

ICS 35.200

**Open Traffic Systems –
OTS 2-Schnittstellenspezifikation –
Teil 2: Technische Spezifikation für Implementierer**

Open Traffic Systems –
OTS 2 Interface Specification –
Part 2: Technical Specification for Implementers

Open Traffic Systems –
OTS 2 Spécification d'interface –
Partie 2: Spécification technique pour des développeurs

Zur Erstellung einer DIN SPEC können verschiedene Verfahrensweisen herangezogen werden:
Das vorliegende Dokument wurde nach den Verfahrensregeln einer PAS erstellt.

Gesamtumfang 130 Seiten

Bereich Innovation

Inhalt

	Seite
Vorwort.....	5
Einleitung	6
1 Anwendungsbereich	7
2 Normative Verweisungen.....	7
3 Begriffe.....	7
4 Abkürzungen	11
5 Lesehinweise	12
5.1 Externe und interne Schnittstellen	12
5.2 Beschreibungssystematik der OTS-Protokollschichten.....	12
5.3 Sequenzdiagramme, interne und externe Kommunikation	13
5.4 Formatierungen	14
5.5 Schichtenaufbau der Schnittstelle.....	15
5.6 Systemarchitektur	16
5.7 Erweiterungskonzepte	17
5.7.1 Erweiterbarkeit einer Schicht	17
5.7.2 Daten- und Adressierungsschicht	17
5.7.3 Baukastenprinzip.....	17
5.8 Datentypen	17
6 OTS Management-Funktionalität	17
6.1 Konfigurationsmanagement.....	17
6.1.1 Allgemeines	17
6.1.2 Verhalten	18
6.1.3 Ablauf	19
6.1.4 Bindung an Übermittlungsschicht.....	19
6.2 Fehlermanagement.....	20
6.2.1 Allgemeines	20
6.2.2 Methoden	21
6.2.3 Bindung an Transportschicht	22
7 OTS-Transportschicht.....	23
7.1 Allgemeines	23
7.2 Methodenübersicht	24
7.3 Adressierung von OTS-Transport Nutzern	24
7.4 TransportID	25
7.5 OTS-Transportdienst und Aufgabenbereiche der OTS-Transportschicht	26
7.5.1 Sequenzdiagramm.....	26
7.5.2 Zustandsdiagramm	28
7.6 OTS-Transport Protokollbindung	30
7.6.1 Allgemeines	30
7.6.2 Encoding	31
7.6.3 Prozedur: Error Handling.....	32
7.6.4 Prozedur: Passive Open („Server“)	34
7.6.5 Prozedur: Passive Close („Server“)	37
7.6.6 Prozedur: Active Open („Client“).....	38
7.6.7 Prozedur: Message Exchange.....	44
7.6.8 Prozedur: Disconnect	49
7.6.9 Prozedur: Keep Alive	52
8 OTS-Sessionschicht.....	53
8.1 Allgemeines	53
8.2 Methodenübersicht	53
8.3 SessionID	54

8.4	OTS-Sessiondienst und Aufgabenbereiche der OTS-Sessionschicht	55
8.4.1	Sequenzdiagramm	55
8.4.2	Zustandsdiagramm	57
8.5	OTS-Session Protokollbindung	59
8.5.1	Allgemeines	59
8.5.2	Encoding	60
8.5.3	Prozedur: Error Handling	60
8.5.4	Prozedur: Accept Sessions („Server“)	62
8.5.5	Prozedur: Reject Sessions („Server“)	63
8.5.6	Prozedur: Open Session („Client“)	64
8.5.7	Prozedur: Close Session	67
8.5.8	Prozedur: Asynchronous Message Exchange	68
8.5.9	Prozedur: Synchronous Message Exchange	69
8.5.10	Behandlung von allgemeinen Fehlern der OTS-Transportschicht	72
9	OTS-Aktivitätsschicht	72
9.1	Allgemeines	72
9.2	Rollen	72
9.2.1	Allgemeines	72
9.2.2	Subscriber	72
9.2.3	Distributor (für Subscriber)	72
9.2.4	Distributor (für Publisher)	73
9.2.5	Publisher	73
9.2.6	Sender	73
9.2.7	Mediator (für Sender)	73
9.2.8	Mediator (für Receiver)	73
9.2.9	Receiver	73
9.3	Kommunikationsmuster	73
9.4	Methodenübersicht	75
9.5	Aufgaben der Aktivitätsschicht	76
9.5.1	Sequenzdiagramme	76
9.5.2	Zustandsdiagramme	79
9.5.3	Zuordnung OTS 1 und OTS 2	85
9.6	OTS-Aktivitätsschicht Protokollbindung	86
9.6.1	Allgemeines	86
9.6.2	Encoding	90
9.6.3	XML-Schema	90
9.6.4	Prozedur: Error Handling	90
9.6.5	Prozedur: Passive Open („Server“, Verbindungsannahme ermöglichen)	91
9.6.6	Prozedur Passive Close: Ablehnen von Verbindungen	93
9.6.7	Prozedur: Active Open („Client“, Verbindung aufbauen)	94
9.6.8	Prozedur: Close (Verbindung beenden)	97
9.6.9	Prozedur: Query (Stellen einer Anfrage)	98
9.6.10	Prozedur: Subscribe (Bestellen von Daten)	100
9.6.11	Behandlung von allgemeinen Fehlern der OTS-Sessionschicht	107
9.7	Aktivitäts-Inhalte	107
9.7.1	Allgemeines	107
9.7.2	Datenmodelle	107
9.7.3	Grundlegende Unit-basierte Datentypen	108
9.7.4	application	109
9.7.5	query	109
9.7.6	response	110
9.7.7	Subscription	112
9.7.8	data	115
9.7.9	command	116
9.7.10	answer	117
10	Implementierungshinweise	118

	Seite
Anhang A (normativ) Fehlermeldungen	119
A.1 Allgemeines	119
A.2 Fehlermeldungen Transportschicht	119
A.3 Fehlermeldungen Sessionschicht	119
A.4 Fehlermeldungen Aktivitätsschicht	119
Anhang B (informativ) Kataloge	120
Anhang C (informativ) Erweiterbarkeit.....	122
C.1 Baukastenprinzip.....	122
C.2 Erweiterbarkeit durch neue OTS 2-Bausteine.....	123
C.3 Erweiterbarkeit durch neue Protokollbindungen	124
Anhang D (informativ) Schichtübergreifende Darstellung	125
D.1 Verbindungsaufbau (Active Open)	125
D.1.1 Allgemeines zu den Sequenzdiagrammen in Anhang C	125
D.1.2 Verbindungsaufbau (Passive Open/Passive Close).....	127
D.1.3 Verbindungsabbau (Close).....	127
D.1.4 Anfrage (Query)	128
D.1.5 Bestellung (Subscription).....	128
D.1.6 Befehl (Command).....	129
Literaturhinweise.....	130

Vorwort

Diese DIN SPEC nach dem PAS-Verfahren enthält die OTS 2-Schnittstellenspezifikation, die ein Teil des OTS-Rahmenwerks (siehe <http://www.opentrafficsystems.org/>) ist. Zu diesem gehören:

- OTS-Kommunikationsarchitektur, (Protokollspezifikation, Testrahmenwerk, Referenzimplementierung);
- OTS-Instrumente, (System- und Vorgehensmodell, Leitfaden);
- OTS-Prozess, (Leitbild, Anwendungs- und Pflegeprozess, Standardisierung und Zertifizierung).

Wesentliches Ziel der Anwendung der Schnittstellenspezifikation ist der Schritt von der Herstellermischung zur Herstellermischbarkeit.

Es soll der derzeitige Zustand verlassen werden, in dem die Kopplung von Komponenten verschiedener Hersteller mit aufwendigen Integrationsprojekten verbunden ist. Stattdessen wird eine Situation erreicht, in der Komponenten verschiedener Komponenten gekoppelt werden können.

Teil 1 der DIN SPEC 91213 erläutert Entscheidungsträgern den Anwendungsbereich und potenziellen Nutzen bzw. Mehrwert der OTS 2-Schnittstellenspezifikation.

Koordiniert wurde die Erarbeitung des Dokumentes durch die nachfolgend genannten Personen, die allein verantwortlich für den Inhalt sind:

- Dr. rer. nat. Josef Kaltwasser; Christian Lüpkes
AlbrechtConsult GmbH

Verfasser des Dokumentes sind:

- Thilo Schön
GEVAS software GmbH, München
- Daniel Bültmann
RWTH Aachen, Communication Networks ComNets

Die Zusammenarbeit in einem festen Arbeitskreis bestand aus folgenden Unternehmen und Institutionen:

- Anton Bauer
Landeshauptstadt München
- Christoph Engel, Jan Krause
Institut für Automation und Kommunikation e. V. ifak
- Henry Jahnke
Senatsverwaltung für Stadtentwicklung Berlin
Verkehrslenkung Berlin
- Volker Kanngießer
Stadt Frankfurt am Main
Straßenverkehrsamt

- Jörg Lange
Senatsverwaltung für Stadtentwicklung Berlin
Verkehrslenkung LVB
- Frank Leonhardt
Stadtlicht GmbH, Berlin
- Dr.-Ing. Michael Ortgiese
PTV Planung Transport Verkehr AG, Karlsruhe
- Holger Rosier
RWTH Aachen, Communication Networks ComNets
- Thilo Schön
GEVAS software GmbH, München
- Markus Tollmann
Fachbereich Verkehrssteuerung
Freie und Hansestadt Hamburg, Landesbetrieb Straßen, Brücken und Gewässer

Diese DIN SPEC ging aus dem Projekt „OTS 2 – OTS-Kommunikation in Verbundsystemen für Verkehrsinformation und Verkehrsmanagement“ im Rahmen der vom Bundesministerium für Wirtschaft und Technologie (Förderkennzeichen 19 P 7050) geförderten Initiative „VM 2010“ hervor. Die Vorhabensbeschreibung des Projektes beinhaltet ausdrücklich die Dokumentation und Verwertung der F&E-Ergebnisse in Form einer Spezifikation.

Zu diesem Thema bestehen derzeit keine Normen.

DIN Deutsches Institut für Normung e. V.
Bereich Innovation
2010-12

Einleitung

Ziel der Erstellung der OTS 2-Schnittstellenspezifikation war die Entwicklung einer offenen Schnittstelle für den verkehrstechnischen Bereich, die die Herstellermischbarkeit verschiedener Komponenten durch Funktionalität, Interoperabilität, Flexibilität, Erweiterbarkeit und Testbarkeit bestmöglich unterstützt. Für die zu übertragenden Daten wird auf existierende Spezifikationen zurückgegriffen.

Die in dieser Spezifikation zitierten Hilfsmittel zur Implementierung, z. B. xml-Schemadateien, sind auf <http://www.opentrafficsystems.org/> zugänglich. Über diese Webseite können auch Fehlermeldungen und Anmerkungen zu technischen Inhalten, die bei weiteren Bearbeitungen der Spezifikation berücksichtigt werden können, gemeldet werden.

1 Anwendungsbereich

Diese DIN SPEC legt die OTS 2-Schnittstelle zwischen zwei Kommunikationspartnern, die über ein Netzwerkprotokoll miteinander kommunizieren, und deren Implementierung über den OTS 2-Protokollstapel, fest.

Die Schnittstelle dient vorrangig zur Übertragung von Daten aus dem Verkehrsbereich. Sie findet Anwendung in der Kommunikation von Komponenten im Bereich Verkehrsmanagement, -steuerung und -information.

Weitere Anwendungsbereiche sind aufgrund der Modularität und Erweiterbarkeit des spezifizierten Protokollstapels möglich.

Diese DIN SPEC beschreibt den Aufbau eines OTS 2-Protokollstapels, der die externe Schnittstelle zwischen OTS 2-Kommunikationspartnern durch aufeinander aufbauende Schichten realisiert. Die Schichten stellen jeweils verschiedene Funktionalitäten bereit. Hierfür werden Methoden, interne Schnittstellen zwischen den Schichten und das geforderte Verhalten beschrieben.

Die oberhalb des OTS 2-Protokollstapels liegende Anwendungs- und Übermittlungsschicht sind nicht Gegenstand der DIN SPEC. Ebenso wenig werden Datenmodelle für die zu übertragenden Informationen spezifiziert, hierzu wird auf die vorhandenen OCIT-I/OTS und DATEX II Spezifikationen referenziert.

Die Spezifikation richtet sich an Softwareentwickler und andere Personen mit einem fundierten technischen Hintergrundwissen.

2 Normative Verweisungen

Die folgenden zitierten Dokumente sind für die Anwendung dieses Dokuments erforderlich. Bei datierten Verweisungen gilt nur die in Bezug genommene Ausgabe. Bei undatierten Verweisungen gilt die letzte Ausgabe des in Bezug genommenen Dokuments (einschließlich aller Änderungen).

ISO/IEC 10646 *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

3 Begriffe

Für die Anwendung dieses Dokuments gelten die folgenden Begriffe.

3.1

Anfrage

Nachricht in der OTS-Aktivitätsschicht an den Kommunikationspartner, auf die eine einmalige synchrone Rückmeldung erfolgt

BEISPIEL Anfrage nach Objekten, für die Daten verfügbar sind.

3.2

Aktivitätsinhalt

spezieller, über eigene XML-Schemata definierter Inhalt von allgemeinen Methoden der Aktivitätsschicht

BEISPIEL Versendung verschiedener Arten von Befehlen, die jeweils gesondert definiert sind.

3.3

Befehl

Nachricht in der OTS-Aktivitätsschicht an den Kommunikationspartner, auf die als einmalige synchrone Rückmeldung nur eine Empfangsbestätigung (Quittung) erfolgt

BEISPIEL Befehl zum Umschalten einer Variotafel in einen anderen Zustand.

3.4

Bestellung

Nachricht in der OTS-Aktivitätsschicht an den Kommunikationspartner, die mehrfache, asynchrone Rückmeldungen auslöst

BEISPIEL Bestellung von laufend aktualisierten Zustandsdaten bestimmter Objekte.

3.5

Client

Programm, das von anderen (Servern) angebotene Dienste in Anspruch nehmen will und diese dazu aktiv aufruft

3.6

Dienstprimitiv

Realisierung der Kommunikation zwischen benachbarten Schichten am Dienstzugangspunkt

ANMERKUNG Diese bilden die abstrakte Beschreibung einer implementierungsspezifischen API.

3.7

Dienstzugangspunkt

implementierungsunabhängige Schnittstelle, an der zwei Schichten lokal miteinander kommunizieren

ANMERKUNG Eine untere Schicht erbringt ihren Dienst einer höheren Schicht am Dienstzugangspunkt.

3.8

Feature

versionierte Funktionalität oder (optionale) Erweiterung innerhalb einer Schicht des Protokolls

ANMERKUNG Siehe auch Konfigurationsinformationen.

3.9

Initialisierungszustand

vollständige Zustandslieferung bezüglich der gewünschten Datenarten zu den gewünschten Objekten als erste Lieferung zu einer Bestellung

ANMERKUNG Der Initialisierungszustand umfasst die aktuellen Werte aller bestellten Zustandsvariablen, auch wenn sich diese Werte seit längerer Zeit nicht geändert haben sollten.

3.10

Kommunikationsmuster

Charakterisierung einer Kommunikationsverbindung in der OTS-Aktivitätsschicht zwischen zwei Partnern, die jeweils eine bestimmte Rolle einnehmen

ANMERKUNG In einer Verbindung hat jeder der beiden Partner eine feste Rolle. Wollen die zugehörigen Programme auch in anderen Rollen kommunizieren, eröffnen sie zusätzliche Verbindungen.

3.11

Konfigurations-Abgleich

Austausch der benötigten bzw. beherrschten Features der Kommunikationspartner beim Verbindungsaufbau, woraus sich die verwendete gemeinsame Schnittmenge ergibt

3.12

Konfigurations-Vorabfrage

Ermittlung der Konfigurationsinformationen eines potenziellen Kommunikationspartners über eine einfache Abfrage ohne Verbindungsaufbau

3.13

Methode

abstrakte Beschreibung einer implementierungsspezifischen API

ANMERKUNG Eine Methode kann an einem Dienstzugangspunkt aufgerufen werden. Im ISO/OSI Referenzmodell wird von einem Dienstprimitiv gesprochen. In jedem Fall handelt es sich um die abstrakte Beschreibung einer implementierungsspezifischen API.

3.14

Nachricht

über eine Kommunikationsverbindung gesendetes Datenpaket

ANMERKUNG auch Bezeichnung von Datenpakete für solche Nachrichten, die für die betrachtete Protokollschicht keine spezielle Bedeutung haben, sondern beliebige Nutzdaten enthalten.

3.15

Open Traffic System

durchgängiges, einheitliches Architektur- und Schnittstellenkonzept für Systemlandschaften zur Verkehrssteuerung und zum Verkehrsmanagement; umfasst eine Methode, mit dem Baulasträger Anforderungen an (herstellergemischte) Systeme spezifizieren und prüfen können

ANMERKUNG Die OTS 1-Schnittstelle für Prozessdaten entspricht OCIT-I mit OTEC-Erweiterungen für Befehle, Betriebsmeldungen und Verkehrsmanagement-Daten.

3.16

Protocol Data Unit

Datenpaket, das zwischen Kommunikationspartnern derselben Schicht ausgetauscht wird

ANMERKUNG Es enthält sowohl Nutzdaten der höheren Schicht, als auch Protokollsteuerinformationen der eigenen Schicht.

3.17

Protokollbindung

Verbindung mit dem Dienstzugangspunkt der unteren Schicht, die die Dienste der darüberliegenden Schicht realisiert

ANMERKUNG Weiterhin beschreibt die Protokollbindung die Struktur und das Format der Protokolldateneinheiten (Protocol Data Units, PDU), die die Schicht über die Dienste der unteren Schicht mit Ihrem Kommunikationspartner austauscht.

3.18

Provider-Port

Dienstzugangspunkt einer Schicht, an dem definierte Methoden für den Aufruf durch einen Nutzer bzw. eine höhere Schicht zur Verfügung gestellt werden

3.19

Prozedur

definierte Teilmenge der Kommunikationsabläufe im Rahmen der Schichtspezifikation

BEISPIEL Eine (Kommunikations-)Prozedur kann das Öffnen einer Verbindung, das Übertragen von Nachrichten oder das Schließen einer Verbindung sein. Alle Prozeduren zusammen genommen erbringen in ihrer Gesamtheit den Dienst der Schicht.

3.20

Rolle

Art der gewünschten Kommunikation

BEISPIEL Beispiele für Rollen sind Subscriber, Publisher oder Distributor.

ANMERKUNG Eine Rolle in der OTS-Aktivitätsschicht wird definiert durch einen Satz von verfügbaren Methoden und ein bestimmtes Verhaltensmuster, welches über ein Zustandsdiagramm festgelegt wird.

3.21

Rückrufobjekt

Objekt in einer Implementierung, welches die Methoden des User-Ports der nächsttieferen Schicht implementiert

ANMERKUNG Beim Aufruf einer Provider-Port-Methode dieser Schicht kann eine Referenz auf das Rückrufobjekt übergeben werden, welches dann asynchrone Rückmeldungen als Aufruf seiner User-Port-Methoden verarbeiten kann.

3.22

Übermittlungsschicht

Basis der OTS-Schnittstelle über ein standardisiertes Netzwerkprotokoll, hier SOAP und Socket-Verbindungen auf TCP-Basis

3.23

OTS-Transportschicht

OTS-Schicht für Datenübertragung und zur Abstraktion der Übermittlungsschicht

3.24

OTS-Sessionschicht

OTS-Schicht der Sitzungsverwaltung

3.25

OTS-Aktivitätsschicht

OTS-Schicht mit anwendungsbezogenen Funktionalitäten

3.26

OTS-Daten- und -Adressierungsschicht

Schicht, die Datenarten und Objektadressierung enthält

ANMERKUNG Das betrifft die Inhalte der von den Aktivitäten der Aktivitätsschicht ausgetauschten Datenpakete; sie wird übernommen von OTS 1/OCIT-I PD und ergänzt um das DATEX II-Modell.

3.27

Anwendungsschicht

Anwender der OTS-Aktivitätsschicht und der OTS-Daten- und -Adressierungsschicht

ANMERKUNG Die Anwendungsschicht ist keine standardisierte Schicht der OTS-Schnittstelle.

3.28

Server

Programm, das bestimmte Dienste anbietet und dazu passiv auf eingehende Aufrufe (von Clients) wartet

3.29

Session

Verbindung zwischen Quelle und Ziel oberhalb der Transportschicht, über die Nachrichten ausgetauscht werden können

ANMERKUNG Mit einer Session können Zustände der Anwendung in einem Kommunikationspartner verknüpft werden. Diese Zustandsdaten haben über die Lebensdauer einer Transportverbindung Bestand. Eine Session kann mehrere Transportverbindungen umfassen (Erweiterung multiple Transportverbindungen), eine Transportverbindung kann aber nur einer Session zugeordnet sein.

3.30

Transportkanal

Kanal, der angibt, welches Protokoll auf der Übermittlungsschicht verwendet wird

BEISPIEL SOAP/HTTP, TCP/Socket

3.31

Transportverbindung

Verbindung zwischen einer Quelle und einem Ziel auf einem bestimmten Transportkanal

3.32**Uniform Resource Identifier**

Addressierung, bestehend aus Schema-Feld zur Identifizierung der Protokollbindung und Schema-spezifischem Teil zur Identifizierung des anzusprechenden Objekts

3.33**User-Port**

Dienstzugangspunkt einer Schicht, an dem definierte Methoden von einem Nutzer bzw. einer höheren Schicht zur Verfügung gestellt werden müssen, um z. B. für Rückmeldungen aufrufbar zu sein

3.34**Verbindung**

Sitzung in der Aktivitätsschicht mit zwei Kommunikationspartnern, in der diese jeweils eine feste Rolle haben

ANMERKUNG Wollen die zugehörigen Programme auch in anderen Rollen kommunizieren, eröffnen sie zusätzliche Verbindungen.

3.35**Verbindungsaufbau**

Vorgang, bei dem zwei Kommunikationspartner eine Kommunikationsverbindung herstellen, wobei ein Konfigurationsabgleich stattfindet, und diese offen halten bzw. sich die Verbindungsparameter merken, wenn die nächsttiefere Schicht nicht verbindungsorientiert arbeitet

3.36**aktiver Verbindungsaufbau**

Versuch eines Programmes, von sich aus eine Verbindung aufzubauen

3.37**passiver Verbindungsaufbau**

Reaktion eines Programmes auf eingehende Verbindungswünsche

4 Abkürzungen

API	Application Programming Interface (Programmierschnittstelle)
http	Hypertext Transfer Protocol (zustandsloses Protokoll zur Datenübertragung in Netzwerken)
ID	Eindeutige Identifikationsnummer
LSA	Lichtsignalanlage
OCIT	Open Communication Interface for Road Traffic Control Systems
OSI	Open Systems Interconnection Reference Model (Referenzmodell für das Design von Netzwerk-Kommunikationsprotokollen)
OTS	Open Traffic Systems
PDU	Protocol Data Unit; Protokolldateneinheiten
SID	SessionID zur eindeutigen Identifikation einer Session
SOAP	Simple Object Access Protocol (Protokoll zur Realisierung entfernter Methodenaufrufe)
SSL	Secure Sockets Layer

TCP	Transmission Control Protocol (Netzwerkprotokoll zur Herstellung einer Verbindung zwischen zwei Endpunkten)
TID	TransportID zur eindeutigen Identifikation einer Transportverbindung
TLS	Transport Layer Security
URI	Uniform Resource Identifier
WSDL	Webservice Description Language (XML-Beschreibungssprache für Webservices)
XML	Extensible Markup Language
XSD	XML-Schemadatei

5 Lesehinweise

5.1 Externe und interne Schnittstellen

Die in dieser DIN SPEC definierte externe OTS 2-Schnittstelle dient der Kommunikation zwischen verschiedenen Applikationen bzw. Implementierungen auch von verschiedenen Herstellern. Sie wird beschrieben über XML-Schemadateien (XSD), welche die jeweilige Protocol Data Unit (PDU) (ausgetauschte Nachrichten) definieren, und eine WSDL-Datei.

Weiterhin ist im vorliegenden Dokument der Aufbau eines OTS 2-Protokollstapels beschrieben, der die definierte externe Schnittstelle durch aufeinander aufbauende Schichten realisiert, die jeweils verschiedene Funktionalitäten bereitstellen. Hierfür werden Methoden, Abläufe und interne Schnittstellen zwischen den Schichten beschrieben.

Die genaue Realisierung der internen Schnittstellen ist nicht festgeschrieben. Daher sind hier keine festen Datentypen vorgegeben. Ein Austausch einzelner Schichten oder Erweiterungsmodule zwischen verschiedenen Implementierungen ist nicht vorgesehen.

Die Existenz der verschiedenen Schichten und ihrer Methoden ist Voraussetzung für die leichte Erweiterbarkeit einer Protokollimplementierung und für die vollständige Durchführbarkeit von Konformitätstests (mit Hilfe eines Testadapters, der die Eigenheiten der jeweiligen Implementierung kapselt). Die Implementierung der für die einzelnen Methoden beschriebenen Abläufe stellt sicher, dass auch die Abläufe auf der externen Schnittstelle standardkonform sind.

Zum Unterschied der externen und internen Schnittstelle siehe auch Bild 1.

5.2 Beschreibungssystematik der OTS-Protokollschichten

Jede Schicht wird eigenständig beschrieben. Ein Rückgriff auf die Funktionen der jeweils tiefer liegenden Schicht findet statt. Die dadurch entstehenden Aufrufketten werden in der Spezifikation nicht aufgelöst, da dies für die Beschreibung und Sicherstellung der Funktionalität nicht nötig ist, solange jede Schicht die von ihr anbietenden Funktionalitäten gegenüber dem Aufrufer (der höheren Schicht) garantiert.

Methoden der internen Schnittstellen werden durch einen Funktionskopf in spezieller Formatierung dargestellt. Rückgabewerte von Methoden sind mit einem Pfeil → gekennzeichnet.

Beispiel:

SSyncMessage (

 sessionID SessionID der Session, für die eine Nachricht verschickt werden soll

 message Inhalt der Nachricht

) → (

 returnMessage Inhalt der Antwort-Nachricht

)

Für die externe Schnittstelle wird festgelegt, welche PDU, d. h. Nachrichten, die über einen Kommunikationskanal übertragen werden, mit dem Kommunikationspartner ausgetauscht werden. Das Format und der Inhalt der Nachrichten sind spezifisch für die jeweilige Protokollbindung und werden daher auch eigens für jede Bindung spezifiziert.

Die Spezifikation der PDU erfolgt durch XML-Schemata (XSD Dateien). Das XML-Schema für die jeweils betrachtete PDU wird im Dokument abgebildet.

5.3 Sequenzdiagramme, interne und externe Kommunikation

Bei den in dieser DIN SPEC eingebundenen Sequenzdiagrammen ist zu beachten, dass ihre Lebenslinien oft zwei Protokollschichten in jeweils zwei Instanzen darstellen.

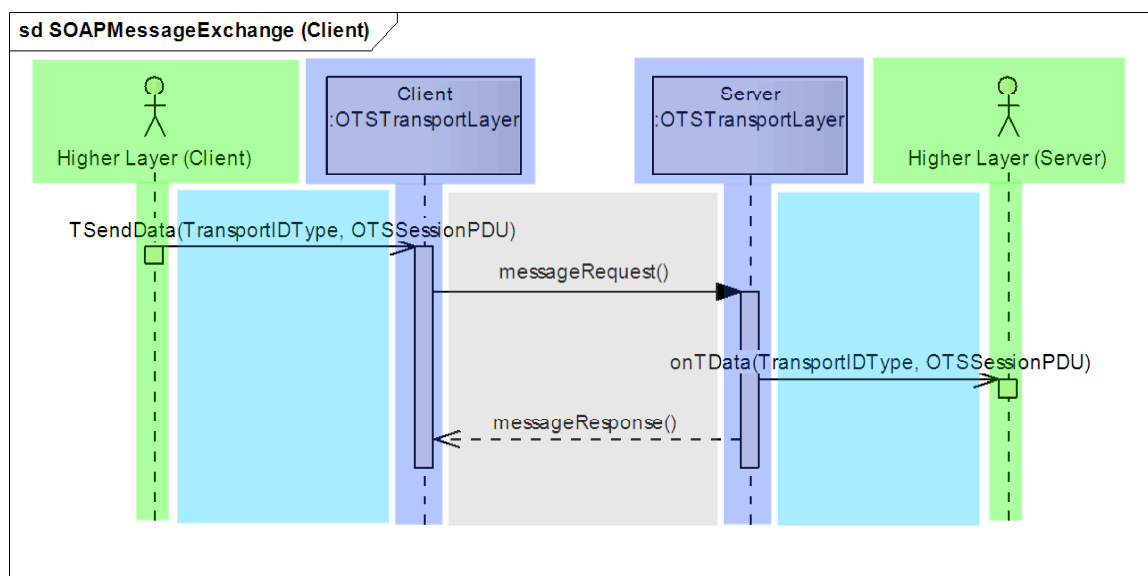


Bild 1 — Beispiel für interne und externe Kommunikation im Sequenzdiagramm

BEISPIEL In Bild 1 wird ein Kommunikationsablauf gezeigt, den die OTS-Transportschicht unterstützt. Die OTS-Transportschicht tritt in zwei Instanzen (Client und Server) auf, die durch die beiden mittleren Lebenslinien dargestellt werden (im Beispiel dunkelblau markiert).

Zwischen diesen beiden Instanzen (die zwei Installationen oder zwei Implementierungen entsprechen können) findet die externe Kommunikation mittels der externen Schnittstelle statt (grau markiert).

Jede Instanz der OTS-Transportschicht ist in einen Protokollstapel eingebunden und wird von einer höher angesiedelten Schicht verwendet. Diese höheren Schichten werden durch die beiden äußeren Lebenslinien dargestellt (Higher Layer auf Client- und Server-Seite, grün markiert).¹⁾

Zwischen der jeweils übergeordneten Schicht und der darunterliegenden Schicht findet eine *interne Kommunikation* mittels der für die Schichten definierten (aber nicht öffentlichen) *Methoden* statt (türkisblau markiert). Diese Kommunikation läuft normalerweise innerhalb eines Prozesses (einer Implementierung) ab.

5.4 Formatierungen

In dieser DIN SPEC werden zur besseren Lesbarkeit spezielle Formatierungen verwendet.

Literale, Dateinamen, Parameter u. a. werden durch eine Schriftart mit fester Laufweite gekennzeichnet.

Methodennamen werden wie Literale gekennzeichnet, aber zusätzlich mit **Fettdruck** und unterschiedlicher Einfärbung je nach Zugehörigkeit:

- Für die Transportschicht dunkelblaue Farbe
- Für die Sessionschicht grüne Farbe
- Für die Aktivitätsschicht rote Farbe
- Für die Aktivitätsinhalte braune Farbe

Protokolldateneinheiten (PDU) und ihre Parameter werden blau eingefärbt.

Allgemeine Hervorhebungen sind kursiv gekennzeichnet.

Fehler- und Ausnahmebehandlungen werden durch einen rötlichen Kasten mit gestrichelter Umrandung gekennzeichnet.

Verweise auf zusätzliche externe Dateien im Text (XML, Schemadateien) werden durch einen gelben Kasten mit durchgehender Umrandung gekennzeichnet.

1) In einer vollständigen OTS 2-Implementierung wird die der OTS-Transportschicht übergeordnete Schicht die OTS-Sessionschicht sein. Damit die Spezifikation einer Schicht aber nicht von der darüberliegenden Schicht abhängig wird, wird auf diese explizite Benennung verzichtet.

5.5 Schichtenaufbau der Schnittstelle

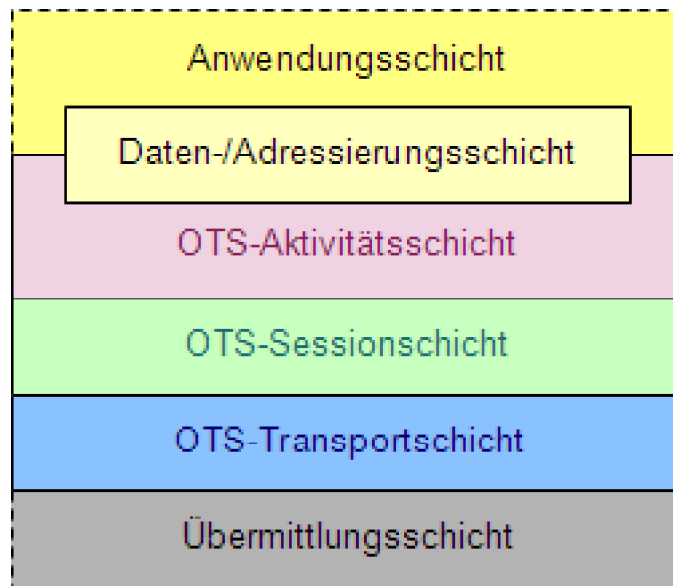


Bild 2 — Schichtenaufbau des OTS 2-Protokollstapels

Die Schnittstelle wird wie in Bild 2 dargestellt in folgenden *Schichten* aufgebaut:

- 1) Übermittlungsschicht
(standardisiertes Netzwerkprotokoll, hier SOAP- und Socket-Verbindungen auf TCP-Basis)
- 2) OTS-Transportschicht
(Datenübertragung, Abstraktion der Übermittlungsschicht)
- 3) OTS-Sessionschicht
(Sitzungsverwaltung, synchrone und asynchrone Nachrichten)
- 4) OTS-Aktivitätsschicht
(anwendungsbezogene Funktionalitäten wie Bestellungen oder Befehle)
- 5) OTS-Daten- und -Adressierungsschicht
(Datenarten und Objektadressierung, betrifft die Inhalte der von den Aktivitäten der Aktivitätsschicht ausgetauschte Datenpakete; übernommen von OCIT-I PD)
- 6) Anwendungsschicht
(nicht standardisiert, wird als Anwender der Aktivitätsschicht und der Daten- und Adressierungsschicht betrachtet)

Anwendungs- und Übermittlungsschicht sind nicht Gegenstand der OTS-Spezifikation.

Die Daten- und Adressierungsschicht wird aus OTS 1 übernommen und durch das DATEX II-Datenmodell ergänzt.

ANMERKUNG Eine direkte Kommunikation zwischen einer Quelle und einem Ziel findet auf der Übermittlungsschicht statt, indirekt kommunizieren jeweils die zugehörigen Schichten beider Seiten miteinander, wie in Bild 3 dargestellt.

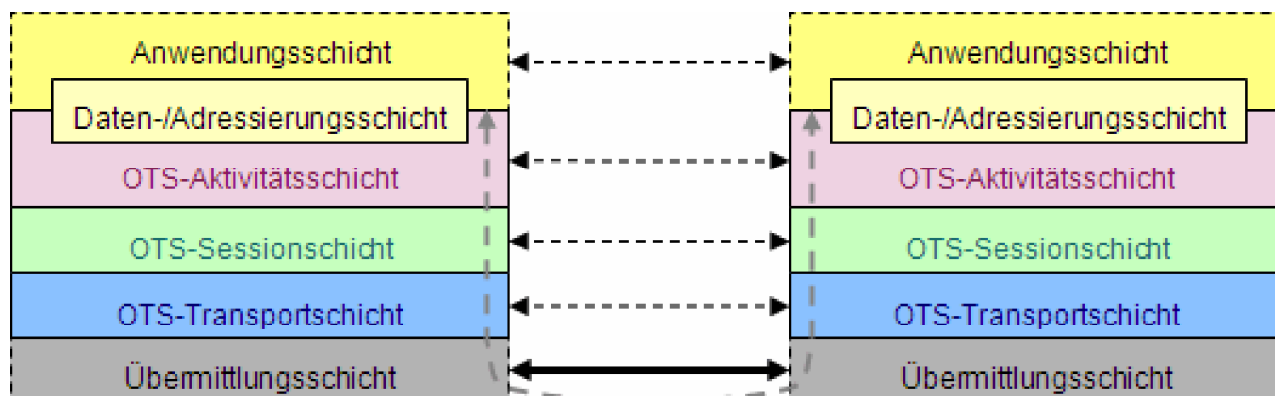


Bild 3 — Kommunikation der Schichten

5.6 Systemarchitektur

OTS 2-Systeme werden durch Auswahl geeigneter Protokollbausteine realisiert. Diese DIN SPEC spezifiziert die Schichten und Bausteine OTS-Transportschicht (englisch OTS-Transport), OTS-Sessionschicht (OTS-Session) und OTS-Aktivitätsschicht (OTS-Activity), wobei letztere verschiedene Unterbausteine (Kommunikationsrollen) zur Verfügung stellt. Die Aktivitätsschicht stellt ein Publish-Subscribe- und eine Send-Receive-Kommunikationsmuster zur Verfügung.

BEISPIEL Rechner einer Systemlandschaft können verschiedene Bausteine einsetzen, um ihren Beitrag zum Verbundsystem zu realisieren. Eine beispielhafte Systemlandschaft ist in Bild 4 dargestellt.

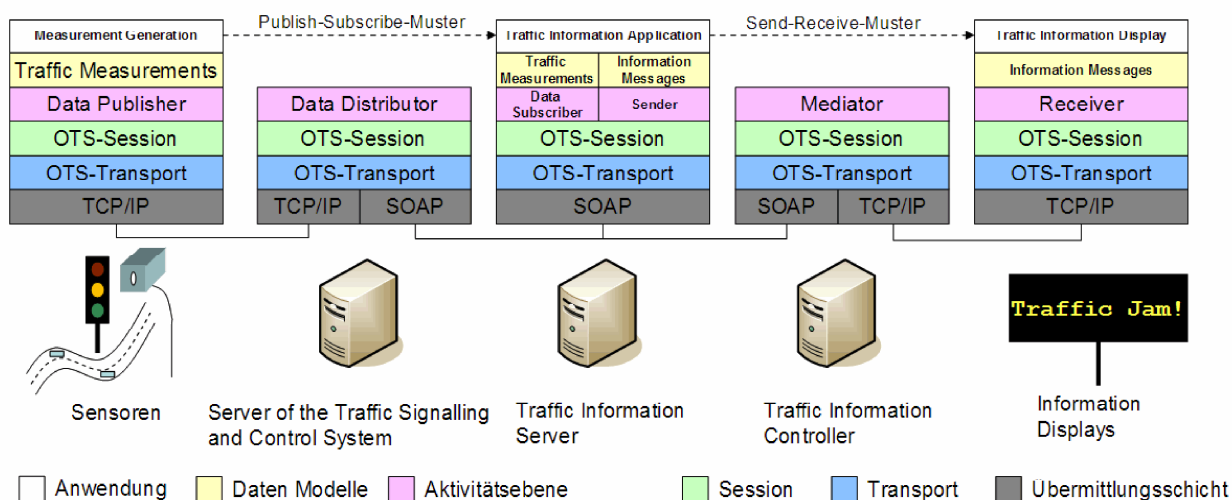


Bild 4 — OTS 2-Systemarchitektur

5.7 Erweiterungskonzepte

5.7.1 Erweiterbarkeit einer Schicht

Die vorhandenen Schichten können durch die Erweiterung der angebotenen Dienste erweitert werden.

Um die Kompatibilität des Gesamtsystems sicherzustellen, müssen die Schichten für die Dauer einer Migration über Abwärtskompatibilität verfügen, d. h. eine Schicht die einen erweiterten Dienst realisiert, muss in der Lage sein mit einer Schicht, die den erweiterten Dienst noch nicht realisiert, zu kommunizieren.

Dazu werden die verschiedenen Versionen einer Schicht mit Versionsnummern versehen. Durch Konfigurationsabfragen vor oder während eines Verbindungsaufbaus wird dann ein geeigneter Modus der Kommunikation (kompatible Versionen) ausgewählt. Auf diese Weise können auch Implementierungen unterschiedlicher Versions- oder Erweiterungsstände miteinander kommunizieren, indem sie sich automatisch auf die gemeinsame Schnittmenge verständigen. Siehe dazu auch 6.1.

5.7.2 Daten- und Adressierungsschicht

Die Datenschicht kann sehr einfach erweitert werden, indem neue Datenarten definiert werden. Die Vorgehensweise hierzu wird aus OTS 1 (OCIT-I PD-DM, Abschnitt 4) übernommen. Wenn DATEX II-Datenarten verwendet werden, gelten die entsprechenden DATEX II-Regelungen.

Erweiterungen in der Adressierung (der Daten) entsprechen einer Erweiterung der Aktivitätsschicht im Bereich der Aktivitätsinhalte.

5.7.3 Baukastenprinzip

Die Möglichkeit zur Erweiterung durch Hinzufügen von neuen OTS 2-Bausteinen und Protokollbindungen ist Bestandteil des OTS 2-Protokollstapels. Normative Regelungen zur Durchführung von Erweiterungen auf dieser Ebene sind noch zu treffen. Die Grundprinzipien werden in Anhang C beschrieben.

5.8 Datentypen

Verwendete Datentypen beziehen sich auf Typen aus der XML Schema Spezifikation des W3C. Diese kann als Namespace (xsd:) referenziert werden mit der URI <http://www.w3.org/2001/XMLSchema>.

6 OTS Management-Funktionalität

6.1 Konfigurationsmanagement

6.1.1 Allgemeines

Konfigurationsinformationen werden schichtübergreifend verwendet, um Informationen über benötigte oder unterstützte Funktionalitäten und Versionen der Schnittstellenimplementierungen auszutauschen. Sie bestehen aus einer Liste mit unterstützten Versionen der verschiedenen Protokollschichten, sowie deren unterstützten Erweiterungen.

Die unterstützten Funktionalitäten oder Erweiterungen, im folgenden Feature genannt, werden jeweils mit Name und minimaler und maximaler Versionsnummer angegeben:

- Der *Name* wird als String angegeben (xsd:string). Im Namensstring werden kleingeschriebene Bezeichner in Englisch verwendet, möglichst kurz, aber aussagekräftig. Untergeordnete Erweiterungen werden im Namensstring mit einem Unterstrich '_' gekennzeichnet. Als Kennzeichnung der Schicht, auf die sich die Bezeichnung bezieht, wird „t_“ (Transportschicht), „s_“ (Sessionschicht) oder „a_“ (Aktivitätsschicht) vorangestellt, die Version der entsprechenden Protokollschicht selbst wird mit „layer“ gekennzeichnet. Beispiele: „t_layer“, „t_feature_subfeature“.
- Die *minimale* und *maximale Versionsnummer* sind Ganzzahlen (xsd:int) größer gleich 1.

Bei jeder Änderung eines Features, die nur genutzt werden kann, wenn beide Seiten sie beherrschen, ist die Nummer zu erhöhen.

6.1.2 Verhalten

Folgendes Verhalten in der Schnittstelle wird festgelegt:

- Die verschiedenen Schichten einer Implementierung hinterlegen Konfigurationsinformationen über *alle* beherrschten Features und Versionen in einem *Konfigurationsobjekt*.
- Erster Anwendungsfall (Konfigurations-Vorabfrage):
 - Die Konfigurationsinformationen werden über eine einfache Abfrage (ohne Verbindungsaufbau) von anderen (potenziellen) Kommunikationspartnern ermittelt.
- Zweiter Anwendungsfall (Konfigurations-Abgleich):
 - Beim Verbindungsaufbau übermittelt (auf jeder Schicht) die Seite, die den Kontakt aufnimmt (Quelle), die von ihr für die konkrete Verbindung angeforderten Features jeweils mit minimaler und maximaler unterstützter Version. Dies kann eine Teilmenge der tatsächlich beherrschten Features sein.
 - Die empfangende Seite (Ziel) wertet die Informationen aus und gleicht sie mit den eigenen Fähigkeiten ab. Es wird der bestmögliche gemeinsame Nenner gebildet und in der entsprechenden Antwort versendet. Beherrschen beide Seiten eine Erweiterung, aber in unterschiedlichen Versionen, wird die höchste Version verwendet, die beide beherrschen.²⁾
 - Beherrscht eine Seite eine Erweiterung nicht oder überschneiden sich die verstandenen Versionsbereiche nicht, so kann die Verbindung nicht aufgebaut werden (falls sie aus technischen Gründen schon besteht, wird sie automatisch wieder geschlossen) und es wird ein entsprechender Fehler gemeldet.
 - Die vom Ziel gemeldete Konfiguration (minimale = maximale Version = verwendete Version) entspricht der im weiteren Ablauf verwendeten.
- Ob eine Versionsänderung kompatibel oder inkompatibel ist, muss die jeweilige Implementierung selbst festlegen.³⁾

2) Beispielsweise könnte die Quelle mitteilen, dass sie ein Feature in den Versionen 5 bis 10 beherrscht (und benötigt). Das Ziel beherrscht das Feature in den Versionen 8 bis 12 und meldet daraufhin zurück, dass die Version 10 verwendet wird.

3) Wird beispielsweise in einer Funktion ein optionales Element zusätzlich eingeführt, so hängt es von der Implementierung ab, ob dieses einfach ignoriert oder darüber gestolpert wird, dass es im alten Schema nicht vorgesehen ist. Dadurch ist es schwer, allgemein festzulegen, ob eine Änderung kompatibel ist.

6.1.3 Ablauf

Bei der Konfigurations-Vorabfrage werden alle Konfigurationsinformationen aller Schichten gemeinsam in einem XML-String des Typs `ConfigListType` übertragen (siehe 6.1.3).

Beim Konfigurations-Abgleich während des Verbindungsaufbaus werden die Konfigurationen Schicht für Schicht verglichen:

- In jeder Schicht werden beim passiven und aktiven Verbindungsaufbau (Passive und Active Open) die gewünschten Konfigurationsparameter als Liste nur für die jeweilige Schicht angegeben. Der Benutzer der Schicht gibt dabei nur wahlfreie Features an und setzt deren Versionsnummern auf 0 (= beliebig), da die Aushandlung der Versionsnummern und der verpflichtenden Features mit dem Kommunikationspartner die Schicht selbstständig übernimmt. Die Angabe dieser verkürzten Featurelisten geschieht in folgenden Methoden:
 - Für die Transportschicht in **TAccept** (7.6.4.2) und **TConnect** (7.6.6.2);
 - Für die Sessionschicht in **SAccept** (8.5.4.1) und **SOpen** (8.5.6.1);
 - Für die Aktivitätsschicht in **AAccept** (9.6.5.2) und **ATie** (9.6.7.2).
- Beim Active Open wird vom aktiven Kommunikationspartner (Client) zum passiven (Server) die vollständige Liste aller verpflichtenden und benötigten Features inklusive beherrschter Versionsnummern übermittelt (wiederum XML-Typ `ConfigListType`).
- Der Empfänger (Server) behandelt die empfangenen Konfigurationslisten pro Schicht separat, aber immer gleichartig. Die jeweilige Liste wird intern mit der vollständigen und versionierten Liste der eigenen Fähigkeiten unter Berücksichtigung der beim Passive Open angegebene wahlfreie Features abgeglichen (s.o., 6.1.2). Daraus entsteht eine neue, abgeglichene Liste von Features.
- Die abgeglichene Liste legt die verwendete Konfiguration in der zu öffnenden Verbindung (Transportverbindung, Session, Verbindung auf Aktivitätsschicht-Ebene) fest.
- Die abgeglichene Liste wird als Antwort auf die Verbindungsanfrage an den aktiven Kommunikationspartner (Client) zurückgeschickt:
 - Für die Transportschicht im Rückgabewert `tConnectRType` (7.6.6.2);
 - Für die Sessionschicht in der *openSessionResponsePDU* (8.5.6.1);
 - Für die Aktivitätsschicht in der *tieResponsePDU* (9.6.7.2).
- Server und Client bekommen die abgeglichene Liste in einer reduzierten Form in den Callback-Methoden zurückgemeldet, die ihn über den erfolgreichen Verbindungsaufbau informieren. Dabei wird nur die Schnittmenge aus den beim Verbindungsaufbau angegebenen wahlfreien Features und der abgeglichenen vollständigen Liste zurückgemeldet, und zwar in den folgenden Methoden:
 - Für die Transportschicht **onTAccepted** (Server) und **onTConnected** (Client);
 - Für die Sessionschicht **onSAccepted** (Server) und **onSOpened** (Client);
 - Für die Aktivitätsschicht **onAAccepted** (Server) und **onATied** (Client).

6.1.4 Bindung an Übermittlungsschicht

Standardcodierung für eine Liste von Konfigurationsinformationen ist ein XML-String nach dem Schema in Bild 5.

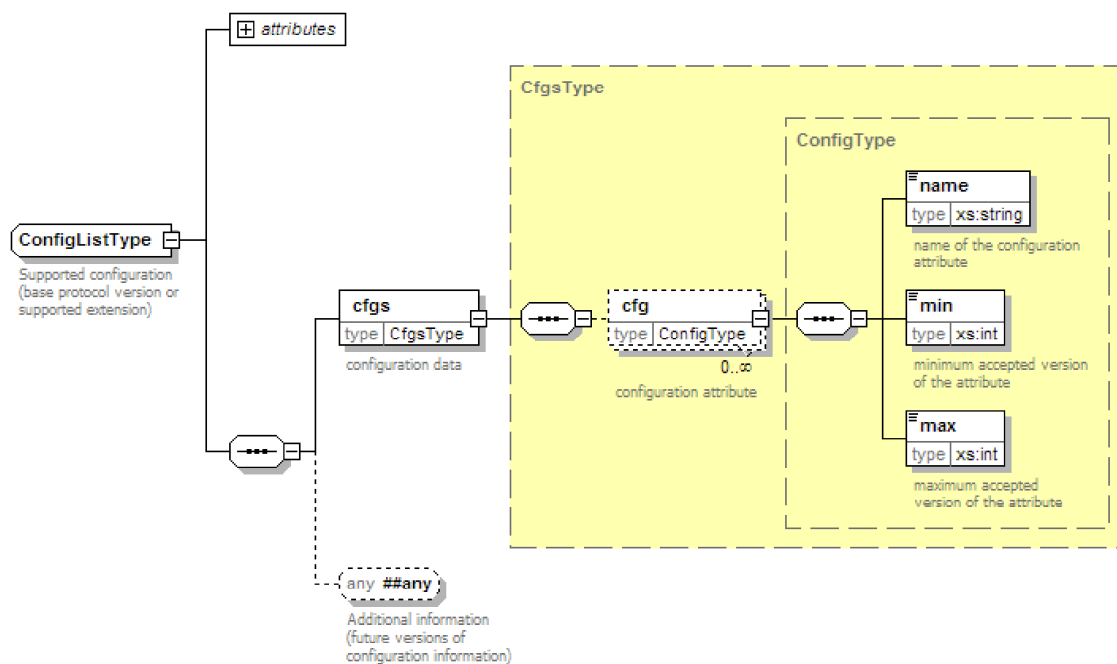


Bild 5 — ConfigListType Datentyp

OTS-Basis-Schemadatei

Datei OTS_Base.xsd:

Schemadatei mit Basis-Typen, enthält ConfigType und ConfigListType

Zukünftige Versionen können am Ende der Konfigurationsinformationen zusätzliche Informationen enthalten. Diese müssen überlesen werden, wenn sie nicht interpretiert werden können.

Der Konfigurations-Abgleich ist fester Teil des Verbindungsaufbaus (siehe 7.6.6, 8.5.6 und 9.6.7) und muss von jeder OTS-Implementierung beherrscht werden.

Die Konfigurations-Vorabfrage muss ebenfalls von jeder OTS-Implementierung beherrscht werden, ist aber in der Anwendung (vor dem Aufbau einer Verbindung) optional.

Die Konfigurations-Vorabfrage wird standardmäßig durch HTTP realisiert. Ein Server veröffentlicht Konfigurationsinformationen unter dem URI

<http://DomainnameDesServers/config/>

Die Information wird von einem Client durch ein http-get auf diesem URI abgerufen. Der Server muss http-get-Anfragen der http-Versionen 1.0 und 1.1 beantworten können. Bei der Übertragung der Antwort wird die oben beschriebene Codierung verwendet.

6.2 Fehlermanagement**6.2.1 Allgemeines**

Fehlerinformationen werden auf verschiedenen Schichten gleichermaßen eingesetzt.

Fehlermeldungen, die zwischen den Kommunikationspartnern ausgetauscht werden müssen, werden als Kombination aus *Fehlernummer*, *Fehlertext* und einer Liste von optionalen *Fehlerparametern* angegeben.

Tauschen zwei OTS-Kommunikationspartner Fehlerinformationen aus, so müssen die übermittelten Nachrichten streng diesem Format entsprechen (XML-Typ `ErrorType`, siehe auch 6.2.3).

Der Fehlertext muss eine schnelle, lesbare Ausgabe und eine Verifizierung der verwendeten Nummern ermöglichen. Fehlertexte müssen in Englisch und so knapp wie möglich verfasst werden (stichpunktartig).

Konkrete Fehlermeldungen und ihre jeweiligen Parameter werden in XML-Dateien definiert, siehe dazu Anhang A.

Für intern (zwischen den Schichten) und extern (zwischen Kommunikationspartnern) ausgetauschte Fehlermeldungen wird in der Spezifikation das gleiche Format verwendet.

ANMERKUNG Wie der interne Austausch von Fehlermeldung zwischen Schichten in einer Implementierung konkret realisiert wird (z. B. per Exception-Handling), bleibt dem Entwickler überlassen.

Folgendes Verhalten für Fehlerinformationen wird festgelegt:

- Fehlermeldungen werden über eine separate Fehlernachricht (Exception) übermittelt.⁴⁾ Schnittstellenmethoden enthalten daher keine Parameter für Fehlerinformationen.
- Im Fehlerfall sollte nur die Fehlermeldung und nicht die im Normalfall erwartete Rückmeldung gesendet werden.
- Die benötigten Informationen, um eine Fehlermeldung richtig zuordnen zu können, werden immer in den Parametern übermittelt (z. B. TransportID).
- Es können mehrere Fehlermeldungen aus dem gleichen Anlass auf einmal ausgelöst werden (z. B. von verschiedenen Schichten).
- Alle Rückrufobjekte müssen dem Interface zur Annahme von Fehlermeldungen genügen. Fehlermeldungen werden an das zugehörige Callback-Objekt zugestellt.
- Auch im Fall synchroner Rückmeldungen werden diese über eine Exception übersprungen – in jedem Fall ist der Inhalt der erwarteten Rückmeldung undefiniert, es gilt nur die Fehlerinformation.
- Fehlermeldungen dürfen nicht zur Übermittlung anderer Informationen missbraucht werden – es dürfen nur „echte“ Fehler definiert werden (z. B. „angeforderte Daten nicht definiert“), die als Reaktion auf ein Fehlereignis ausgelöst werden. Andere Informationen müssen über die Methoden des Protokolls abgehandelt werden (z. B. „angeforderte Daten nur teilweise vorhanden“).
- Fehlermeldungen an den Kommunikationspartner werden durch die Schicht, in der ein Fehler aufgetreten ist, an den Kommunikationspartner übermittelt.

6.2.2 Methoden

Folgende interne Methoden werden für den Austausch von Fehlerinformationen definiert:

onError (code, text, params)	Übergeben bzw. Empfangen einer internen Fehlermeldung beim Anwender der Schicht, in der der Fehler auftritt
sendError (connectionID, code, text, params)	Übermittlung einer Fehlermeldung an den Kommunikationspartner (interne Methode).

4) Dies erleichtert das Auffinden von Fehlern in Protokolltraces.

onRemoteError(connectionID, code, text, params)

Empfangen einer Fehlermeldung vom Kommunikationspartner

Folgende Parameter werden dabei verwendet:

connectionID	ID der Verbindung je nach Schicht: — TransportID (siehe 7.4) für die Transportschicht (xs:TransportIdType) — SessionID für Session- und Aktivitätsschicht (xs:SessionIdType)
code	Fehlernummer (xsd:int)
text	Fehlertext (xsd:string)
params	Liste von optionalen Fehlerparametern, jeder Parameter besteht aus Name (xsd:string) und Wert (xsd:string)

6.2.3 Bindung an Transportschicht

Fehlermeldungen an den Kommunikationspartner werden als XML-String vom Typ `ErrorType` nach dem folgenden Schema übermittelt:

- `code` : Fehlercode
- `text` : Fehlerbeschreibung
- `param` : Zusätzliche Parameter der Fehlernachricht. Jeder Parameter hat einen Wert und einen (als Attribut angegebenen) Namen.

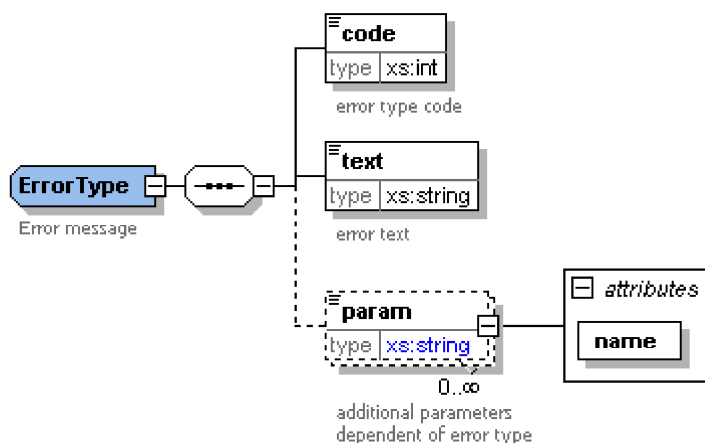


Bild 6 — ErrorType Datentyp

Die `connectionID` ist in diesem Typ nicht enthalten. In der Transportschicht wird die benötigte TransportID zusätzlich übertragen (siehe 7.6.3), ansonsten ergibt sich die SessionID automatisch aus der genutzten Verbindung (wie bei anderen Nachrichten auch).

OTS-Basis-Schemadatei

Datei `OTS_Base.xsd`: Schemadatei mit Basis-Typen, enthält `ErrorType`

7 OTS-Transportschicht

7.1 Allgemeines

Konfigurationsinformation: „`t_layer`“ = 1.

Die *OTS-Transportschicht* ist für die Abwicklung der Netzwerk-Kommunikation auf der Übermittlungsschicht zuständig und verbirgt deren Besonderheiten vor den übergeordneten Schichten.

ANMERKUNG 1 Die Transportschicht stellt eine Abstraktion verschiedener Transport-Protokolle für die weiter oben im Protokollstapel liegenden Schichten dar. Verschiedene Transport-Protokolle (TCP, SOAP) können über einen einheitlichen Dienstzugangspunkt angesprochen werden.

Die Transportschicht kann Transportverbindungen herstellen und beenden sowie Datenpakete senden und empfangen. Der Inhalt der Datenpakete ist zunächst beliebig.⁵⁾

Eine *Transportverbindung* ist eine Verbindung zwischen einer Quelle und einem Ziel auf einem bestimmten Transportkanal. Der *Transportkanal* gibt an, welches Protokoll auf der Übermittlungsschicht verwendet wird. Möglich sind hier zunächst SOAP/HTTP und TCP/Socket (siehe Bild 7).

ANMERKUNG 2 Die Protokollbindung an TCP/IP und Ethernet ist hier beispielhaft zur Vervollständigung des Protokollstapels angegeben und ist die präferierte. Es ist durchaus möglich, andere Protokollbindungen in den unteren Schichten anzuwenden (z. B. TCP/IP und GPRS für Geräte, die über das zellulare Mobilfunknetz angebunden werden).

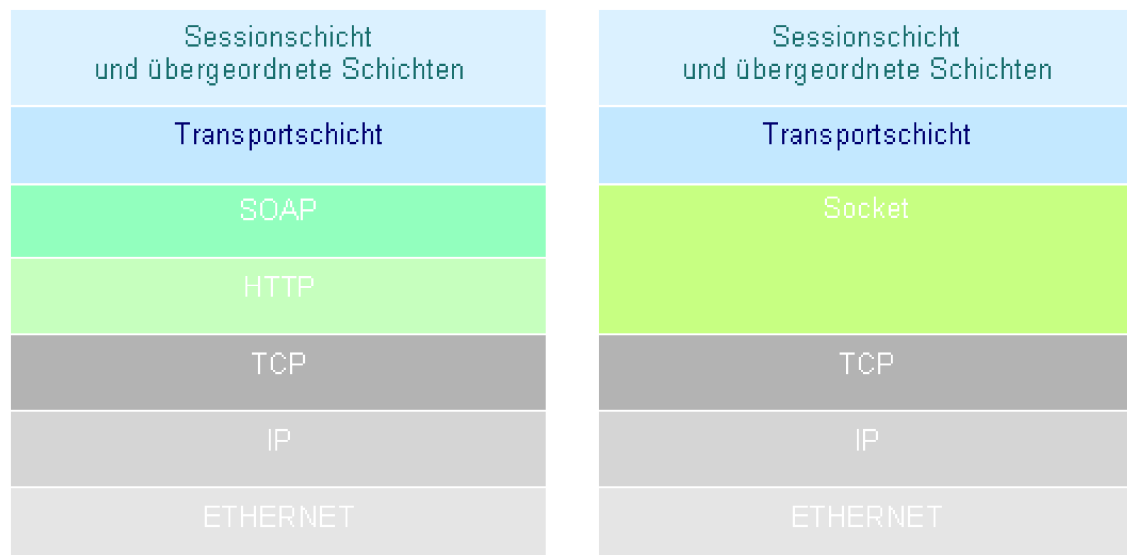


Bild 7 — Protokollbindung SOAP/HTTP (links) und TCP/Socket (rechts)

5) In einem spezifischen Protokollstapel, der OTS 2 mit allen drei Schichten implementiert, sind die zulässigen Datenpakete effektiv eingeschränkt auf die PDU der Sessionschicht.

Die Transportschicht bietet der übergeordneten Schicht ein asynchrones Verhalten an. Beherrscht das unterliegende Netzwerkprotokoll dies nicht vollständig (z. B. SOAP/HTTP), so wird das Verhalten Schicht-intern durch entsprechende Maßnahmen nachgebildet.

7.2 Methodenübersicht

Im Dienstzugangspunkt der OTS-Transportschicht werden die in Tabelle 1 und Tabelle 2 aufgeführten **Methoden** definiert.

ANMERKUNG Der Buchstabe T in den Methodennamen kennzeichnet die Zugehörigkeit zur Transportschicht.

Die Methoden des Provider-Port (siehe Tabelle 1) werden von der Transportschicht implementiert und von ihrem Nutzer⁶⁾ aufgerufen.

Tabelle 1 — Methoden der Transportschicht (Provider-Port)

Nr.	Methode	Beschreibung	Abschnitt
1	TAccept	Annahme von Transportverbindungen	7.6.4.2
2	TReject	Eingehende Transportverbindungen ablehnen	7.6.5
3	TConnect	Transportverbindung anfordern	7.6.6.2
4	TDisconnect	Transportverbindung beenden	7.6.8.2
5	TSendData	Daten über Transportverbindung versenden	7.6.7.2

Die Methoden des User-Port (siehe Tabelle 2) werden umgekehrt vom Nutzer der Transportschicht implementiert und von der Transportschicht aufgerufen.

Tabelle 2 — Benötigte Methoden eines Nutzers der Transportschicht (z. B. Sessionschicht) (User-Port)

Nr.	Methode	Beschreibung	Abschnitt
1a	onTAccepted	Eingehende Transportverbindung wurde angenommen	7.6.4.3
3a	onTConnected	Angeforderte Transportverbindung wurde aufgebaut	7.6.6.3
4a	onTDisconnected	Transportverbindung wurde beendet	7.6.8.3
5a	onTData	Daten wurden über Transportverbindung empfangen	7.6.7.3
6	onError	Ein lokaler Fehler ist aufgetreten	6.2
7	onRemoteError	Ein Fehler ist bei der Kommunikation mit dem Kommunikationspartner aufgetreten	7.6.3

7.3 Adressierung von OTS-Transport Nutzern

Zur Adressierung von OTS-Transport Nutzern wird ein *Uniform Resource Identifier* (URI) verwendet. Ein URI hat immer die Form in Tabelle 3.

6) Im Falle eines vollständigen OTS 2-Protokollstapels ist die Sessionschicht der Nutzer.

Tabelle 3 — URI

Schema	Schema-spezifischer Teil
http:	//www.comnets.rwth-aachen.de

Das Schema ist die höchste Ebene der URI Namensstruktur. Es bezeichnet einen Namensraum für den Schema-spezifischen Teil.

In OTS-Transport wird das `Schema` zur Identifikation der Protokollbindung verwendet. Tabelle 4 zeigt die Zuordnung des Schema-Feldes zur jeweiligen Protokollbindung.

Tabelle 4 — Das Schema-Feld in OTS-Transport URI

Protokollbindung	Schema-Feld für OTS-Transport URI
SOAP/HTTP	soap:
TCP/Socket/XML	xml:
SOAP/HTTP mit TLS Verschlüsselung (optional)	soap.tls:
TCP/Socket/XML mit TLS Verschlüsselung (optional)	xml.tls:

Eine Implementierung der OTS-Transportschicht muss die Protokollbindungen SOAP/HTTP und TCP/Socket/XML implementieren. Die Protokollbindungen mit Transport Layer Security (TLS) Verschlüsselungen sind optional.

Die Protokollbindungen mit TLS sind identisch zu denen ohne, mit der Besonderheit, dass anstatt eines unverschlüsselten TCP Sockets ein TLS verschlüsselter Socket eingesetzt wird. Das zugehörige Zertifikatsmanagement ist nicht Bestandteil dieser Spezifikation.

Für den Schema-spezifischen zweiten Teil des URI wurde eine generische Syntax von der IETF entwickelt.⁷⁾ Diese hat die Form wie in Tabelle 5.

Tabelle 5 — URI Syntax

Schema		Username	Password		Hostname	Port	Path	Query	Fragment
http:	//	daniel	: secret	@	example.com	: 80	/ dbn	? a=b	# Example

Die Felder `Username`, `Password`, `Hostname` und `Port` finden in OTS-Transport dieselbe Anwendung wie normalerweise auch im World Wide Web. Der `Path` findet bei der SOAP Protokollbindung Anwendung. Die Felder `Query` und `Fragment` werden nicht benutzt.

7.4 TransportID

Die Transportschicht vergibt für jede Transportverbindung eine so genannte *TransportID*. Die TransportID hat die folgenden Eigenschaften:

- Sie identifiziert eindeutig eine Transportverbindung zwischen einem Client und einem Server.

⁷⁾ siehe Literaturhinweise

- Sie ist auf Client- und auf Server-Seite gleich. Wenn Nutzer der Transportschicht sich über eine bestimmte Transportverbindung verständigen wollen, so können sie dies durch Übermittlung der TransportID tun.
- Die TransportID ist unabhängig von der Implementierung und dient als abstrakte Identifikationsnummer für eine bestimmte Transportverbindung

Die TransportID wird beim Verbindungsaufbau festgelegt.

Damit die Eindeutigkeit sichergestellt ist, wird die TransportID aus einem *ClientPart* und einem *ServerPart* zusammengesetzt. Bild 8 zeigt die Struktur der TransportID. *ClientPart* und *ServerPart* werden durch ein Trennzeichen getrennt (hier „:“).



Bild 8 — TransportID

Die Eindeutigkeit wird dadurch hergestellt, dass sowohl der Client als auch der Server einen lokalen *ID-Pool* verwalten, in dem nachgehalten wird, welche ClientParts und ServerParts schon vergeben wurden.

Bei der Implementierung des ID-Pools ist zu beachten, dass wenn ein *ClientPart* oder *ServerPart* freigegeben wird, dieser nicht sofort wieder verwendet werden darf, sondern es muss eine bestimmte Zeit (60 Sekunden, o. ä.) gewartet werden.

Dies ist notwendig, weil es in einem IP basierten System zur Duplizierung oder Überholung von Paketen kommen kann. Es sind Situationen denkbar, in denen noch Pakete einer OTS-Verbindung im IP-Netz vorhanden sind, nachdem die Verbindung bereits abgebaut wurde. Werden die ClientParts und ServerParts einer *TransportID* zu früh zur Wiederverwendung freigegeben, kann es passieren, dass diese „alten“ Pakete der „neuen“ Verbindung zugeordnet werden. Dies gilt es zu vermeiden. Aus Effizienzgründen sollten aber, obwohl dies das genannte Problem auch lösen würde, keine „ewig“ eindeutige, aber dafür sehr lange ID (z. B. UUID) verwendet werden.

7.5 OTS-Transportdienst und Aufgabenbereiche der OTS-Transportschicht

7.5.1 Sequenzdiagramm

Bild 9 zeigt das Sequenzdiagramm der Kommunikation zweier Nutzer des OTS-Transportdienstes (Client und Server) an der OTS-Transportschicht.

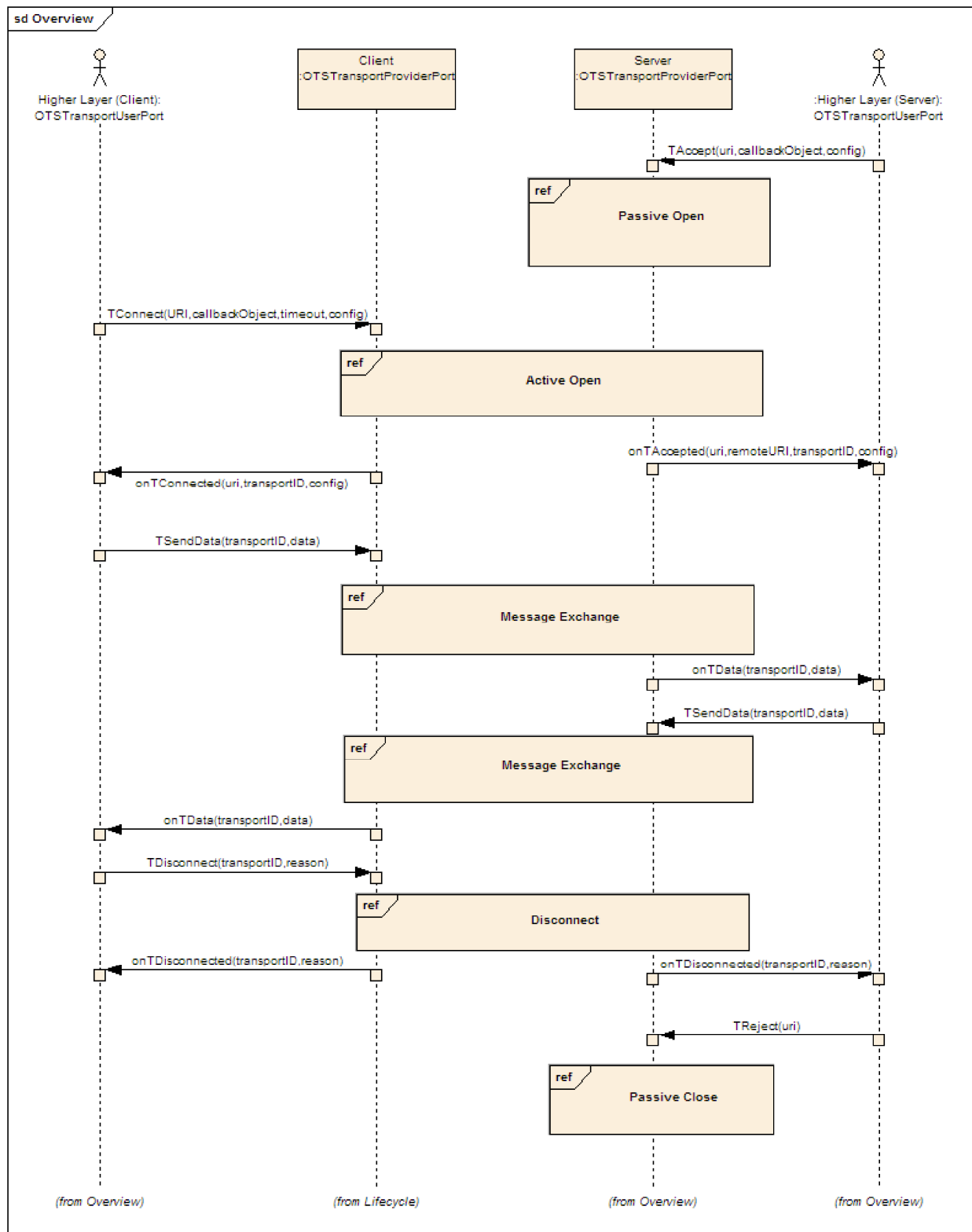


Bild 9 — Dienstangebot der OTS-Transportschicht und deren Aufgabenbereiche

Das Sequenzdiagramm zeigt nicht alle möglichen Abläufe, sondern einen Beispielfall.

Nutzer des OTS-Transportsdienstes werden über einen URI (Definition siehe 7.3) adressiert.

Ein Server, also ein Nutzer, der anderen Nutzern einen Dienst zur Verfügung stellen will, zeigt der OTS-Transportschicht durch den Aufruf von **TAccept**(URI) an, dass er OTS-Transportverbindungen unter der Adresse URI annimmt. Die OTS-Transportschicht führt die *Passive Open* Prozedur durch. Nach Ausführung von *Passive Open* werden eingehende Verbindungsanforderungen für den URI dem Nutzer angezeigt.

Ein Client, also ein Nutzer, der den Dienst eines anderen Nutzers nutzen will, baut durch Aufruf von **TConnect**(URI) eine OTS-Transportverbindung zu diesem auf. Die OTS-Transportschicht im Client kontaktiert daraufhin die OTS-Transportschicht im Server und führt die *Active Open* Prozedur aus. Im Server wird die Annahme einer neuen Verbindung dem Nutzer durch **onTAccepted**(URI, TransportID) angezeigt. Im Client wird der Verbindungsaufbau positiv bestätigt durch **onTConnected**(URI, TransportID). Beiden Nutzern wird die eindeutige *TransportID* mitgeteilt und zu welchem URI die Antwort auf den Verbindungsaufbau gehört. Der Verbindungsaufbau ist damit abgeschlossen. Im Fehlerfall werden statt den positiven Bestätigungen Fehlermeldungen ausgelöst.

Ist eine Verbindung aufgebaut, so können Client und Server asynchron, unabhängig voneinander und beliebig oft Nutzdaten austauschen. Daten werden durch Aufruf der Methode **TSendData**(TransportID, Data) versendet. Die OTS-Transportschicht führt die *Message Exchange* Prozedur aus. Eingehende Daten werden dem Nutzer durch den Aufruf von **onTData**(TransportID, Data) angezeigt. Die *TransportID* erlaubt die eindeutige Zuordnung der Nutzdaten (Data) zu einer OTS-Transportverbindung. Im Sequenzdiagramm wird exemplarisch der Austausch zweier Nachrichten zwischen Client und Server dargestellt.

Wird die OTS-Transportverbindung nicht mehr benötigt, löst der Server oder der Client die Verbindung auf, indem er die Methode **TDisconnect**(TransportID) aufruft. Die OTS-Transportschicht führt die *Disconnect* Prozedur durch und zeigt beiden Nutzern durch Aufruf von **onTDisconnected**(TransportID) an, dass die Verbindung geschlossen wurde und die *TransportID* ab sofort ungültig ist.

Stellt ein Server einen angebotenen Dienst ein, so teilt er der OTS-Transportschicht durch Aufruf von **TReject**(URI) mit, dass er zukünftig keine Verbindung mehr unter dem URI annimmt. Die OTS-Transportschicht führt die Prozedur *Passive Close* durch.

7.5.2 Zustandsdiagramm

Bild 10 zeigt das Zustandsdiagramm eines OTS-Transport Kanals in einer implementierungsunabhängigen Darstellung. Der Zustandsautomat gilt sowohl für die Client- als auch für die Server-Seite.

Die OTS-Transportschicht besitzt in der Regel viele OTS-Transportverbindungen und betreibt diese konkurrent.

Initiiert ein Client eine OTS-Transportverbindung, dann wird auch eine Instanz des Zustandsautomaten generiert. In diesem Fall wird der Eingangspunkt *Active Open Mode* gewählt. Das Verhalten dieser Transportverbindung ist dann im Folgenden durch diese Instanz des Zustandsautomaten festgelegt. Wird die Verbindung abgebaut, so baut auch die OTS-Transportschicht den entsprechenden Zustandsautomaten ab.

Auf Server-Seite wird eine Instanz des Zustandsautomaten erzeugt, wenn ein Client den Verbindungsaufbau initiiert hat. Dies ist durch den Empfang von *Active Open Request* ausgedrückt. Die konkrete übertragene Nachricht wird durch die jeweilige Protokollbindung festgelegt (siehe unten). Der Zustandsautomat wird durch den Eingangspunkt *Passive Open Mode* betreten. Analog zu der Client-Seite wird der entsprechende Zustandsautomat bei der Auflösung einer Verbindung gelöscht.

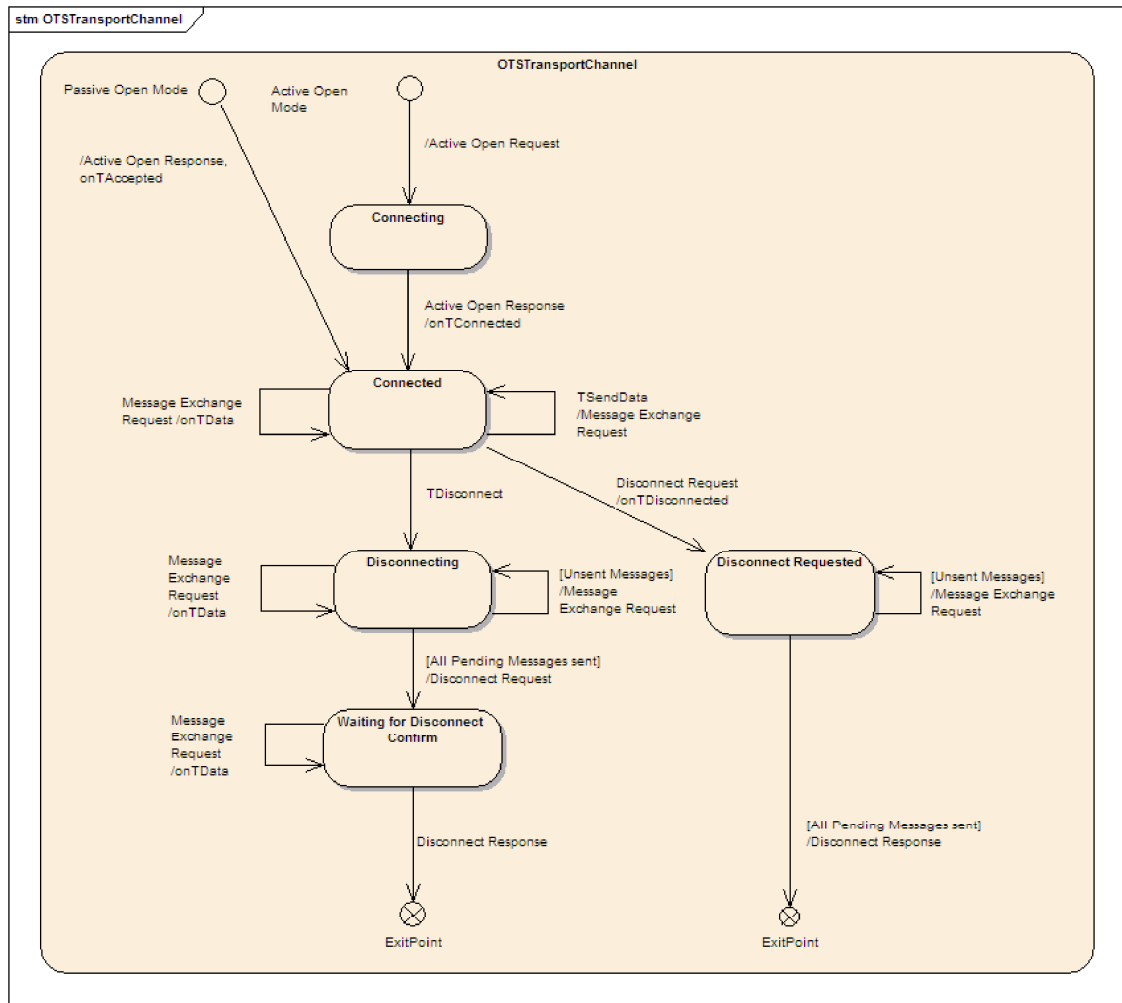


Bild 10 — Zustandsdiagramm OTS-Transportschicht

Ein Zustandsübergang kann durch verschiedene *Ereignisse* ausgelöst werden:

- 1) Der lokale Nutzer initiiert einen Zustandsübergang durch Aufruf einer der Dienstprimitive (**TConnect**, **TAccept**, **TSendData**, **TDisconnect** oder **TReject**).
- 2) Ein Zustandsübergang wird im Rahmen der Kommunikation mit dem Kommunikationspartner durchgeführt. Dies geschieht durch erfolgreiche Durchführung einer der OTS-Transportschicht Prozeduren (*Passive Open*, *Active Open*, *Message Exchange*, *Disconnect* oder *Passive Close*).

Prozedurname Request: Die Prozedur wurde vom Kommunikationspartner aktiviert.

Prozedurname Response: Die Prozedur *Prozedurname* wurde lokal aktiviert und die Ausführung der Prozedur wurde durch den Kommunikationspartner erfolgreich bestätigt.

Zustandsübergänge können *Aktionen* bewirken. Diese sind von den Ereignissen durch „/“ abgetrennt. Mögliche Aktionen sind:

- 1) Der lokale Nutzer wird über einen Zustandsübergang informiert. Dies geschieht durch Aufruf einer der Dienstprimitive am Dienstzugangspunkt (**onTConnected**, **onTAccepted**, **onTData**, **onTDisconnected**).
- 2) Die OTS-Transportschicht initiiert eine ihrer Prozeduren (*Active Open Request*, *Passive Open Request*, *Message Exchange Request*, *Disconnect Request* oder *Passive Close Request*).

7.6 OTS-Transport Protokollbindung

7.6.1 Allgemeines

Die Beschreibung des Dienstzugangspunktes von OTS-Transport erfolgt durch die UML. Es werden Schnittstellenklassen spezifiziert, die die Kommunikation an der Schichtengrenze zwischen der OTS-Transportschicht und den OTS-Transport Nutzern definieren. Diese Schnittstellen werden als *ProviderPort* und als *UserPort* bezeichnet.

Der Dienstanbieter (hier OTS-Transportschicht) stellt Implementierungen der *ProviderPorts* bereit. Um asynchrone Rückmeldungen von der OTS-Transportschicht entgegenzunehmen, stellen OTS-Transport Nutzer Implementierungen der *UserPorts* bereit (vgl. Bild 10)⁸⁾

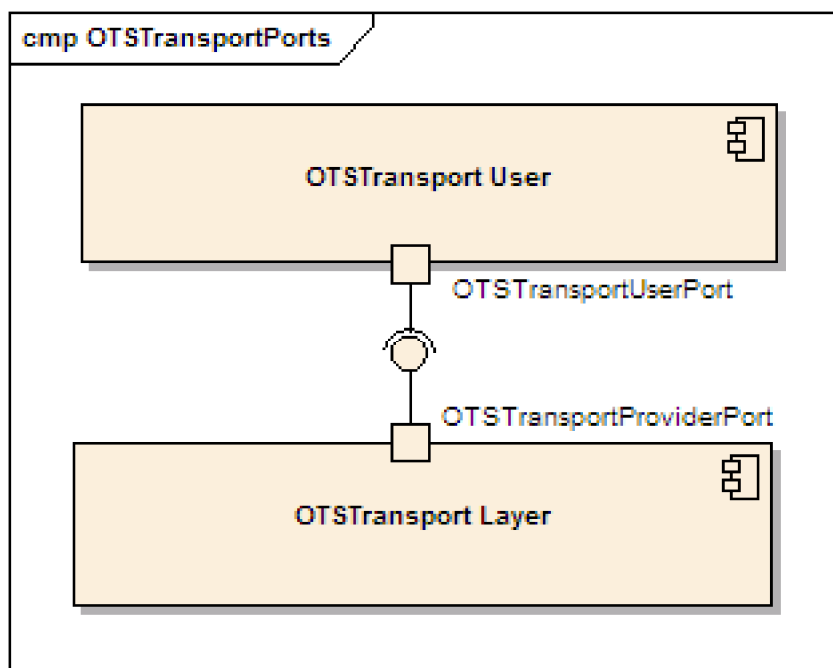


Bild 11 — Beschreibung des OTS-Transport Dienstzugangspunktes

Bild 11 zeigt die Spezifikation der Schnittstellen am OTS-Transport Dienstzugangspunkt. sowohl für den *ProviderPort* (OTS-Transportschicht) als auch für den *UserPort* (Nutzer der OTS-Transportschicht).

8) Es handelt sich hier um eine Beschreibungssystematik (Provider- und User-Ports). Die konkrete Realisierung in einer Implementierung des OTS-Protokollstapels kann dieser Systematik folgen, muss es aber nicht. OTS-Konformität muss lediglich bezüglich mit anderen OTS-Kommunikationspartnern ausgetauschter Nachrichten und bezüglich des Verhaltens (Zustandsautomat) der OTS-Transportschicht sichergestellt sein. Wie die Kommunikation zwischen Schichten im lokalen Knoten realisiert wird, bleibt vollständig unter Kontrolle des Realisierers.

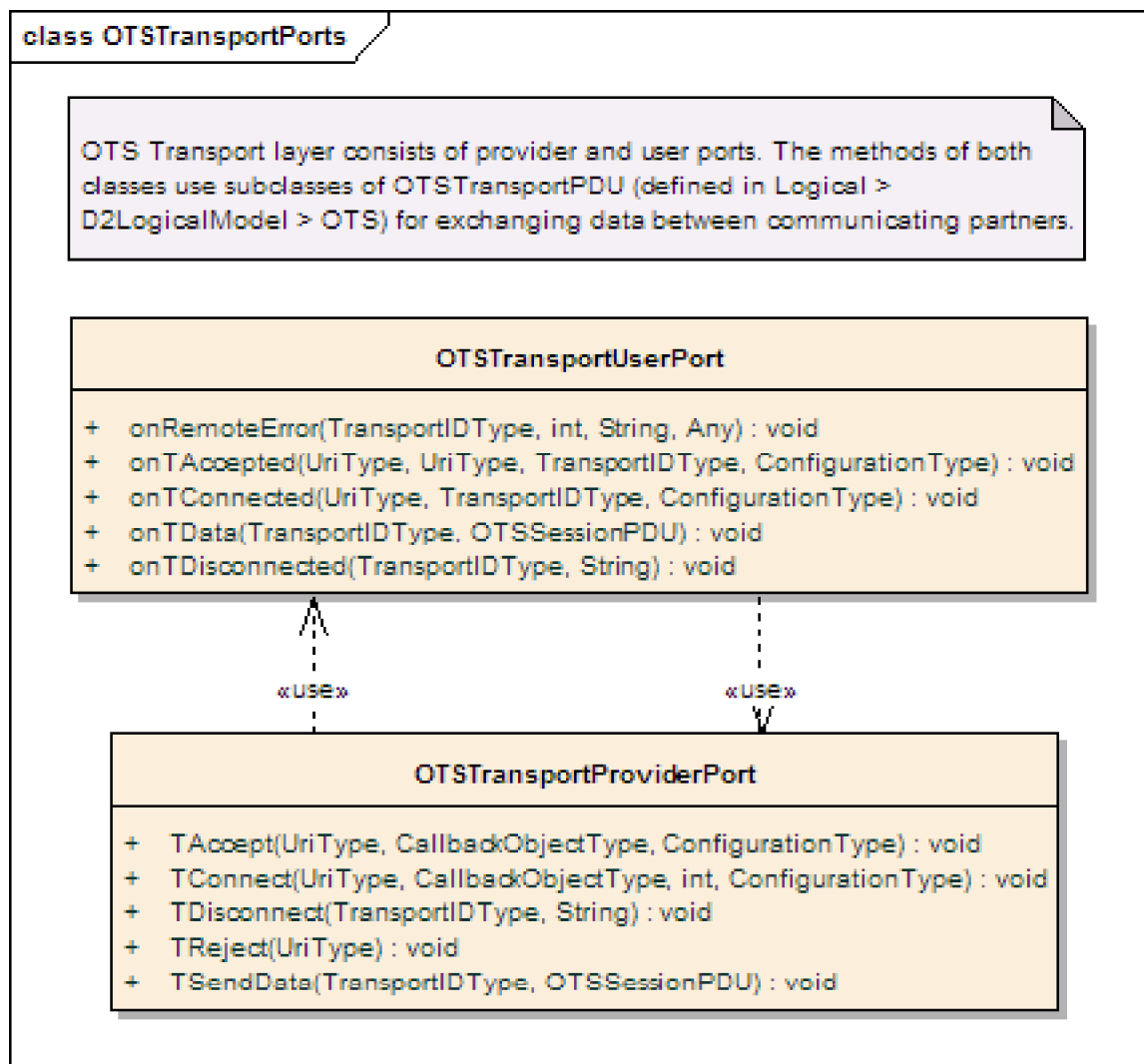


Bild 12 — OTS-Transport Ports

7.6.2 Encoding

Als *Encoding* für alle Zeichenketten wird, soweit nicht an der jeweiligen Stelle anders angegeben, *UTF-8* verwendet gemäß:

- RFC 3629 / STD 63 (2003)⁹⁾
- The Unicode Standard, Version 4.0, §3.9 bis §3.10 (2003)¹⁰⁾
- ISO/IEC 10646

9) siehe Literaturhinweise

10) siehe Literaturhinweise

7.6.3 Prozedur: Error Handling

7.6.3.1 Allgemeines

Die Prozedur realisiert die Zustellung von Fehlernachrichten (siehe A.2) der OTS-Transportschicht an den Kommunikationspartner.

7.6.3.2 Interne Schnittstelle bzw. Methode

`onError()`

Siehe 6.2.

`onRemoteError (`

<code>transportID</code>	TransportID der Transportverbindung bei der der Fehler auftrat
<code>code</code>	Fehlercode
<code>text</code>	Fehlerbeschreibung
<code>params</code>	Zusätzliche Parameter der Fehlernachricht

`)`

Der Benutzer der OTS-Transportschicht wird von seinem Kommunikationspartner über einen Fehler in Kenntnis gesetzt.

Um einen solchen Fehler zu übermitteln, wird die Methode `sendError()` benutzt. Sie ist schichtintern und steht dem Benutzer der OTS-Transportschicht nicht zur Verfügung.

`sendError (`

<code>transportID</code>	TransportID der Transportverbindung bei der der Fehler auftrat
<code>code</code>	Fehlercode
<code>text</code>	Fehlerbeschreibung
<code>params</code>	Zusätzliche Parameter der Fehlernachricht

`)`

7.6.3.3 Bindung an die Übermittlungsschicht

Fehlermeldungen werden dem Kommunikationspartner nicht als Nutzdaten übermittelt, sondern in eigenständigen Datagrammen.

ANMERKUNG Dies erleichtert die Beobachtbarkeit von Fehlersignalisierung auf der Übermittlungsstrecke.

7.6.3.4 SOAP

7.6.3.4.1 Allgemeines

Wird die Methode `sendError()` aktiviert, so wird zunächst festgestellt, ob der lokale Endpunkt der Transportverbindung `transportID` ein SOAP-Client oder ein SOAP-Server ist.

7.6.3.4.2 SOAP-Client

Die lokale OTS-Transportschicht (Client) sendet dem Kommunikationspartner (Server) die SOAP-Nachricht `tSendError()`. Der Parameter dieser Methode ist vom Typ `tSendErrorType` (vgl. Bild 13). Die Parameter werden entsprechend den Aufrufparametern gesetzt. Der URI des Servers ist durch die OTS-Transportverbindung `transportID` gegeben.

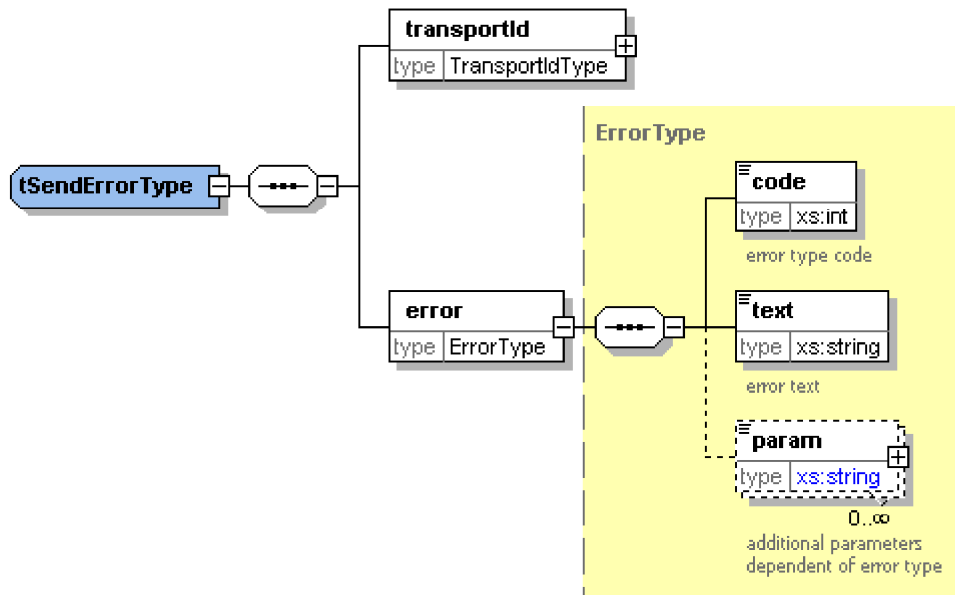


Bild 13 — tSendErrorType Datentyp (SOAP-Bindung)

7.6.3.4.3 SOAP-Server

Der Server generiert eine Nachricht vom Typ `tSendErrorType` (siehe Bild 13) und stellt diese in den Nachrichtenpuffer, der vom Client in regelmäßigen Abständen gepollt wird ein. Siehe dazu die detaillierte Beschreibung zum Pollingmechanismus (siehe 7.6.7).

Empfängt der Server die SOAP-Nachricht `tSendError()` oder der Client eine Nachricht vom Typ `tSendErrorType`, so wird dem OTS-Transportbenutzer durch Aufruf der Methode `onRemoteError()` diese Fehlernachricht angezeigt. Die Parameter werden entsprechend der empfangenen Fehlermeldung gesetzt und die TransportID der aktuellen Verbindung über den Parameter `transportID` an den OTS-Transportbenutzer weitergegeben.

OTS-Transport SOAP-Bindungs-Schemadatei

Datei <code>OTS_Transport_SOAP.xsd</code> :	Schemadatei zur SOAP-Bindung der Transportschicht (<i>PDU</i>)
Datei <code>OTS2.wsdl</code> :	Web Service Definition für die Transportschicht

7.6.3.5 Socket

Fehlernachrichten werden wie in der SOAP-Bindung in XML-Tags gekapselt. Konkret wird eine XML-Nachricht vom Typ `tSendErrorType` verschickt (siehe Bild 14). Diese wird über die durch die TransportID angegebene Socket-Verbindung verschickt.

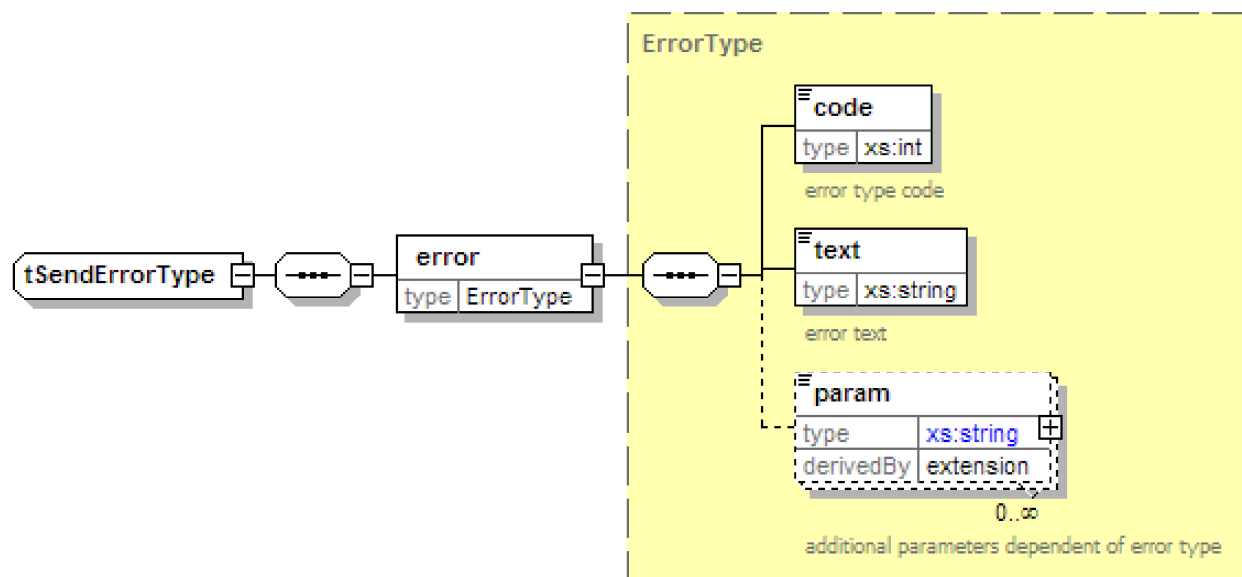


Bild 14 — tSendErrorType Datentyp (Socket-Bindung)

OTS-Transport Socket-Bindungs-Schemadatei

Datei OTS_Transport_Socket.xsd: Schemadatei zur Socket-Bindung der Transportschicht ([PDU](#))

Wird auf dem Socket einer Transportverbindung ein XML `<error>` tag empfangen, so wird dem OTS-Transportbenutzer durch Aufruf der Methode `onRemoteError()` diese Fehlernachricht angezeigt. Die Parameter werden entsprechend der empfangenen Fehlermeldung gesetzt und die TransportID der aktuellen Verbindung über den Parameter `transportID` an den OTSTransportbenutzer weitergegeben.

7.6.4 Prozedur: Passive Open („Server“)

7.6.4.1 Allgemeines

Diese Prozedur realisiert die Annahme von eingehenden OTS-Transportverbindungen durch Nutzung der angebundenen Übermittlungsschicht.

7.6.4.2 Provider-Port: Annehmen von Transportverbindungen

7.6.4.2.1 Interne Schnittstelle bzw. Methode

TAccept (

URI	Adressangabe, auf der Transportverbindungen angenommen werden („scheme://hostname:port“)
callbackObject	Rückrufobjekt, welches die Schnittstelle des OTS-Transport User Ports implementiert und die aufzubauende Transportverbindung repräsentiert
config	Wahlfreie Konfigurationsinformationen (Liste zu verwendender wahlfreier Funktionalitäten der OTS-Transportschicht)

)

Ein Nutzer, der einen Dienst zur Verfügung stellt, teilt OTS-Transport durch Aufruf der **TAccept** Methode mit, dass er nun eingehende Transportverbindungen auf einer bestimmten Adresse **URI** akzeptiert.

Der Aufrufer muss im Anschluss über das angegebene `callbackObject` empfangsbereit für die angegebenen Ereignisse sein. Als erstes wird **onTAccepted()** erwartet, das ausgelöst wird, wenn ein Client sich erfolgreich per **TConnect()** verbindet (siehe 7.6.6). Im Fehlerfall wird stattdessen eine Fehlermeldung geworfen.

7.6.4.2.2 Bindung an Übermittlungsschicht

Auf der durch den **URI** angegebenen Adresse wird mit dem ebenfalls durch den **URI** identifizierten Netzwerkprotokoll (`scheme`, siehe Tabelle 4) auf eingehende Verbindungen gewartet.

ANMERKUNG Der zur Verbindungsaufnahme verwendete Transportkanal ist ein vorab festzulegendes Konfigurationsdatum.

Die OTS-Transportschicht muss das Format der **URI** überprüfen. Entspricht das Format der **URI** nicht dem Format wie in 7.3 beschrieben, so wird **onError(1001, „Invalid URI“, ...)** aufgerufen. Die Prozedur *Passive Open* wird nicht durchgeführt.

Ist die **URI** syntaktisch korrekt, entspricht das Protokollfeld aber nicht einem der Werte aus Tabelle 4, dann wird ebenfalls **onError(1001, „Invalid URI“, ...)** aufgerufen. Die Prozedur *Passive Open* wird nicht durchgeführt.

Das Feld `uri` wird jeweils mit der aktuellen **URI**, die den Fehler verursacht, belegt. Das Feld `method` wird mit „TAccept“ belegt.

7.6.4.2.3 SOAP

Die Identifikation einzelner Verbindungen bei der SOAP-Protokollbindung wird durch Mitführen der OTS-TransportID in den übertragenen Nachrichten realisiert.

Nutzer der OTS-Transportschicht, die eingehende OTS-Verbindungen annehmen, teilen dies der OTS-Transportschicht durch den Aufruf von **TAccept()** mit. Dabei wird der **URI**, auf dem eingehende OTS-Transportverbindungen angenommen werden, als Parameter übergeben. Sie enthält neben `Schema` und `Hostname` (IP-Adresse) zusätzlich eine Portnummer. Falls das `Path` Feld vom Nutzer gesetzt wurde, wird es von OTS-Transport beim Aufruf von **TAccept()** verworfen.

Der SOAP Dienst ist immer unter dem Pfad `/OTSTransport` erreichbar.

Die SOAP Protokollbindung nimmt beim Aufruf von **TAccept()** Verbindungsanfragen (SOAP Methode **TConnect**) an, die an eine SOAP-Server Instanz gehen, die auf in **URI** angegebenem **URI** und Portnummer bindet (`Soap::soap_bind()` o.ä.). Ein OTS-Transport-SOAP-Dienst, d. h. ein Webservice, ist dann unter diesem **URI** erreichbar.

Der Webservice wird spezifiziert über die `OTS2.wsdl` (s.o.).

Dieser Dienst bietet folgende SOAP-Nachrichten an (siehe `OTS_Transport_SOAP.xsd`):

- **tConnect** (Antwort **tConnectR**)
- **tDisconnect** (Antwort **tDisconnectR**)
- **tSend** (Antwort **tSendR**)
- **tSendError** (Antwort **tSendErrorR**)
- **tGet** (Antwort **tGetR**)

Kann keine SOAP-Server Instanz auf dem angegebenen Port gestartet werden, da der Port schon belegt ist, wird der `onError(1101, „TAccept failed“, ...)` aufgerufen. Die Prozedur *Passive Open* wird abgebrochen und alle innerhalb dieser Prozedur allokierten Ressourcen freigegeben.

Das Feld `uri` wird mit der URI, die den Fehler verursacht, belegt, das Feld `reason` mit "Port already in use". Ist die Fehlerursache unbekannt, wird `reason` mit „Unknown reason“ belegt.

Eine neue OTS-Transportverbindung wird von einem Client durch Aufruf der SOAP-Nachricht `tConnect()` auf dem URI eines Servers gestartet (siehe 7.6.6). Der Server vergibt dann eine neue TransportID und übermittelt diese an den Client.

Das SOAP Protokoll benutzt HTTP und TCP/IP zur Realisierung seiner Dienste. Ein SOAP Server akzeptiert eingehende Verbindungen mittels HTTP-Server. Der HTTP-Server öffnet eine passive TCP/IP Verbindung und wartet auf eingehende Verbindungen. Besitzt ein Host mehrere Netzwerkkarten und mehrere IP Adressen, so bestimmt die Angabe des Hostnamens im URI des `TAccept()` Aufrufs, auf welcher IP Adresse auf eingehende TCP/IP Verbindungen gewartet wird.

Der HTTP-Server muss beim Aufruf von `TAccept()` dementsprechend konfiguriert werden, so dass eingehende Verbindungen nur auf der IP Adresse entgegengenommen werden, die mit dem Hostnamen im URI assoziiert ist.

7.6.4.2.4 Socket

Bei Benutzung der Socket-Protokollbindung benutzt OTS-Transport TCP/IP Mechanismen, um eingehende Verbindungen entgegenzunehmen. Beim Aufruf von `TAccept()` öffnet die OTS-Transportschicht eine passive TCP/IP Verbindung und wartet auf eingehende Verbindungen. Der Hostname des URI bestimmt, auf welcher IP Adresse (falls im Host mehrere vorhanden sind) eingehende Verbindungen akzeptiert werden.

Mit jeder eingehenden Verbindung ist ein Verbindungsendpunkt (ein aktiver Socket) eindeutig verknüpft und dieser kann eindeutig mit einem OTS-Transport Kanal verknüpft werden, wenn eine neue Verbindung akzeptiert wird.

Für den Versand über die Socket-Verbindung sind folgende XML-Nachrichten definiert (siehe `OTS_Transport_Socket.xsd`):

- `tConnect` (Antwort `tConnectR`)
- `tDisconnect`
- `tSend`
- `tSendError`
- `tPing` (Antwort `tPong`)

Kann der TCP Socket auf dem angegebenen Port nicht geöffnet werden, da der Port schon belegt ist, wird der `onError(1101, „TAccept failed“, ...)` aufgerufen. Die Prozedur *Passive Open* wird abgebrochen und alle innerhalb dieser Prozedur allokierten Ressourcen freigegeben. Das Feld `uri` wird mit der URI, die den Fehler verursacht, belegt, das Feld `reason` mit "Port already in use". Ist die Fehlerursache unbekannt, wird `reason` mit „Unknown reason“ belegt.

7.6.4.3 User-Port: Erfolgte Annahme einer Transportverbindung

7.6.4.3.1 Interne Schnittstelle bzw. Methode

`onTAccepted (`

<code>URI</code>	URI, der beim Aufruf von <code>TAccept</code> angegeben wurde
<code>RemoteURI</code>	URI vom Kommunikationspartner, der die Verbindung aufbaut
<code>transportID</code>	Bei erfolgreichem Connect vergebene TransportID, sonst 0
<code>config</code>	Verwendete wahlfreie Konfigurationsinformationen (Liste unterstützter Funktionalitäten und Versionen der OTS-Transportschicht)

`)`

Nach einem vorangegangenen `TAccept()` wurde von einer Quelle eine Transportverbindung per `TConnect()` angefordert. Es wurde die Authentifizierung durchgeführt und die Transportverbindung aufgebaut. In diesem Fall wird auf Ziel-Seite das Ereignis `onTAccepted()` bei dem CallbackObject ausgelöst, welches bei `TAccept()` angegeben wurde.

Während einer bestehenden Transportverbindung müssen beide Seiten empfangsbereit für Ereignisse `onTData()` und `onTDisconnected()` sein.

7.6.5 Prozedur: Passive Close („Server“)

7.6.5.1 Allgemeines

Diese Prozedur beendet die Annahme von eingehenden Transportverbindungen und macht eine vormals durchgeführte *Passive Open* Prozedur rückgängig.

7.6.5.2 Provider-Port: Ablehnung von Transportverbindungen

7.6.5.2.1 Interne Schnittstelle bzw. Methode

`TReject (`

<code>URI</code>	Adressangabe, auf der keine Transportverbindungen mehr angenommen werden dürfen („scheme://hostname:port“)
------------------	--

`)`

Es wird die Annahmefähigkeit für eingehende Transportverbindungen auf einer bestimmten Adresse `URI` widerrufen (die einer früher bei `TAccept()` angegebenen Adresse entspricht). Dies hat keine Auswirkung auf aktive bestehende OTS-Transportverbindungen.

7.6.5.2.2 Bindung an Übermittlungsschicht

Das Warten auf neu eingehende Verbindungen auf der angegebenen Adresse `URI` wird eingestellt. Der Widerruf der Annahme von Transportverbindungen hat keine Auswirkung auf bereits bestehende OTS-Transportverbindungen.

Die OTS-Transportschicht muss das Format der URI überprüfen. Entspricht das Format der URI nicht dem Format wie in 7.3 beschrieben, so wird `onError(1001, „invalid URI“, ...)` aufgerufen. Die Prozedur *PassiveClose* wird nicht durchgeführt.

Ist die URI syntaktisch korrekt, entspricht das Protokollfeld aber nicht einem der Werte Tabelle 4, dann wird ebenfalls `onError(1001, „invalid URI“, ...)` aufgerufen. Die Prozedur *Passive Close* wird nicht durchgeführt.

Das Feld `uri` wird jeweils mit der aktuellen URI, die den Fehler verursacht, belegt. Das Feld `method` wird mit „TReject“ belegt.

7.6.5.2.3 SOAP

Gibt es keine aktiven OTS-Transportverbindungen unter dem angegebenen URI, dann wird der OTS-Transport Dienst unter dem gegebenen URI beendet. Es dürfen keine weiteren Anfragen unter diesem URI angenommen werden.

Gibt es aktive OTS-Transportverbindungen unter dem angegebenen URI, dann darf der OTS-Transport Dienst zunächst nicht beendet werden. Jedoch darf der OTS-Transport Dienst keine neuen Verbindungen annehmen. Erst wenn alle noch aktiven OTS-Transportverbindungen beendet wurden (durch `TDisconnect` oder durch einen Timeout), wird auch der OTS-Transport Dienst beendet.

Kann der SOAP Server auf dem angegebenen Port nicht geschlossen werden, da er nie geöffnet wurde, so wird `onError(1201, „TReject failed“, ...)` aufgerufen. Die Prozedur *Passive Close* wird abgebrochen. Das Feld `uri` wird mit der URI, die den Fehler verursacht, belegt. Das Feld `reason` wird mit „Port not in use“ belegt. Ist die Fehlerursache unbekannt, wird das Feld `reason` mit „Unknown reason“ belegt.

Wird eine neue Verbindung von einem Client durch die SOAP-Nachricht `tConnect()` angefordert, während der OTS-Transport Dienst auf das Beenden aller aktiver OTS-Transportverbindungen wartet und keine neuen Verbindungen mehr annimmt, so wird dem Client der Fehler `(5102, „rejected“, ...)` mitgeteilt (siehe auch `TConnect`).

7.6.5.2.4 Socket

OTS-Transport schließt die passive TCP/IP Verbindung, die vormals durch einen `TAccept()` Aufruf geöffnet wurde. Es werden keine Verbindungen mehr angenommen.

Kann der TCP Socket auf dem angegebenen Port nicht geschlossen werden, da der Port nie geöffnet wurde, so wird `onError(1201, „TReject failed“, ...)` aufgerufen. Die Prozedur *Passive Close* wird abgebrochen. Das Feld `uri` wird mit der URI, die den Fehler verursacht, belegt. Das Feld `reason` wird mit „Port not in use“ belegt. Ist die Fehlerursache unbekannt, wird das Feld `reason` mit „Unknown reason“ belegt.

7.6.6 Prozedur: Active Open („Client“)

7.6.6.1 Allgemeines

Diese Prozedur stellt eine OTS-Transportverbindung zu einem Kommunikationspartner her. Der Kommunikationspartner muss eingehende Verbindungen akzeptieren, d. h. die *Passive Open* Prozedur muss von ihm ausgeführt worden sein.

7.6.6.2 Provider-Port: Aufbau einer Transportverbindung

7.6.6.2.1 Interne Schnittstelle bzw. Methode

TConnect (

<code>targetURI</code>	Vollständige Ziel-Adressangabe inkl. Authentifizierung („scheme://user-name:password@target:port“)
<code>callbackObject</code>	Rückrufobjekt, welches die Schnittstelle des OTS-Transport User Ports implementiert und die aufzubauende Transportverbindung repräsentiert
<code>timeout</code>	Timeout-Zeit in Sekunden, nach der beim Ausbleiben von Nachrichten der Gegenseite die Verbindung als gestört erkannt wird
<code>config</code>	Wahlfreie Konfigurationsinformationen (Liste zu verwendender wahlfreier Funktionalitäten der OTS-Transportschicht)

)

Es wird eine neue Transportverbindung zu einem Ziel `targetURI` aufgebaut. Der Aufruf kann nur bei korrekter Authentifizierung erfolgreich abgeschlossen werden.

Die beherrschten *und* gewünschten Konfigurationsparameter `config` werden angegeben. Das Ziel muss in einer Antwort die im Weiteren zu verwendenden Konfigurationsparameter zurückmelden.

Der Parameter `timeout` dient der Verbindungsüberwachung.

Der Aufrufer muss im Anschluss über das angegebene `callbackObject` empfangsbereit für die angegebenen Ereignisse sein. Zuerst wird das Ereignis `onTConnected()` erwartet. Im Fehlerfall wird stattdessen eine Fehlermeldung geworfen.

7.6.6.2.2 Bindung an Übermittlungsschicht

Aus dem URI werden Ziel-Adresse und Ziel-Port extrahiert. Die Quell-Adresse wird als bekannt angenommen. Das `scheme` Feld des URI bestimmt, welche Protokollbindung benutzt wird (siehe 7.3). Die Authentifizierung wird auf Ziel-Seite durchgeführt. Schlägt die Authentifizierung fehl, so wird der Verbindungsaufbau abgebrochen.

Je nach Bindung stellen sich Quelle und Ziel darauf ein, über den neuen Kontakt Daten zu empfangen. Ggf. muss das geforderte bidirektionale asynchrone Verhalten mit geeigneten Maßnahmen nachgebildet werden. Ist das zugrunde liegende Netzwerkprotokoll nicht verbindungsorientiert (z. B. SOAP), muss dies ebenfalls nachgebildet werden (z. B. durch Mitschicken der TransportID).

Die OTS-Transportschicht muss das Format der URI überprüfen. Entspricht das Format der URI nicht dem Format wie in 7.3 beschrieben, so wird der `onError(1001, „invalid URI“, ...)` aufgerufen. Die Prozedur *Active Open* wird nicht durchgeführt.

Ist die URI syntaktisch korrekt, entspricht das Protokollfeld aber nicht einem der Werte aus Tabelle 4, dann wird ebenfalls `onError(1001, „invalid URI“, ...)` aufgerufen. Die Prozedur *Active Open* wird nicht durchgeführt.

Das Feld `uri` wird jeweils mit der aktuellen URI, die den Fehler verursacht, belegt. Das Feld `method` wird mit „TAccept“ belegt.