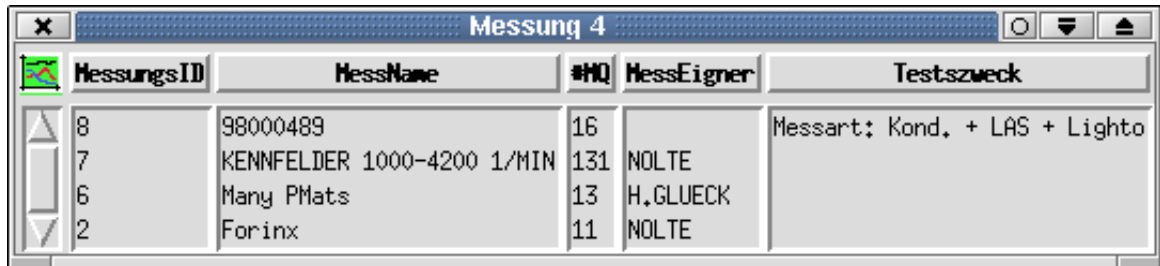
The definition used:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE AOXS SYSTEM "file:/home/horst/dev/applications/aoxs/test/aoxs.dtd">
<AOXS>
  <AE ID="A1" NAME="System" TYP="ENV"><?map A1?>
    <AA ID="A1.0" NAME="SystemID" TYP="ID" DTYP="INT"><?map A1.0?></AA>
    <AA ID="A1.1" NAME="SystemName" TYP="NAME" DTYP="STRING"/>
    <AA ID="A1.2" NAME="SystemInfo" TYP="DESCRIPTION" DTYP="STRING"/>
    <AA ID="A1.91" NAME="Testrange" DTYP="STRING"><?map {
  // get date range for all tests
  var first="9999";
  var last="";
  var tests=aop.getIEIds(id(aid,iid),"A3");
  for (var i = 0; i < tests.length; i++) {
    out(i);
    var beg = aop.getIEValue(tests[i], "VersBeginn");
    out (beg);
    if (first > beg)
      first=beg;
    var end = aop.getIEValue(tests[i], "VersEnde");
    out(end);
    if (last < end)
      last=end;
  }
  // yyyyymmdd only
  return first.substring(0,8) + " - " + last.substring(0,8);
}?></AA>
  <AS ID="S1.13" NAME="Quantities(SystemID)" TYP="ENVID" REF="R7.1"/>
  <AS ID="S1.3" NAME="Test(System)" REF="R3.1"/>
  <AS ID="S1.0" NAME="MotId(SystemID)" TYP="ENVID" REF="R2.0"/>
  <IE ID="I1.1">
    <IV>1</IV>
    <IV>AOXS</IV>
    <IV>Mapped server using MVA-PC backend</IV>
  </IE>
</AE>
```

The description of functions `aop.getIEIds()`, `aop.getIEValue()` to access data and `out` to perform execution tracing can be found within AOXS core documentation. The function `id(aid, iid)` is just a script defined within startup file:

```
function id(aid, iid) {
  return "I"+aid+"."+iid;
}
```

`aop.getIEIds()` may be used to show counts, e.g. assume You want to know how many measured quantities are available when looking into measurement, e.g.



MessungsID	MessName	#MQ	MessEigner	Testszweck
8	98000489	16		Messart: Kond. + LAS + Lighto
7	KENNFELDER 1000-4200 1/MIN	131	NOLTE	
6	Many PMats	13	H.GLUECK	
2	Forinx	11	NOLTE	

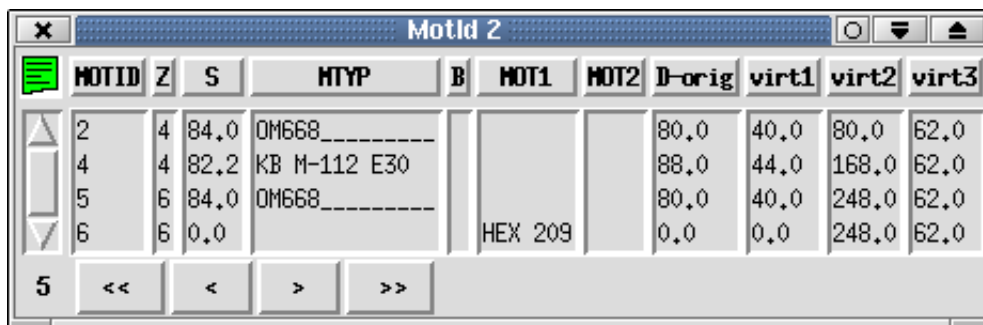
Figure 2.3: counting children

To do so, just put a virtual attribute

```
...
<AA ID="A4.91" NAME="#MQ" DTYP="INT"><?map {
// count number of MQ's
var mqs=aop.getIEIds(id(aid,iid),"A5");
return mqs.length;
}?></AA>
...
```

at appropriate place.

The next example takes one column (D-orig) to create 3 other columns showing a rescaled value (virt1), the accumulated value (virt2) and the mean value (virt3):



MOTID	Z	S	HTYP	B	MOT1	MOT2	D-orig	virt1	virt2	virt3
2	4	84,0	OM668_				80,0	40,0	80,0	62,0
4	4	82,2	KB M-112 E30				88,0	44,0	168,0	62,0
5	6	84,0	OM668_				80,0	40,0	248,0	62,0
6	6	0,0			HEX 209		0,0	0,0	248,0	62,0

Figure 2.4: arithmetics

The scriptlets used:

```

...
  <AA ID="A2.91" NAME="D-orig" DTYP="DOUBLE"><?map A2.4?></AA>
  <AA ID="A2.92" NAME="virt1" DTYP="DOUBLE"><?map {
    return aop.getIEValue(id(aid,iid),"D")/2.0;
  }?></AA>
  <AA ID="A2.93" NAME="virt2" DTYP="DOUBLE"><?map {
    var val=aop.getIEValue(id(aid,iid),"D");
    if (snr == 1) set(val);
    else val = sum(val);
    return val;
  }?></AA>
  <AA ID="A2.94" NAME="virt3" DTYP="DOUBLE"><?map {
    return mean();
  }?></AA>
...

```

together with utility methods in outer script:

```

...
// accumulator
var acc = 0.0;
var cnt = 0;
function sum(val) {
  acc += val;
  cnt++;
  return acc;
}
function set(val) {
  acc = val;
  cnt = 1;
  return acc;
}
function mean() {
  return acc/cnt;
}
...

```

The virt3 shows that evaluation occurs column by column. If virt3 column would be placed before virt2 column, virt3 would be NaN (0/0).

2.5 Common mistakes

- XML syntax errors, e.g. ID's not unique, open-close braces mismatch, ...: The XML parser will raise an exception on startup.
- ECMA-Script errors in server startup script: No startup occurs, as usually standard error log has not yet been redirected to a file, ensure to keep console output to identify reason.
- ECMA-Script errors in scriptlets: Will cause script compile errors on startup.

- Runtime script errors when calculating virtual attributes return constant values depending on datatype of attribute (e.g. Float: NaN).
- Virtual attributes implicitly mapped: May cause mysterious data updates and shall be used for special reasons only.

Chapter 3

Extensions

Extensions have been offered separately and will be made available in a second phase. Some of the extensions will be part of AOXS core part.

3.1 Performance features

3.1.1 Caching

For “readonly” data, e.g. units, quantities, the map server might cache the informations fetched, e.g. might keep a copy internally by performance reasons. The current caching implemented in aop session applies to names only (needed to replace id's bei names in AopWeb views). The extended solution shall hold complete sets of data within AOXS XML data (indexed by Id as usual). This feature is different to phase 1 implementation where constant instances are preset within the data model and are not dynamically added in running service. This feature is useful only when using multiple backend sessions, e.g. one for national language encoded dictionary instances.

3.1.2 Additional indices

Aoxs shall get an additional (beside ID) index to support hash code access to attributes by name (see AOXS core).

3.1.3 Dynamic reloads

Currently service has to be stopped when configuration is changed, e.g. when new map shall be used or dictionaries shall be extended/replaced. That extension is related to the need for separated operator interface and is tied to servlet implementation of AOXS-GW as indicated below.

3.2 Additional interfaces (beyond ONCRPC)

Later revisions will include firewall tunneling and related security features. HTTP/HTTPS will be used to encapsulate RPC when leaving intranet¹. Those

¹Resembling SOAP (former XML-RPC)

functionalities will be added incrementally, starting by replacing the main batch process (Control) by a servlet. The conceptual background: The object AOS now instantiated in a batch process (Control) becomes part of a servlet implementation

3.2.1 Running as servlet

The transparent server might be implemented as a servlet, providing informations using http get and accepting e.g. changed mappings using http put. This way AOX-S-GW gets access to features like SSL and to HTTP protocol usage for easiertunneling.

3.2.2 Firewall handling

Chaining might be used for proxy/firewall concepts: The mapping servers are located within intranet and communicate e.g. using http tunneling thru internet. To allow single port assignement for secure channels, HTTP-POST-requests and associated reply encapsulate the RPC-requests and replies. The HTTP-RPC encapsulation is made by changing RPC layer implementation (the server resp. client stub). The RPC Portmapper is part of the server implementation (e.g. the servlet accessed by the end user application) while the ODS server side servlet uses the portmapper (registry) in use by ODS server.

There are two scenarios:

1. Each of the two endusers (ODS Server and ODS Client application) are within distinct intranets behind firewalls. The two intranets are connected by secure channels. In that case it is the responsibility of the firewall to provide the channel connecting the local ports. No SSL handling is required by gateway. The firewall administrators just assign a port at each side. That port number is assigned to the two tunneling servlets. Beside protocol conversion/wrapping, any (even both) of the servlets can perform ODS data structure conversion too.

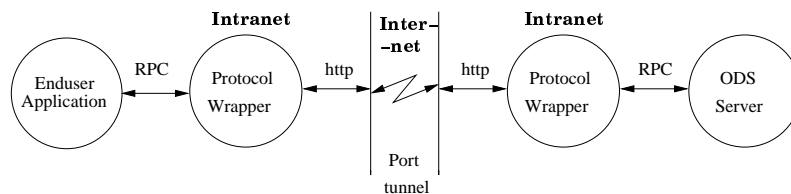


Figure 3.1: Connecting two intranets

2. The application (e.g. a browser like AopWeb) is part of Internet accessing ODS server thru one firewall. In that case the Servlet has to RPC requests have to be encapsulated and feeded thru SSL to defined port offered at firewalls internet side. This usually means, that the target servlet becomes part of "outside" web services offered and has to be installed at

firewall (creating higher risks and higher administration efforts to get it secure). Enabling usage of proper client authentication (to get the https connection) is only extra effort (compared to above scenario) to be made by servlet provider, any other administration efforts apply to security administration.

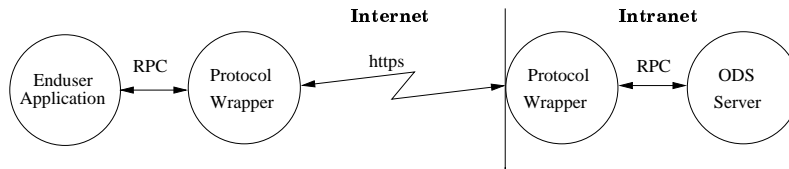


Figure 3.2: Connecting an outside client

A third case (client application inside and server part of internet) is considered irrelevant and is covered by above scenarios.

For some applications it may be useful to include the protocol conversion into the original application as a library to enhance performance by eliminating one hop within total message path and to make deployment easier (no extra setup of servlet engine at application side).

Quite sure ODS data structure conversion (if necessary) has to be provided by the other servlet within message path.

The most uptodate approach for HTTP encapsulating ONC-RPC protocol might be usage of SOAP protocol promoted by Microsoft and IBM and proposed to W3C for standardisation. Using SOAP is first choice for servlet - servlet and servlet - client application protocol.

3.2.3 Registry handling

There is no automatic global announcement of ODS service availabilities². The ODS protocol wrappers are prepared to handle the protocol defined by ASAM-ODS 3.x. Extending the AVL specific “Registry” will need to create extra versions of protocol wrappers and the original ONC-RPC based program together with wrappers has to be deployed to systems within client applications scope. To overcome that administrative nightmare one shall consider to introduce Web-suitable registries, e.g. LDAP or UDDI. Up till that point the availability of services has to be parametrized, e.g. the client side accesses the local servlet (localhost) which offers some preset RPC numbers. Each of those rpcnumbers has a remote host servlet URI parametrized. On openenv request, the URI is used including openenv authentication fields. The addressed servlet accesses the ODS server associated to given URI, creates an ODS session and an http session. Any requests coming within same http session are associated to the proper ODS session. No RPC-programnumbers are transferred thru internet, e.g. the numbers offered by client side servlet are independent from program numbers used by server side.

²The ODS server side has no mean to create a new protocol wrapper instance at the client side

In a firewall tunneling configuration the mapping node **M** of figure 1.1 is split into two parts, but equivalent registry nodes **R** are totally distinct. The client side **R** offers just the “tunneled” services while the server side **R** offers the original (and potentially the mapped versions). The client side servlet offering RPC accessibility is (shall be as project offer includes that functionality) able to register itself with an local (in same intranet) AVL registry, e.g. may publish its served RPC program numbers there.

Nevertheless, that neither can guarantee availability of remote RPC service nor can that feature be used to dynamically reflect changes in service availability at remote side. The availability can be determined only by accessing the service. For load balancing, the server side servlet might use a pool of RPC program numbers (round robin), but that has nothing to do with a “global” AVL registry.