Specification of a communication protocol

Preliminary draft proposal prepared by DE experts

Technical contacts:

Germany: Sebastian Kaldeweide, Infotech GmbH
kaldeweide@infotech.de

Patrick Noetzel, Infotech GmbH
noetzel@infotech.de

Contents

A. Introduction

This Specification of a communication protocol complements the “Specification of a data model”. It defines a secure and homogenous electronic data interchange (EDI) protocol. The protocol is used to transfer messages created according to the data model (EDI-Messages) from one party to another.

This document gives an overview of the electronic data interchange (EDI) protocol.
Different use cases will be discussed.

B. Conventions

The names of entities are italicized.

This specification uses the key words (MUST, SHALL, etc.) of [RFC 2119[1]] to indicate the requirement levels.

C. Definitions

EDI-Message - describes the transferred XML-Message as defined in the data model.

Party - means a person (natural or juristic) who wants to transfer EDI-Messages.

Message Broker – is a central system, which is permanent available and provides inboxes for parties. A Message Broker delivers EDI-Messages to inboxes.

Provider System – is a representative of another party. A provider acts on behalf of another party at least in communication matters.

D. Delimitations

This specification describes the transport-protocol of EDI-Messages. There are no assumptions about the transferred payload.

In this document, the Payload[2] is defined as the xml message instance conforming to the Data Model Specification.

The used certificates and the Public Key Infrastructure (PKI[3]) are not target of this specification.

This specification assumes that the certificates are exchanged in a secure way and the quality is sufficient for both parties involved.

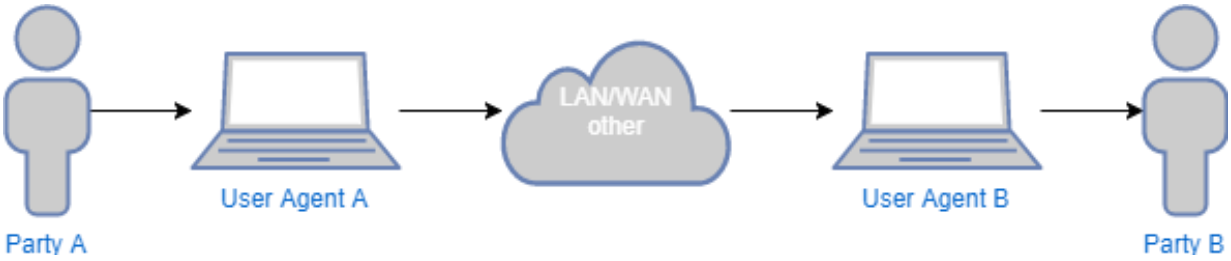
---

[1] https://www.ietf.org/rfc/rfc2119.txt
[2] https://en.wikipedia.org/wiki/Payload_(computing)
[3] https://en.wikipedia.org/wiki/Public_key_infrastructure

E. Use Cases

This chapter describes the different use cases. At the beginning, the simplest case with peer to peer transfer of EDI-Messages is shown. Further the more complex scenarios with one or many message brokers are discussed.

I. Peer to Peer

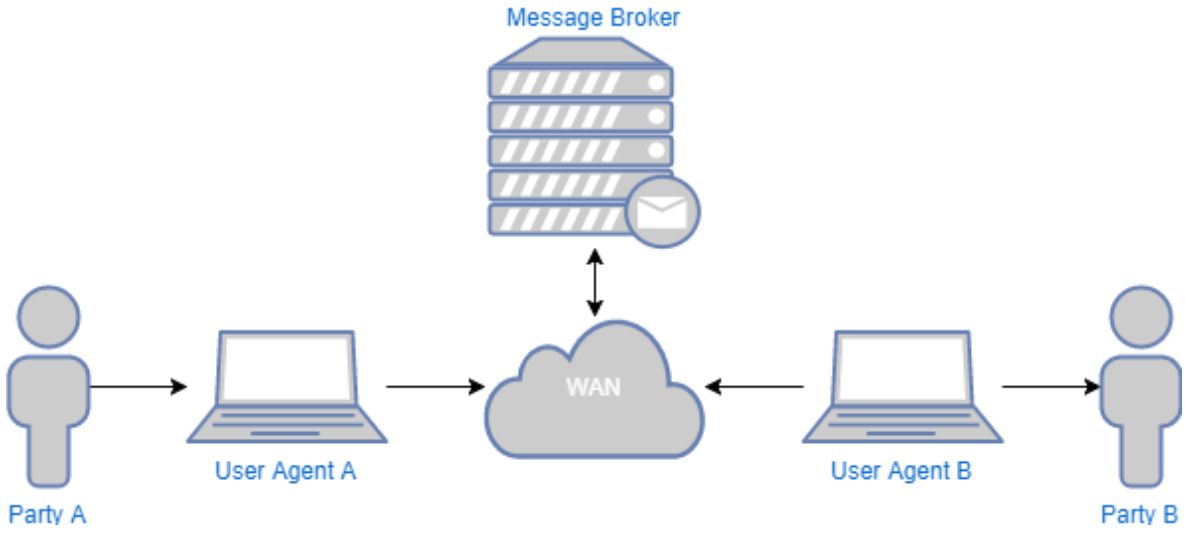


The peer-to-peer case describes the direct transfer of an EDI-Message between two parties. There should be no technical restrictions for the transmission layer. For example, a direct transfer between two mobile phones with active bluetooth pairing should be sufficient.

The User Agent A is the software which generates the EDI-Messages, encrypts them and send them to the other endpoint. User Agent B has to provide an active endpoint for the EDI-Protocol. In this case the transfer is synchronous. If Party A expects a response from Party B, the use case is reversed. Party B sends a regular EDI-Message to Party A.

In this point to point transfer the additional encryption of the EDI-Message can be omitted if the underlaying transport layer has a sufficient encryption and authentication mechanism. This is part of the agreement between the parties involved.

II. Single Message Broker



This scenario describes the transmission of EDI-Messages with the assistance of a Message Broker.

The Message Broker (MB) addresses the problem of the permanent availability on the receiver side. The broker adds an asynchronous transfer mode to the EDI-transmission.

Party A can send a Message to the always public available MB.
Party B can receive new EDI-Messages due to asking the MB for new Messages.
Party B can download the message immediately after the upload from *Party A* is finished.
Party B has only access to transmissions which are associated with his party identifier.
This have to be ensured by the MB due to an inbox registration process.

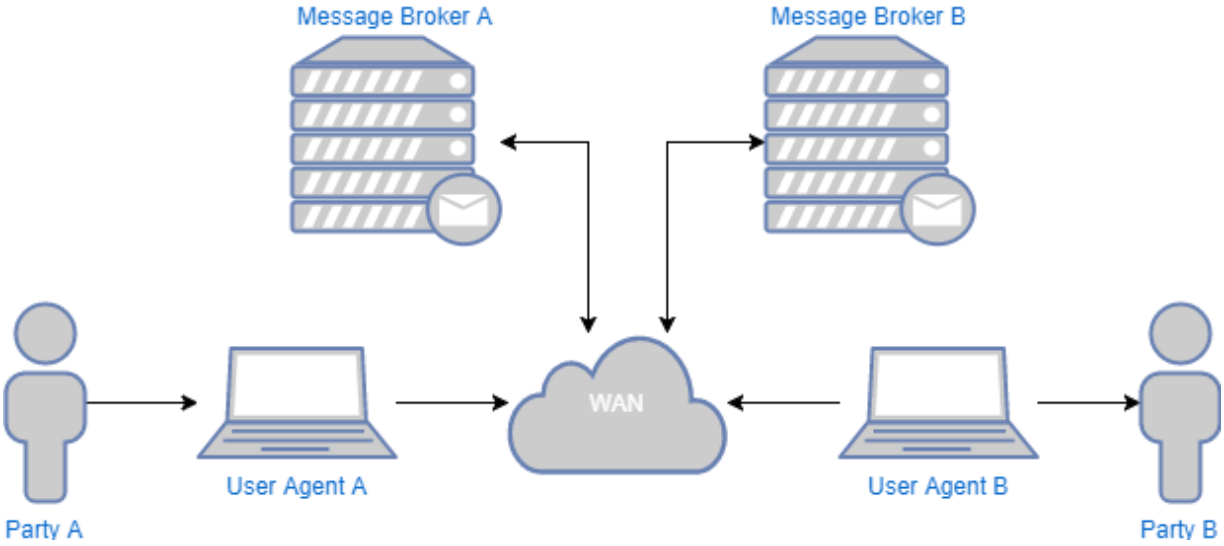
The inbox registration and retrieval protocol are not part of this specification[4].
This is an implementation detail of any concrete MB. Every MB can specify its own mechanism.

The receipt is not directly acknowledged from *Party B,* like in the peer to peer example. In this case a distinction between *transferred* and *delivered* EDI-Messages have to be made. After the upload the EDI-Message is in the state *transferred* and after the download through the receiver the message is in the state *delivered*.

This use case is an implicit part of the multi broker scenario if both parties are using the same message broker.

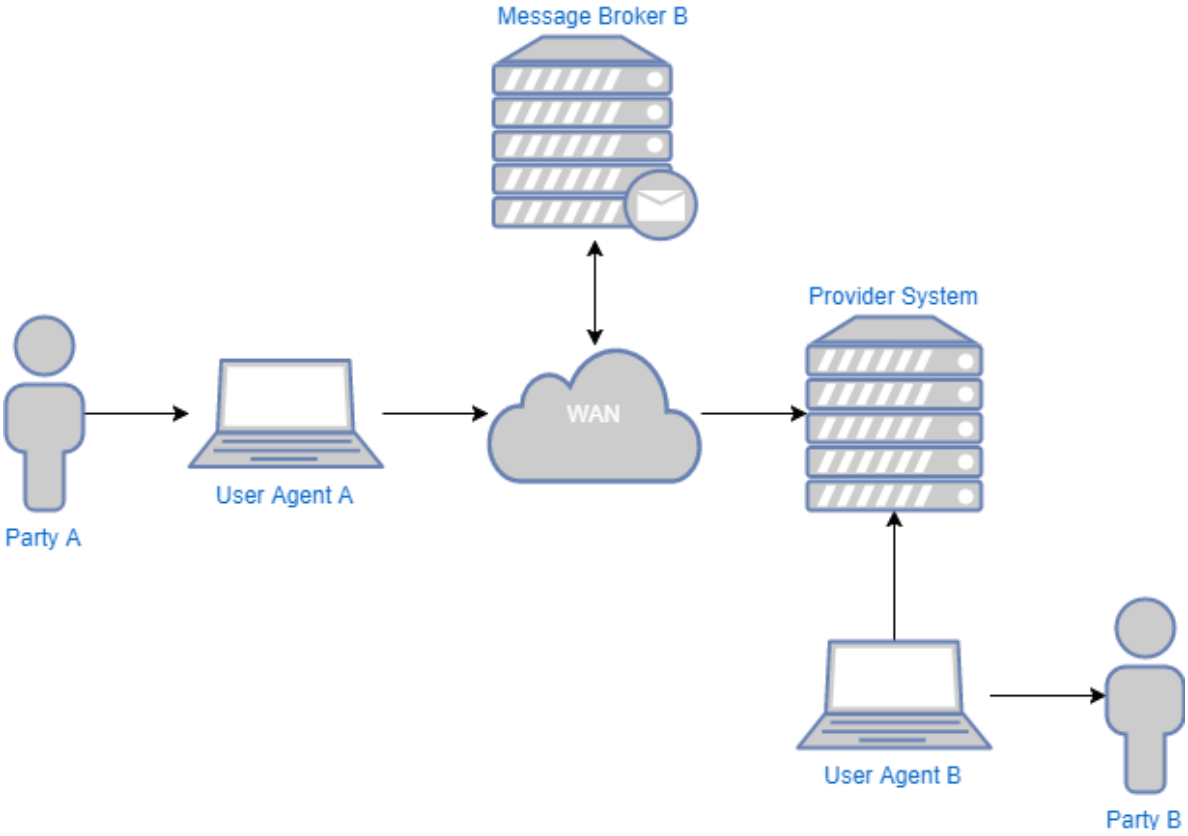
[4] Example definition and implementation will be provided

III. Multi Message Broker



This use case eliminates the global single point of failure of the MB. Each party can be registered on a separate MB. To transfer an EDI-Message from *Party A* to *Party B, Party A* creates a transmission on MB B and uploads the data. If *Party A* wants a response for the transmission *Party B* have to create a new separate transmission on MB A, where the inbox of *Party A* is registered. *Party A* creates a direct connection to the Message Broker of *Party B* and vice versa. If a Message Broker is unavailable only the parties with the inbox on this broker are affected. For a higher redundancy, if needed, a second inbox on a separate MB can be opened. In this case the party has two inboxes to check and multiple target addresses.

IV. Message Broker with Party behind Provider

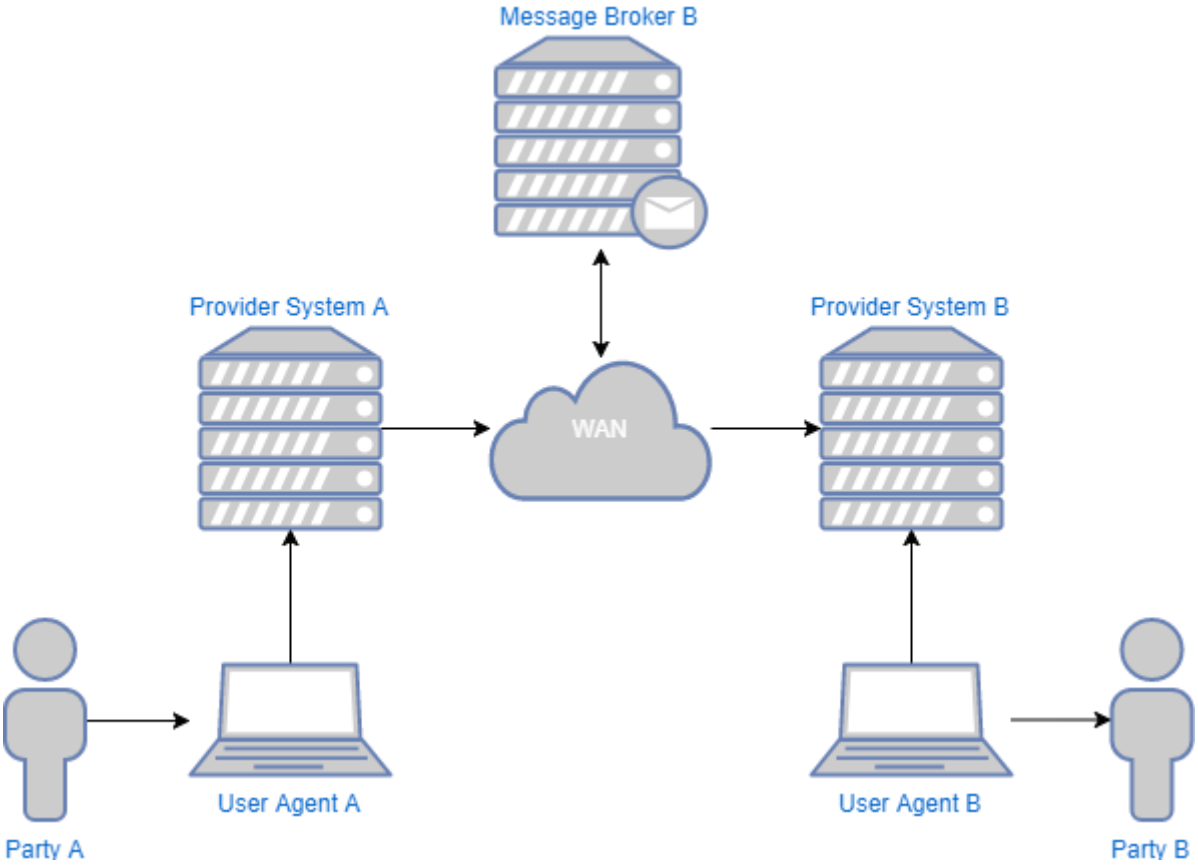


With the help of a Provider System (PS) it is possible to delegate the handling of the EDI-protocol to another party. A PS acts like any other party in the EDI-Protocol but legally on behalf of an actual party. If a PS is used, the PS is the effective endpoint of the EDI-Protocol and target of the message encryption. Any further protocols or mechanisms between the *Party B* and the PS are not part of this specification. The PS is registered to a MB like any other Party in the System. If a party is behind a PS than the messages have to be enriched about a sub address. A PS can have multiple parties.

In the case Party A wants to send an EDI-Message to Party B, Party A has to create a transmission on the MB with the PS as the transfer-target.

The PS can provide the EDI-Protocol and can be a message broker too.
The PS is free to implement its own inbox mechanism to deliver the messages to *Party B*.

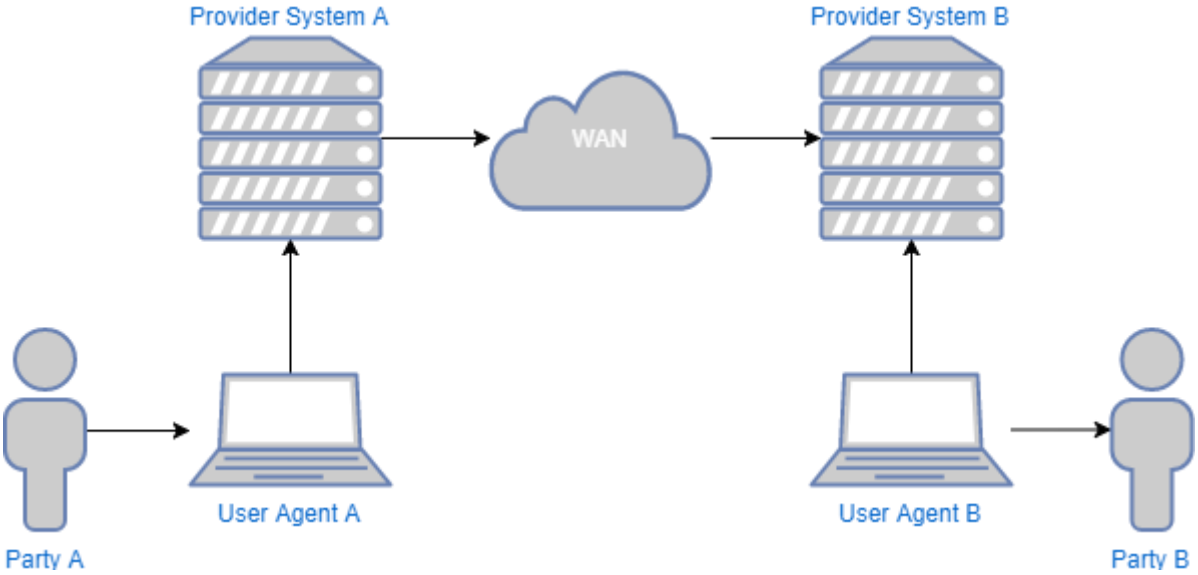
V. Provider to Provider with external Message Broker



The case of provider to provider communication will be the most common way.
Each party is registered on a Provider System (PS) and each PS is registered on a MB.
If *Party A* wants to send a message to *Party B*, then *Party A* uploads the EDI-Message to *PS A*.
PS A will sign the message with its own certificate and encrypt the Data with the key of PS B.

After the encryption, the message is transmitted to the MB B where PS B is registered.
PS B gets the message and decrypts it. PS B provides the message to Party B.
If Party B sends a response, the process is reversed.

VI. Provider to Provider

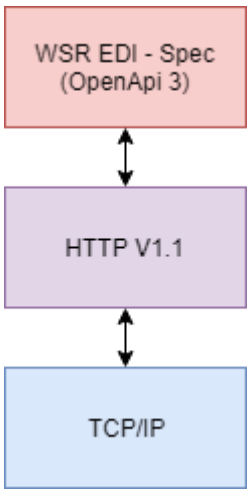


In this case, two providers communicate directly together. Because both providers are a normal party in the system, this scenario is covered due to the peer to peer case.
The sub addressing on the provider is embedded in the transmission message.

F. Transmission Protocol

This chapter describes the concrete functions and technologies, which are necessary for a transmission of an EDI-Message.

I. Technology Considerations



The used base technology is *OpenAPI 3*[5]. *OpenAPI* is based on the widely used *Swagger2* Specification.
Swagger was generalized and renamed with the switch to Version 3.
There is a great toolset around *Swagger* and *OpenAPI*. An application protocol is defined in a single *.yaml*-File like the webservice definition language (wsdl) in *SOAP*[6]. This file can be used to generate basic client and server code or an extensive development documentation. The core principle of *OpenAPI* is the *RESTful*[7] api design, which is broadly used in many other protocols.
Additionally the protocol is available on modern language and frameworks like *C# .NET Core* and the *go*[8] programming language. *SOAP* has a lack of support on these languages. Even if there is no *OpenAPI* support in the targeted language, it is possible to write the protocol on your own.

---

[5] https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md
[6] https://de.wikipedia.org/wiki/SOAP
[7] https://de.wikipedia.org/wiki/Representational_State_Transfer
[8] https://golang.org/

OpenAPI is based on HTTP as transport layer, which is proxy aware and easy to implement.
It is possible to use standard load balancers to scale up your message broker respectively the provider system.

II. Addressing Scheme

It is necessary to use an addressing scheme, which uniquely addresses a party´s inbox or the network endpoint in case of the peer to peer communication model. If the party is a provider an additional sub target has to be provided.

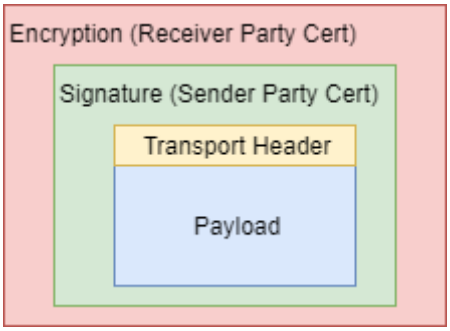
The addressing scheme consists of three values. The *broker* is a URI [RFC 3986[9]] which specifies the network endpoint with the embedded transport protocol. The party is the target of the encrypted EDI-Message and the *sub_target* is the party behind a provider, if one is used.

Example:

```
{
    "broker": "https://edi.wsr.eu",
    "party": "Provider A",
    "sub_target": "XYZ"
}
```

Sub addressing of parties will not be provided to the Message Broker directly. If a sub address is specified for an EDI-Message the address is embedded in the encrypted Message.

III. Transmission Message Format



The inner layer of the Transmission Message Format (TMF) consists of an optional header and the EDI-Message as payload. The header consists of a JSON object [RFC 8259[10]] with a possible sub address, if the target party is located behind a provider.
If the sender wants a response to the EDI-Message an additional *response_to* field can be provided in the header. The *response_to* field have to provide a full target address. *sub_target* and *response_to* addresses are within the signature and any modifications will be detected. To make sure that a message comes from a known party, the certificate of the signature can be checked. The outermost encryption layer guarantees that the Message Broker can´t read the specific message or even knows who is the communication partner of the receiver party.

[9] https://tools.ietf.org/html/rfc3986
[10] https://tools.ietf.org/html/rfc8259

Example transmission without signature and encryption:

```
{
    "sub_target": "<sub_target>",
    "response_to":
    {
        "broker": "<broker>",
        "party": "<provider_id>",
        "sub_target": "<sub_target>"
    }
}
some binary payload
```

The fields *sub_target* and *response_to* are optional.

IV. Functions

In the following part, all functions of the transmission protocol are listed.
A transmission is split into three calls for technical reasons to achieve a reliable transmission.
Semantically the transmission is only one function call.

1. CreateTransmission

This function creates a new transmission and returns a new transmission id.

The returned id is referenced in any further operations to associate the function call with this transmission. The id has to be not guessable because there is no authentication
for the sender. Everyone who knows the id can upload the data or get the current state of this transmission. A universal unique identifier (UUID) which met the [RFC 4122[11]] should be sufficient.

Path: /transmissions/create

Method: POST

Parameter:

Name	Type	Description
party	String	target party identifier

[11] https://tools.ietf.org/html/rfc4122

Responses:

Code 200:

Name	Type	Description
tid	string	The id of the created transmission.

Code 429:

Too many requests

2. Upload Data

After the creation of the transmission this function can be called to upload the message in the transmission message Format.

Path: /transmissions/{tid}/upload

Method: POST

Parameter:

Name	Type	Description
tid	string	ID of the transmission where the data should be attached.
data	TMF	The encrypted transmission message.

Responses:

Code 200:

OK, data uploaded

Code 404:

transmission does not exist

Code 412:

data already uploaded

3. GetTransmissionState

This function can be called to get the current state of the transmission.
The response will be filled with a new timestamp for the reached state on each time the state changes. The states will change in the following way: *created -> transferred -> delivered*.
In the case of peer to peer communication model, the state delivered is reached immediately after the transfer.

Path: /transmissions/{id}/state

Method: GET

Parameter:

Name	Type	Description
tid	string	ID of the transmission to which the data should be attached to.

Responses:

Code 200:

Name	Type	Description
created	string	Contains a timestamp upon the creation of the transmission
transferred	string	Occurs if the upload function has been called on the transmission and the transmission is in the inbox of the receiver.
delivered	string	Occurs if the receiver has confirmed to have downloaded the data successfully.

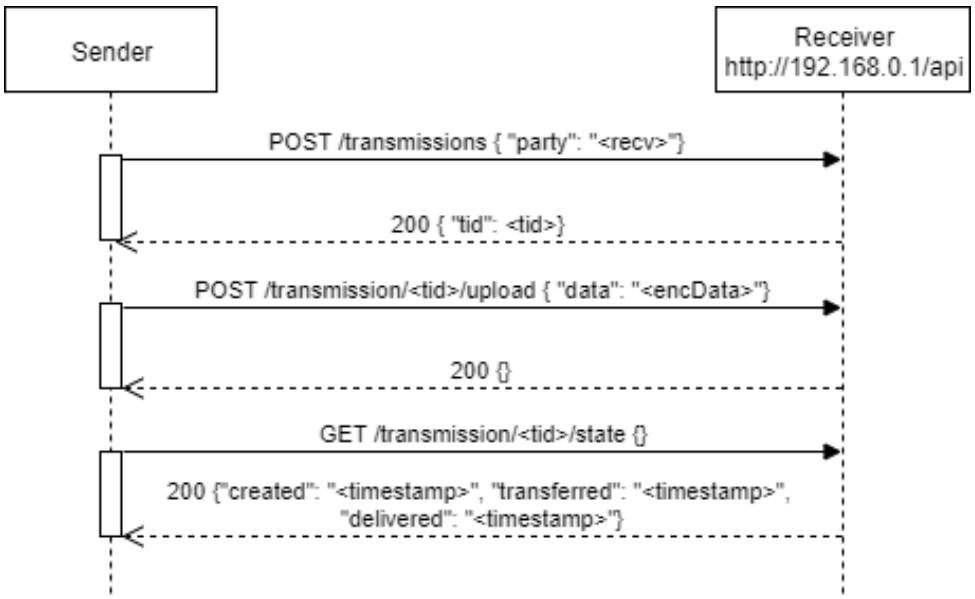
Code 404:

transmission not exists

V. Example Transmissions

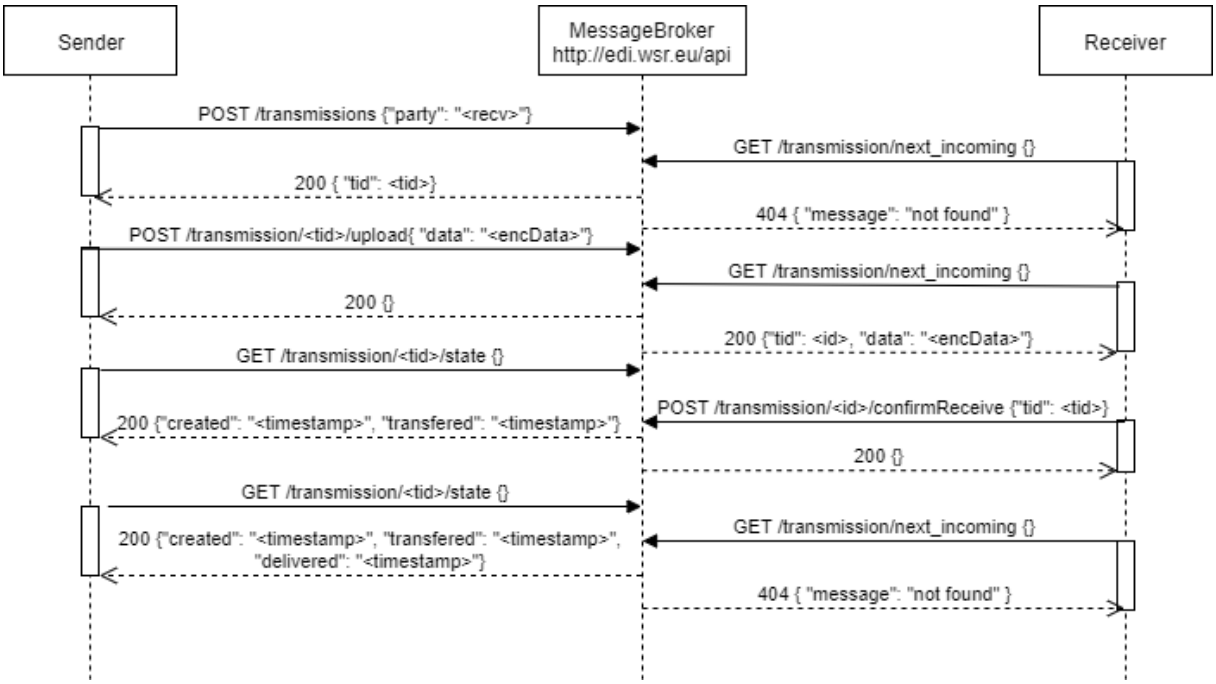
This chapter shows some example transmissions. All functions between receiver and message broker are not part of this specification. The receiver is described in the next chapters to provide an example that is as most complete as possible. Each Message Broker can provide its own inbox interface.

1. Normal Message Transmission



This scenario describes the transmission between two parties without failures in peer to peer mode where the receiver implements the transmission protocol directly.
The first network-call creates a new unique Transmission-ID. This ID is used as a reference for subsequent calls. With the additional party parameter, the transmission is created for the specified party with the *<recv>* ID. If the receiver rejects the creation, the sender might have addressed the wrong network endpoint. The state of the transmission is going to delivered immediately, because the target party has received the EDI-Message already. The protocol is synchronous with the sender as the leader of the dialog.

2. Normal Message Transmission with Message Broker



In the case with a Message Broker in between of sender and receiver the receiver does not have to be online at sending time. With a central instance, both sides have to poll the state of the transmission. The sender transmits the message to the broker in the same way as in the scenario without message broker. Transmission state is *transferred* instead of *delivered*. The receiver is addressed. After the data transfer, the receiver can get the message as the *next_incoming*-one. Each call to */transmission/next_incoming* will transfer the same message until its receipt is confirmed. The receiver has to confirm the transmission to get the next message. If no more transmissions are available the function returns „not found“. The time between GET */transmission/next_incoming* with 204 error and the next one should at least be one minute. The delivered state of the message is reached after the receipt is confirmed by the receiver.

3. Asynchronous Message Transmission

To reduce the state polling on sender side, the receiver can send a response to the sender's inbox. The sender can check his own inbox before the state check on the message broker is performed. This widely eliminates all the polling on a message broker, but if the response keeps absent the sender can check the current state of the transmission.
No additional functionality is required on the protocol layer to achieve asynchrony.
In peer to peer mode, no polling is required at all.

G. Security Considerations

In this chapter common security aspects will be discussed.

I. Payload Encryption

Each transferred EDI-Message has to be encrypted with the certificate of the receiving party or the certificate of the party´s provider system.
Because the system is designed to use an end to end cryptography, the method has to be negotiated between the two parties involved in the transfer.
For the encryption the Cryptographic Message Syntax (CMS[12]) format is used.
The used algorithms have to be recommended in the actual ETSI specification[13].

II. Authenticity

A PKCS7 signature of the payload is performed before the transmission of an EDI-Message. The party´s certificate is used for the signature. If a provider is involved, the provider has to sign the message instead. The certificates have to be exchanged in a trusted way. The integrity of the message is ensured because of the electronic signature. Any modifications will be detected on the receiver side.

III. Transport Layer Security

Even if the EDI-Message is encrypted separately, the transport layer should be encrypted too.
In this case the transferred meta data is fully encrypted and secured.
Additionally the client can verify by itself to speak with the right network endpoint.

IV. Perfect forward secrecy

Perfect forward secrecy should be activated on transport layer.

The use of perfect forward secrecy is hard to achieve in a communication scenario where one partner is offline. If perfect forward secrecy is a requirement for two parties on payload level, then

---

[12] https://tools.ietf.org/html/rfc5652
[13] https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.02.02_60/ts_119312v010202p.pdf

they should communicate in the peer to peer model without the use of an additional message broker.

V. Data Privacy

In the message brokers point of view, there is no personal data used or stored.
The inbox names are normally juristic persons or freely assigned names.

Between a provider and the represented parties, a data protection agreement has to be created. The provider decrypts the EDI-Messages and has access to the entire data, including the communication list in the message.

VI. Spam Protection

Whitelists or blacklists have to be realised on the user agent of the party or the provider system. Message Brokers have no access to any information to decline a message automatically. The sender certificates are only visible to the provider or party.

Instead of the distinct choice between a whitelist or a blacklist there could be a hybrid way. Generally, a whitelist can be used and each certificate with its root is in the eIDAS trusted list could automatically be accepted.

VII. Denial of Service Protection

A public inbox is a possible target for Denial of Service (DoS[14]) attacks.
It should be considered to use a memory limit per inbox or a max message count. But this can be used to denial the service of a receiver, if an attacker keeps the inbox of a party at the limit. If the limit is reached easily, a send limit on the sender´s IP address can be installed.

[14] https://de.wikipedia.org/wiki/Denial_of_Service