SIEMENS

Communication Interface between Central Units and from Central Units to other Systems

Author: Senninger Markus

Version: 2
03.03.2002
State: Zwischenstand

Filename: Soap_Protocol.doc

1. Inhaltsverzeichnis

Seite

1 Introduction

The communication between Central Units and communication interfaces to these Central Units should be implemented unique in all Central Units.

As an approach the communication interface Soap can be used. The Soap protocol is the common communication interface over them all communications are done. This communication interface is called OCPI.

These interfaces are declared to be open. Therefore they can be used by external systems. Concert does not know any other interfaces (besides some exceptions) but Soap.

The task of this document is to describe this open protocol and its usage.

It is not the task of this document to describe data structures of the particular data.

1.1 History of Change

Version	Date	Name	Change
3	2008	Senninger	Formal adaptions to OCPI
2	10.03.2002	Senninger	Revised chapter 4
1	02.03.2002	Senninger	Create document

1.2 Definitions

Statement	Definition
SOAP	„Simple Object Access Protocol. Transfers commands with the means of XML by using httpIt is described within http://www.w3.org/TR/SOAP.
Protocolmanager	Protocollayer which implements commands and the buffer.
SoapServer-Interface	Includes Soap and ProtocolManager at the server site
SoapClient-Interface	Includes Soap and ProtocolManager at the client site

1.3 Abbrevations

Abbrevation	Description
SSL	Secure Socket Layer.

2 Transfer Protocol Soap

This chapter describes the Soap interface.

2.1 Technology

Data is encoded with XML. This includes the following advantages:

- usable protocol for all components (no dependency on used data)
- tracable
- plattform independency.
- easily expandable.

The protocol includes easy commands like write, get or delete. FTP is not sufficient.

As transfer media

- SOAP on the base of http is used

Soap itself uses XML to build its data.

2.2 Protocol Requirements

- data is represented as XML data at the output interface.
- data is accepted as XML at the input interface (configuration)
- commands are embedded within XML. (e.g. post,get)
- Objects are identified by external identifiers.
- one cannot mix different objecttypes within one request
- the protocol is stateless within the server. This meaans that the server does not know anything about the client.

2.3 Concept and Security

The protocol is a client server protocol.

Security will be implemented by using username and password in plain text.

Three classes of unintended foreign accesses:

- **The aggressor uses the transmission link to intrude into the computer (client or server), for example to steal or modify data.** This scenario is the most frequent (and the most severe in terms of the consequences). To put a stop to this under TCP/IP, it is necessary for only one port at most to be enabled towards the outside and for this port not to be a default port. In the protocol described above (with http as the underlying layer), a firewall for the client can be made so restrictive that no connections can be opened towards the clients and only one specific port is opened out to the control centre. The best possible protection for the client is provided with a suitably configured firewall. In this case, SOAP is not configured to port 80, but to a project-specific port number. The control centre can also be fairly reliably protected by i) not installing SOAP to port 80 in the specific project and ii) by implementing SOAP via its own classes and not via the IIS.

- **The aggressor attempts to gain access to SOAP to execute OCIT commands there as an OCIT client.** (This scenario does not call for a particularly trained aggressor.) In this scenario, the aggressor does not even attempt to reach default ports and is not even able to steal data or to manipulate files. Such an attack is also only suitable for the server (and not for the client). Such an attack is relatively simple, but is made practically impossible by the user name and the password, provided the attacker is not aware of a valid pair.
- **The (professional) aggressor intercepts a TCP connection and determines a user name and password** (professionals only). This scenario does not permit any theft either and also no file manipulation, but does allow the aggressor to pretend to be an OCIT client without knowing a password and user name. You have to do a cost/benefit calculation here. As it is not quite easy, except for specialists, to intercept and read a TCP connection, the data really ought to be interesting enough to make such a break-in worthwhile. To circumvent this, an HTTPS connection can be used with moderate effort instead of an http connection (this entails additional programming effort because SSL libraries have to be incorporated) and, when using Verisign, for example, a public key has to be requested for the control centre and has to be paid for, unless communication partners agree on a self-generating key and enter it manually. In such a case, it is practically no longer possible to intercept and read a connection between the client and the control centre. In my opinion, though, such a changeover is not worth the effort.

The server saves a list of user names and their associated passwords. The server also stores details of which operations are permitted by which user. Access from more than one client is possible with the same name, with the result that not all clients have to be made known in Concert.

2.4 Commands

With the protocol you are able to read and configure data Besides it is possible evaluate objects and instances concerning availability and structure dynamicly during runtime..

Every command consists of two XML structures

On call a XML structure is transfered from Client to the Server, The result of the call is put into a Result or also called a response structure and sent back to the client.

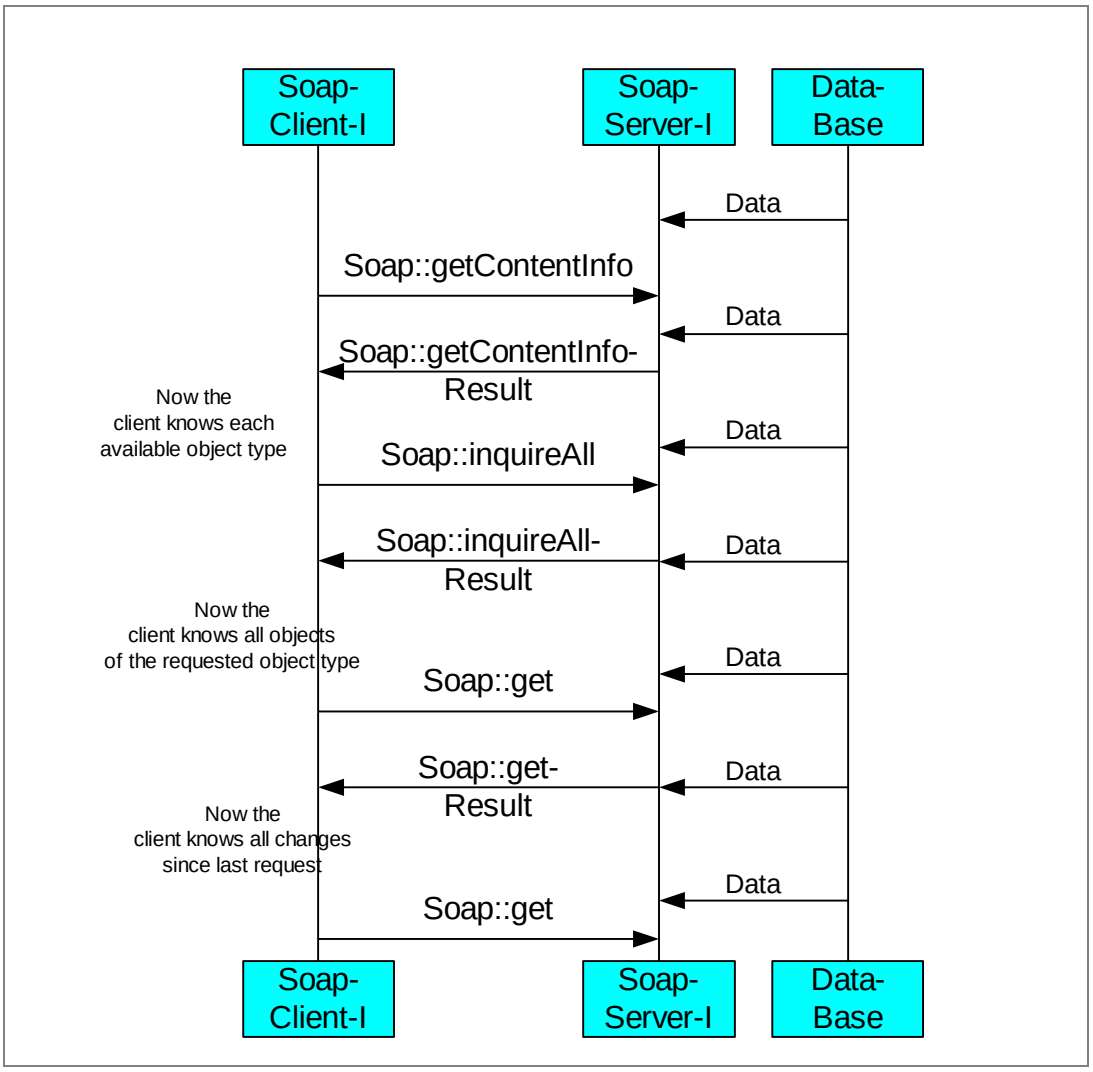


Figure 1: Usual Sequence to Read Data From SoapserverInterface

In the above image you see the sequence of commands to read data by a client.

2.4.1 Reading data by a Client

All commands to read data include a filter as parameter.This filter is a field of objects which are to be read. In case the filter is empty all objects will be returned in any other case only those objects included in the filter list.

In order to ease the resynchronisation a start information is added to each response of a read access With this information the client is able to decide when to resynchronise (in case of changed start string).

The server offers all readable data at its outer interface . The server offers data of several object types.

The Client has to queury all available object types with the command getContentInfo

Those objecttypes which are contained in the answere can be read by the Client (if he has read access to it). either by the command get or inquireAll

The difference between inquireAll and get is as follows:

.InquireAll delivers all objects of the requested object type with the last state/contents of the objects. It has to be used in any case of resynchronisation (e.g. restart of the server or the client).

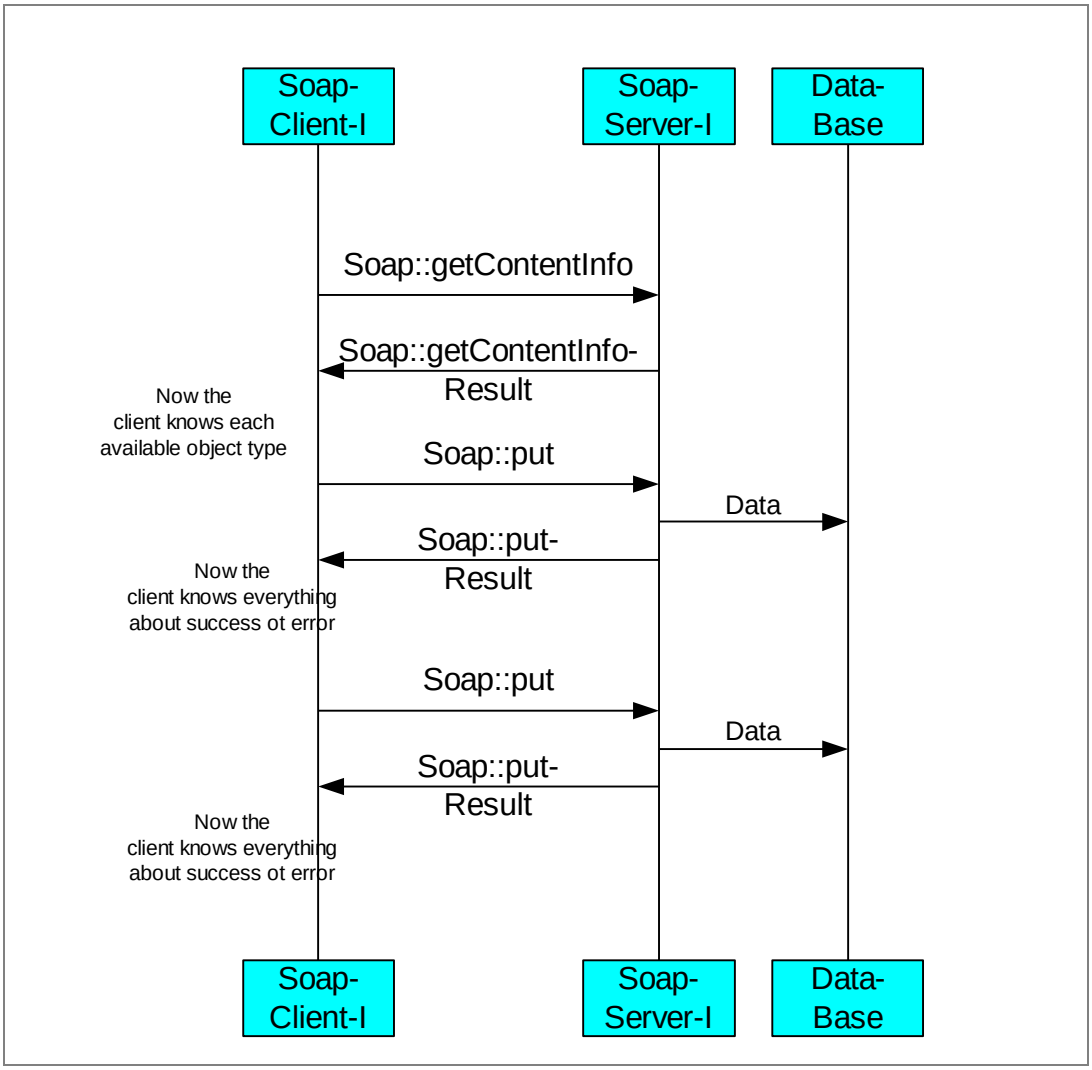


Figure 2: Usual Sequence to Write/Configure Data in the SoapserverInterface

2.4.2 Configuration of Data in the Server

It is possible to configure data at the server. The command put is to be used.

The behaviour of the Server depends on the object type, In case of unknown objects in the put command.. Either the object will be created or an error will be returned. To delete objects the command delete is required

The configuration interface is rather simple using the command put. The configuration data can be placed into it and is to be sent towards the server. The server accepts it or rejects all unconfigurable objects in the putResult-list.

2.5 Sequence Control

The protocol itself is connection less. To achieve sequence control it is sufficient to wait for the response of the according request. This is achieved by http. So no sequence control is required.

2.6 OSI – Layering

This what we call server is devided into different parts. First we have the Soap functionality which is covered by the Soap component. In addition to the Soap a protocol manager is used, whose task it is to implement all commands including the required data buffer for server functionality. Finally a data access layer is required to link from XML to the used database (e.g. Object Manager).

The following image describes this in a layer modelling.

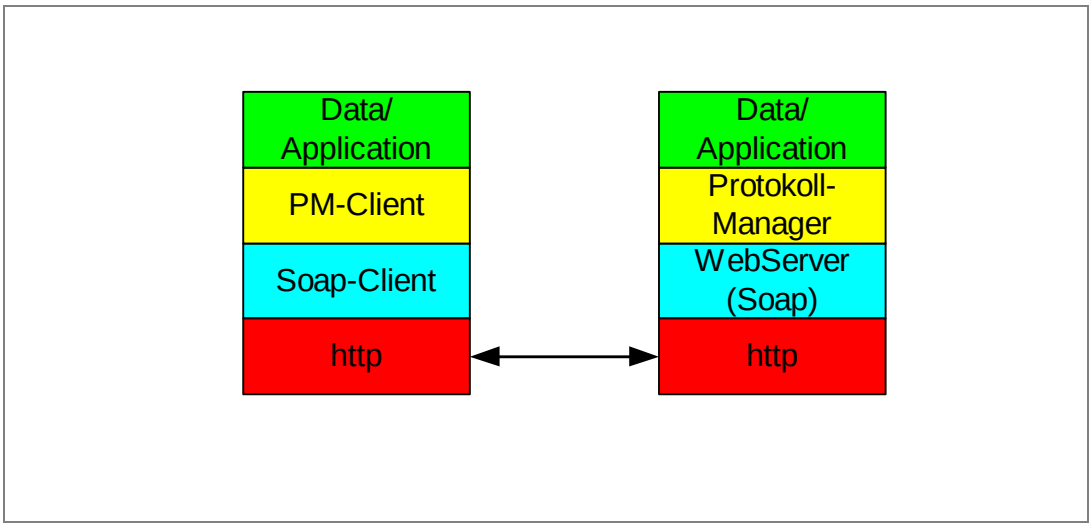


Figure 3: Layers

The client covers similar layers.

2.7 Commands in Detail

The commands in detail will be described here

We have defined getContentInfo, get, inquireAll, put and delete. The functions have similar meaning for all available objecttypes. They differ only in parameters..

element **Protokoll**

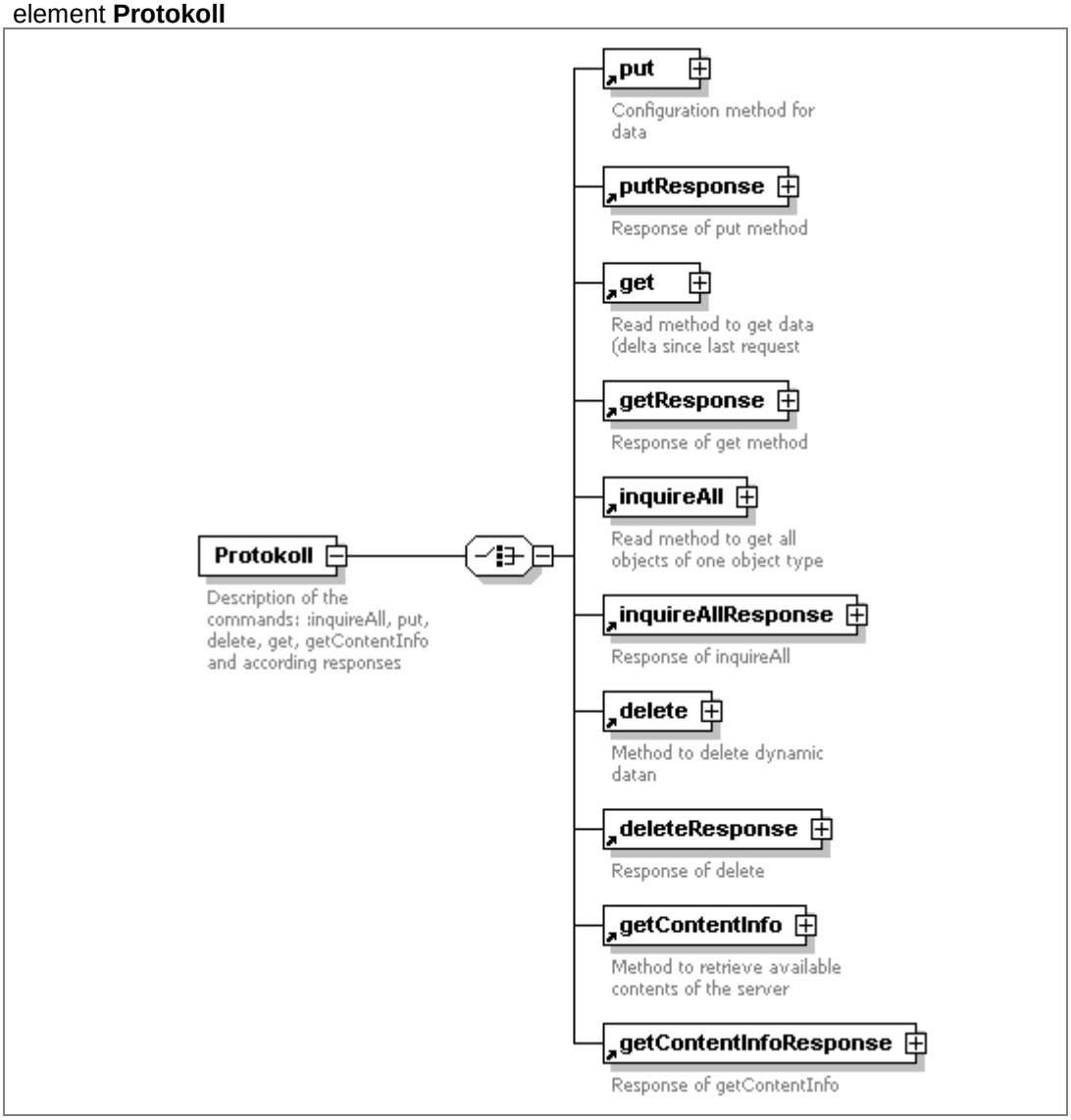
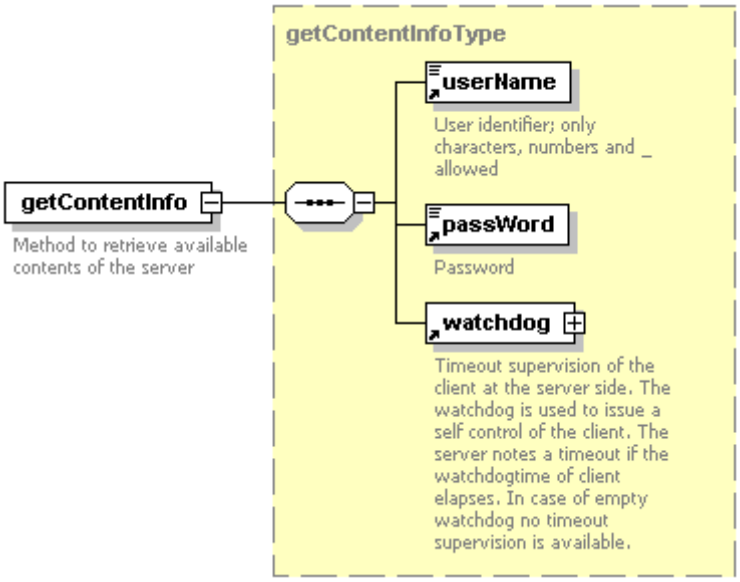


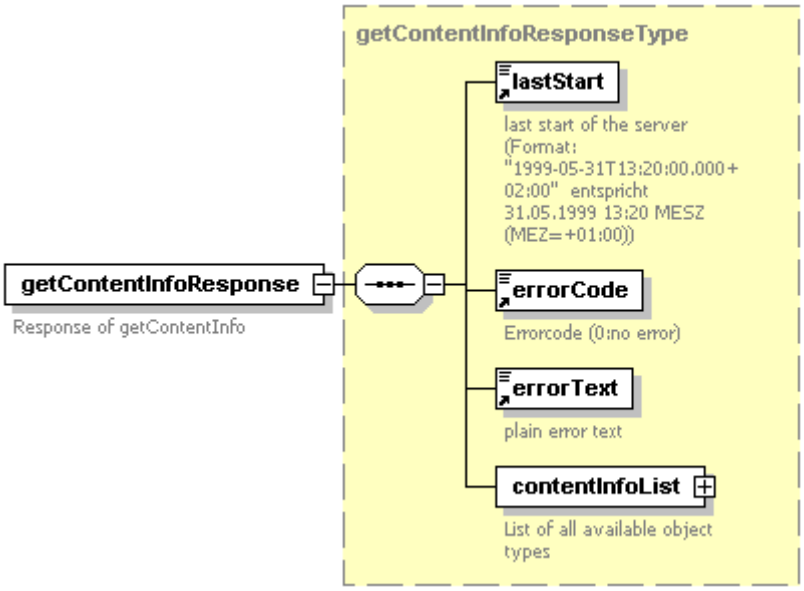
Figure 4: Available Commands

2.7.1 getContentInfo

getContentInfo has no parameters, besides name of the client and the password and optional the watchdog.



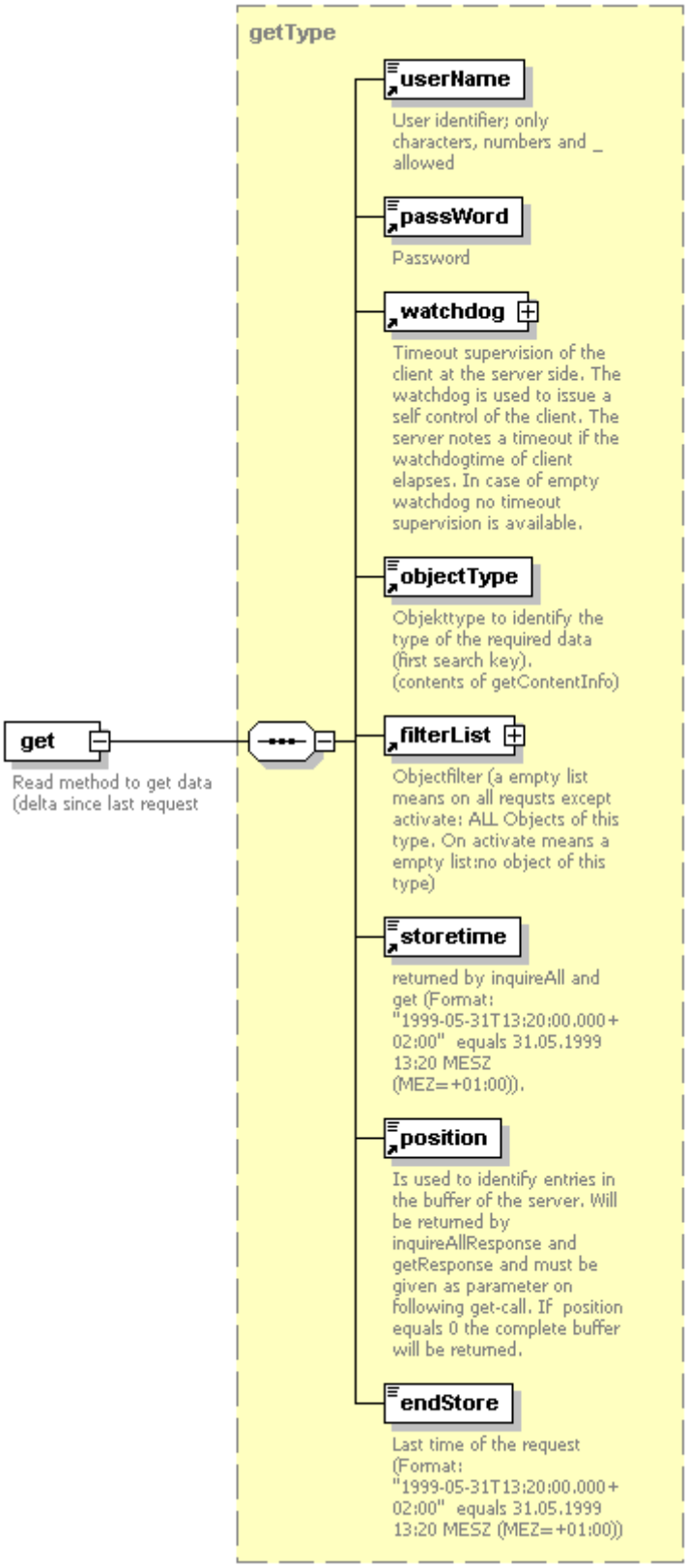
As response you receive a list of all available object types including access rights and recommended update cycle to them.

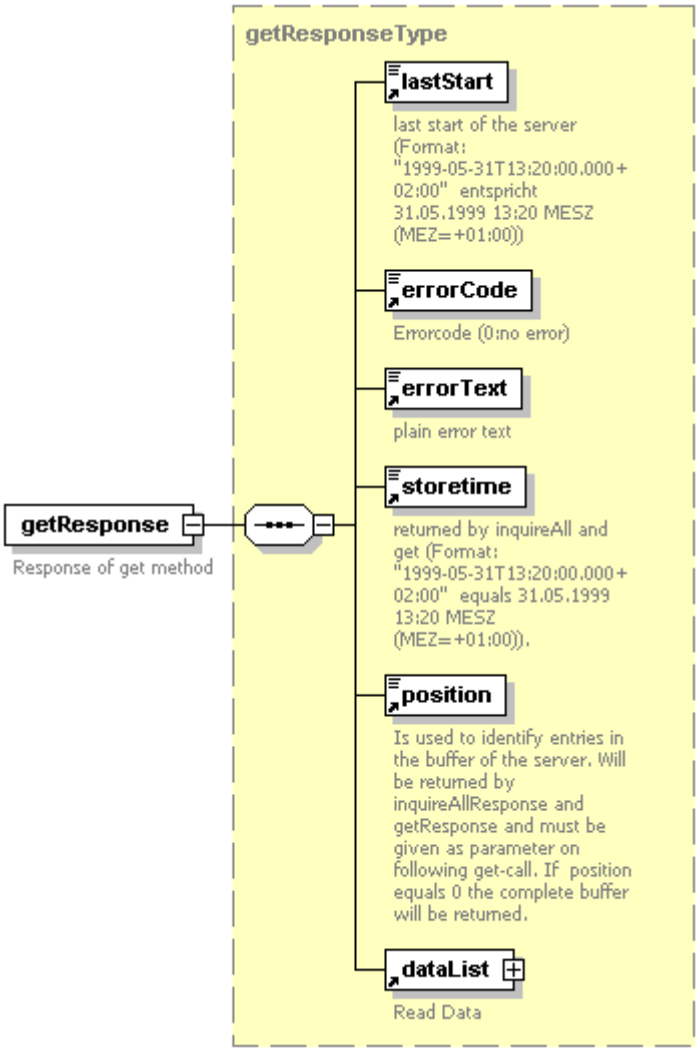


2.7.2 get

get has besides the standard parameters either start and endtime to get all values within this time range or it includes the position number from where the request is related to. Usualy the position number is the positionnumber returned by the latest inquireAllResponse or getResponse.

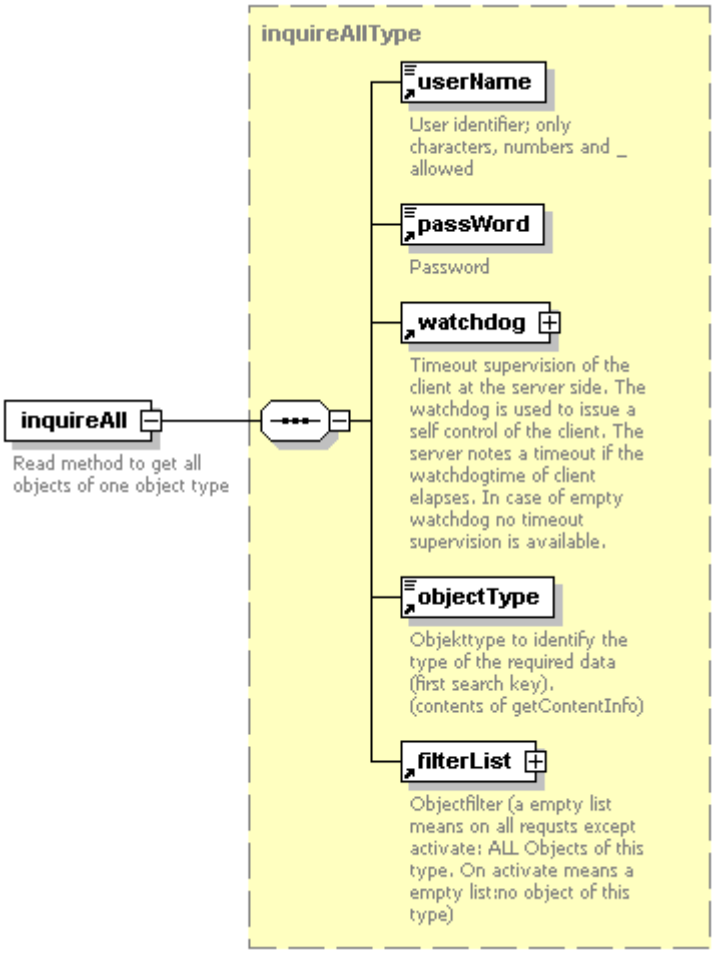
So the inquiry returns all values since last request. This is the recommended way how to use this interface.



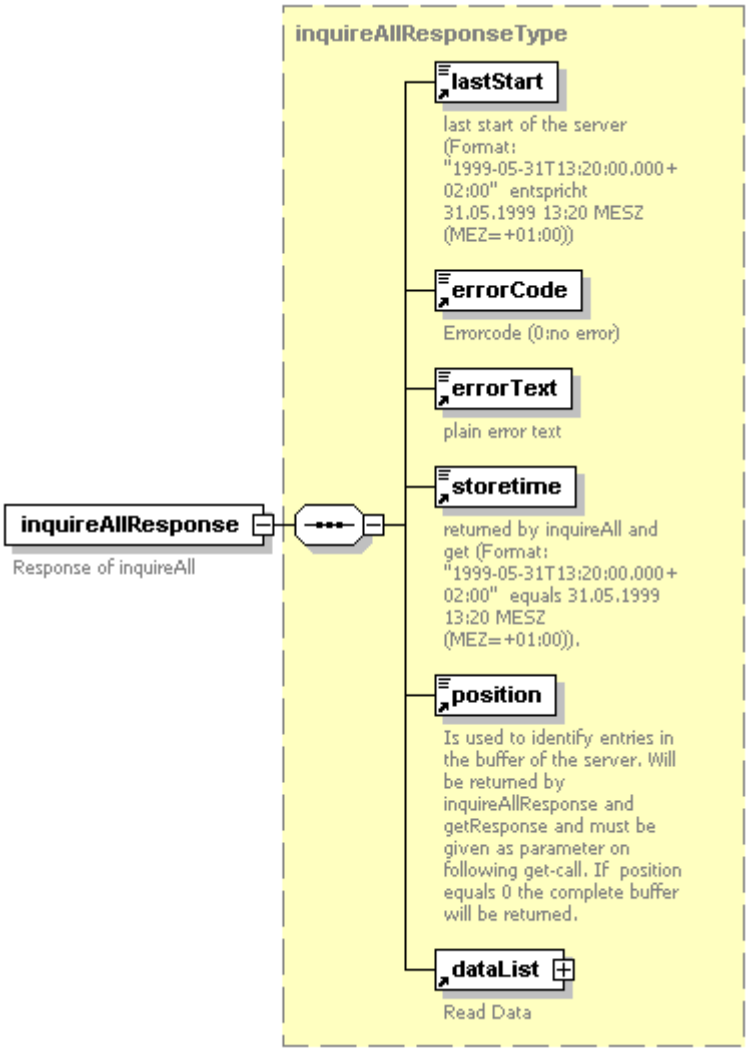


2.7.3 inquireAll

Inquire all includes only standard parameters.

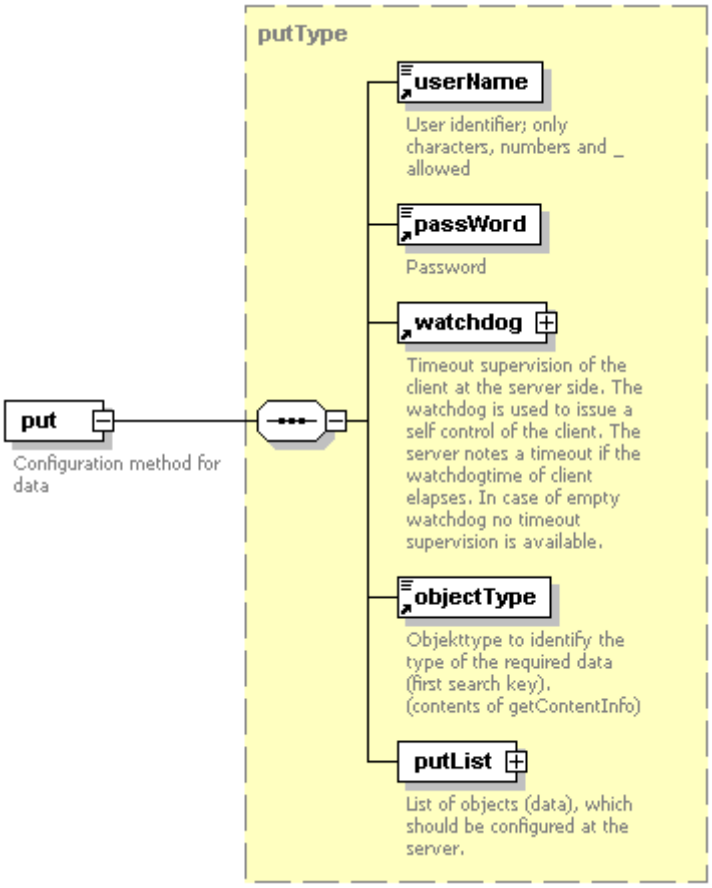


As response all topical values will be returned.

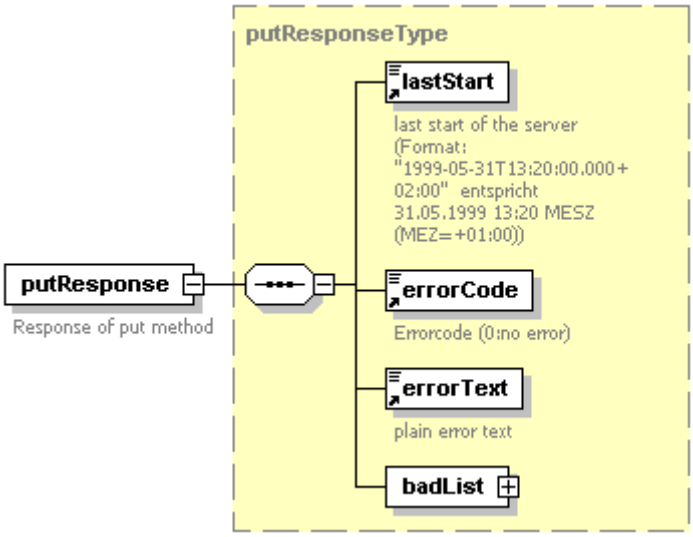


2.7.4 put

put includes all data instances which should be configured.

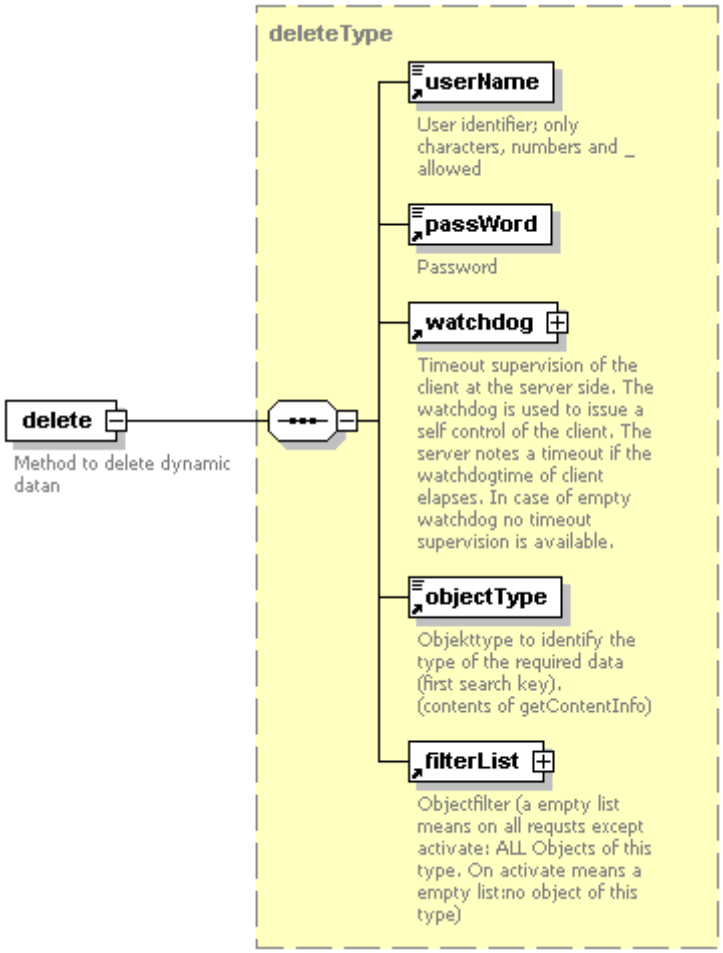


As response you get unconfigurable data instances (usually none).

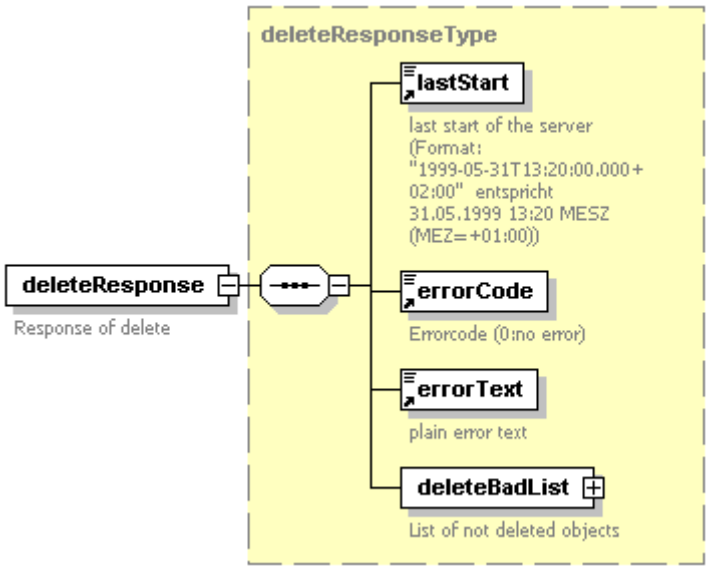


2.7.5 delete

delete removes data ion case this is possible, Data instances to delete have to be added to the filterList.



As response you get the data instances, which could not be deleted.



2.7.6 activate/deactivate/deactivate/deactivateResponse

Usually the data source (the SoapServerInterafce) represents all objects of an objecttype at its interface. For some objecttypes it makes sense to represent only those objects which are required by the client. Then the client has to activate those objects which are required, after usage these objects have to be deactivated concerning the representation at the interface.

Note: These commands are not yet implemented !!!

2.7.7 General Parameter

There are standard parameters mentiona above, but defined here:

Inputparameter

- UserName is used to define proper access rights for different users
- UserPasswd authentifies the user
- Watchdog is a structure with which the client tells the server when it calls the server again. It is to be used for timeout supervision.
- start_store – Start of store
- end_store – End of store
- position – Startposition for this access
- filterlist – List of objects, which are to be read

Outputparameter

- LastStart is the date/time stamp of the last server start. In case of a change of lastStart the client has to resynchronize by using the command inquireAll.
- Errorcode is an error code, in case of an error executing a command. In case of wrong XML-Structures errorcode will not be used, SOAP-Fault will be returned instead..
  The following errorcodes will be used:

Errorcode	Description
0	no error
1	wrong user name or wrong password (deliberately combined to one error for secrurity reasons)
2	overflow of ringbuffer
10	data record not available
11	data record cannot be changed
12	data record cannot be deleted
13	wrong value for attribute
13	
15	
16	
17	
18	

19	
20	
21	
22	
23	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	

- Errortxt is a human readable description of the error code.
- position – Position of last data accessed. Is to be used for following accesses
- Datalist – returned data

3 Data structures

Data structures embedded in the protocol (e.g. the mentioned commands put, inquireAllResponse …) are defined in own scheme definitions.

They are not scope of this document (...).

4 How to Use

This chapter describes how to use server and client depending on different use cases. This can be only a recommendation. The real architecture has to be decided on each certain case.

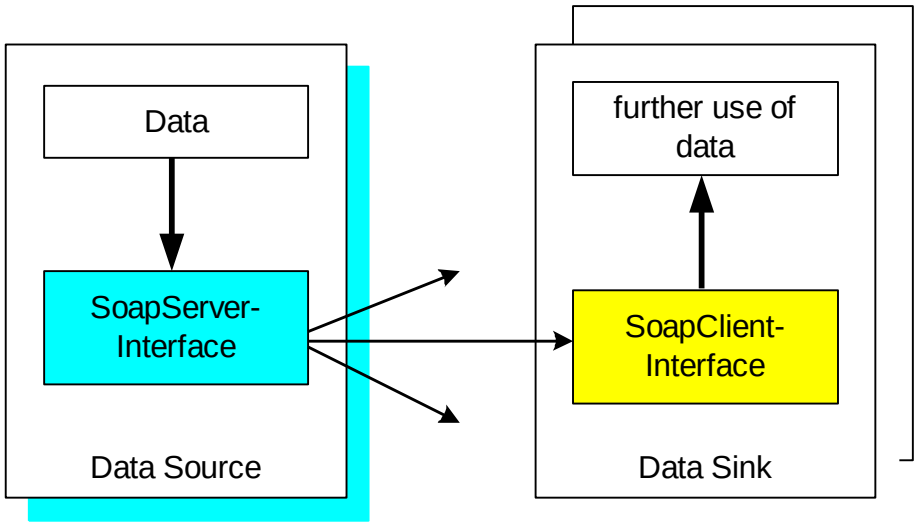
4.1 Data delivery interface / one or more Consumer

In case we have several consumers requiring data from time to time and not under real time requirements the following architecture is recommended.

- The data source includes the SoapServerInterface, which implements functionality like data buffering.

- The data consumers include the SoapClientInterface, which implement the access to the soap server in case the data is required (commands inquire and get).

This includes the following advantages:
- reduces costs because the client can access the server on demand
- improves the acceptance of the protocol because the SoapClientInterface is easier to implement
- there is no need that the data source knows anything about the consumers.
- only one communication interface at the data source



Example:

This interface is used in the following applications:

- the Prediction System Monet requires data of the VMZ Concert (measured data like occupancy, speed and traffic count) . Those values are inquired every view minutes by Monet. There is no need to update the data immediately. The Concert is Server and the Monet is Client in consequence.
- In any case of internet providing the data source has to be server and the internet service is client.

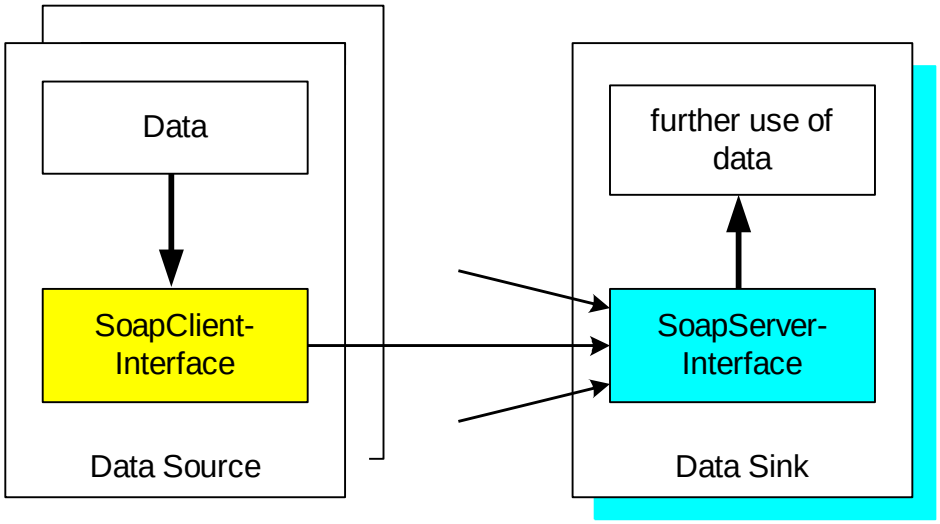
4.2 Configuration Interface

In case a system (data source) requires a configuration interface at the data sink, the following architecture is recommended:

- The data source is client
- The data sink is server

This includes the following advantages:

- transmission on demand
- realtime configuration (no polling required)
- more than one configuration interface at one Data Sink with the same communication interface



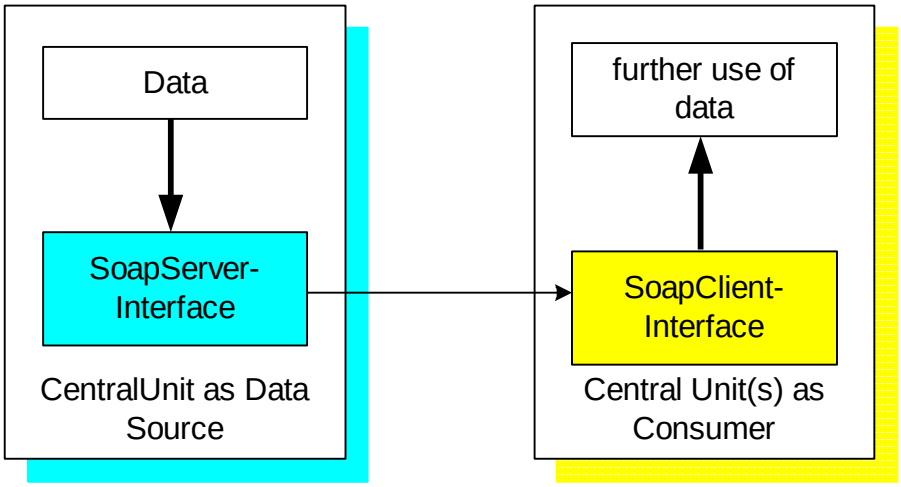
This interface is used in the following applications:

- in case a subsystem likes to get rid of measured data like roadwork management systems or subsystems with a kind of data forwarding interface, then it uses the client as a simple way to forward data to the data sink.
  It should only be used for a fiew object types to avoid overload situations.

4.3 Interface between Central Units to update its data (unidirectional)

In case of data exchange between Central units the following interface is recommended.

- Data source is Server
- Data sink is client



In fact we have the same configuration as mentioned within chapter Data delivery interface / one or more Consumer.

4.4 Interface between Central Units to update its data (bidirectional)

In case a interface in two directions is required it is clear to implement the interface twice, because the Central unit is server and client at the same time.