Open Communication Interface for Road Traffic Control Systems

Offene Schnittstellen für die Straßenverkehrstechnik

OCIT-C Center to Center

Transport Protokoll

OCIT-C_Protokoll_V1.2_R1

OCIT Developer Group (ODG)&Partner

OCIT-C Center to Center

Transport Protokoll

Document: OCIT-C_Protokoll_V1.2_R1

Herausgeber: ODG & Partner

Kontakt: www.ocit.org

Inhalt

Dokumentenstand

Version Zustand	Datum	Verteiler	Kommentar
V1.1_R1	**30.10.2014**	**PUBLIC**	**Version 1.1 Ausgabe 1**
			2.2.1: inquireAll Zustand der Objekte in der Vergangenheit (Textergänzung) 2.3.5: Textkorrektur 2.3.4: Neue Textfassung 2.3.5 Textänderung 2.3.8.1 Korrektur der Bilder 1 bis 17
V1.2_R1	**16.12.2016**	**PUBLIC**	**Version 1.2 Ausgabe 1**

1 Einführung

OCIT-C steht für Open Communication Interface for Road Traffic Control Systems – Center to Center. Mit OCIT-C werden die Funktionen zur Kommunikation zwischen zentralen Verkehrssteuerungs- und Verkehrslenkungssystemen abgedeckt:

- Verkehrsrechnerzentralen und Verkehrsmanagementzentralen (kommunal, regional, überregional)
- Verkehrsingenieurarbeitsplatz mit Verkehrsrechnerzentralen
- Parkleitsysteme, Parkhaussysteme
- Baustellenmanagementsysteme
- Lokale Internetanwender (städtische Info im Internet)

Die Definition und Pflege der Schnittstelle OCIT-C wird von der ODG und ihren Partnern durchgeführt.

Mit OCIT-C steht ein Standard zur Verfügung, der OCIT-O passgenau ergänzt. Mit OCIT-C und OCIT-O für die Kommunikation von Zentralen zu Feldgeräten werden alle Anforderungen von der Verkehrssteuerung bis hin zum übergeordneten Verkehrsmanagement abgedeckt.

OCIT-C orientiert sich konsequent an den praktischen Anforderungen. Durch niedrige Implementierungskosten ist der Einsatz auch für Lösungen mit schmalen Budgets geeignet.

Charakteristische Eigenschaften von OCIT-C sind:

- Auf dem Standard SOAP basierendes Austauschprotokoll mit einfachem Request-Response-Kommunikationsmuster (direktes Abfragen von Daten).
- Definition eines umfassenden, alle Teilbereiche der Verkehrssteuerung und Verkehrslenkung enthaltenen Datenmodells im Prozessdatenbereich.
- Systemintegration und gewünschte Anpassungen werden vorab über Projektierung geregelt.
- Konformitätstests des Protokolls werden in einer über www.ocit.org bereitgestellten Testumgebung durchgeführt. Tests von gesamten Implementierungen (Protokoll und Dateninhalte) werden projektbezogen durchgeführt.
- Erweiterungen um DATEX II Bestandteile sind je nach Projektanforderungen möglich.

Die Kommunikationsschnittstelle soll in allen zentralen Einheiten in gleicher Weise implementiert werden. Dazu wird das SOAP-Protokoll als übergeordnete Kommunikationsschnittstelle, über die die gesamte Kommunikation erfolgt, verwendet. Die hier beschriebene Ausprägung wird als OCIT-C Protokoll bezeichnet.

Diese Schnittstelle ist offen und kann in diversen Systemen, vorwiegend im Bereich der Straßenverkehrstechnik, eingesetzt werden. Die Aufgabe dieses Dokuments ist es, das OCIT-C Protokoll und seine Verwendung zu beschreiben. Es ist nicht die Aufgabe dieses Dokuments, die Datenstrukturen der zu übertragenden Daten zu beschreiben. Diese sind im Dokument „OCIT-C Daten“ beschrieben.

1.1 Begriffe und Abkürzungen

Begriff / Abkürzung	Beschreibung
AP	Anwenderprogramm
Client	Ein Programm, das von anderen (Servern) angebotene Dienste in Anspruch nehmen will und diese dazu aktiv aufruft.
DATEX II	Spezifikationen des Technischen Komitees 278 des Europäischen Komitees für Normung (CEN) zum Austausch von Verkehrsinformationen zwischen Verkehrszentralen.
FTP	File Transfer Protocol, ein Netzwerkprotokoll zur Dateiübertragung
http	Hypertext Transfer Protocol, ein Protokoll zur Übertragung von Daten über ein Netzwerk.
LSA	Lichtsignalanlage
Methode	Die einer Klasse von Objekten zugeordneten Algorithmen. Auch synonym gebraucht als Funktion, Prozedur, Befehl, Aktion.
ÖV	Öffentlicher Nahverkehr
OCIT	Open Communication Interface for Road Traffic Control Systems / Offene Schnittstellen für die Straßenverkehrstechnik.
OCIT-C	Open Communication Interface for Road Traffic Control Systems – Center to Center. Mit OCIT-C werden die Funktionen zur Kommunikation zwischen zentralen Verkehrssteuerungs- und -Verkehrslenkungssystemen abgedeckt.
OCIT-O	OCIT-Outstations Schnittstelle zwischen Verkehrssteuerungszentralen und Lichtsignalsteuergeräten zur Steuerung und Versorgung der Lichtsignalsteuergeräte
ODG	OCIT Developer Group
OSI	OSI-Schichtenmodell (auch OSI-Referenzmodell; englisch Open Systems Interconnection Reference Model), ein Kommunikationsmodell der Internationalen Organisation für Normung (ISO) für Kommunikationsprotokolle in Rechnernetzen.
OTS 2	Open Traffic Systems, Version 2
Server	Ein Programm, das bestimmte Dienste anbietet und dazu passiv auf eingehende Aufrufe (von Clients) wartet.
SOAP	SOAP (Simple Object Access Protocol), ist ein Protokoll mit dessen Hilfe Daten zwischen Systemen ausgetauscht werden können. SOAP verwendet „Remote Procedure Call“ und ermöglicht dadurch den Aufruf von Funktionen in anderen Computern. Siehe http://www.w3.org/TR/SOAP

SSL	Secure Socket Layer.
Soap-Server-Interface	Soap and Protocolmanager auf der Server-Seite
Soap-Client-Interface	Soap and Protocolmanager auf der Client-Seite
Protocolmanager	Protokollschicht zur Implementierung von Kommandos im Puffer
TLS	Technische Lieferbedingungen für Streckenstationen. Die TLS sind ein Standard für den Aufbau von Verkehrsbeeinflussungsanlagen an Bundesfernstraßen. Herausgeber: Bundesanstalt für Straßenwesen.
TCP / IP	Transmission Control Protocol / Internet Protocol, eine Familie von Netzwerkprotokollen für das Internet.
VDV	Verband Deutscher Verkehrsunternehmen
WSDL	Web Services Description Language, eine plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Netzwerkdienste (Webservices) zum Austausch von Nachrichten auf Basis von XML.
XML	Extensible Markup Language, eine Auszeichnungssprache zur Darstellung strukturierter Daten in Form von Text. XML wird u. a. für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt. Ein XML-Dokument besteht aus Textzeichen, im einfachsten Fall in ASCII-Kodierung, und ist damit maschinenlesbar. Binärdaten enthält es nicht. Die XML-Spezifikation wird vom World Wide Web Consortium (W3C) als Empfehlung (Recommendation) herausgegeben.
XSD	XML Schema, eine Empfehlung des World Wide Web Consortium (W3C) zum Definieren von Strukturen für XML-Dokumente. Die Struktur wird in Form eines XML-Dokuments beschrieben. Darüber hinaus wird eine große Anzahl von Datentypen unterstützt. In der XSD Schemasprache werden Datentypen, einzelne XML-Schema-Instanzen (Dokumente) und Gruppen solcher Instanzen beschrieben. Ein konkretes XML Schema wird auch als eine XSD (XML Schema Definition) bezeichnet und hat als Datei üblicherweise die Endung „.xsd“.

2 Protokoll

Die gesamte Kommunikation über die Schnittstelle wird mittels des SOAP Protokolls abgewickelt.

In diesem Kapitel wird die Anwendung des Protokolls für die Schnittstelle OCIT-C beschrieben. Diese Konfiguration muss in allen Clients und Servern installiert werden.

Die genaue Beschreibung des dem Protokoll zugrunde liegenden Datenmodells, sowie die elementare Beschreibung der Attribute und Elemente erfolgt vollständig innerhalb der einzelnen Schemadefinitionen in Form von XML Schemadefinitionen (XSD). Diese sind sowohl maschinell verarbeitbar als auch als Text lesbar. Die Schemadefinitionen wurden in englischer Sprache erarbeitet und werden nicht übersetzt.

2.1 Übertragungsprotokoll SOAP

Dieses Kapitel beschreibt die SOAP Schnittstelle.

2.1.1 Technik

Die zu übertragenden Daten werden als XML codiert. Dies hat folgende Vorteile:

- Übliches Protokoll für alle Bereiche,
- Unabhängigkeit von der Art der Daten,
- plattformunabhängig,
- einfach zu erweitern.

Als Übertragungsverfahren wird SOAP auf der Basis von http benutzt. SOAP verwendet ebenfalls XML, um seine Daten aufzubauen.

Das Protokoll enthält einfache Befehle wie Empfange (get) oder Lösche (delete).

2.1.2 Anforderungen ans Protokoll

- Das Protokoll ist ein Server-Client Protokoll.
- Daten werden an der Ausgangs-Schnittstelle als XML dargestellt.
- Daten werden an der Eingangs-Schnittstelle als XML akzeptiert.
- Befehle sind in XML eingebettet.
- Objekte werden durch externe Bezeichner identifiziert.
- Unterschiedliche Objekttypen dürfen nicht auf einmal angefordert (in einem „request“) werden.
- Das Protokoll ist innerhalb des Servers zustandslos. Der Server weiß nichts über den Client.

2.1.3 Sicherheit

Der Server enthält eine Liste der Benutzernamen und der mit ihnen verbundenen Passwörter, sowie die dem Nutzer erlaubten Operationen und Zugänge zu Clients.

2.1.4 Erforderliche Bandbreite

Die erforderliche Bandbreite hängt von der Anzahl der Clients, der Objekttypen und Objekte im System ab. Daher können hier keine Angaben zur Bandbreite gemacht werden. Ein Local Area Network (LAN) zwischen den zentralen Anwendungen wird jedoch eine ausreichende Übertragungskapazität bieten.

2.2 Protokollfunktionen

Das Protokoll erlaubt Lesen und Konfigurieren von Daten. Außerdem ist es ist möglich, Objekte hinsichtlich der Verwendbarkeit auszuwerten und während der Laufzeit dynamisch zu strukturieren. Jeder Befehl besteht einer Anforderung (Request) und einer Antwort (Respond).

Bei einem Request wird eine XML Struktur vom Client zum Server geschickt. Das Ergebnis wird vom Server als „Resultat“ (auch „response structure“ genannt) zum Client zurückgeschickt.

Verfügbare Methoden:

Aufruf (request)	**Antwort (response)**	**Funktion**
put	putResponse	Konfigurieren von Objekten
get	getResponse	Abfragen von geänderten Daten seit der letzten Abfrage
inquireAll	inquireAllResponse	Abfragen aller Objekte eines Objekttyps

Aufruf (request)	Antwort (response)	Funktion
delete	deleteResponse	Löschen von dynamischen Daten
getContentInfo	getContentInfoResponse	Abfragen von Objektinhalten
wait4Get	wait4GetResponse	Abfragen von geänderten Daten seit der letzten Abfrage (wie get) mit dem Unterschied, dass die Response verzögert wird, bis Daten zur Verfügung stehen.

2.2.1 Lesen von Daten durch den Client

Alle Protokollfunktionen zum Lesen von Daten beinhalten einen Parameter (Filter), der die Objekte kennzeichnet, die gelesen werden sollen.

Falls der Filter leer ist, werden alle Objekte in der zugehörigen Antwort zurückgesendet. Damit sich der Client im Falle einer Unterbrechung wieder synchronisieren kann, werden die Startinformationen der vorhergehenden Antwort eines Lesezugriffs mit übertragen.

Der Server bietet alle lesbaren Daten der vorhandenen Objekte an seiner äußeren Schnittstelle an. Die vorhandenen Objekt-Typen können vom Client mit dem Befehl getContentInfo abgefragt werden. Diese können vom Client mit „get“ oder „inquireAll“ gelesen werden (wenn der Lesezugriff erlaubt ist).

Der Unterschied zwischen inquireAll und get ist:

- inquireAll (Resynchronisierungsfunktion) liefert alle Objekte des abgefragten Objekt-Typs mit dem letzten Zustand bzw. dem Inhalt der Objekte. inquireAll muss in jedem Fall einer Synchronisierung (z. B. Wiederanlauf des Servers oder Clients) verwendet werden. get (hier als Lesen von Änderungen gebraucht) liefert alle seit der letzen Abfrage vorgenommen inhaltlichen Änderungen des abgefragten Objekt-Typs. Dieser Mechanismus wird im Detail im Kapitel 2.3.1, Datenpufferung und Positionsbehandlung, beschrieben.

Das folgende Ablaufdiagramm Bild1 zeigt, wie Daten des Servers periodisch mithilfe der Protokollfunktionen inquireAll und get angefordert werden.

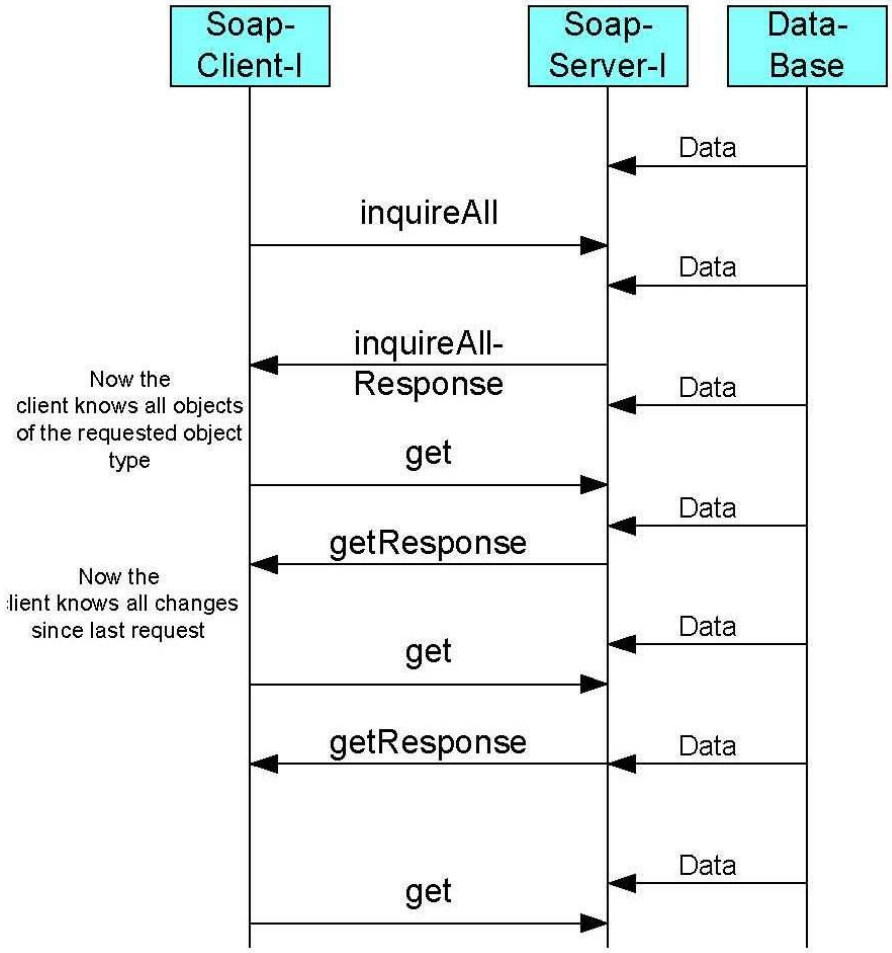


Bild 1: Übliche Reihenfolge zum Lesen von Daten vom SOAPserverInterface

2.2.2 Senden von Daten zum Server

Es ist möglich, Daten vom Client zum Server zu senden. Dazu wird die Protokollfunktion „put“ verwendet. Das Verhalten des Servers hängt vom Objekt-Typ ab. Im Falle von unbekannten Objekten im put Befehl wird entweder das Objekt erzeugt oder eine Störung zurückgesendet. Um Objekte zu löschen wird der Befehl „delete“ abgesetzt.

Zur Konfiguration der Schnittstelle werden die entsprechenden Daten mit put zum Server gesendet. Der Server nimmt sie an oder weist alle nicht konfigurierbaren Objekte zurück und listet sie in der putResultlist auf.

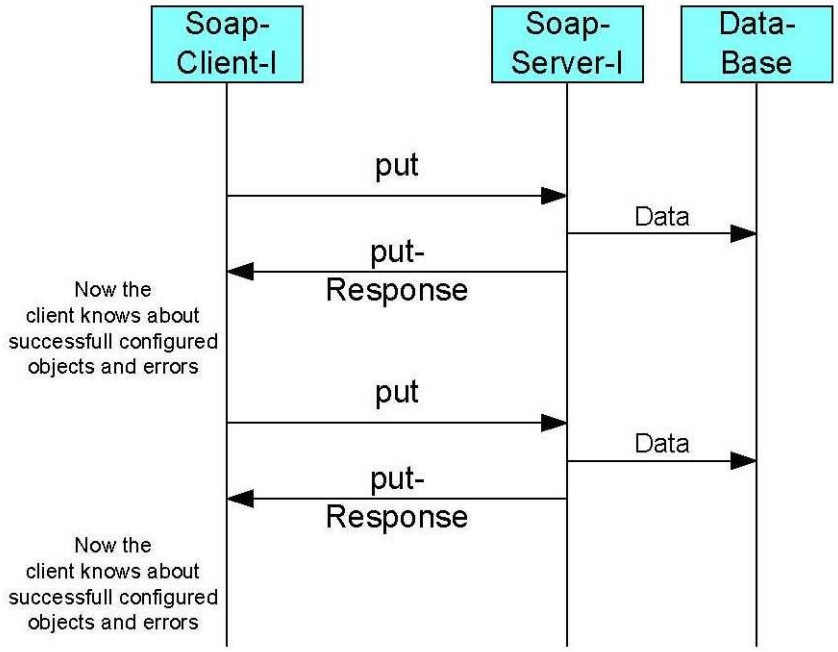


Bild 2: Übliche Reihenfolge zum Schreiben oder Konfigurieren von Daten zum SOAPserverInterface

2.3 Ablaufsteuerung

Das Protokoll ist verbindungslos. Zu Ablaufsteuerung (sequence control) reicht es aus auf die Antwort einer Anfrage zu warten. Dies wird durch das Protokoll http geregelt. Eine zusätzliche Ablaufsteuerung wird nicht benötigt. Passende Abläufe werden in folgenden Kapiteln beschrieben.

2.3.1 Datenpufferung und Positionbehandlung

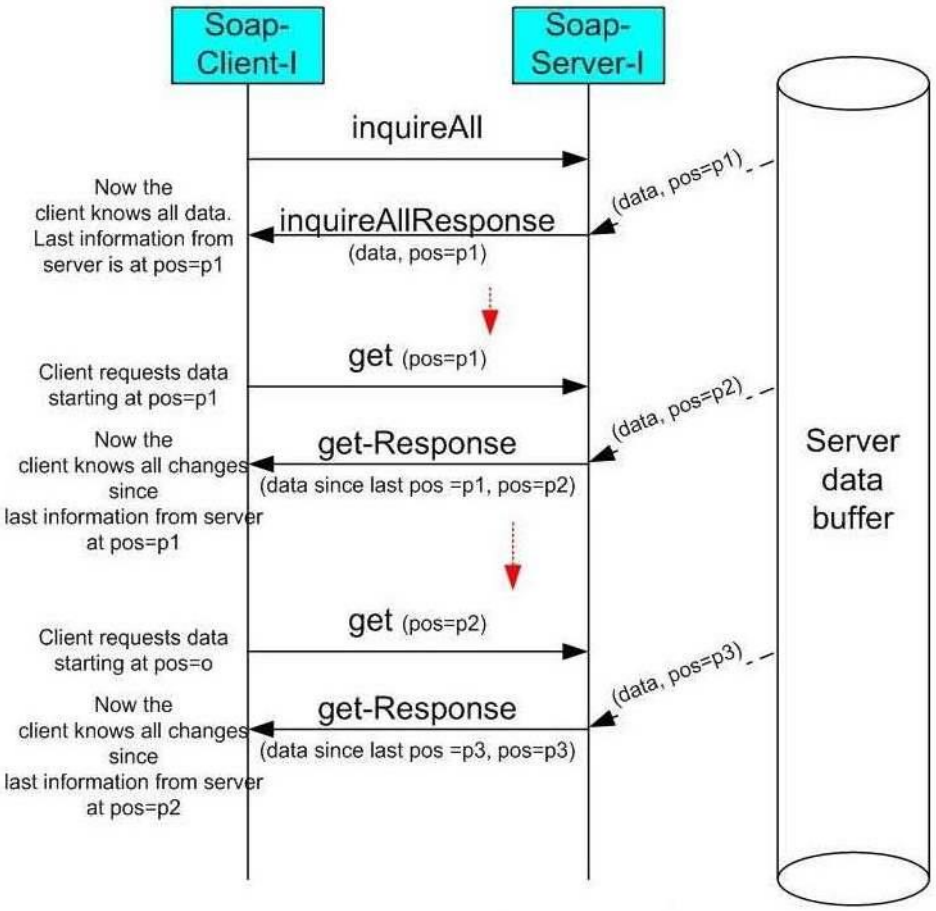


Bild 3: Datenpufferung am Server und Positionsbehandlung am Client

Der Server legt die vom Client angeforderten Daten aus seiner Datenbank in einem Ringpuffer ab. Nach der Durchführung eines inquireAll, kennt der Client den letzten Zustand der der Datenbank. Außer der InquireAll Antwort empfängt der Client die letzte Positionszahl und zeigt die letzten Informationen vom Server an (Zeiger zu den letzten empfangenen Daten im Datenpuffer). Mithilfe dieser Positionszahl erzeugt der Client die nächste Anforderung (get). Durch die Positionszahl weis der Server, welche Daten zur Verfügung gestellt werden müssen, um alle aufgetreten Änderungen zu liefern. Die erhaltene Antwort umfasst die gewünschten Daten und eine neue Positionszahl, die für die nachfolgende Abfrage verwendet wird.

Wenn ein Client mehr als einen Objekt-Typ abfragen will, müssen mehrere Anfragen benutzt werden.

Jede Anfragenreihe muss eine eigene Positionsbehandlung durchführen, um Objekt-Typ bezogene Deltawerte zu erhalten. Dies gilt auch im Falle der Verwendung von unterschiedlichen Filterlisten und bei der Verwendung von Parametern mit dem optionalen XML Element get→data.

2.3.2 Zu lange Transaktionszeit

Normalerweise enthält eine Antwort die angeforderten Daten. Falls der Server mehr Zeit als zulässig benötigt (> 60s), wird eine leere Antwort wird zum Client zurückgegeben. Die überschrittene Transaktionszeit wird durch einen Störungscode angezeigt. Trotzdem fährt der

Sever fort, weiter angeforderten Daten von seiner Datenbank (oder vom Archiv) zu holen. Der Client wiederholt seine Anfrage nach einer bestimmten Zeit. Nach dieser Zeit sollte der Server in der Lage sein, die Daten zu liefern. Falls der Server noch immer nicht liefern kann, wird wieder ein Störungscode angezeigt.

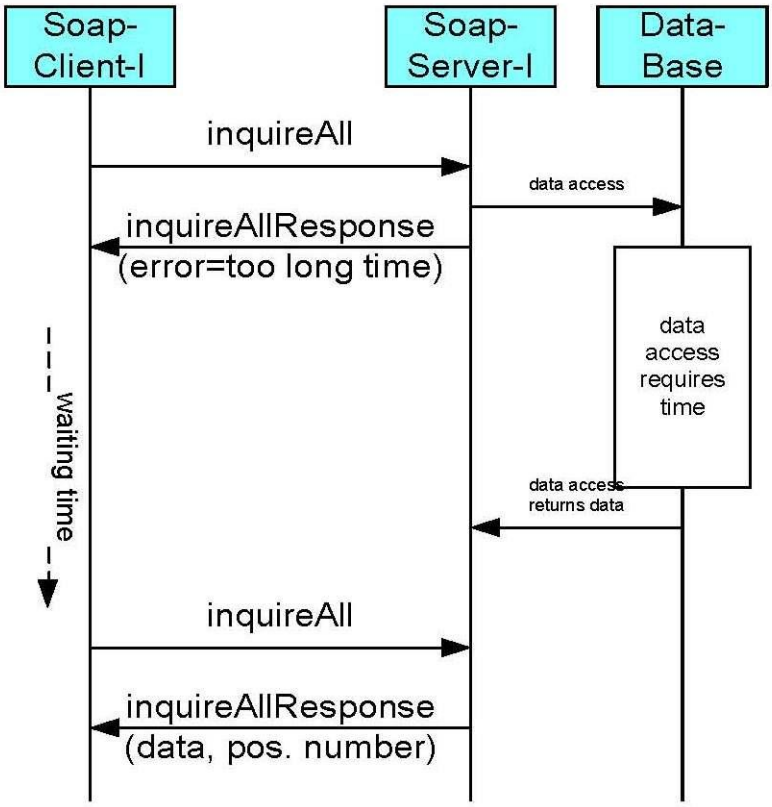


Bild 4: Handhabung langer Transaktionszeiten (ähnliche Reihenfolge gilt auch für „get“)

2.3.3 Zu lange Abfragen

Der Server liefert einen Störungscode zurück, falls der Inhalt einer Antwort eine bestimmte Schwelle übersteigt. Dieses kann geschehen, falls

- zu viele historischen Daten angefordert werden (Zeitbereich zu groß), oder
- zu viele Objekte abgefragt werden.

Der Client muss seine Anforderung durch Reduktion der Anzahl der Objekte oder des Abfragezeitraums entsprechend verringern.

2.3.4 Zu viele Änderungen

Sollten seit der letzten Abfrage zu viele Änderungen angefallen sein, wird die Abfrage so weit als möglich befriedigt, indem entweder nicht alle Objekte oder nicht der gesamte Zeitbereich zurückgegeben wird.

Die unvollständige Rückgabe wird über den errorCode (in der protocol.xsd) signalisiert (errorCode: missingDatasets)

2.3.5 Handhabung von Abfragen historischer Daten

- Der Befehl „get“ wird verwendet, falls ein Zugang zu den historischen Daten angefordert wird.

- Das Element „storetime“ wird benutzt, um den Anfangszeitpunkt der gespeicherten historischen Daten zu definieren.

- Das Element „endStore“ wird benutzt, um den Endezeitpunkt der gespeicherten historischen Daten zu definieren.

- Die Antwort des Servers enthält die Zustandsänderungen (keine Initialwerte) von „storetime“ bis „endStore“, sofern das Abfrageintervall nicht zu groß definiert wurde.

- Sollte das Abfrageintervall zu groß definiert worden sein, wird die Abfrage so weit als möglich befriedigt, indem entweder nicht alle Objekte oder nicht der gesamte Zeitbereich zurückgegeben wird. Die unvollständige Rückgabe wird über den errorCode (in der protocol.xsd) signalisiert (errorCode: missingDatasets)

- Wenn storetime gleich endStore ist, wird der Zustand zu diesem Zeitpunkt gesendet. Der Wert erhält dann als Zeitstempel entweder exakt den Originalzeitstempel des Wertes oder, sollte dieser nicht vorliegen, wird der initiale Zeitstempel nicht aufgeführt (Zeitstempel ist optionales Element in der protocol.xsd).
  Kann der Initialwert nicht geliefert werden, oder macht deren Lieferung keinen Sinn (z.B. bei dem Objekttyp „operatingMessages“), so wird kein Wert geliefert.

Es wird empfohlen:

- Den Zeitbereich so kurz wie möglich zu wählen

- Filter zu verwenden und damit die Menge der abzufragenden Objekte zu verringern.

2.3.6 Multi Client Fähigkeit

Der Server arbeitet verbindungslos. Durch die Positionszahl ist es nicht notwendig auf der Serverseite Wissen über den Client aufzubauen.

Bei der Implementierung von OCIT-C muss sichergestellt werden, dass die die benutzten Bibliotheken der unteren Kommunikationsschichten einen mehrfachen Client-Zugang erlauben.

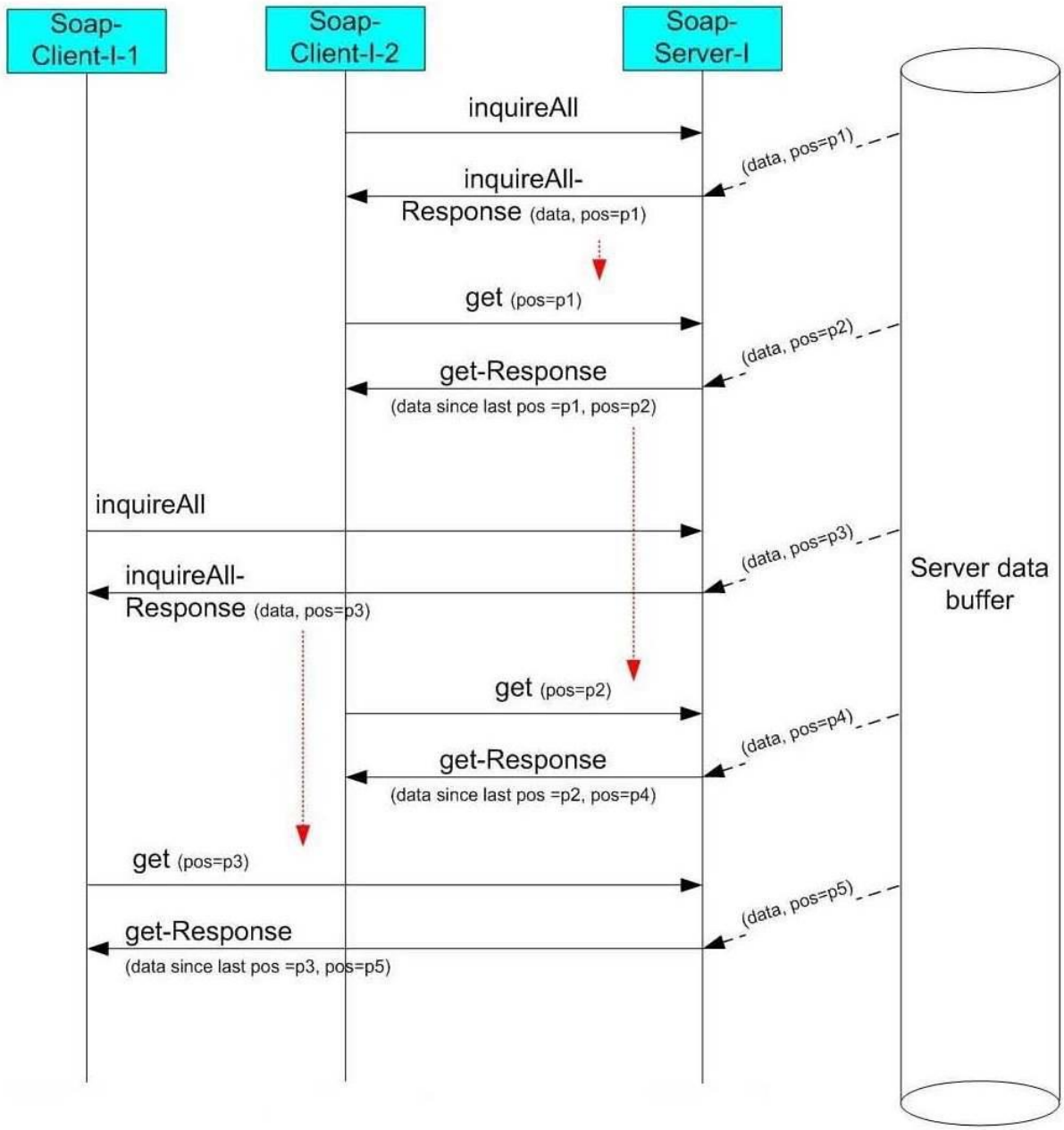


Bild 5: Multi Client Fähigkeit

2.3.7 Resynchronisieren

Es gibt verschiedene Gründe für eine erneute Synchronisierung (Resynchronisation):

- Neustart des Clients
  - Der Client erkennt dies und wird daher mit „inquireAll“ resynchronisieren.
  - Der Server kann möglicherweise den Neustart des Clients mittels eines optional implementierbaren Watchdogs erkennen.
- Neustart des Servers

- Der Client erkennt möglicherweise über „socket timeout“, dass der Server nicht erreichbar ist.
- Der Server reagiert auf jede mögliche Protokollfunktion (inquireAll, get oder put), einschließlich lastStart, das anzeigt, wenn der Server gestartet wurde.
- Der Client muss mit inquireAll resynchronisieren, falls sich lastStart von vorherigen unterscheidet.

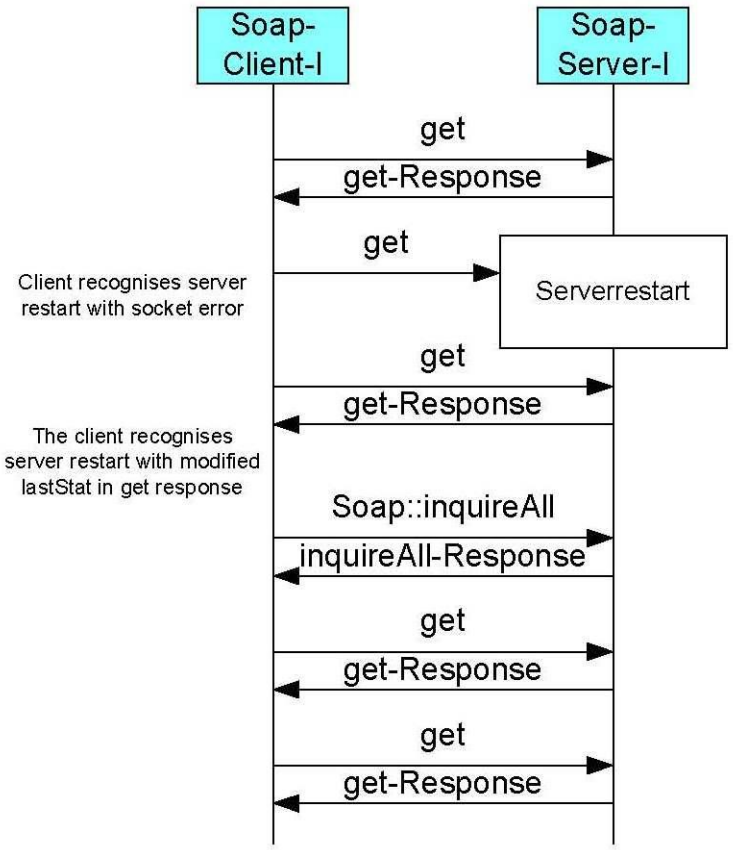


Bild 6: Neustart des Servers

2.3.8 Bidirektionale Kommunikation

Die folgenden Anwendungsfälle benötigen bidirektionale Kommunikation:

- LSA Steuerung oder Schildersteuerung:
  - Schaltbefehl wird übertragen von Zentrale nach Geräteserver
  - Schaltzustand wird asynchron übertragen von Geräteserver nach Zentrale
- CCTV
  - PTZ (pan/til/zoom) Befehl wird übertragen von Zentrale nach Geräteserve
  - - Aktueller Zustand wird asynchron übertragen von Geräteserver nach Zentrale

Die folgenden Kapitel beschreiben mögliche Konfigurationen. Als Beispiel wird die „Schildersteuerung“ verwendet.

2.3.8.1 Bidirektionale Kommunikation mit Client und Server Paar

Schalten eines Schildes (Ablaufreihenfolge im Gutfall)

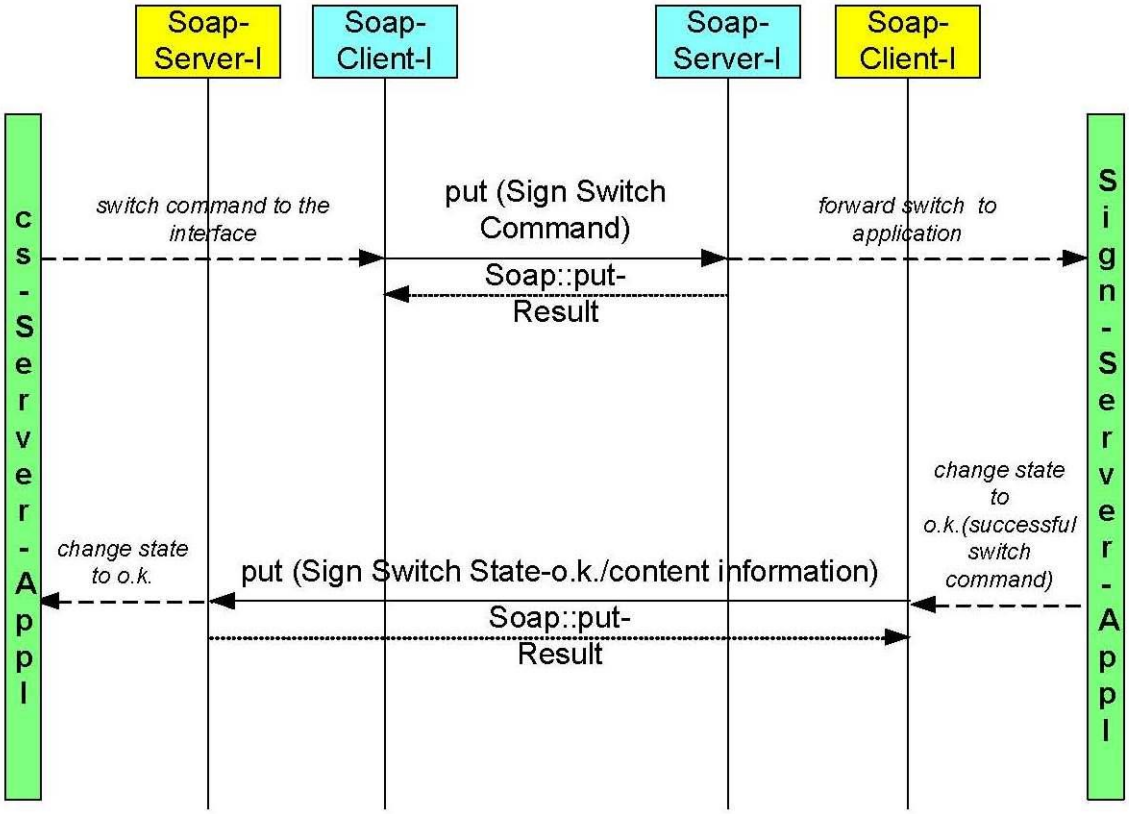


Bild 7: Schalten eines Schildes (Ablaufreihenfolge im Gutfall)

Schalten eines Schildes (Ablaufreihenfolge im Fehlerfall)

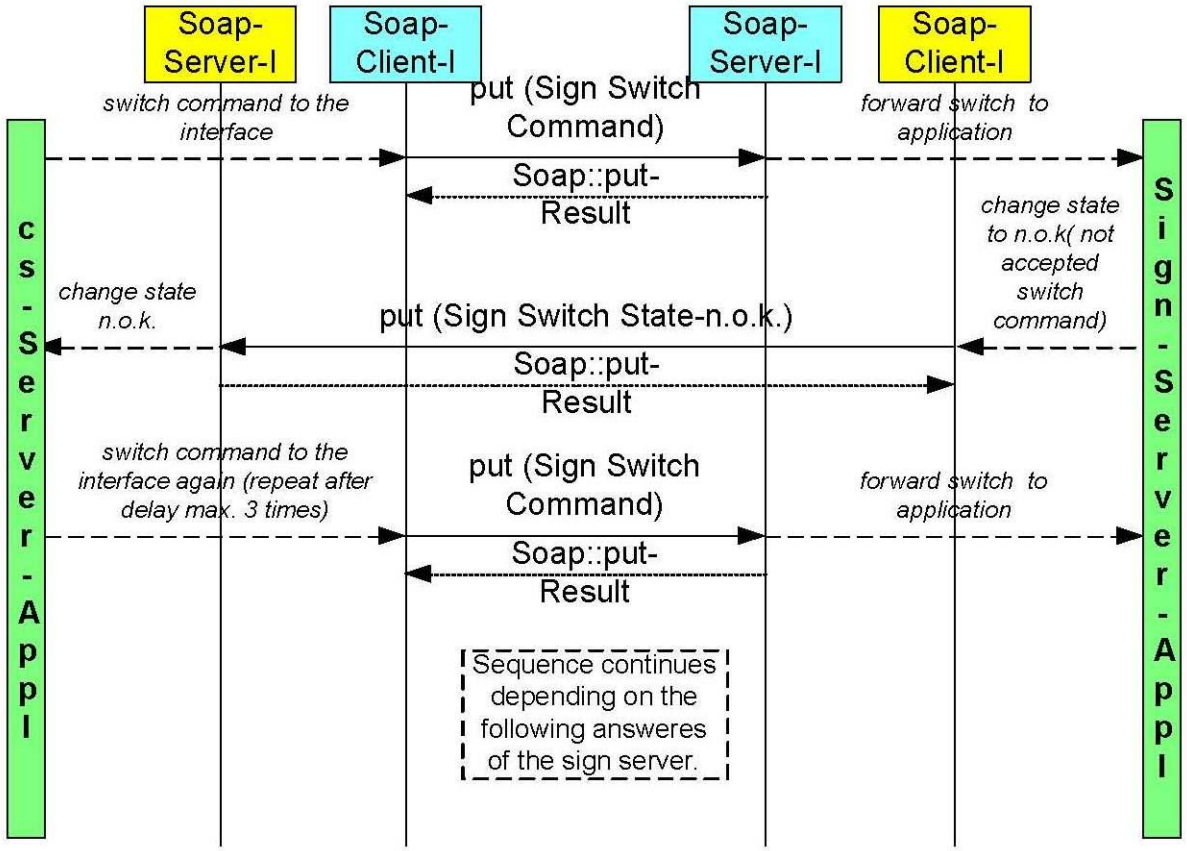


Bild 8: Schalten eines Schildes (Ablaufreihenfolge im Fehlerfall)

Im Falle einer fehlenden Antwort werden die Schaltbefehle nach einer konfigurierbaren Verzögerung wiederholt. Dies gilt unabhängig davon wo die Störung auftritt (es erfolgt kein Zustandsübergang zum Zustand „busy“ oder „o.k.“). Das Wiederholen wird nachdem drei erfolglosen Versuchen gestoppt. Dann ändert sich der aktuelle Zustand an der Seite des zentralen Systems zu n.o.k.

Zustand-, Inhalts- und Verbindungsüberwachung

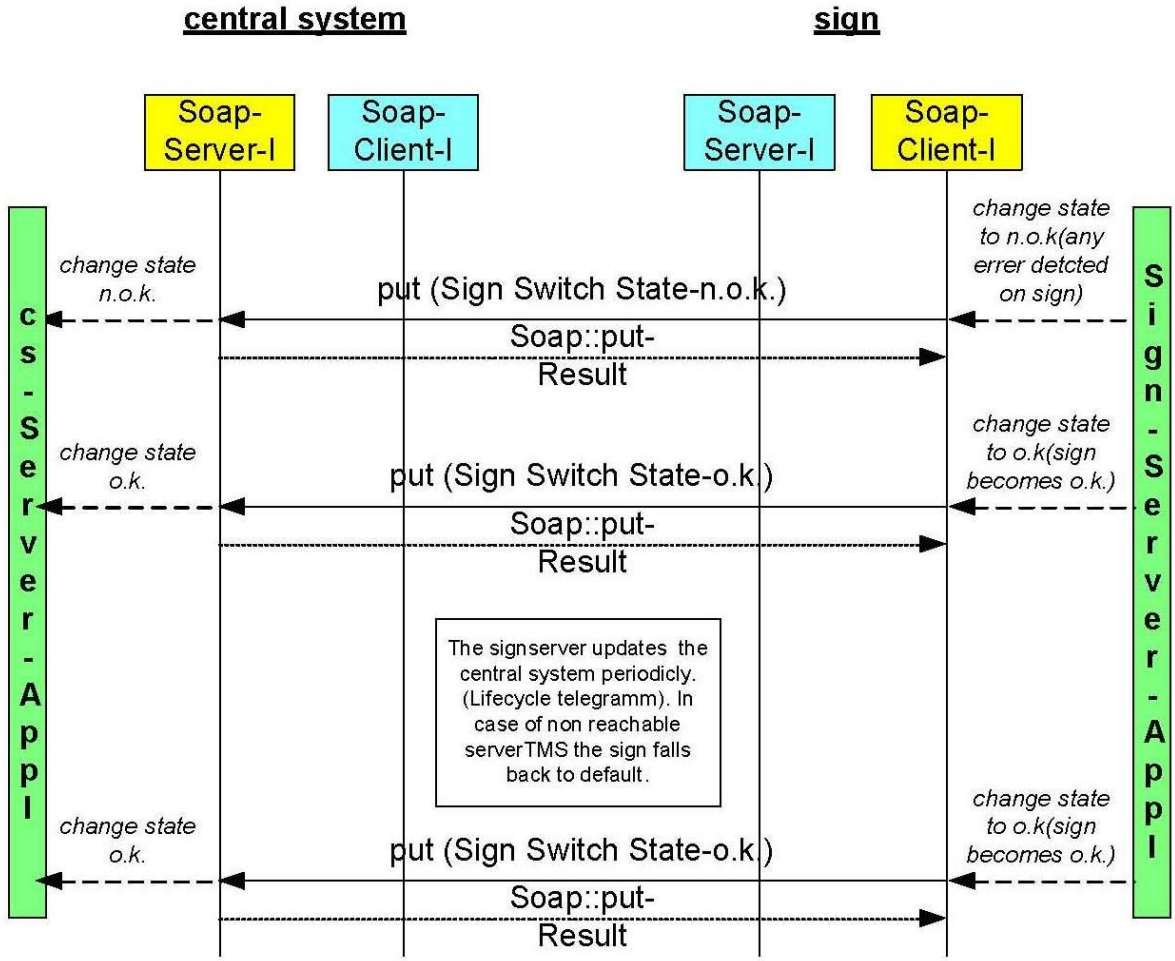


Bild 9: Schaltzustand

Der Schilder-Server (sign server) aktualisiert das zentrale System (central system) periodisch, auch wenn sich der Schaltzustand nicht ändert. Folglich erhält das zentrale System Nachricht:

- falls ein Schild, das an das Schilder-Server angeschlossen wird, seinen Zustand ändert
- falls ein Schild seinen Inhalt ändert
- falls der Schilder-Server unerreichbar ist (die max. Wartezeit gibt das zentrale System vor)

Falls das zentrale System unerreichbar ist, setzt der Schilder-Server die Schilder auf einen vordefinierten Anzeigezustand.

2.3.8.2 Bidirektionale Kommunikation mit regelmäßiger Zustandsabfrage (polling)

Die Auswertung der Zustände erfolgt sinngemäß wie in Kapitel 2.3.8.1.

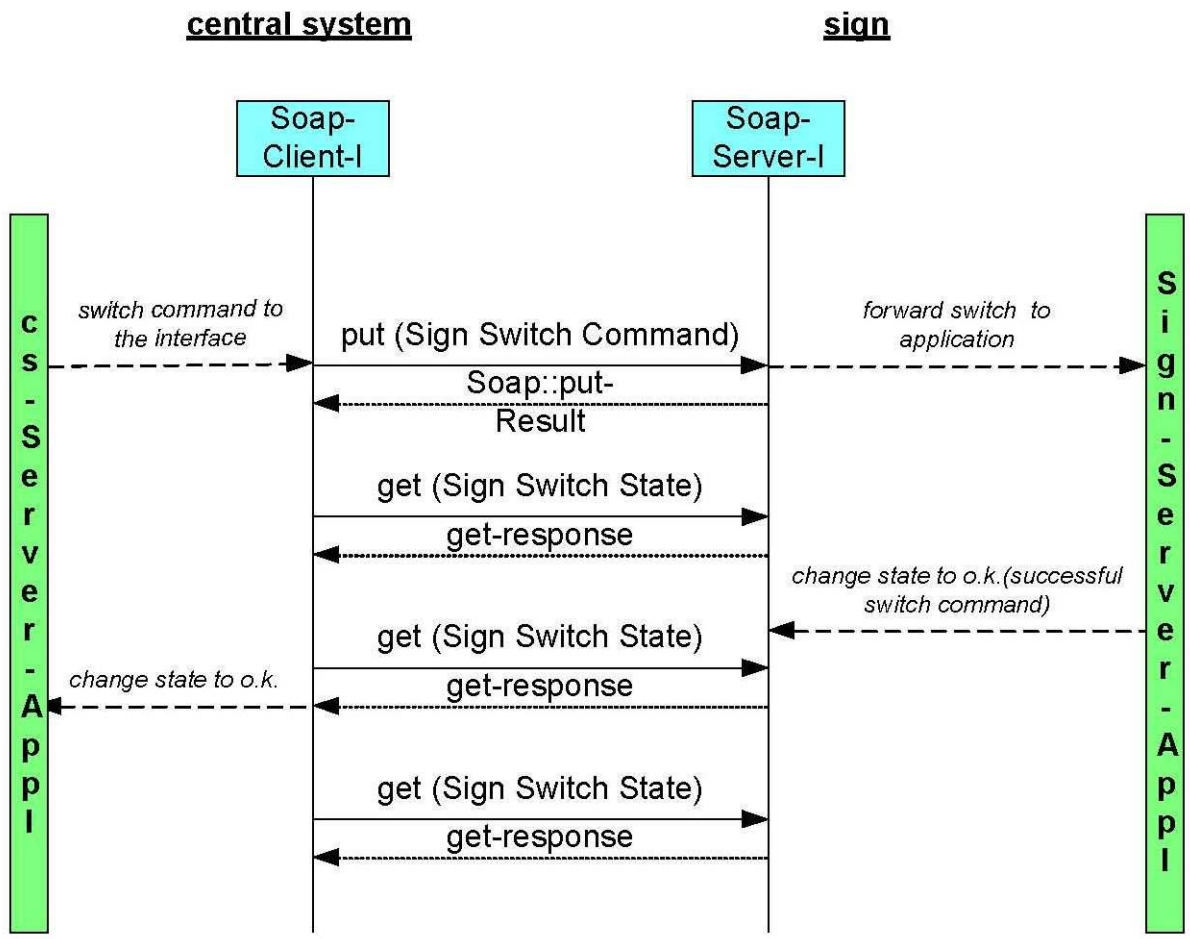


Bild 10: Regelmäßige Zustandsabfrage

2.3.9 Vermeidung von Abtastverzögerungen

Zur Vermeidung von ungewollt grossen Abtastverzögerungen durch zyklische Abfragen vom Client zum Server wird eine neue Funktion im Protokoll eingeführt. Diese Funktion arbeitet nach folgendem Prinzip:

- Aufbau wie die Funktion „get“
  (d.h. gleiche Parameter, nur andere Funktionsbezeichnung)

  Name der Funktion „wait4Get“

- Die neue Funktion wait4Get arbeitet wie die Funktion get mit dem Unterschied, dass im Falle keiner vorliegenden Daten am Server (d.h. die Rückgabeliste wäre leer) die Response aus dem wait4Get verzögert wird bis entweder ein Timeout auftritt oder Daten am Server verfügbar werden.

- Der Client würde also unmittelbar nach Empfang einer Antwort die Funktion wait4Get wiederum aufrufen, um dann den Rückkanal indirekt permanent geöffnet zu halten.
- Zur Vermeidung von mehreren Abfraqen innerhalb einer Sekunde kann der Server Schutzmaßnahmen ergreifen. Eine mögliche Schutzmaßnahme wäre nur eine Antwort pro Sekunde zuzulassen und die sich in dieser Sekunde angesammelten Werte in der Antwort zurückzugeben.

- Der Server kann die Anzahl der Clients bzw. Objekttypen/Objekte, die auf diese Weise abgefragt werden, einschränken.

- Zur Vermeidung zu vieler geöffneter Sockets ermöglicht die Funktion wait4Get die gleichzeitige Abfrage verschiedener Objekttypen. Dies wurde entsprechend in den Datenstrukturen der Funktion berücksichtigt.

Das folgende Sequenzdiagramm verdeutlicht das Vorgehen (vereinfacht für die Abfrage eines Objekttyps:

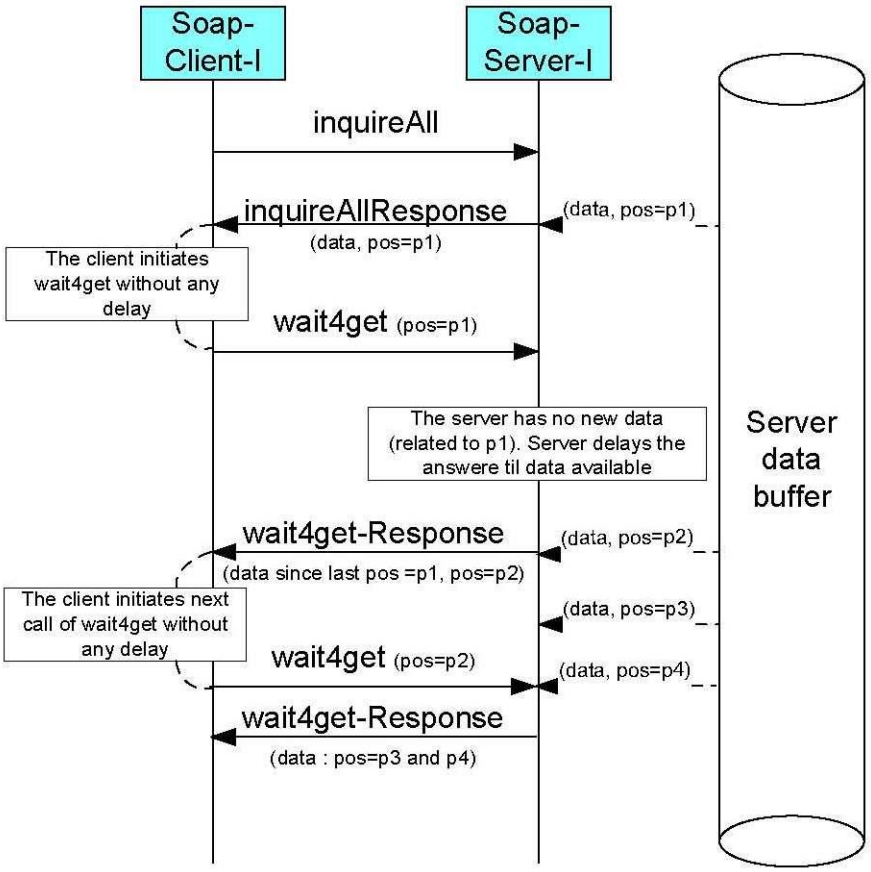


Bild 11: Kommunikationschichten Client und Server

2.4 OSI – Schichten

Server und Client besitzen entsprechend dem OSI Modell in verschiedene Abschnitte mit unterschiedlicher Funktion unterteilt.

Die unterste Protokollschicht bildet das Protokoll http, das für die Datenübertragung im Netzwerk zuständig ist.

Darüber liegt das SOAP-Protokoll als Client- oder Server Ausführung.

Der Protokollmanager hat die Aufgabe, alle Befehle einschließlich den erforderlichen Datenpuffer für die Serverfunktionalität bereit zu stellen.

Die Applikationsschicht stellt die Verbindung zur Datenbank her.

Das folgende Bild beschreibt zeigt diese Schichtmodell.

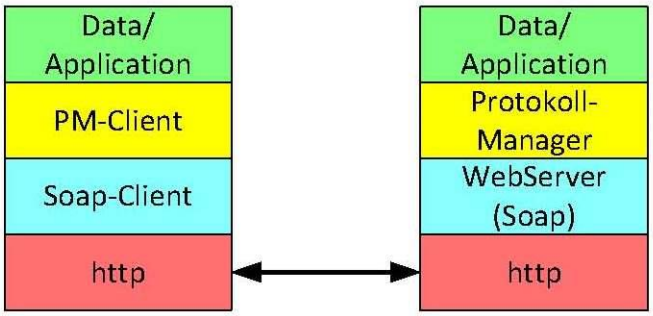


Bild 12: Kommunikationschichten Client und Server

2.5 Protokollfunktionen im Detail

Verfügbare Methoden:

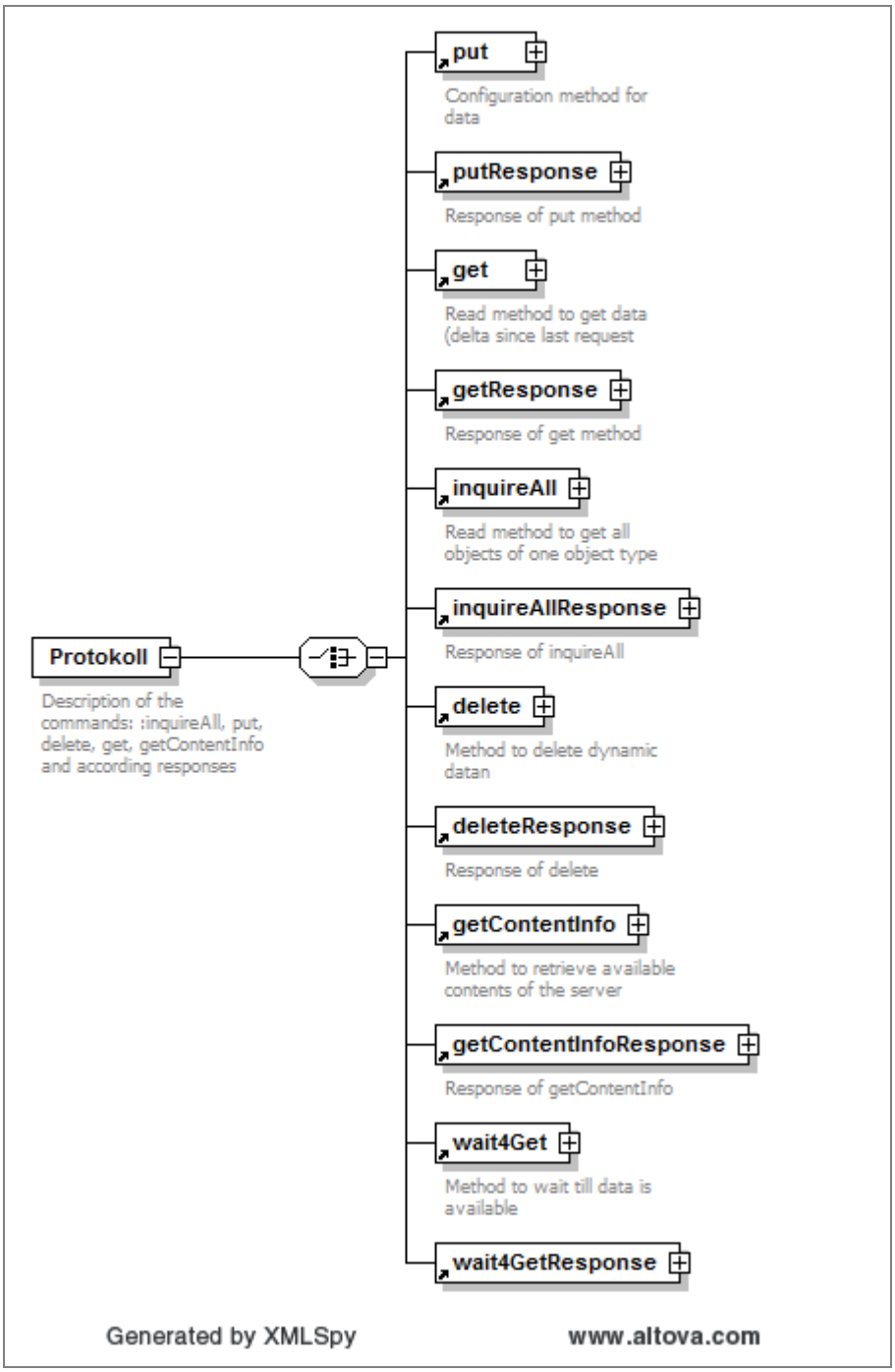


Bild 13: Verfügbare Methoden

2.5.1 Standard-Parameter

Die Standard-Parameter der Methoden sind:

Eingangs-Parameter

- **UserName** und **UserPasswd** authentifizieren den Benutzer
  UserName und UserPasswd werden als normaler Text übertragen. Diese Authentifizierung sollte daher nicht für hohe Sicherheitsanforderungen verwendet werden.

- **watchdog** ist eine Struktur mit der der Client dem Server mitteilt, wenn der nächste Aufruf erwartet werden kann. Dies kann zur zeitlichen Überwachung (timeout) des Client durch den Server verwendet werden.

- **storetime** bezeichnet den Start der angeforderten oder gesendeten Daten. Diese Methode wird nur bei Zugriffen auf gespeicherte historische Daten verwendet.

- **endStore** bezeichnet das Ende der angeforderten Daten. Diese Methode wird nur bei Zugriffen auf gespeicherte historische Daten verwendet.

- **position** bezeichnet die Position eines Zeigers im Puffer des Servers.
  Die Position wird in mit den Methode inquireAll oder getResponse erhalten und für die nächste get Anforderung verwendet (siehe Kapitel 2.3.1, Datenpufferung und Positionsbehandlung).

- **filterList** ist eine Liste der Objekte, welche gelesen werden sollen.
  Damit kann die Datenmenge begrenzt werden.

Ausgangs-Parameter

- **lastStart** ist der Zeitstempel des letzten Starts des Servers. Falls sich LastStart von einer auf die nächste Antwort des Servers ändert, ist das ein Zeichen, dass die Daten neu synchronisiert werden müssen. Der Client resynchronisiert (resynchronise) deswegen mit der Methode inquireAll.

- **errorCode** ist ein Fehlercode, der im Falle von fehlerhaften Befehlen erzeugt wird. Im Falle von fehlerhaften XML-Strukturen wird er nicht verwendet. Dazu gibt es die Meldung aus dem SOAP-Protokoll (Fault).

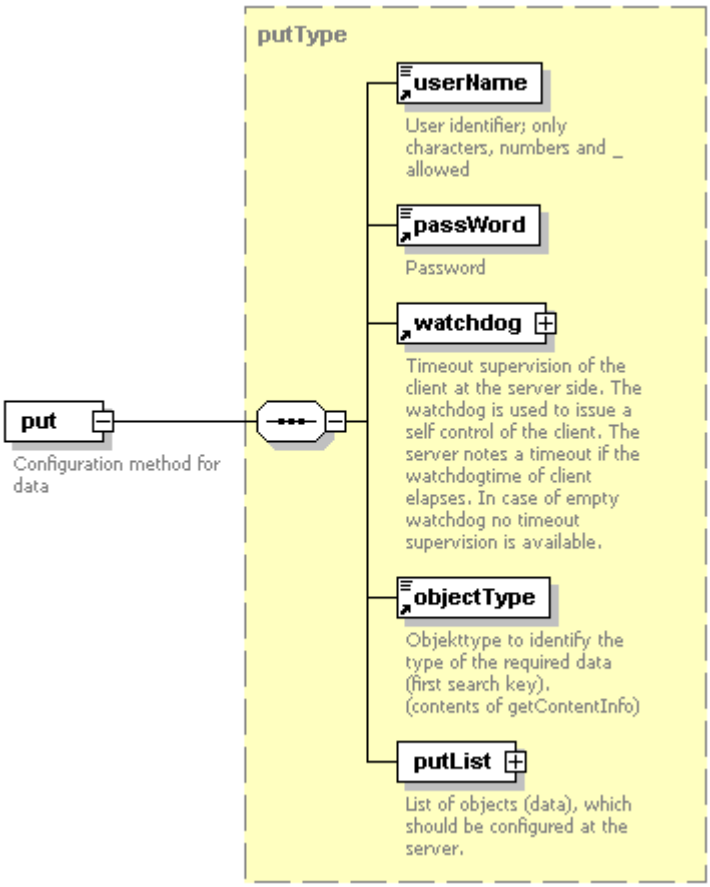
- **errorText** ist eine von Menschen lesbare Beschreibung des errorCode.

- **position** ist die Position des letzten Datenzugriffs. Sie muss für den darauffolgenden Datenzugriff verwendet werden.

- **dataList** ist eine mit den angeforderten Daten.

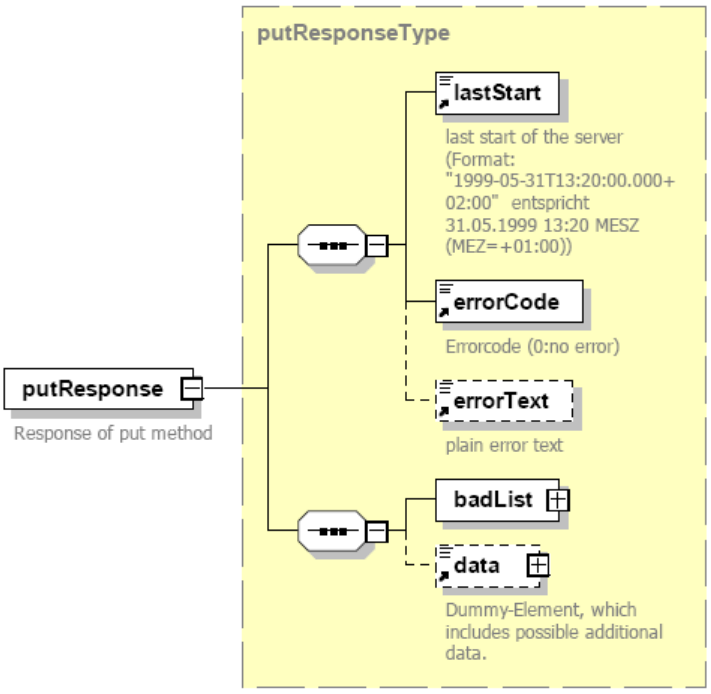
2.5.2 put

Die Methode put dient zum konfigurieren von Objekten. Sie enthält alle Instanzen der Daten, die konfiguriert werden sollen.



putResponse ist die Antwort auf put.

Sie enthält alle nicht-konfigurierbaren Daten, üblicherweise keine.

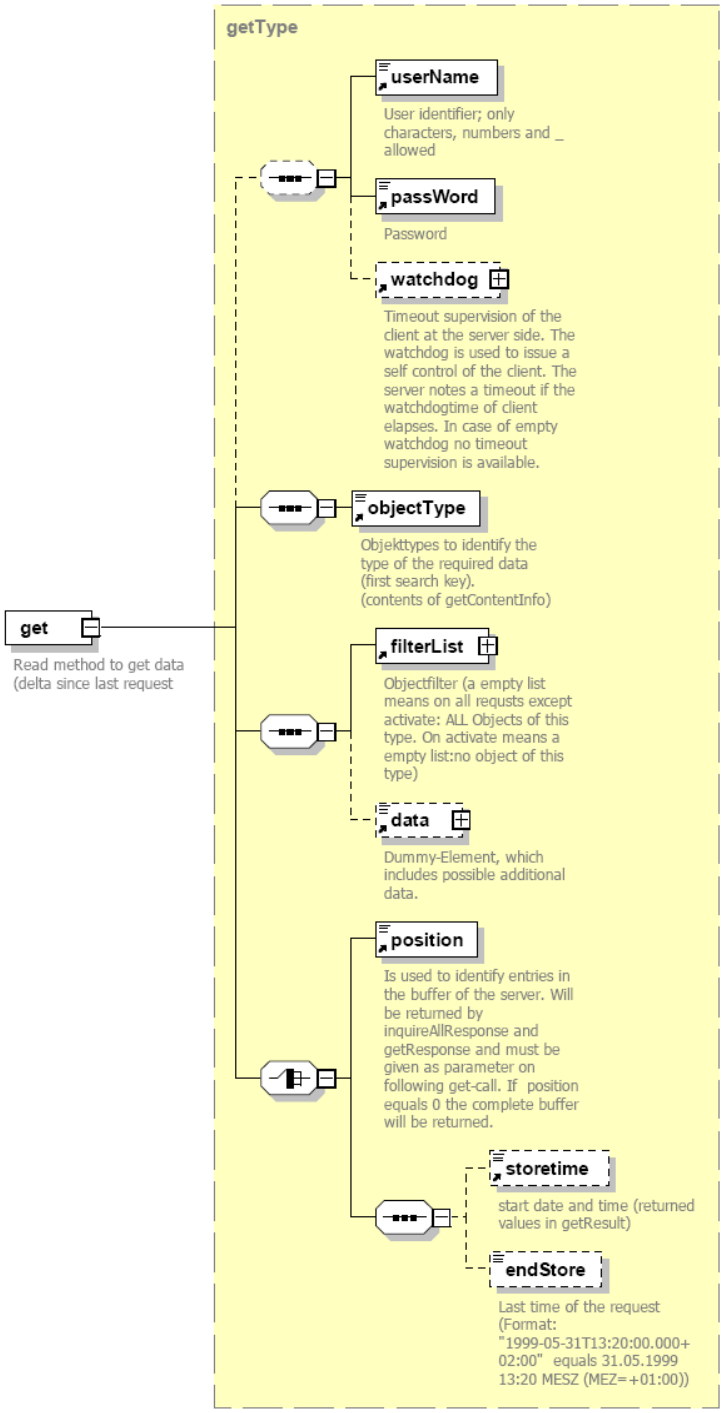


2.5.3 get

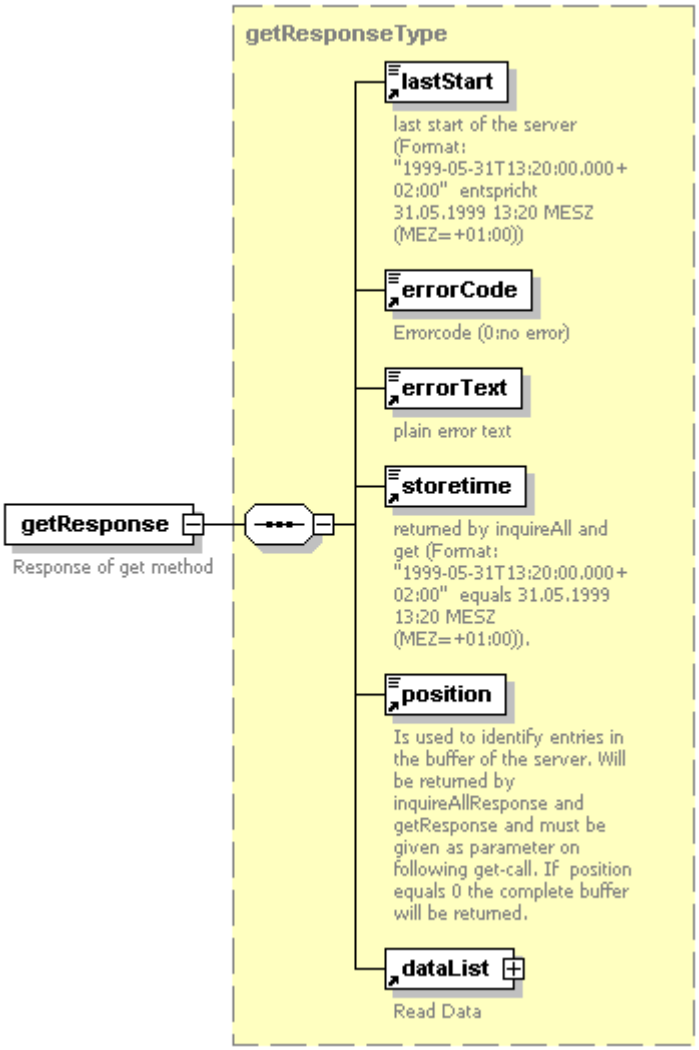
Die Methode get dient zum Abfragen von Daten.

Sie hat außer dem Standardparametern

- entweder Start- und Endezeit um alle Werte innerhalb dieser Zeitspanne zu erhalten
- oder die Positionsnummer der abzufragenden Daten. Normalerweise ist dies die Positionsnummer, die durch letzte Methode inquireAllResponse oder getResponse zurückgebracht wurde.



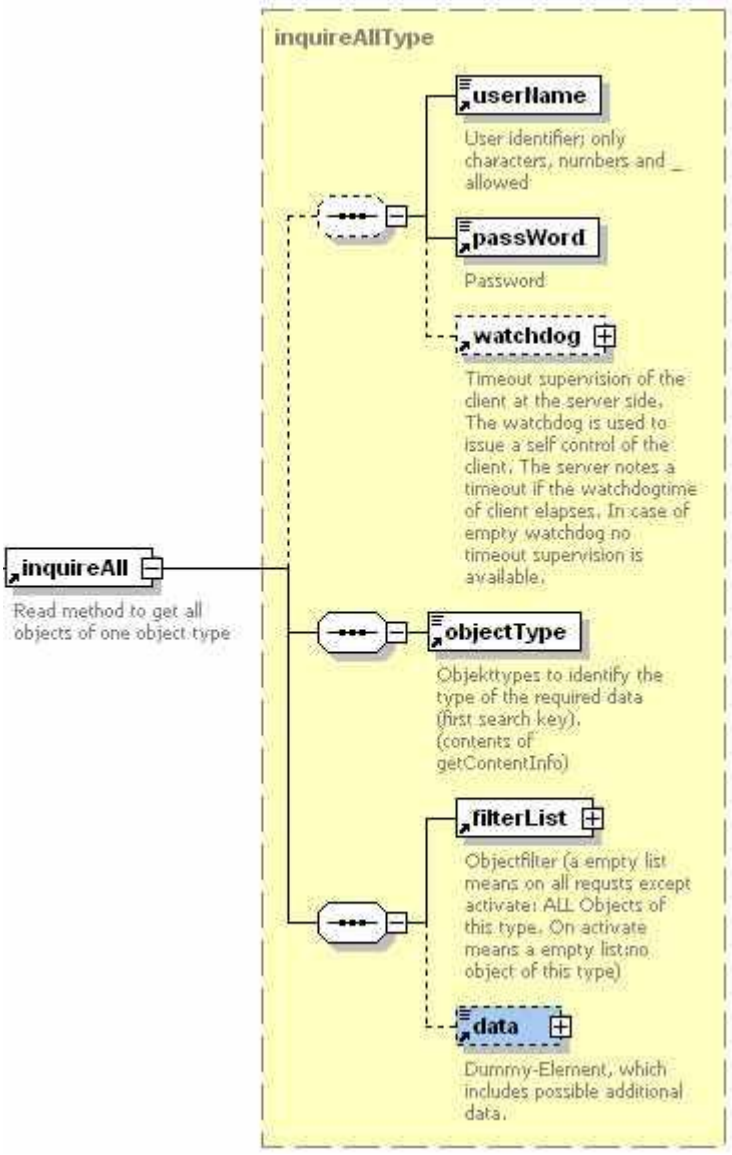
getResponse ist die Antwort auf get



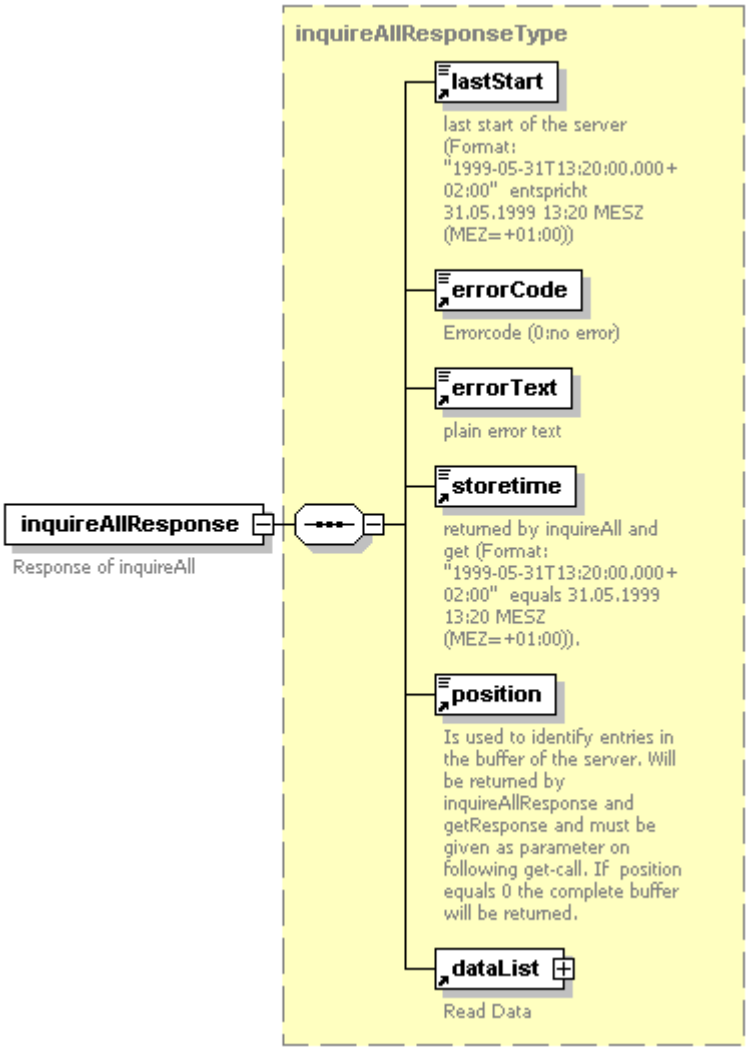
2.5.4 inquireAll

Methode zum Abfragen aller Objekte eines Objekttyps mit dem letzten Zustand bzw. dem Inhalt der Objekte. Diese Methode garantiert dem Client, dass in der Antwort alle abgefragten Objekte in der Antwort enthalten sind.

Die Methode InquireAll enthält nur die Standard Parameter.

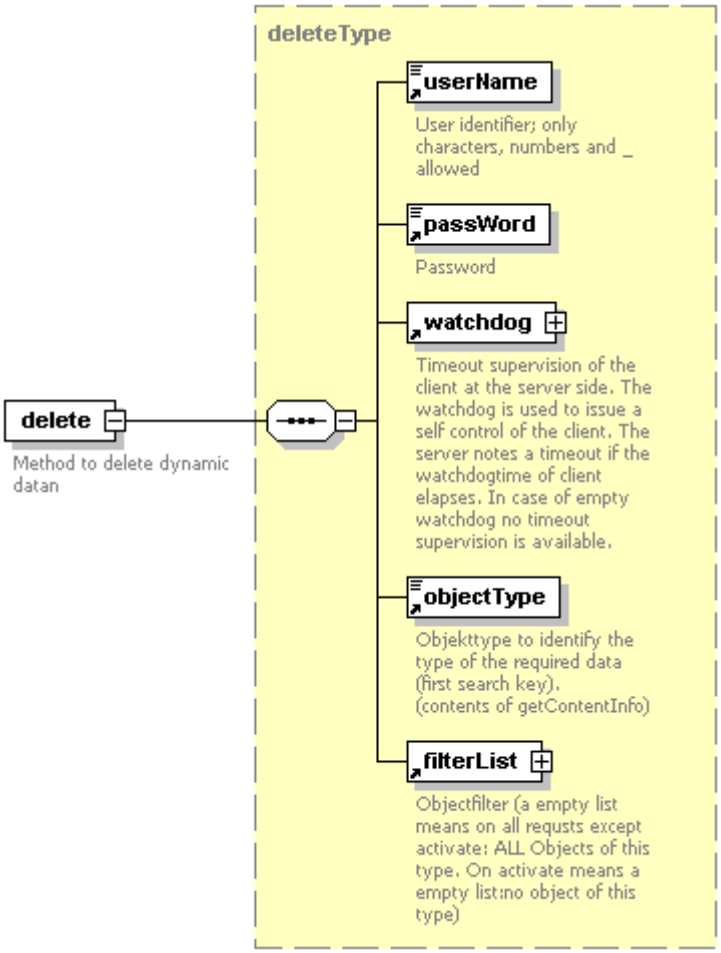


inquireAllResponse ist die Antwort mit allen angeforderten Inhalten.

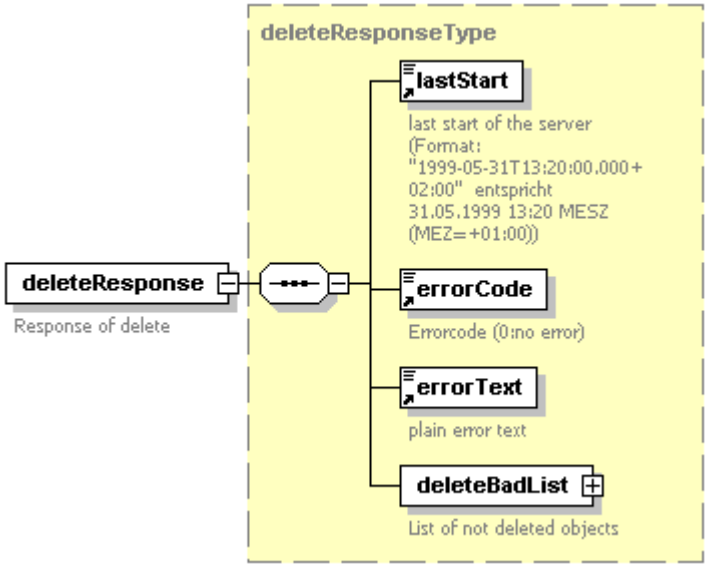


2.5.5 delete

Die Methode delete dient zum Löschen von dynamischen Daten (bei denen dies erlaubt ist). Instanzen der zu löschenden Daten müssen in die Filterliste eingetragen werden.

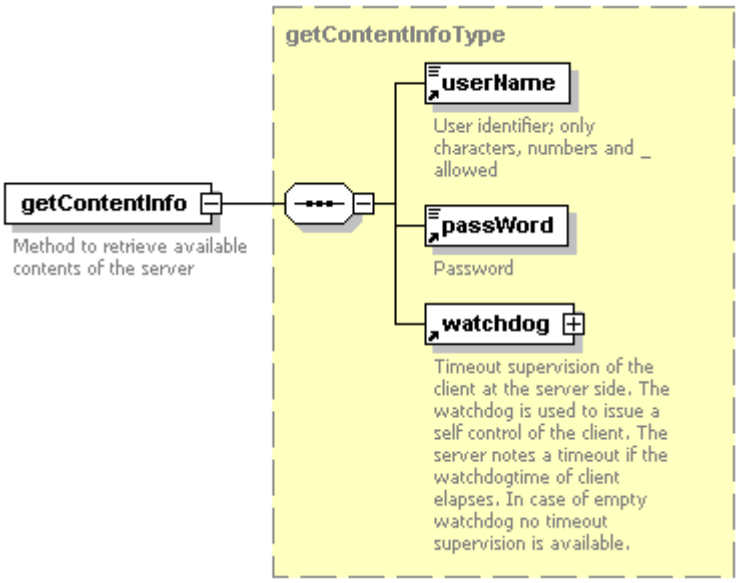


deleteResponse ist die Antwort auf delete. Sie enthält die Instanzen der Daten, die nicht gelöscht werden konnten.

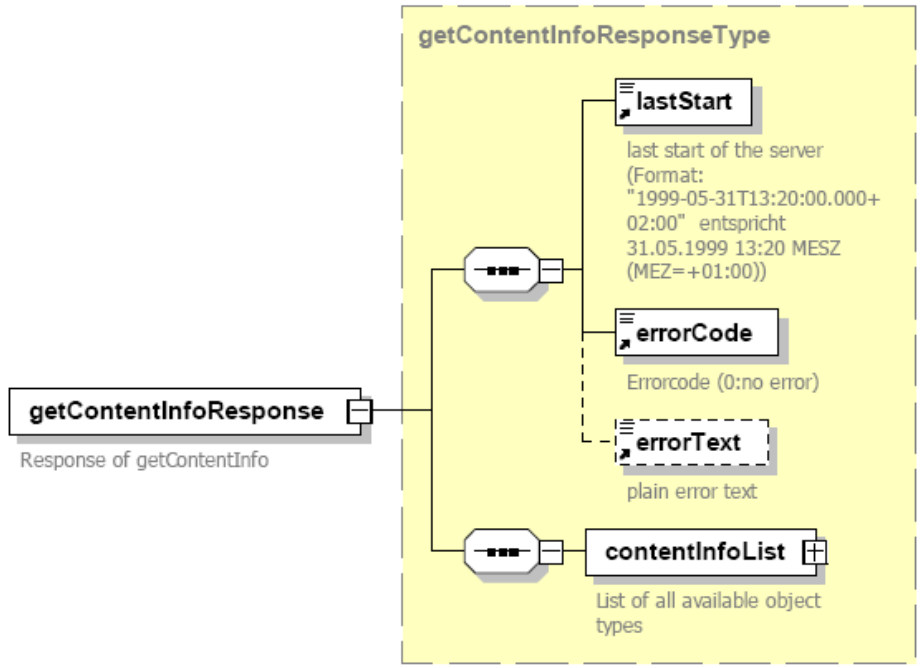


2.5.6 getContentInfo

Methode zum Abfragen von Objektinhalten des Servers. Die Methode hat als Parameter nur den Namen des Clients, das Passwort und optional die Zeit für den „Watchdog“.

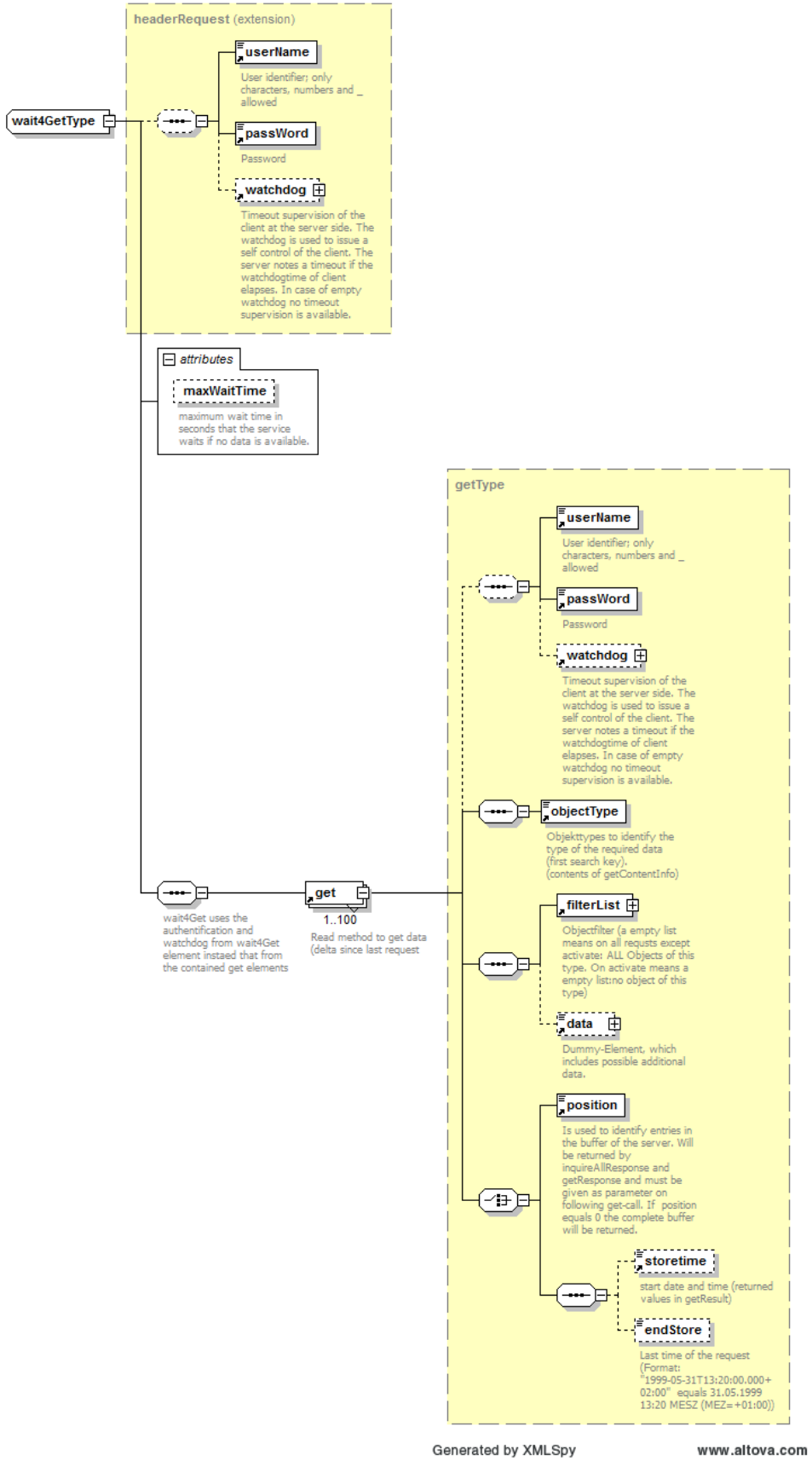


Die Antwort getContentInfoResponse enthält eine Liste der verfügbaren Objekttypen mit ihren Zugriffsrechten und empfohlenen Update-Zyklen.

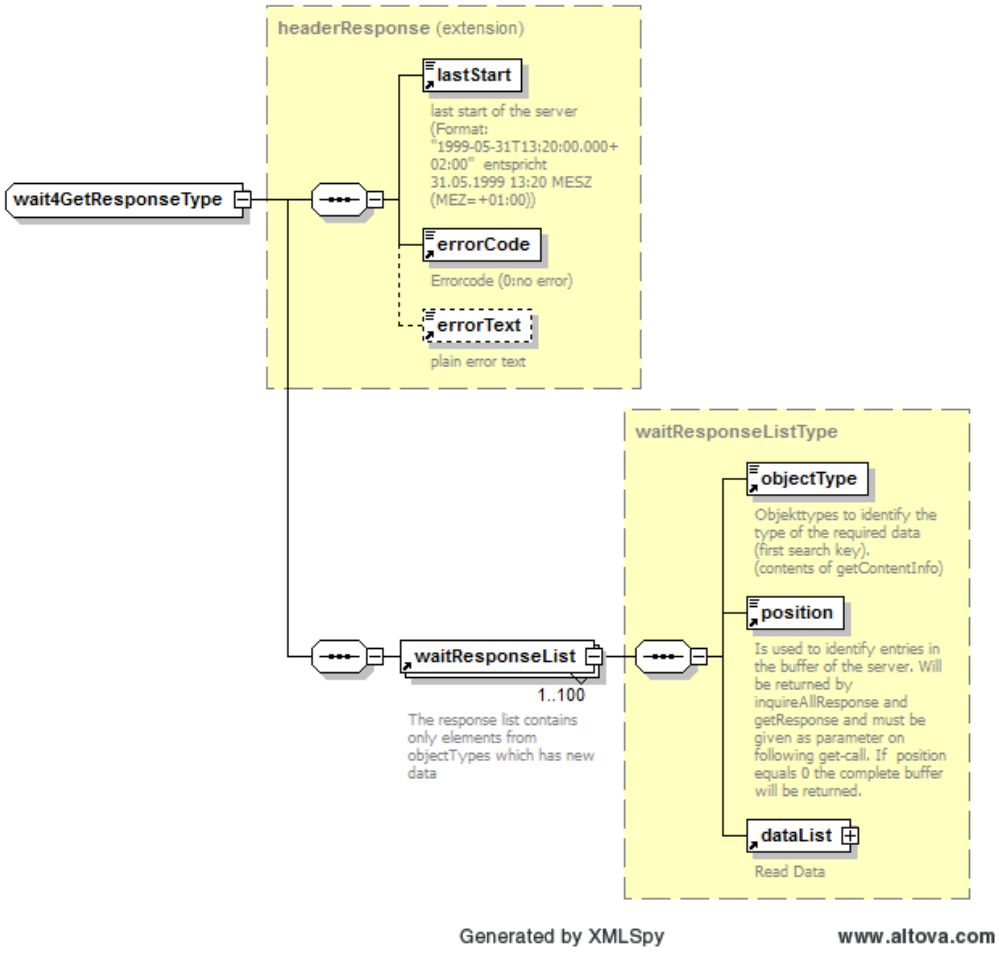


2.5.7 wait4Get

Methode zum Abfragen von Daten des Servers. Die Methode hat als Parameter die gleichen Parameter wie ein get, ermöglicht jedoch die gleichzeitige Abfrage mehrerer Objekttypen.



Wait4GetResponse ist die Antwort mit den Änderungen seit der letzten Abfrage.



2.6 Datenstrukturen

Datenstrukturen, die ins Protokoll eingefügt werden, z. B. die Methoden put, inquireAll-Response usw., werden in eigenen Schemadateien des Protokolls definiert.

2.7 Definition ErrorCodes

Folgende errorCodes sind in der protocol.xsd definiert:

errorCode	Bedeutung
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 Anwendungsempfehlungen

Dieses Kapitel beschreibt, wie Server und Client in verschiedenen Anwendungsfällen gehandhabt werden können. Dies sind nur Empfehlungen, von denen projektspezifisch abgewichen werden kann.

2.8.1 Datenlieferung bei mehreren Abnehmern

In Fällen, bei denen es mehrere Abnehmer von Daten des Servers gibt, die gelegentlich Daten anfordern und keine Echtzeit-Verbindung benötigen, wird die im Folgenden beschriebene Architektur empfohlen.

- Die Datenquelle enthält das sogenannte SOAPServerInterface, welches Funktionalitäten zur Datenpufferung enthält.
  Die Datensenke enthält das sogenannte SOAPClientInterface, welches den Zugriff auf den SOAP Server ermöglicht (Methoden inquireAll und get).

Damit erreicht man dass der Client auf den Server nur bei Verlangen zugreift. Darüber hinaus ist das Protokoll mit geringem Aufwand zu implementieren. Im Falle einer Internetanbindung sind die Datenquelle der Server und ein Internet-Dienst der Client.

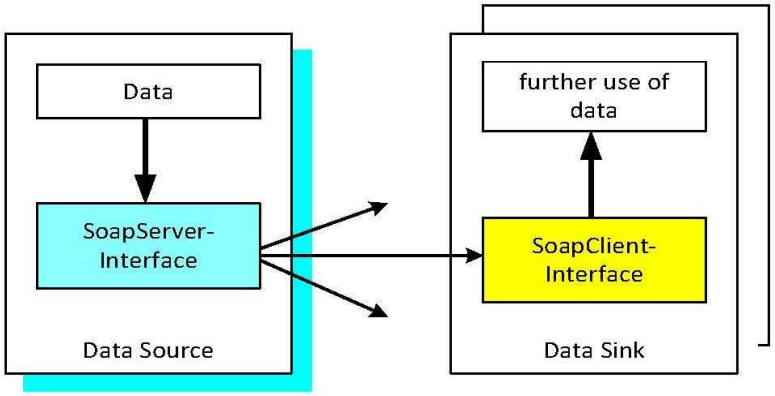


Bild 14: Datenlieferung an mehrere Abnehmer

2.8.2 Konfigurations-Schnittstelle

Falls ein System (als Datenquelle) eine Konfigurations-Schnittstelle an der Datensenke benötigt, empfiehlt sich folgende Architektur:

- Die Datenquelle ist Client
- Die Datensenke ist Server

Eigenschaften:

- Datenübertragung nur auf Verlangen
- Echtzeit Konfiguration, kein „Polling“ erforderlich
- Mehrere Konfigurations-Schnittstellen mit einheitlichem Kommunikations-Interface an einer Datensenke möglich.

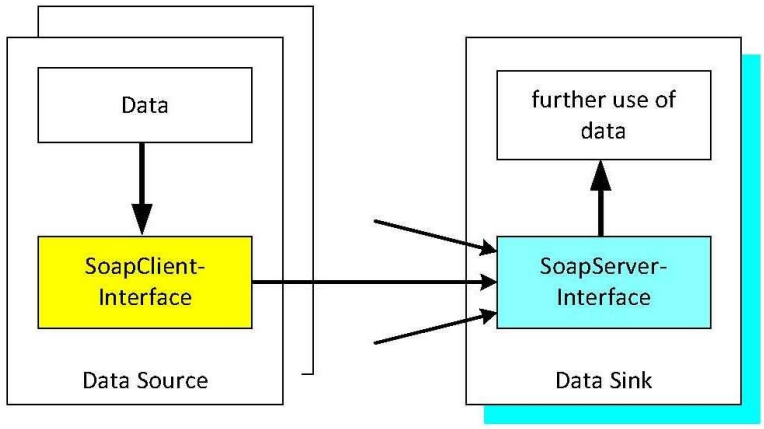


Bild 15: Mehrere Konfigurations-Schnittstellen

Ein Anwendungsfall dafür sind Subsysteme (als Client), die ihre erfassten Daten sofort weitergeben, z.B. Daten über Verkehrsstörungen. Diese Konfiguration soll nur verwendet werden um Überlastungssituationen zu vermeiden.

2.8.3 Daten-Update zwischen zentralen Einrichtungen (unidirektional)

In diesem Fall wird empfohlen:

- Datenquelle ist Server
- Datensenke ist Client

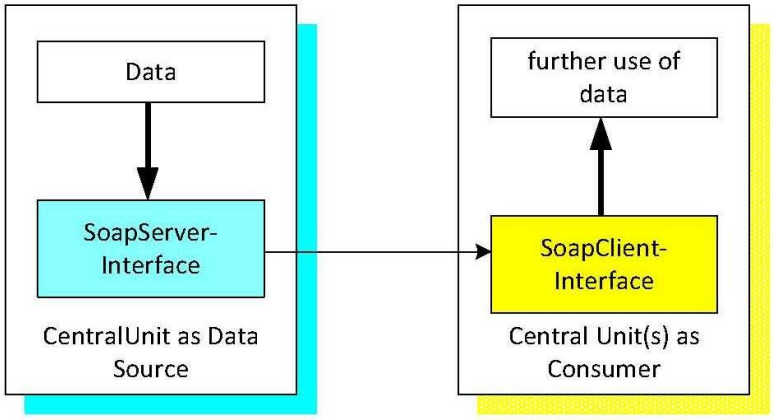


Bild 16: Konfiguration bei Datenlieferung an Client

Praktisch ist dies dieselbe Konfiguration wie bei Datenlieferung bei mehreren Abnehmern.

2.8.4 Daten-Update zwischen zentralen Einrichtungen (bidirektional)

Falls eine bidirektionale Schnittstelle gefordert wird, kann die die Schnittstelle zweimal ausgeführt werden, weil die zentrale Einrichtung gleichzeitig Client und Server ist.

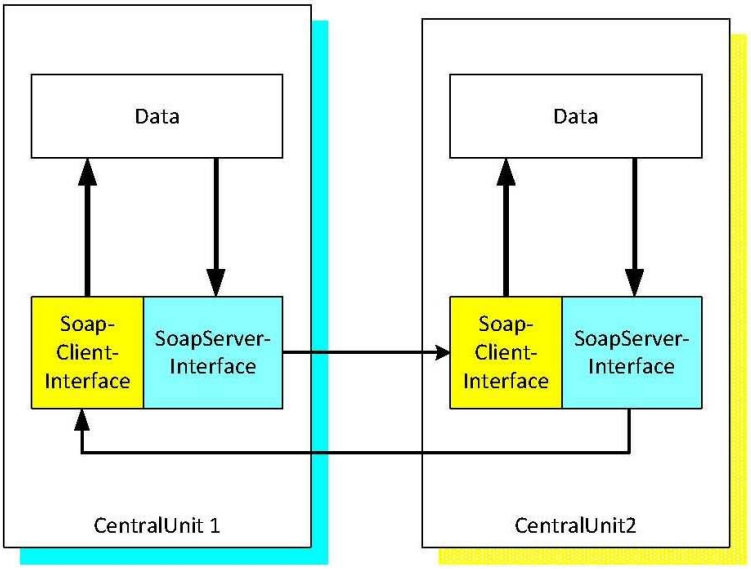


Bild 17: Konfiguration bei Datenaustausch Server - Server

OCIT-C_Protokoll_V1.2_R1