OCIT®

Open Communication Interface for Road Traffic Control Systems

Offene Schnittstellen für die Straßenverkehrstechnik

OCIT-C Center to Center

Transport Protocol

OCIT-C_Protocol_V1.2_R1

OCIT Developer Group (ODG)&Partner

OCIT-C Center to Center

Transport Protocol

Document: OCIT-C_Protocol_V1.2_R1

Editor: ODG & Partner

Contact: www.ocit.org

Contents

Document history

Version Issue	Date	Distribution List	Comment
V1.1_R1	**30.10.2014**	**PUBLIC**	**Version 1.1 Issue 1**
			2.2.1: inquireAll status of the objects in the past (text added) 2.3.5: Text correction 2.3.4: New text version 2.3.5 Text amendment 2.3.8.1 Correction of figures 1 to 17
V1.2_R1	**16.12.2016**	**PUBLIC**	**Version 1.2 Issue 1**

1 Introduction

OCIT-C stands for Open Communication Interface for Road Traffic Control Systems - Center to Center. OCIT-C covers the communication functions between central traffic control and traffic guidance systems:

- Traffic control centers and traffic management centers (urban, regional, interregional)
- Traffic engineer work place with traffic control centers
- Parking guidance systems, parking facility systems
- Roadworks management systems
- Local internet users (city info online)

The definition and maintenance of the OCIT-C interface is carried out by the ODG and their partners.

OCIT-C provides a standard which perfectly supplements OCIT-O. All requirements for traffic control up to the overarching traffic management are covered by using OCIT-C and OCIT-O for communication from control centers to field devices.

OCIT-C is therefore geared towards practical requirements. With its low implementation costs, its use is also suitable for solutions with small budgets.

The featured properties of OCIT-C are:

- An exchange protocol with a simple request-response communication pattern (direct data request) based on the SOAP standard.
- Definition of a comprehensive data model in the process data area containing all subfields of traffic control and traffic guidance.
- System integration and desired adaptations are governed in advance by project planning.
- Conformity tests for the protocol are carried out in a test environment provided at www.ocit.org. Tests of entire implementations (protocol and data contents) are carried out on a project-specific basis.
- Expansions to the DATEX II components are possible based on your project requirements.

The communication interface should be implemented in the same manner in all central units. To do this, the SOAP protocol is used as the primary communication interface, through which all communication is carried out. The specification described here is called the OCIT-C protocol.

This interface is open and can be used in various systems, primarily in the field of road traffic control systems. The aim of this document is to describe the OCIT-C protocol and its application. This document does not aim to describe the data structures of the data to be transferred. These are described in the document "OCIT-C data".

1.1. Terms and Abbreviations

Term / Abbreviation	Description

AP	User program
Client	A program which wishes to use services offered by other (servers) and actively opens them to do so.
DATEX II	Specifications of Technical Committee 278 of the European Committee for Standardization (CEN) for the exchange of traffic-related data between traffic control centers.
FTP	File Transfer Protocol, a network protocol for transferring files
http	HyperText Transfer Protocol, a protocol for transferring data over a network.
TSS	Traffic signal system
Method	The algorithms assigned to a class of objects. Also used as a synonym for function, procedure, command, action.
PT	Public Transport
OCIT	Open Communication Interface for Road Traffic Control Systems
OCIT-C	Open Communication Interface for Road Traffic Control Systems - Center to Center. OCIT-C covers the communication functions between central traffic control and traffic guidance systems.
OCIT-O	OCIT outstations Interface between traffic control centres and traffic signal controllers for controlling and supplying the traffic signal controllers.
ODG	OCIT Developer Group
OSI	Open Systems Interconnection Reference Model, a communication model of the International Organization for Standardization (ISO) for communication protocols in computer networks.
OTS 2	Open Traffic Systems, Version 2
Server	A program that offers certain services and passively waits on incoming calls (from clients) to do so.
SOAP	Simple Object Access Protocol, it is a protocol which enables data to be exchanged between systems. SOAP uses the "Remote Procedure Call", through which it enables the functions in other computers to be called. See http://www.w3.org/TR/SOAP
SSL	Secure Socket Layer.
Soap-Server-Interface	Soap and Protocolmanager on the server side
Soap-Client-Interface	Soap and Protocolmanager on the client side
Protocolmanager	Protocol layer used for implementing commands in the buffer

TLS	Technical delivery terms for roadway stations. The TLS are a standard for the structure of traffic control systems on major German Federal highways. Editor: German Federal Highway Research Institute
TCP / IP	Transmission Control Protocol / Internet Protocol, a family of network protocols for the Internet.
VDV	Verband Deutscher Verkehrsunternehmer (Association of German Transportation Companies)
WSDL	Web Services Description Language, a platform, programming language and protocol independent description language for web services for exchanging messages based on XML.
XML	Extensible Markup Language, a markup language for presenting structured data in the form of text. XML is used among other things for a platform and implementation-independent exchange of data between computer systems. An XML document is made up of text characters, in the most basic case in ASCII coding, and is therefore machine-readable. It does not contain binary data. The XML specification is published by the World Wide Web Consortium (W3C) as a recommendation.
XSD	XML schema, a recommendation of the World Wide Web Consortium (W3C) for defining structures for XML documents. The structure is described in the form of an XML document. Furthermore, it supports a large number of data types. The XSD schema language describes data types, individual XML schema instances (documents) and groups of such instances. A specific XML schema is called an XSD (XML Schema Definition) and the file usually has the ending ".xsd".

2 Protocol

All communication via the interface is processed using the SOAP protocol.

This chapter describes the application of the protocol for the OCIT-C interface. This configuration must be installed in all clients and servers.

The exact description of the underlying protocol of the data model, as well as the basic description of the attributes an elements is given entirely within the individual schema definitions in the form of XML schema definitions (XSD). These are both text and machine readable. The schema definitions have been compiled in English.

2.1. SOAP transfer protocols

This chapter describes the SOAP interface.

2.1.1. Technology

The data to be transferred are coded as XML. This has the following advantages:

- Common protocol for all areas,
- Independence of the type of data,
- platform independent,
- simple to expand.

SOAP is used as the transfer process based on http. SOAP also uses XML to structure its data.

The protocol contains simple commands such as 'get' or 'delete'.

2.1.2. Protocol requirements

- The protocol is a server-client protocol.
- Data are displayed on the output interface as XML.
- Data are accepted at the input interface as XML.
- Commands are embedded in XML.
- Objects are identified by external identifiers.
- It is not possible to request more than one object type at a time (in a "request").
- The protocol inside the server is stateless. The server knows nothing about the client.

2.1.3. Security

The server contains a list of usernames and the associated passwords, as well as the operations the users are allowed to perform and the client accesses.

2.1.4. Required bandwidth

The required bandwidth depends on the number of clients, the object types and the objects in the system. Therefore, it is not possible to provide any specifications regarding bandwidth. A Local Area Network (LAN) between the central applications however will offer sufficient transfer capacity.

2.2. Protocol functions

The protocol allows data to be read and configured. Furthermore, it is possible to evaluate objects in terms of their usability and to structure them dynamically while it is running. Each command consists of a request (Request) and a response (Respond).

In each request, an XML structure is sent from the client to the server. The result is sent back from the server to the client as a "Resultat" (also called "response structure").

Available methods:

Request	**Response**	**Function**
put	putResponse	Configuring objects
get	getResponse	Request for data modified since the last request
inquireAll	inquireAllResponse	Request of all objects of an object type
delete	deleteResponse	Deletion of dynamic data
getContentInfo	getContentInfoResponse	Request for object contents
wait4Get	wait4GetResponse	Request for data modified since the last request (such as get), with the difference that the response is delayed until data are available.

2.2.1. Reading data through the client

All protocol functions for reading data include a parameter (filter), which labels the objects that should be read.

If the filter is empty, all objects in the associated response are sent back. So that the client is able to resynchronise in the event of an interruption, the start information of the previous response of a reading access is included in the transfer.

The server offers all readable data of the available object to its external interface. The existing object types can be requested by the client using the command 'getContentInfo'. These can be read by the client using "get" or "inquireAll" (if read access is allowed).

The difference between inquireAll and get is:

- • inquireAll (resynchronisation function) delivers all objects of the requested object type with the status or content of the object. inquireAll must be used in any case for a synchronisation (e.g. server or client restart). get (here used as read changes) delivers all content changes of the requested object type carried out since the last request. This mechanism is described in detail in chapter 2.3.1, Data buffering and position handling.

The following sequence diagram Figure1 shows how it is possible to periodically request data from the server using the protocol functions 'inquireAll' and "get".

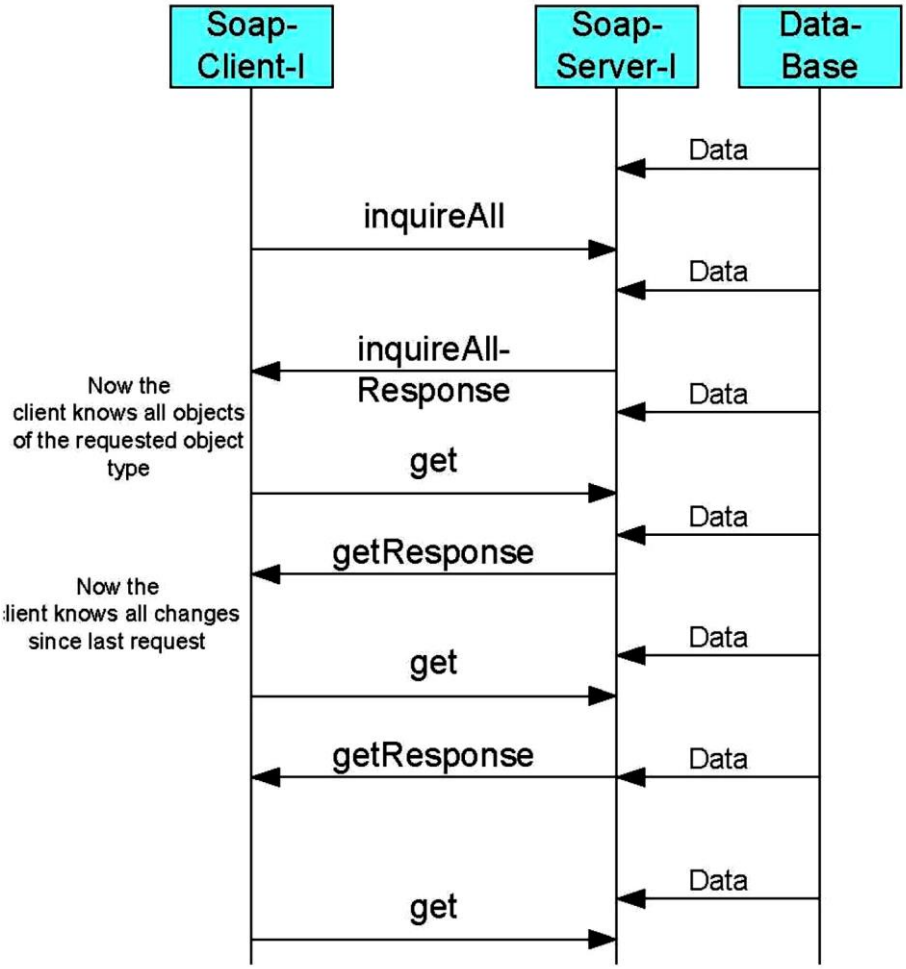


Figure 1: Common sequence for reading data from the SOAPserverinterface

2.2.2. Sending data to the server

It is possible to send data from the client to the server. The "put" protocol function is used to do this. The server's behaviour depends on the object type. In the case of unknown objects in the "put" command, either the object is generated or a malfunction is returned. To delete objects, use the "delete" command.

The appropriate data are sent to the server using the "put" command for the configuration of the interface. The servers accepts them or rejects all non-configurable objects and places them on the putResultlist.

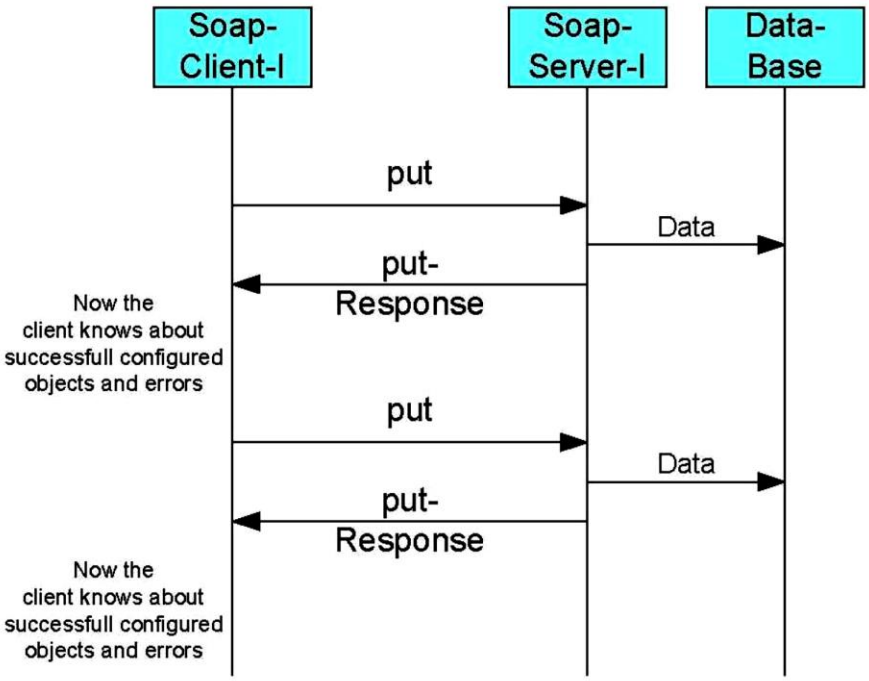


Figure 2: Common sequence for writing or configuring data to the SOAPserverInterface

2.3. Sequence control

The protocol does not have a connection. For sequence control, it is sufficient to wait for the response to a request. This is governed by the http protocol. Additional sequence control is not required. The suitable sequences are described in the following chapters.

2.3.1. Data buffering and position handling

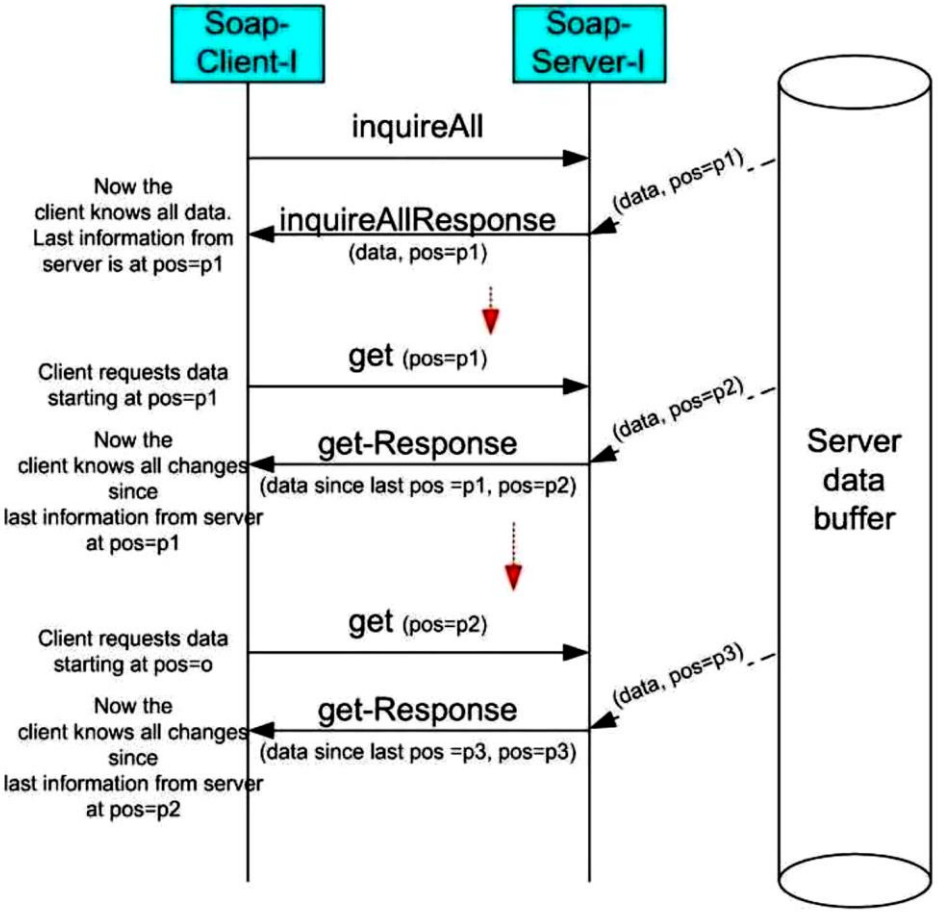


Figure 3: Data buffering on the server and position handling on the client

The server stores the data requested by the client from its database in a ring buffer. After an inquireAll has been carried out, the client knows the last status from the database. Other than the inquireAll response, the client receives the last position number and shows the latest information from the server (indicator to the latest received data in the data buffer). Using this position number, the client generates the next request (get). Through the position number, the server knows which data must be provided in order to deliver all changes that have occurred. The response received includes the desired data and a new position number that will be used for the following request.

If a client wishes to request more than one type of object, it needs to use multiple requests.

Each request sequence must perform its own position handling in order to receive delta values relating to the object type. This also applies in the case of using different filter lists and when using parameters with the optional XML element get→data.

2.3.2. Transaction time too long

Normally a response contains the requested data. If the server requires more time than permitted (> 60 sec), an empty response is returned to the client. The exceeded transaction time is displayed by an error code. Nevertheless, the server continues to retrieve requested data from its database (or from the archive). The client repeats its request after a certain period of time. After this period of time, the server should be able to deliver the data. If the server is still not able to deliver, an error code is displayed once again.

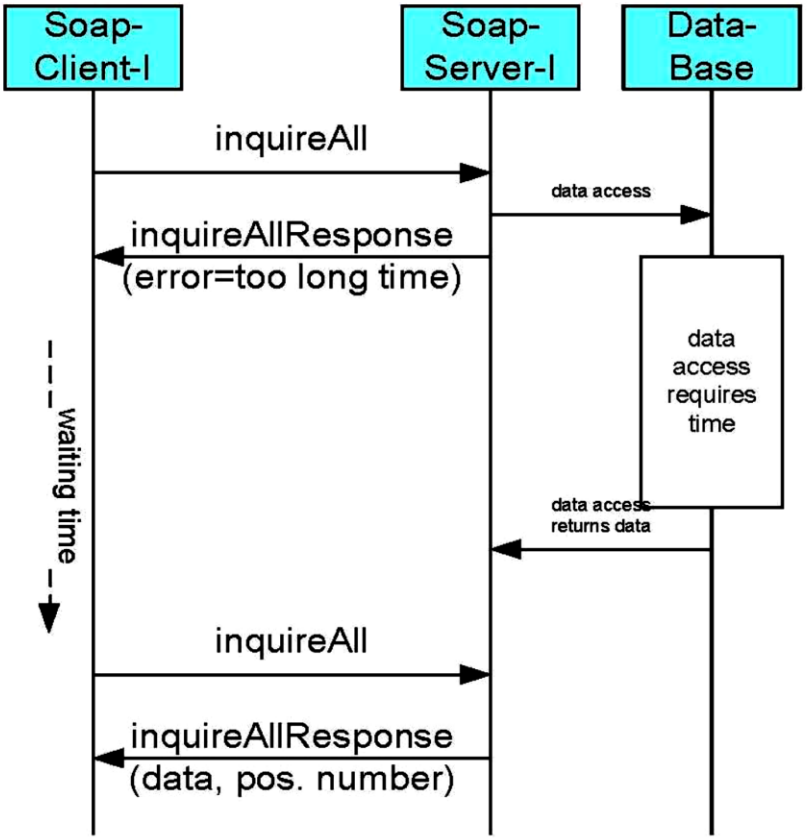


Figure 4: Dealing with long transaction times (similar sequence also applies for "get")

2.3.3. Requests too long

The server returns an error code, if the content of a response exceeds a certain threshold. This may happen, if

- too much historical data is requested (time range too large), or
- too many objects are requested.

The client must narrow its request by reducing the number of objects or the request time range accordingly.

2.3.4. Too many changes

If too many changes have been made since the last request, the request is satisfied as far as possible, by returning not all of the objects or not the entire time range.

The incomplete return is signalled by the errorCode (in protocol.xsd) (errorCode: missingDatasets)

2.3.5. Dealing with requests for historical data

- The "get" command is used, if access to the historical data is requested.
- The "storetime" element is used to define the start time of the saved historical data.
- The "endStore" element is used to define the end time of the saved historical data.
- The server response contains the status changes (no initial value) from "storetime" to "endStore", provided that the definition of the request interval is not too wide.
- Should the definition of the request interval be too wide, the request is satisfied as far as possible, by returning not all of the objects or not the entire time range. The incomplete return is signalled by the errorCode (in protocol.xsd) (errorCode: missingDatasets)
- If storetime is the same as endStore, the status at this time is sent. The value then receives as the time stamp either the original time stamp of the value exactly, should this not be present, the initial time stamp is not executed (the time stamp in protocol.xsd is an optional element).
  If the initial value cannot be delivered, or its delivery doesn't make any sense (e.g. with the "operatingMessages" object type), then no value is delivered.

It is recommended:

- To select as short a time period as possible
- To use filters and with them reduce the quantity of objects requested.

2.3.6. Multi client capability

The server operates without a connection. Due to the position number, it is not necessary to build up knowledge about the client on the server side.

When implementing OCIT-C, it must be ensured that the libraries used in the lower communication layers allow multi client access.

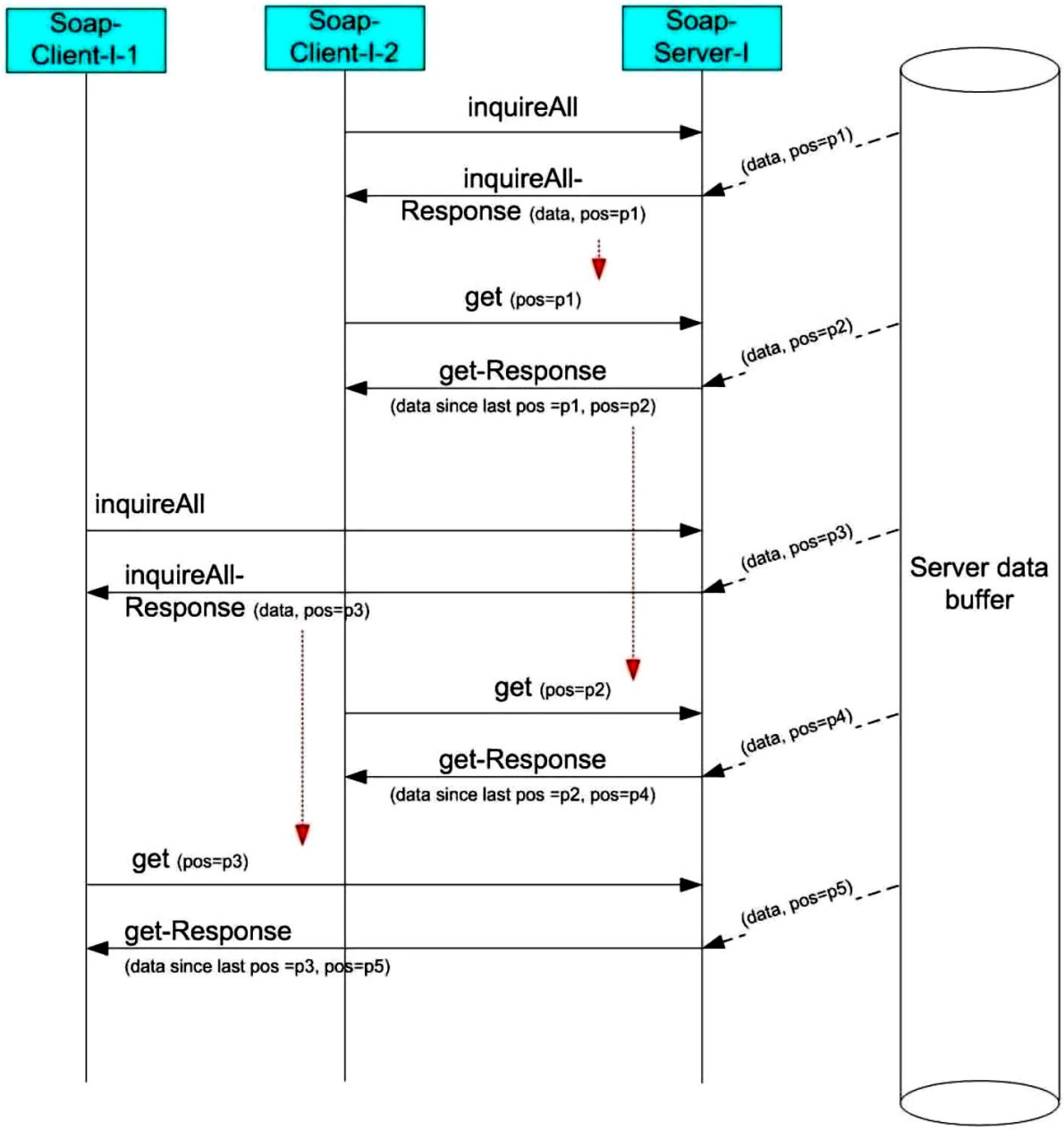


Figure 5: Multi client capability

2.3.7. Resynchronisation

There are various reasons for a resynchronisation:

- Restart of the client
  - The client recognises this and therefore resynchronises using "inquireAll".
  - The server is possibly able to detect the restart of the client using a watchdog which can be optionally implemented.
- Restarting the server
  - The client possibly detects that the server cannot be reached via "socket timeout".

- The server responds to each possible protocol function (inquireAll, get or put), including lastStart, that displays when the server was last restarted.
- The client must resynchronise using inquireAll, if the lastStart does not differentiate from previous entries.

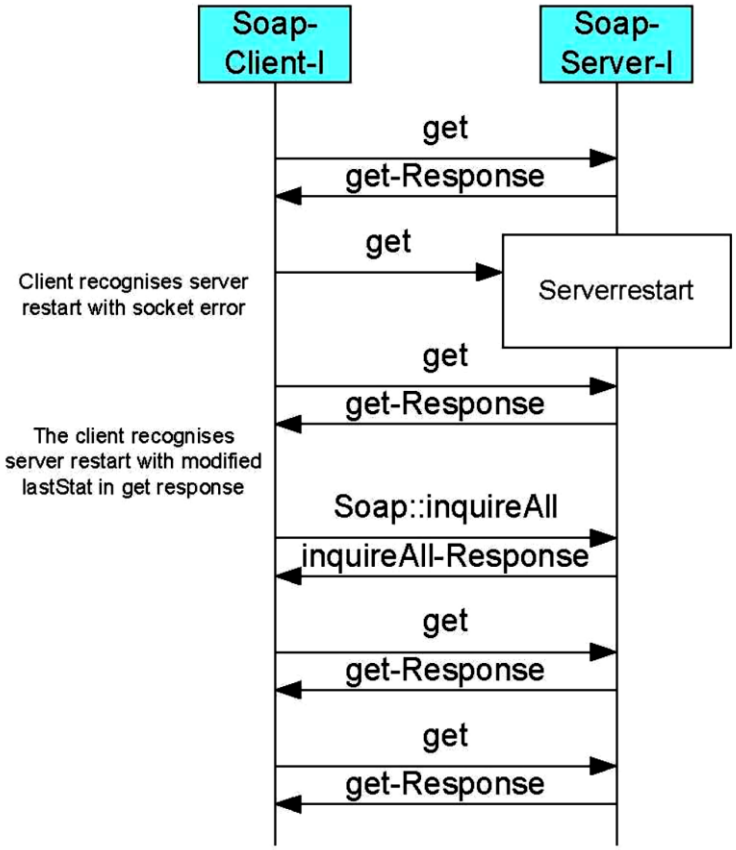


Figure 6: Restarting the server

2.3.8. Bi-directional communication

The following applications require bi-directional communication:

- TSS control or sign control:
  - Switching command is transferred from the control center to the device server or switching status is asynchronously transferred from the device server to the control center
- CCTV
  - PTZ (pan/til/zoom) command is transferred from the control center to the device server
  - - Current status is asynchronously transferred from the device server to the control center

The following chapters describe possible configurations. As an example, the "Sign control" is used.

2.3.8.1. Bi-directional communication with client and server pair

Switching a sign (process sequence is successful)

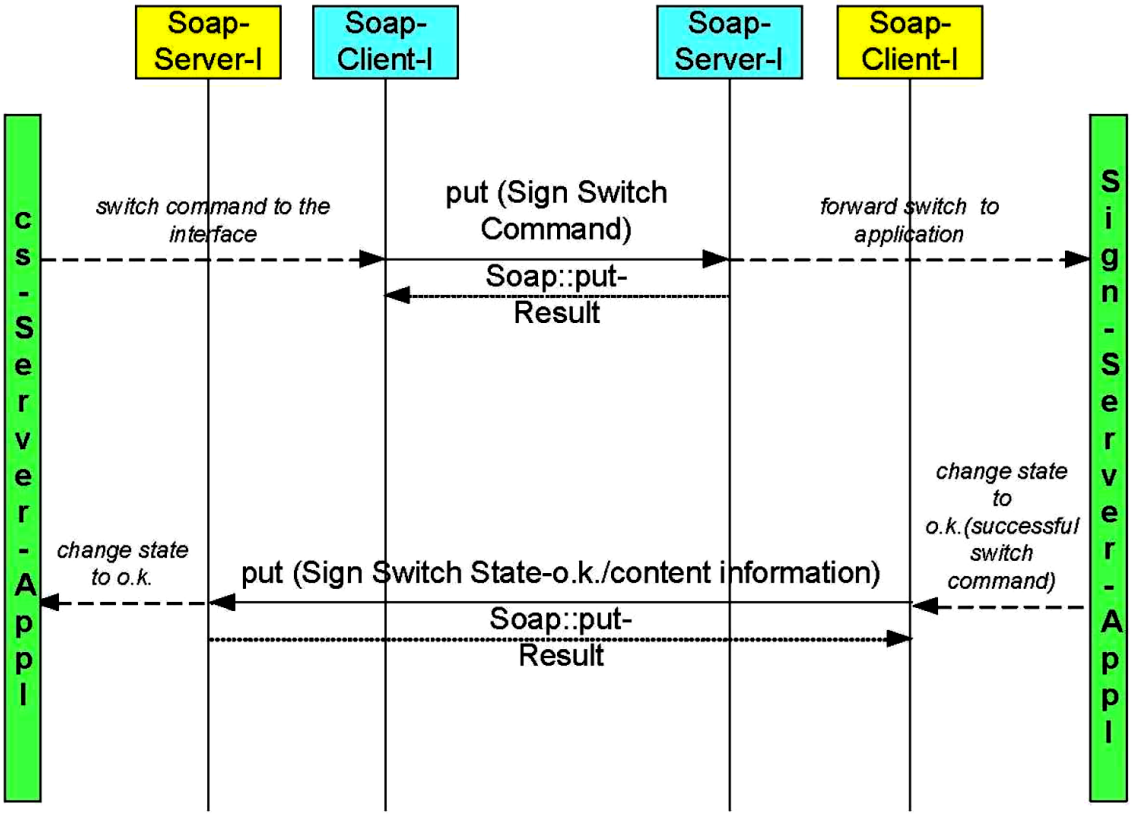


Figure 7: Switching a sign (process sequence is successful)

Switching a sign (process sequence is unsuccessful)

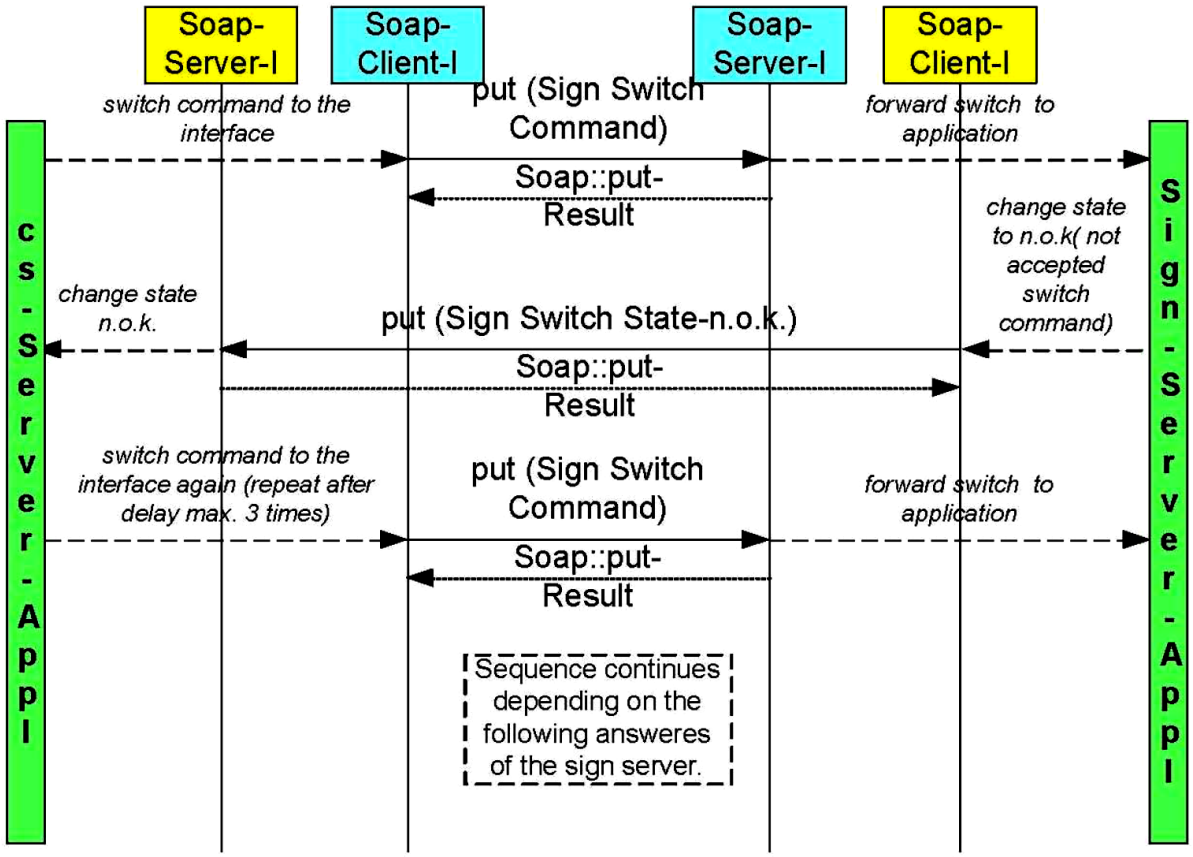


Figure 8: Switching a sign (process sequence is unsuccessful)

In the case of a missing response, the switch commands are repeated after a configurable delay. This applies regardless of where the fault occurs (there is no status transition to the "busy" or "o.k." status). The repetition is stopped after three unsuccessful attempts. The current status on the side of the central system then changes to n.o.k.

Status, content and connection monitoring

central system sign

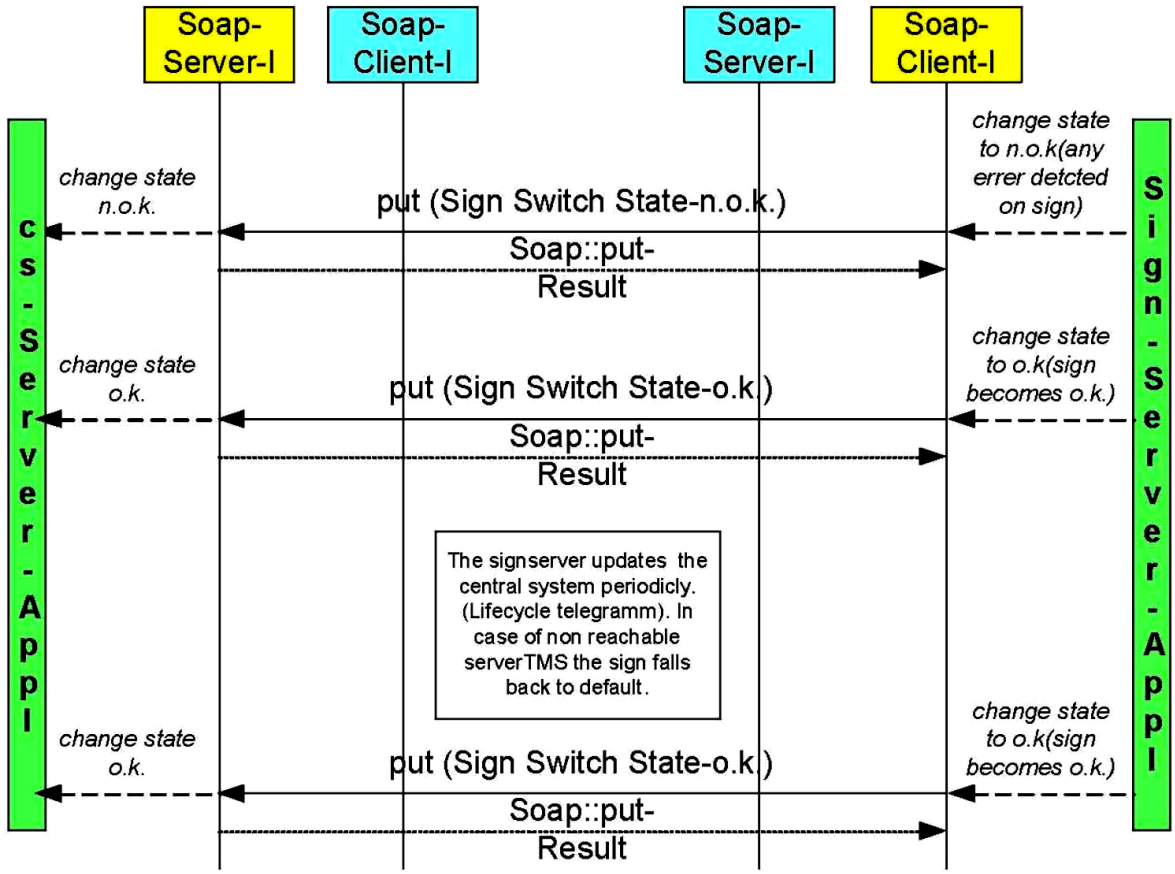


Figure 9: Switching status

The sign server updates the central system periodically, even if the switching status has not changed. Subsequently, the central system receives notification:

- if a sign, which is connected to the sign server, changes its status
- if a sign changes its content
- if the sign server cannot be reached (the max. waiting time is specified by the central system)

If the central system cannot be reached, the sign server sets the signs to a pre-defined display status.

2.3.8.2. Bi-directional communication with regular polling

The statuses are evaluated in the same manner as in chapter 2.3.8.1.

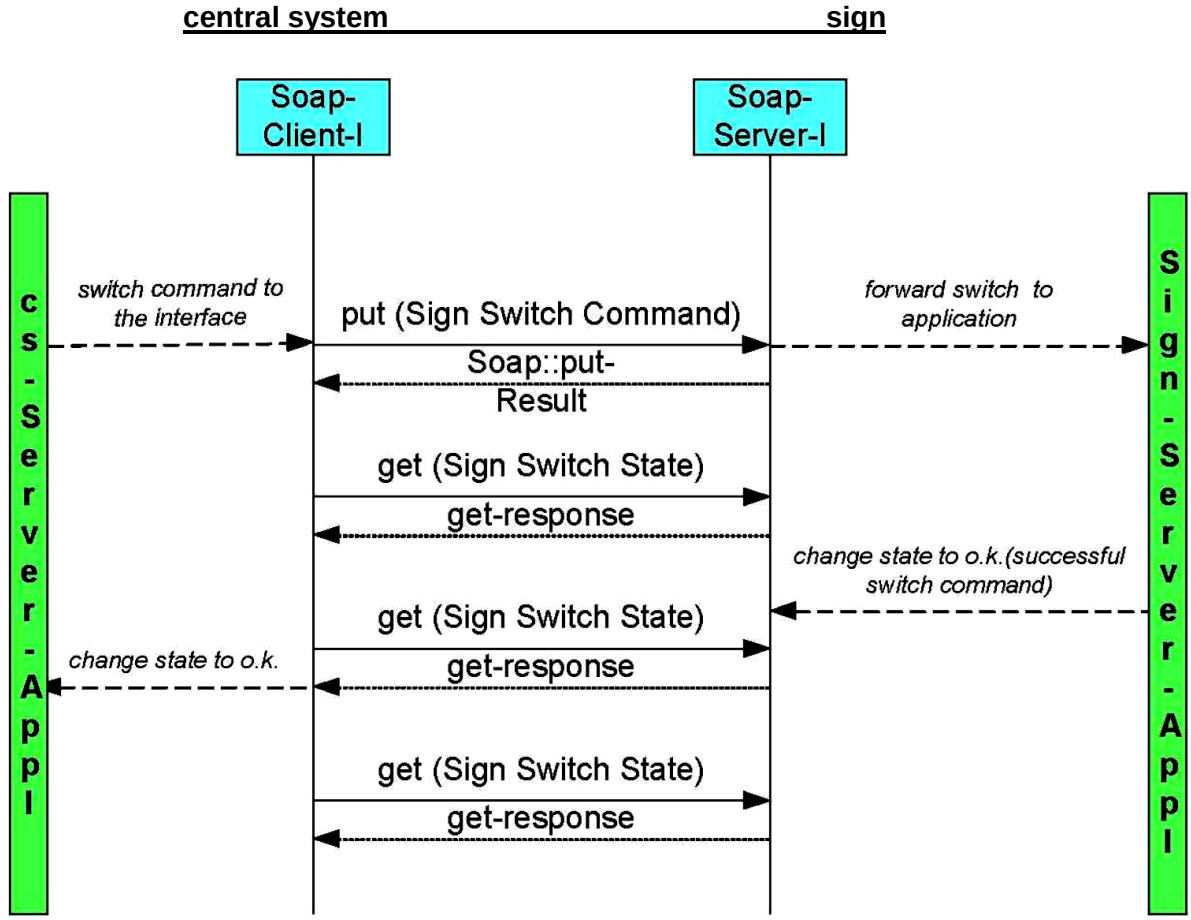


Figure 10: Regular status requests

2.3.9. Avoiding sampling delays

To prevent inadvertently large sampling delays through cyclic requests from the client to the server, a new function is introduced in the protocol. This function works in accordance with the following principle:

- Structure like the "get" function

  (i.e. same parameters, only different function name)

  Name of the function "wait4Get"

- The new function wait4Get operates like the "get" function with the difference, that in the event there is no data present on the server (i.e. the return list were to be empty), the response from the wait4Get is delayed until either a timeout occurs or there is data available on the server.

- The client would therefore invoke the wait4Get function once again immediately after receiving a response, in order to indirectly keep the return channel permanently open.

- To prevent multiple requests within a second, the server can adopt protective measures. A possible protective measure would be to allow one response per second and to return the values accumulated in this second in the response.

- The server is able to restrict the number of clients or object types / objects, which are requested in this manner.

- To prevent too many sockets being opened, the wait4Get function enables different object types to be requested simultaneously. This would be considered accordingly in the data structures of the function.

The following sequence diagram illustrates the procedure (simplified for the request of an object type:

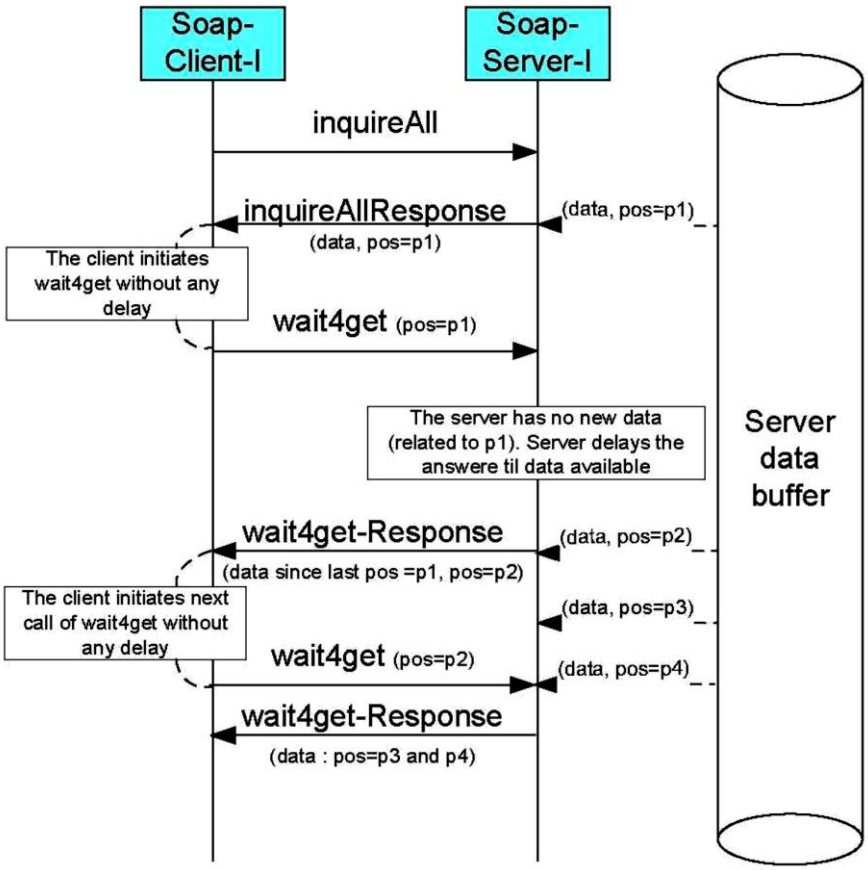


Figure 11: Communication layers client and server

2.4. OSI - Layers

Server and client have different sections with different functions according to the OSI model.

The lowest protocol layer is the http protocol, which is responsible for transmitting data in the network.

Above that is the client or server version of the SOAP protocol.

The protocol manager has the task of providing all commands including the required data buffer for the server functionality.

The application layer establishes the connection to the database.

The following figure describes this layer model.

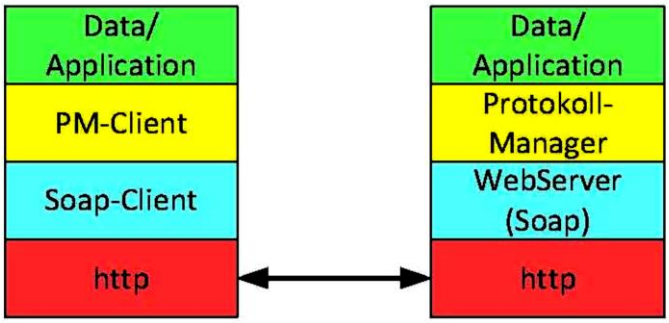


Figure 12: Communication layers client and server

2.5. Protocol functions in detail

Available methods:

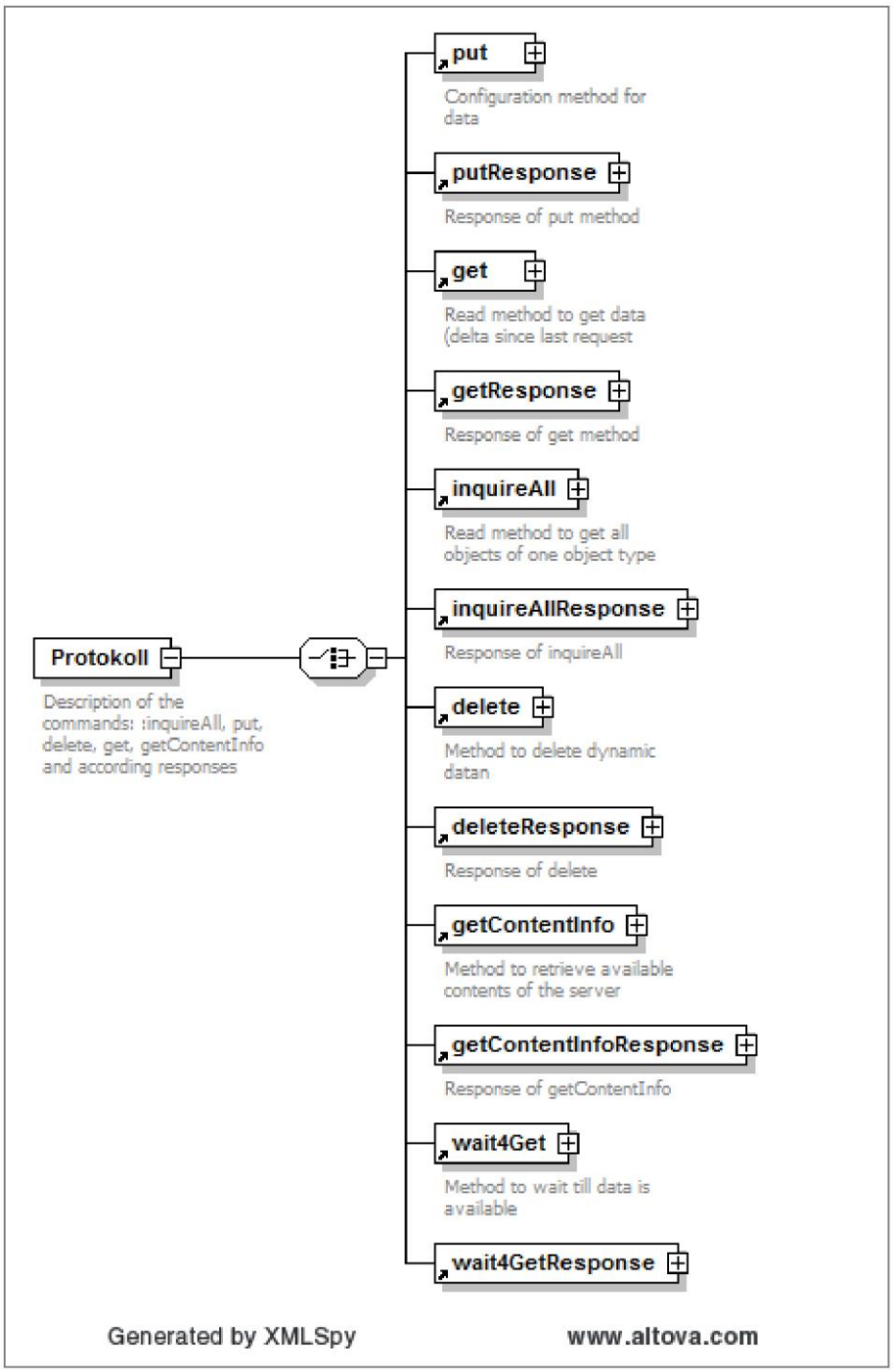


Figure 13: Available methods

2.5.1. Standard parameters

The standard parameters of the methods are:

Input parameters

- **UserName** and **UserPasswd** authenticate the user UserName and UserPasswd are transferred as normal text. This authentication should therefore not be used for high levels of security requirements.

- **watchdog** is a structure used by the client to inform the server when the next call can be expected. This can be used by the server for timeout monitoring of the client.

- **storetime** indicates the start of the requested or sent data. This method is only used when accessing data saved historical data.

- **endStore** indicates the end of the requested data. This method is only used when accessing data saved historical data.

- **position** indicates the position of the indicator in the server buffer.
  The position is received using the methods inquireAll and getResponse and used for the next get request (see chapter 2.3.1, Data buffering and position handling).

- **filterList** is a list of objects which should be read.
  These can be used to limit the quantity of data.

Output parameters

- **lastStart** is the time stamp of the last server restart. If the lastStart changes from one server response to the next, this is a sign that the data must be resynchronised. The client resynchronises because of this using the inquireAll method.

- **errorCode** is an error code which is generated in the case of faulty commands. In the event of faulty XML structures, it is not used. In addition, there is the message from the SOAP protocol (fault).

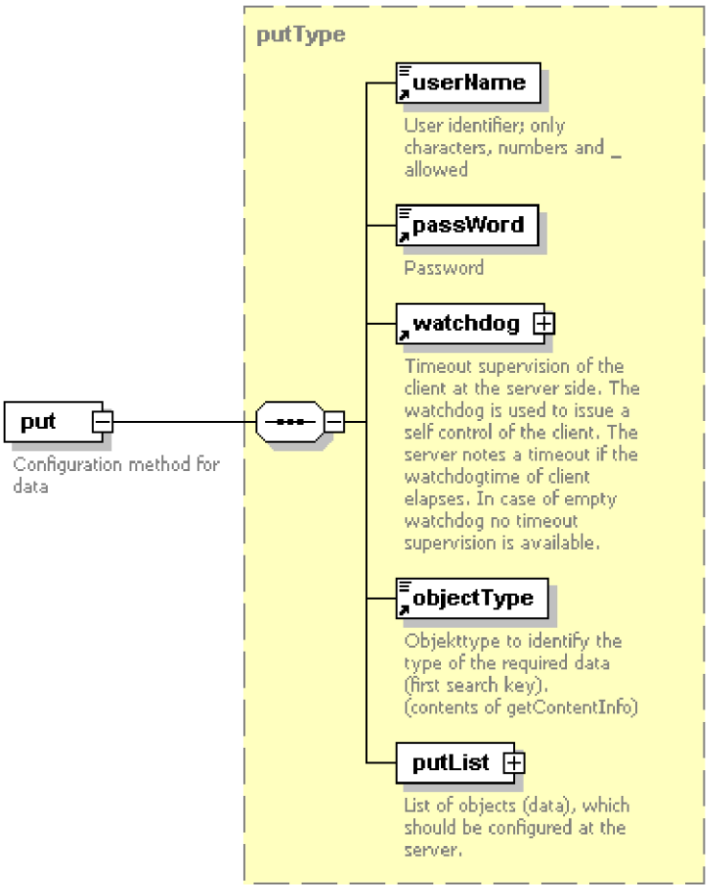
- **errorText** is a description of the errorCode than can be read by humans.

- **position** is the position of the last data access. It must be used for the subsequent data access.

- **dataList** is a list including the requested data.

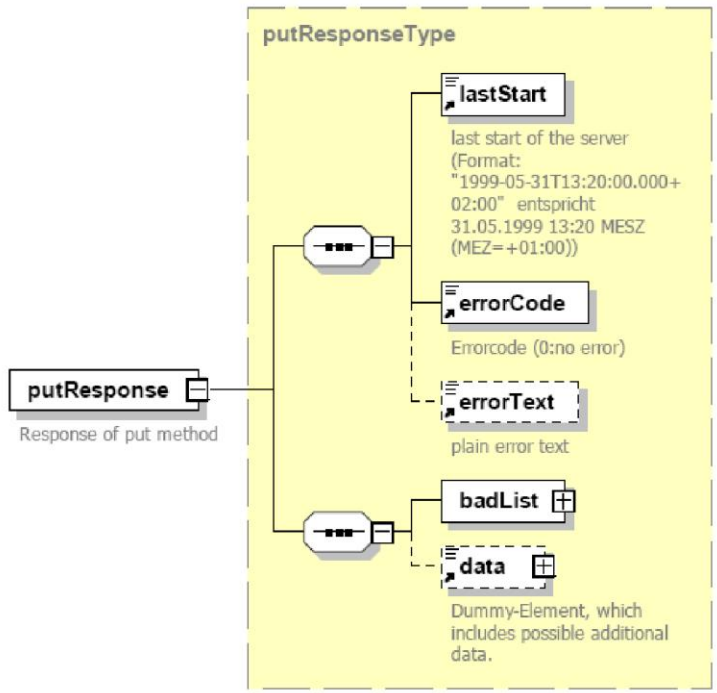
2.5.2. put

The "put" method is used to configure objects. It contains all instances of the data, which should be configured.



putResponse is the response to "put".

It contains all non-configurable data, usually none.

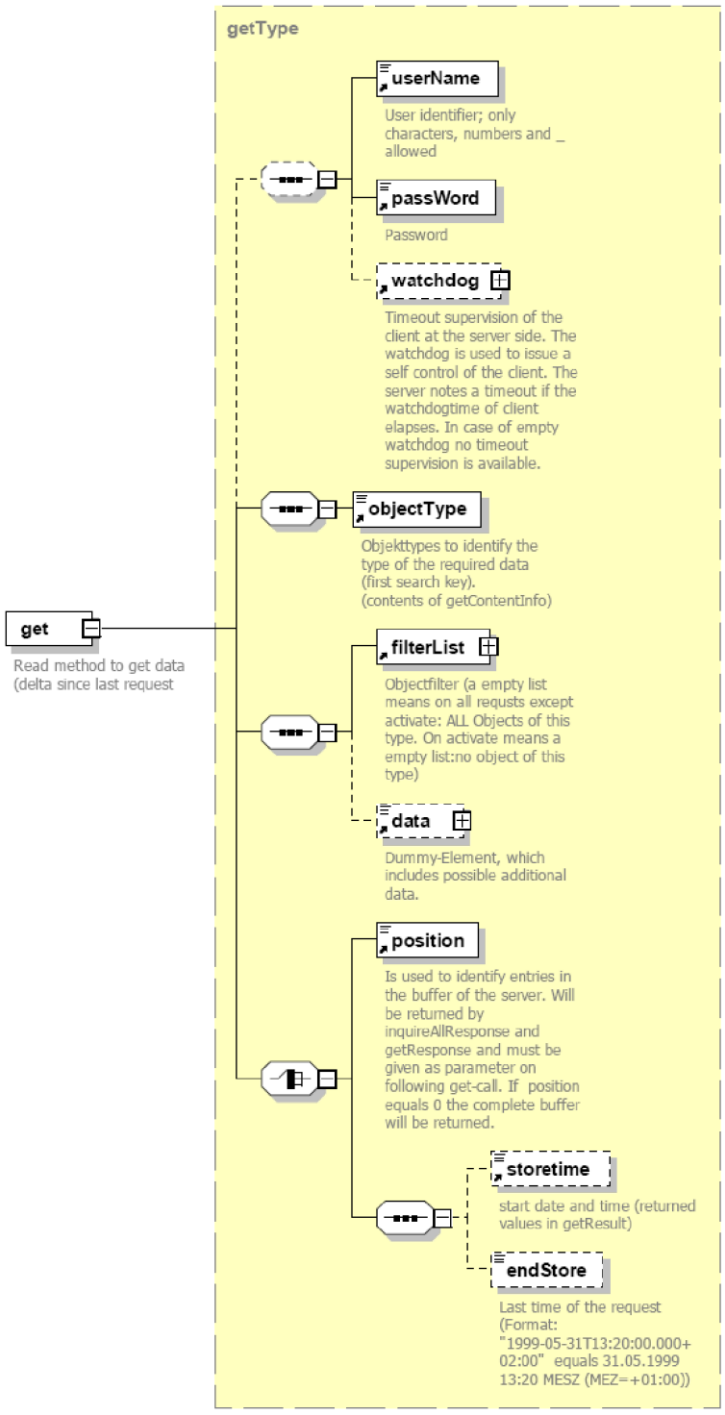


2.5.3. get

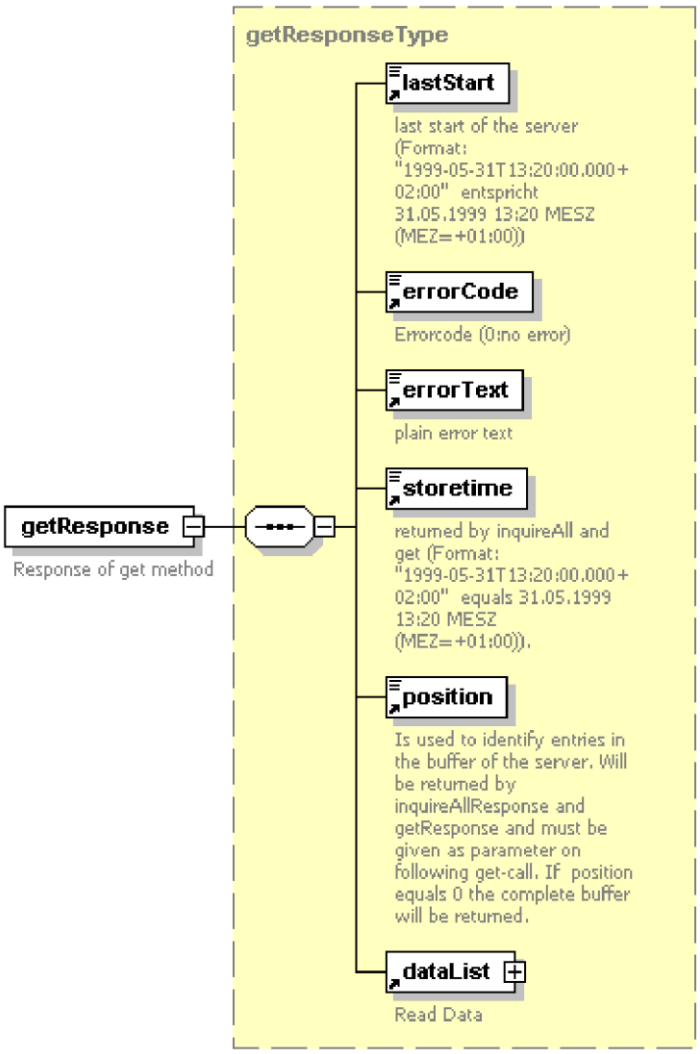
The "get" method is used to request data.

In addition to the standard parameters, it has

- either the start or end time, in order to receive all values within this time span
- or the position number of the data requested. Normally this is the position number returned by the last inquireAllResponse or getResponse method.



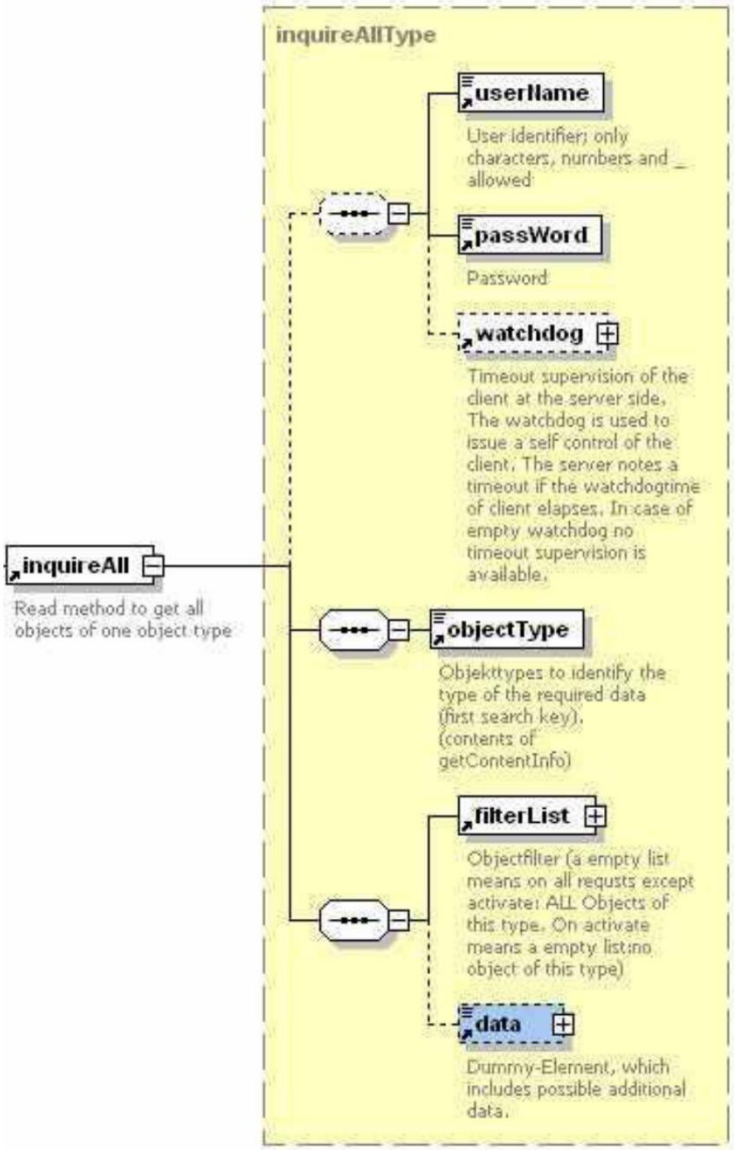
getResponse is the response to "get"



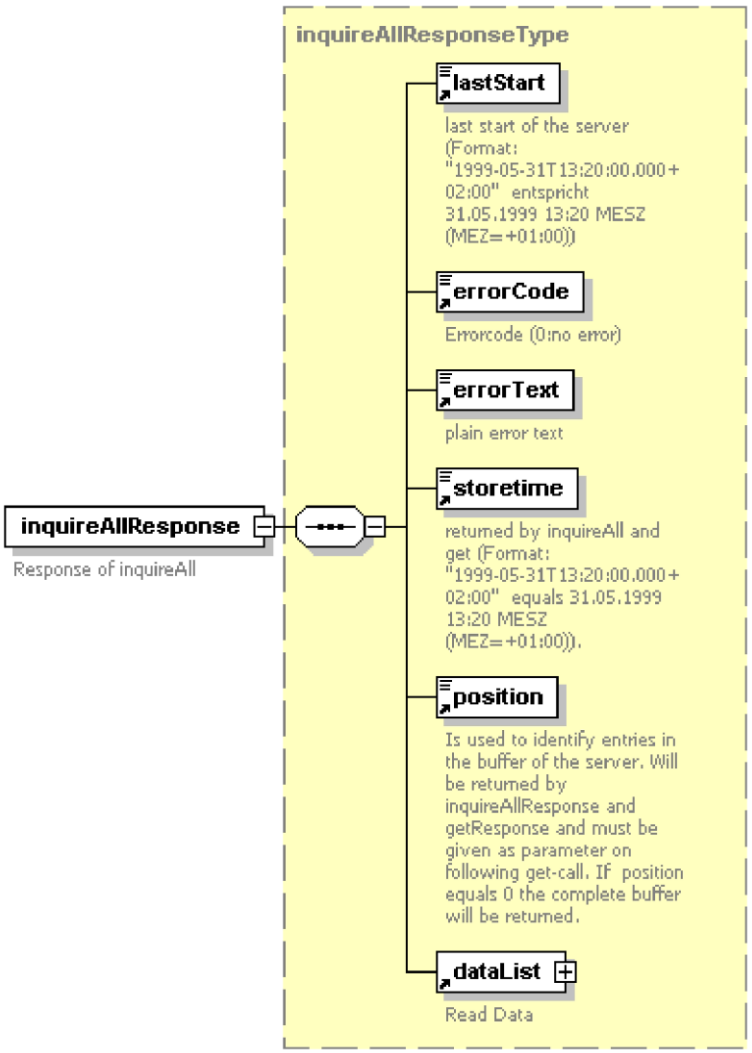
2.5.4. inquireAll

Method for requesting all objects of an object type with the latest status or object content. This method guarantees the client that the response contains all the requested objects.

The inquireAll method only contains standard parameters.

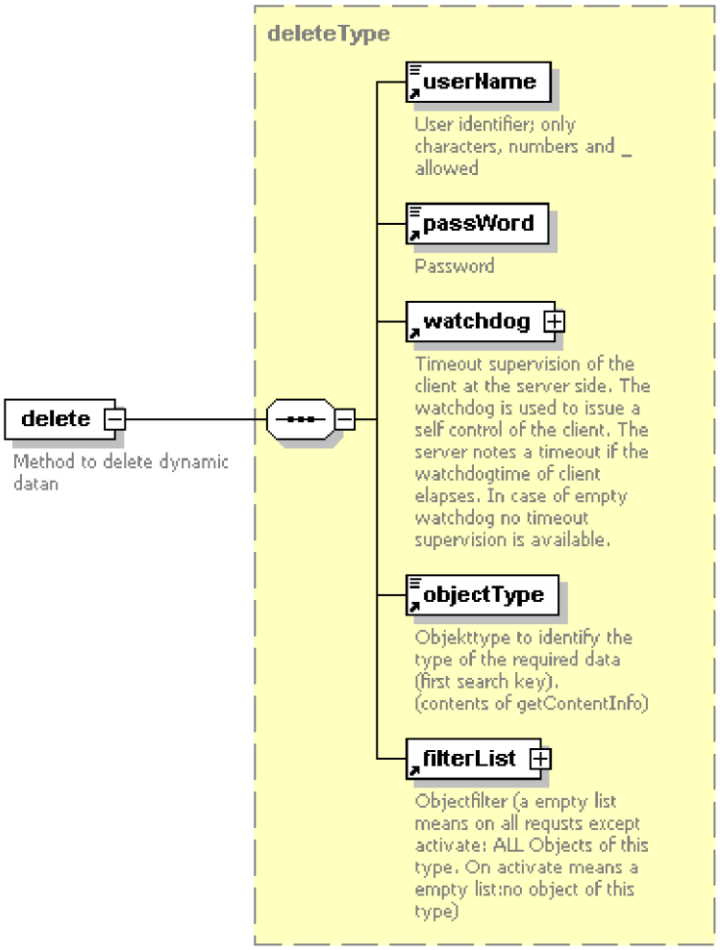


inquireAllResponse is the response with all requested contents.

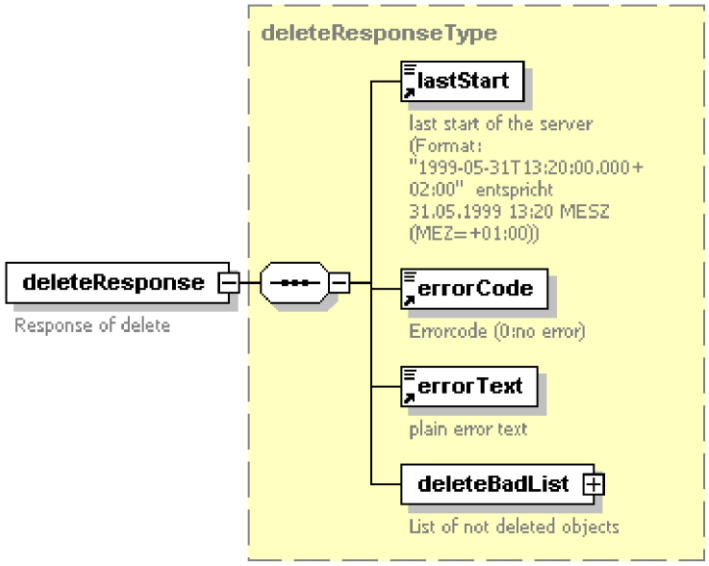


2.5.5. delete

The "delete" method is used to delete dynamic data (where permitted). Instances of the data to be deleted must be entered into the filter list.

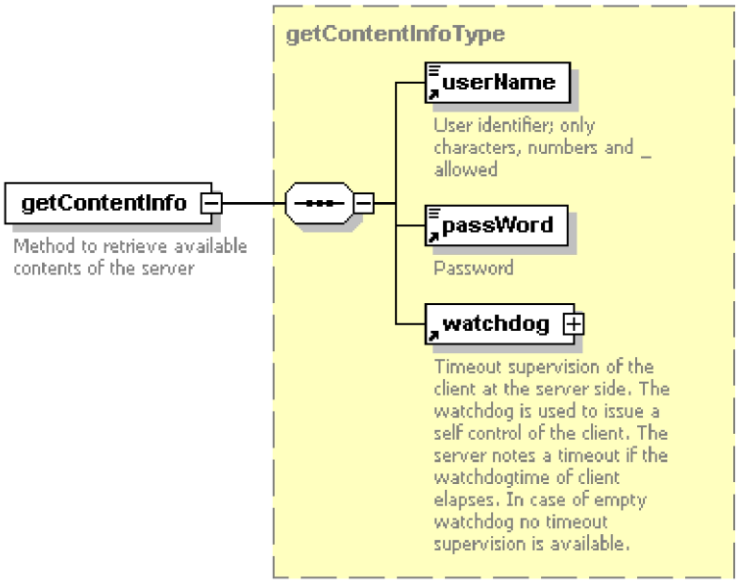


deleteResponse is the response to "delete". It contains the instances of the data, which cannot be deleted.

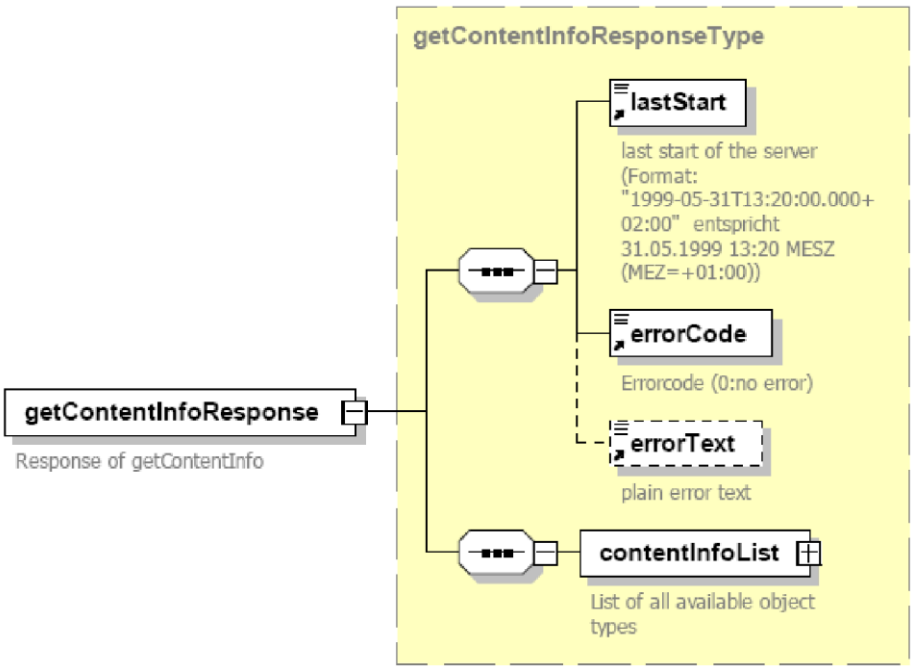


2.5.6. getContentInfo

Method for requesting the object contents of the server. The method's parameters are just the name of the client, the password and optionally the time for the "watchdog".

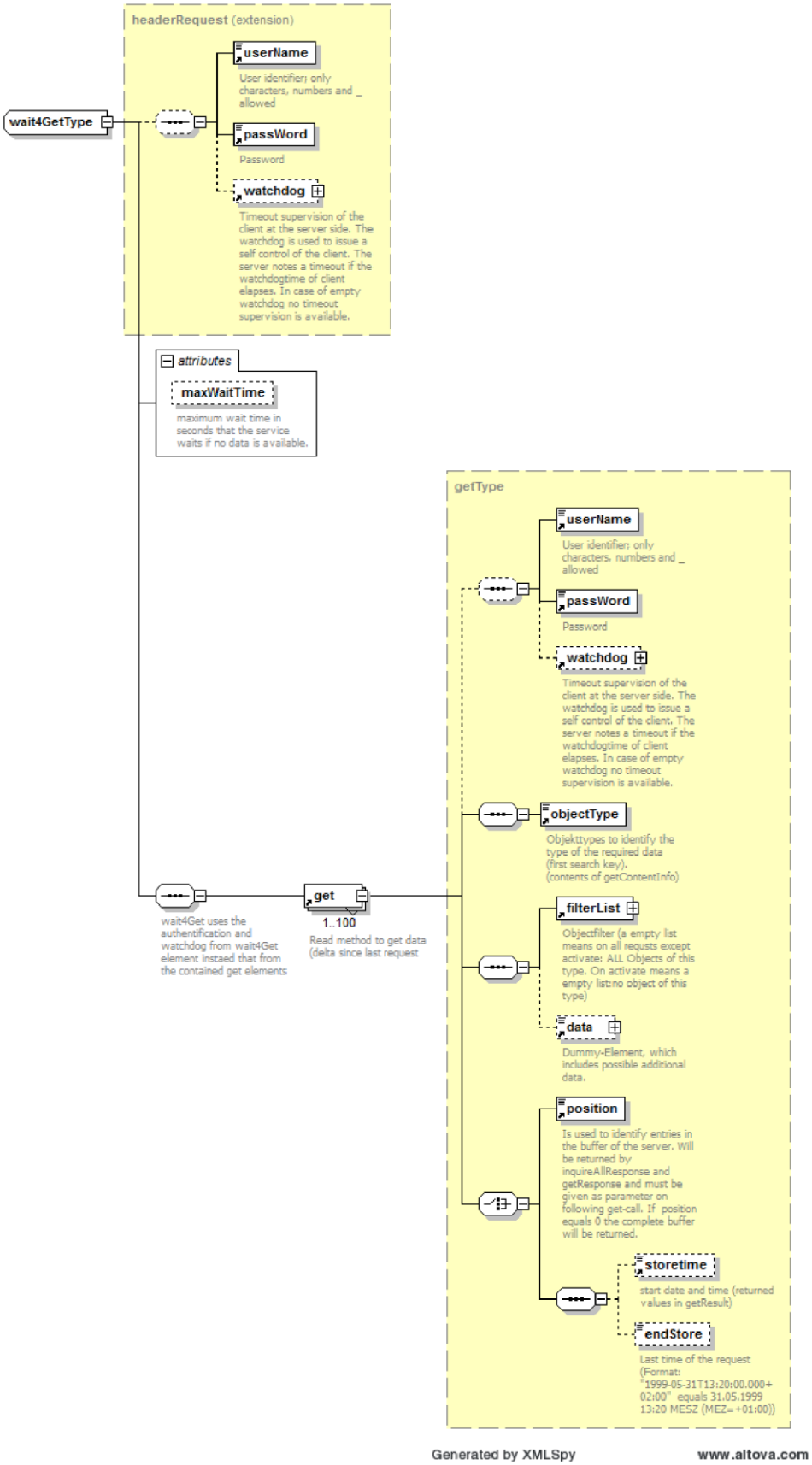


The response getContentInfoResponse contains a list of available object types with their access rights and recommended update cycles.

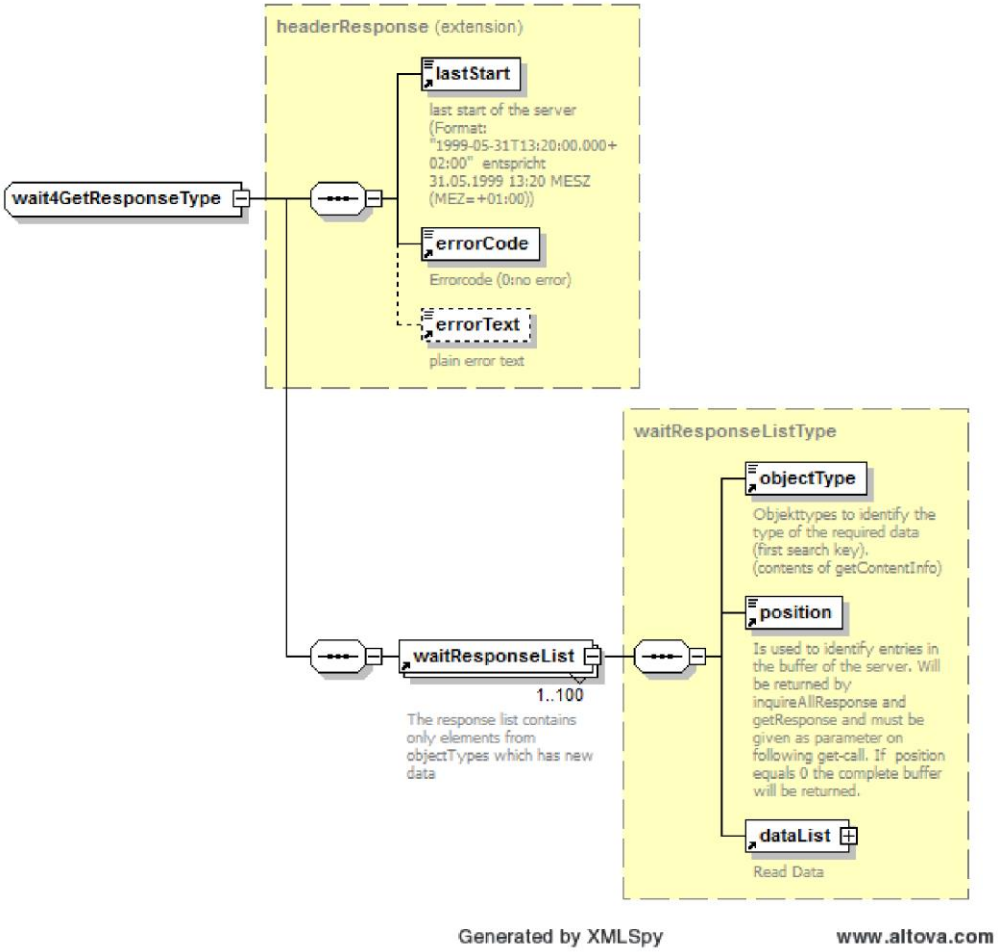


2.5.7. wait4Get

Method for requesting the server data. The method's parameters are the same parameters as a "get", however it enables multiple object types to be requested simultaneously.



Wait4GetResponse is the response with the changes since the last request.



2.6. Data structures

Data structures that are inserted into the protocol, i.e. the methods, put, inquireAll - response, etc, are defined in separate schema files of the protocol.

2.7. Definition of errorCodes

The following errorCodes are defined in protocol.xsd:

errorCode	Meaning
0	no error
1	access error
2	buffer overflow
10	requested data unavailable
11	requested data cannot be sent
12	requested data cannot be deleted
13	values cannot be set
14	found empty object type
15	object type not found
16	error writing data
17	error creating data
18	error deleting data
19	missing filter for deletions
20	server shortly unavailable
21	missing parameters to execute the method
22	internal error
23	other registered accessing client
30	one file cannot be accessed
31	error opening a file
32	error reading a file
33	internal error, reading the archive
34	internal error, parsing the archive
35	error, parsing the archive
36	error activating
37	error deactivating
38	error reading data
39	object not found
40	invalid time range.
41	time range complete (no error)
42	missing data sets
43	returned time range incomplete

2.8. Suggested applications

This chapter describes how to manage servers and clients in various applications. This are just recommendations, which may differ on a project-specific basis.

2.8.1. Data delivery in the case of multiple recipients

In cases where there are multiple recipients for the data from the server, which merely request data and don't require a real-time connection, the architecture described below is recommended.

- The data source contains the so-called SOAPServerInterface, which contains functionalities for buffering data.
  The data sink contains the so-called SOAPClientInterface, which enables access to the SOAP server (inquireAll and get methods).

This makes it that the client only access the server upon request. Furthermore, the protocol can be implemented with little effort. In the case of an internet connection, the server is the data source and the client the web service.

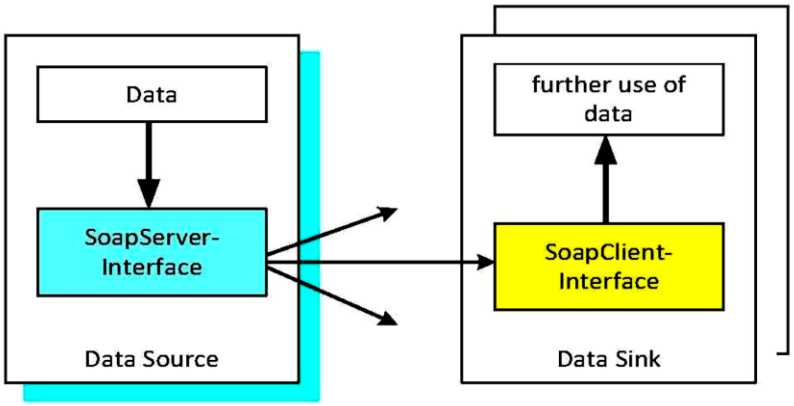


Figure 14: Data delivery to multiple recipients

2.8.2. Configuration interface

If a system (as a data source) requires a configuration interface on the data sink, the following architecture is recommended:

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

Properties:

- Data transfer only on demand
- Real-time configuration, no polling required
- Multiple configuration interfaces with standardised communication interface to a data sink possible.

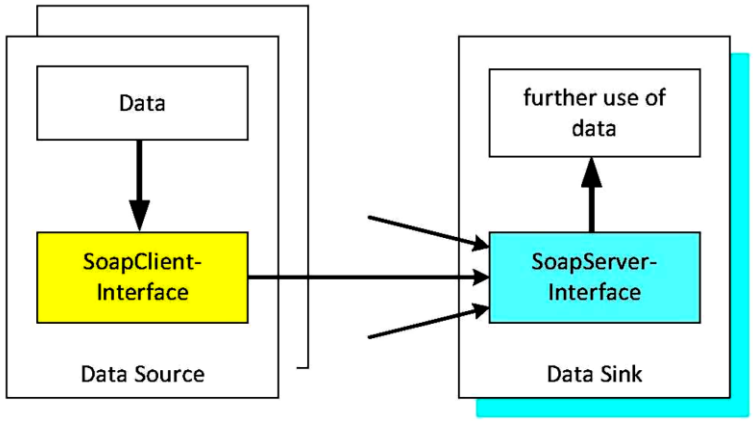


Figure 15: Multiple configuration interfaces

An application for this are subsystems (as the client), which immediately forward their collected data for example data about traffic faults. This configuration should only be used to prevent overload situations.

2.8.3. Data update between central facilities (unidirectional)

In this case, the following is recommended:

- Data source is the server
- Data sink is the client

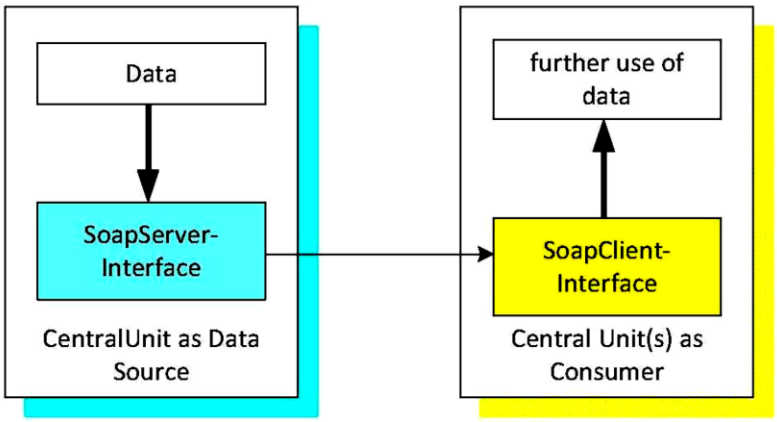


Figure 16: Configuration when delivering data to the client

Practically, this configuration is the same as for data delivery in the case of multiple recipients.

2.8.4. Data update between central facilities (bi-directional)

If a bi-directional interface is required, the interface can be realised twice, since the central system is the client and the server simultaneously.

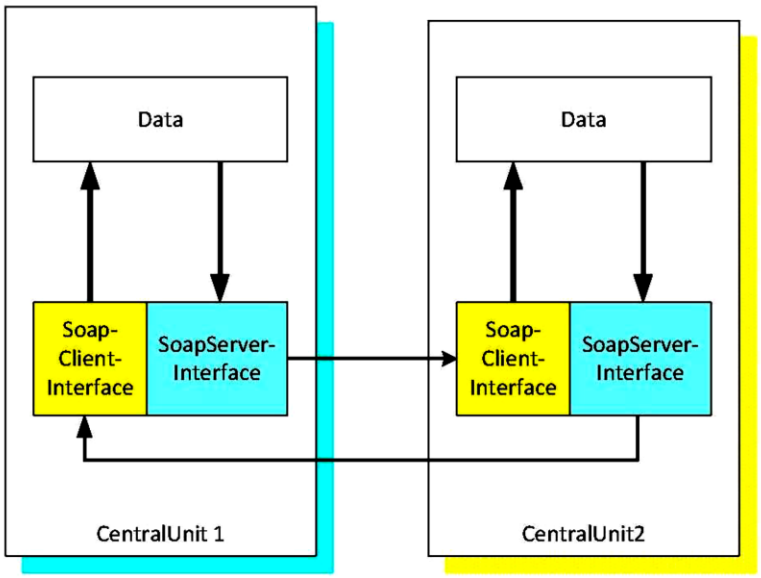


Figure 17: Configuration when exchanging data server – server

OCIT-C_Protocol_V1.2_R1