OJP for EU-Spirit

OJP Profile Definition and Implementation Concept for EU-Spirit

30. August 2019
Version 1.1
Peter von Grumbkow HaCon Ingenieurgesellschaft mbH Lister Straße 1530163 Hannover Germany

Contents

Document Version

Date	Version	Author	Remarks
31.03.2019	0.1	PvG	Initial version
08.04.2019	0.2	PvG	Questions from Johan Rix
26.06.2019	0.3	PvG	Integration of comments from workshop on 04.04.2019; official layout; joining of trains
11.07.2019	0.4	PvG	Clarification concerning system IDs; added details for wishes to OJP v1.1.
29.08.2019	1.0	PvG	Document completed; added EU-Spirit details; defined extensions; added details on references and system IDs
30.08.2019	1.1	PvG	Added chapter on simple profile

1 Introduction

EU-Spirit is a distributed journey planning system, which currently uses a proprietary XML interface for the communication between the different components. This interface shall be exchanged by the CEN standard interface OJP ("Open API for distributed journey planning"). Therefore all components using or serving the interface have to be changed.
This document is not a documentation of OJP. Instead it describes the usage of OJP especially for EU-Spirit. This kind of description is called a “profile”. Everything within OJP, what is not covered by this document shall be handled as described in the OJP documentation. This applies to services and to attributes in requests and responses.

1.1 External Information

The official documentation of OJP can be obtained here:
http://www.normes-donnees-tc.org/wp-content/uploads/2017/01/TC_278_WI_00278420_E-RS-170118-final3.pdf
The official XSD files of OJP can be obtained here:
http://www.normes-donnees-tc.org/wp-content/uploads/2017/01/OJP-xsd_CEN-2016.zip
There is a discussion forum for OJP operated by the VDV in Germany. We propose to use that for discussions and questions dealing with OJP in general or its specific use within EU-Spirit. It is accessible here:
General OJP forum: https://forum.vdv.de/viewforum.php?f=88
Specific OJP in EU-Spirit forum: https://forum.vdv.de/viewforum.php?f=108
OJP is maintained and further developed by the CEN TC278-WG3-SG8. Please consider contributing to this group by being a part of it.

2 General information

2.1 Concerning OJP

Generally, a component should implement its OJP services with the best effort principles: functionality a component can offer, it should offer. I.e. it should accept as many parameters in requests and give as much information possible in responses. The latter includes explicitly sending a complete context structure for the respective response.
Also, attributes and elements not being explicitly mentioned in this document should be handled or filled respectively as far as possible. Please refer to the OJP documentation.

2.2 Concerning EU-Spirit

The technical and algorithmic principle of EU-Spirit stays the same. OJP simply replaces the existing interface. Therefore, this document will not describe in detail the EU-Spirit algorithm and request sequences. Instead it will often refer to the existing knowledge about EU-Spirit.

Nevertheless, chapter 10 contains some information an EU-Spirit specific usage of services.

2.3 EU-Spirit components affected by OJP

There are three types of components in EU-Spirit being affected by the switch of interface to OJP:

- Active Server: Component, which acts as the back-end for the UI, and which uses the services of the RCC (location identification, journey planning, stop events, service information).
- RCC: "Ring connection composer". Component which is asked by the active servers. In order to fulfill queries of active server, it uses the services of the passive servers. I.e. it distributes the queries and composes the corresponding responses.
- Passive Server: Component, which is queried by the RCC, in order to fulfill partial tasks (location identification, partial journey planning, stop events, service information).

Between these three components, two communication paths exist, where currently the legacy DELFI API is used and where in future OJP shall be used: between active server and RCC; and between RCC and passive server. This document will deal only with these two communication paths. Additional ones can be thought of (e.g. third-party access of active servers using OJP) but are not covered in this document.

2.4 Terms

OJP has three bases:

- The structure and general envelope of messages is based on the CEN standard SIRI.
- The structure and content of services is mainly based on the German standard interface TRIAS.
- The terms are mainly based on the CEN standard TransModel.

This leads to some changes of terms compared to the existing EU-Spirit. The following list tries to explain some of the new terms:

- Exchange point
  This is equivalent to the EU-Spirit term “transition point”.
- Place
  TransModel uses the term “place” for what we called a “location” in EU-Spirit. So, stops, addresses, POIs, etc. are a place. Nonetheless, the service for retrieving places is called *LocationInformation*.
- Trip
  In legacy EU-Spirit this was called a “schedule”.
- System ID
  In legacy EU-Spirit this was called “provider code”.

3 Service Overview

3.1 Service Description

Please refer to the OJP documentation concerning the general structure of OJP messages. OJP currently defines 7 services, all of them based on the request/response schema. All corresponding requests of the OJP services are defined within
OJPRequestStructure.RequestGroup.ServiceRequestGroup.ServiceRequest.
The respective response definitions can be found in
OJPResponseStructure.ResponseGroup.DeliveryResponseGroup.ServiceDelivery.ServiceDeliveryBodyGroup.
OJP defines 7 services:

- *OJPLocationInformation*
  Retrieve places (e.g. stops, addresses, POIs). Equivalent to the legacy service *Locations*.
- *OJPTrip*
  Retrieve trips. Equivalent to the legacy service *Connections*.
- *OJPFare*
  Retrieve fare information, either static ones, for a stop, for a trip, or for multiple trips. In parts equivalent to the legacy service *RefineConnections*. Will not be used in EU-Spirit, as fares shall be transported within the services *Trip* and *MultiPointTrip*.
- *OJPStopEvents*
  Retrieve stop events, e.g. for departure or arrival boards. There is no equivalent legacy service.
- *OJPTripInfo*
  Retrieve detailed information on a specific service, e.g. the complete list of calls. There is no equivalent legacy service.
- *OJPExchangePoints*
  Retrieve exchange points, either all of them or for a specific reference location. Equivalent to the legacy services *Transitions* and *AllTransitions*.
- *OJPMultiPointTrip*
  Retrieve trips with multiple origin and/or destination places given. Equivalent to the legacy service *PartialConnections*.

In the following chapters the leading “OJP” of the services is not mentioned anymore. Instead of *OJPTrip* we will talk of the service *Trip*.

3.2 Services offered by the RCC

The following services will be offered by the RCC:

- *LocationInformation*
- *Trip*
- *StopEvents*
- *TripInfo*

3.3 Services needed from passive servers

The following services must be offered by passive servers:

- *LocationInformation*
- *Trip*
- *StopEvents*
- *TripInfo*
- *ExchangePoints*
- *MultiPointTrip*

3.4 Public transport modes and products

Different to the legacy EU-Spirit interface, OJP defines a given set of public transport modes and sub-modes. Most of the service requests include the possibility to apply a filter of public transport modes through the structure *PtModeFilterStructure*. Within this structure filters can be defined on two levels: basic modes and detailed sub-modes. Please refer to the OJP documentation about the meaning of those.
When the services return mode specific information (e.g. on stops or services), the same modes are used, but this time through the structure *ModeStructure*.

4 Services in detail

This chapter contains details concerning each service needed in EU-Spirit. It mentions attributes and elements which are mandatory to be supported for EU-Spirit, even if they are optional within the OJP specification. The lists also mention attributes or elements which are not needed to support.
A statement like “It is required to support the policy *XXX*” referring to a request means, that the meant system not only accepts such requests, but acts accordingly and returns the corresponding results.
In general, all passive servers should follow the best effort principle as far as possible. Accept as many parameters in requests as possible and act accordingly. Give as much information as possible in responses. If a request parameter could not be supported, try to still fulfil the rest of request. E.g. when your system does not support public transport sub-modes, ignore them and try to calculate without them.

4.1 LocationInformation

LocationInformation is a service, which is does not need to be called with references to already existing objects. The usual use-case of this service is to transform a user’s input into possibly meant places. The resulting places can be then used for feeding the services *Trip*, *StopEvents*, and *MultiTrip*.

4.1.1 RCC

- The RCC will pass requests mostly unchanged (except references) from caller to passive servers. It will also pass the respective responses from the passive servers to the caller.

- The RCC will support all filters and policies. The RCC will not apply them itself but pass them to the passive servers.
- The RCC will not support an empty *InitialInput*. It will revoke such requests.
- The RCC will support *InitialInput* with *LocationName* and/or *GeoPosition* and/or *GeoRestriction*.

4.1.2 Passive servers

- Passive servers have to support an empty(!) *InitialInput*. I.e. that all locations can be returned. This applies especially to stops. The service implementation has to support an export of all locations via *LocationInformation*.
- The same applies to topographic places. The passive server implementation has to support an export of all its topographic places via *LocationInformation*. These topographic places should contain all city/town/village names needed, to access all stops/POIs/addresses covered by the passive server for EU-Spirit. This mechanism has to be able to replace the city-name lists currently used in EU-Spirit. (Future use, see 6)
- Passive servers have to support an *InitialInput* with *LocationName* and/or *GeoPosition* and/or *GeoRestriction* (at least up to a sensible detailed level).
- Passive servers have to fully support the following data filters: *Type*, *Usage*, *PtModes.PtMode* and *PtModes.Exclude*, *TopographicPlaceRef*. Other data filters are optional. *PtModes* has to be supported for the future automatic exchange point definition. An automatic mechanism might ask for air/ferry/train stops only (future use, see 6).
- Passive servers have to support all policy parameters in *PlacePolicyGroup*.

4.2 Trip

Trip calculates trips between (in EU-Spirit) one origin place and one destination place. These places need to be the result of a *LocationInformation* request.

4.2.1 RCC

- The RCC will pass requests mostly unchanged (except references) from caller to passive servers. It will also pass the respective responses from the passive servers to the caller.
- The RCC will support all filters and policies, except the policy *AcceptDeferredDelivery*. The RCC will not apply them itself but pass them to the passive servers.
- The RCC will not support more than one *Origin* or *Destination* place.
- The RCC will not support *Via*, *NotVia*, and *NoChangeAt*.
- The RCC will not support *PlaceContextStructure.TimeAllowance*.
- The RCC will always return a *TripResponseContext*, containing the summary of information given by the passive servers.

4.2.2 Passive servers

- It is not required to support *Via*, *NotVia*, and *NoChangeAt*.
- It is not required to support *TripLocation* as *Origin* or *Destination*.
- It is not required to support *TimeAllowance* in *Origin* or *Destination*.
- It is only required to support one single *Origin*, which contains a *PlaceRef* to a regular place. The same applies to *Destination*.
- It is required to support *PtModeFilter.PtMode* and *PtModeFilter.Exclude*.
- It is required to support the policy *NumberOfResults* and *NumberOfResultsBefore*/*After*. Other policies are optional to support.
- It is required to support the content filters *IncludeLegs* and *IncludeIntermediateStops*.
- It is required to support *IndividualTransportOptions*.
- Passive servers have to send a complete *TripContext* within the response, containing all referenced places and situations.
- It is not required to support the policy *AcceptDeferredDelivery*. The RCC will never set this parameter. Therefore, passive servers will never be asked to return *TripSummary* instead of *Trip* within the *TripResultStructure*.

4.3 StopEvents

StopEvents returns stop events for a place. This place needs to be the result of a *LocationInformation* request.

4.3.1 RCC

- The RCC will pass requests mostly unchanged (except references) from caller to passive servers. It will also pass the respective responses from the passive servers to the caller.
- The RCC will support all filters and policies. The RCC will not apply them itself but pass them to the passive servers.
- The RCC will not support *PlaceContextStructure.TimeAllowance*.
- The RCC will always return a *StopEventsResponseContext*, containing the summary of information given by the passive servers.

4.3.2 Passive servers

- It is only needed to support *PlaceRef* within the request. *StopPointRef* and *StopPlaceRef* are necessary, other types of places optional.
- *TimeAllowance* is not needed.
- *IndividualTransportOptions* are only needed, if other places than *StopPoints* or *StopPlaces* are supported.

- Passive servers have to send a complete *StopEventsContext* within the response, containing all referenced places and situations.

4.4 TripInfo

TripInfo returns detailed information for a specific service. The reference to this service has to be obtained from a result of either *Trip* or *StopEvents*.

4.4.1 RCC

- The RCC will pass requests mostly unchanged (except references) from caller to passive servers. It will also pass the respective responses from the passive servers to the caller.
- The RCC will support all filters and policies. The RCC will not apply them itself but pass them to the passive servers.
- The RCC will not support the *TimedVehicleRefGroup*.
- The RCC will always return a *TripInfoResponseContext*, containing the summary of information given by the passive servers.

4.4.2 Passive servers

- It is only needed to support requests with *DatedJourneyRef*.
- All filters and policies should be supported, according to available data.
- Passive servers have to send a complete *TripInfoContext* within the response, containing all referenced places and situations.

4.5 ExchangePoints

4.5.1 RCC

- The RCC will not support or implement this service.

4.5.2 Passive servers

- With an empty *PlaceRef*, passive servers have to return all(!) exchange points, which fit to the given data filters.
- With a given *PlaceRef*, passive servers have to return sensible exchange points, which fit to the given data filters. Sensible exchange points allow good trips to/from the given place.
- The data filter *Type* should be assumed to be always "*stop*" and only "*stop*", also when not given.

- The data filter *Usage* tells, whether the *PlaceRef* is origin, destination, or via. It can be missing.
- It is required to support the data filters *PtModes.PtMode* and *PtModes.Exclude*.
- It is required to support the data filters *DestinationSystem* and *AdjacentSystem*.
- It is required to support all policy parameters.
- Within the response it is required to fill *TravelDurationEstimate* and *BorderPoint* with appropriate values for all returned exchange points. *TravelDurationEstimate* shall contain an estimation (!) of how long the travel from origin to exchange point (or from exchange point to destination respectively) will be. *BorderPoint* tells whether an exchange point is logically located on the border between two regional systems. This is not meant geographically! At a border transition point, the changes times are not used under certain circumstances (see 7.1).
- All returned exchange points have to be either a *StopPlace* or a *StopPoint*. If a passive server has an exchange point defined on stop place level, it returns only the *StopPlace*. If a passive server has exchange points defined on stop point level, it returns those stop points, but also the corresponding stop places.

4.6 MultiPointTrip

4.6.1 RCC

- The RCC will not support or implement this service.

4.6.2 Passive servers

- It is not required to support *Via*, *NotVia* and *NoChangeAt*.
- It is not required to support *TripLocation* as *Origin* or *Destination*.
- There can be either one single *Origin*, which contains a *PlaceRef* to a regular location (not an exchange point), or a set of origins, which each contain a *PlaceRef*, where all referred places are exchange points. Other combinations are not allowed and can be rejected by the passive server.
- The same applies to *Destination*.
- With a set of exchange points as *Origin* given, either all origins contain also a *DepArrTime*, or all origins contain a *TimeAllowance*, or all contain none of both. Other combinations are not allowed and can be rejected by the passive server.
- The same applies to *Destination*.
- It is required to support *PtModeFilter.PtMode* and *PtModeFilter.Exclude*.
- It is required to support the policy *NumberOfResults* and *NumberOfResultsBefore*/*After*. It is only required to support the policy *MultiPointType* and only in the flavor of "*anyPoint*". Other policies are optional to support.
- It is required to support the content filters *IncludeLegs* and *IncludeIntermediateStops*.

- It is required to support *IndividualTransportOptions* for a single, regular location as Origin or Destination.
- It is required to send a complete *MultiPointTripContext* within the response, containing all referenced places and situations.
- It is not required to support the policy *AcceptDeferredDelivery*. The RCC will never set this parameter. Therefore, passive servers will never be asked to return *TripSummary* instead of *Trip* within the *MultiPointTripResultStructure*.

5 Usage of References and System IDs

OJP does not contain any direct way to address different systems. As EU-Spirit is a distributed system it consists of several partial systems. E.g. a *StopEvent* request to the RCC has to be directed to the "correct" passive system in the background. There is no explicit attribute within OJP for the RCC to identify the correct passive system. Instead the different *XXXRefs* have to include an appropriate identifier for the actually addressed passive system.
The RCC will therefore extend all relevant references it gets from the passive systems by a system ID (formerly known as “provider code”). Whenever a passive system sends any kind of reference to the RCC, the RCC will prefix it with "<system ID>:" before sending the response to the active server. Whenever an active server sends a request to the RCC, the RCC strips all contained references by the system ID before sending the corresponding requests to the passive systems.
The prefix can be also interpreted (but never changed!) by active servers in order to recognize, which system originally provided the corresponding information.

The RCC will add/discard the system ID to/from the following references:

- *StopPointRef*
- *StopPlaceRef*
- *TopographicPlaceRef / TopographicPlaceCode*
- *PointOfInterestRef / PointOfInterestCode*
- *AddressRef / AdressCode*
- *SituationBaseIdentityGroup.SituationNumber*
- *OperatingDayRef*
- *VehicleRef*
- *JourneyRef*
- *LineRef*
- *DirectionRef*
- *OperatorRef*
- *FareProductId*
- *FareAuthorityRef*

The actually used system IDs in EU-Spirit will be managed separately and will consist in the first step of the previous provider codes.

6 OJP extensions used in EU-Spirit

The attached file “OJP-Extensions for EU-Spirit.xsd” contains the type definitions for the needed extensions.

6.1 System filter for *LocationInformation*

Define an extension in *OJPLocationInformationRequestStructure*, which takes a sequence of system IDs as a system filter for *LocationInformation* requests. A client system shall be able to look for locations only in specified systems. Therefore such a system filter has to be part of a *LocationInformation* request.

<xs:element name="SystemFilter" type="SystemFilterStructure"/>

6.2 Extensions in *PlaceStructure*

Define an extension in *PlaceStructure*, which takes the waiting time of the exchange point as an *nonNegativeInteger*. For *StopPoints* always 0.
The same extension contains information, whether the place is a city name result or not. If yes, the extension contains the system ID of the system to ask for actual locations in a subsequent request.

<xs:element name="PlaceExtension" type="PlaceExtensionStructure"/>

6.3 Extensions in *TripStructure*

Define an extension in *TripStructure*, which takes the Öresund tariff region for origin and destination. The same extensions can take the line ID at origin and destination as a *normalizedString*.

<xs:element name="TripExtension" type="TripExtensionStructure"/>

6.4 Webview URLs for ticketing

Although not being exactly an extension, we define, that, if filled, the field *FareProductStructure.FareProductBookingGroup.SaleUrl* contains an URL pointing to a fully responsive webview, which can be used within any container. The webview itself should contain further ticketing information and/or the possibility to sell the ticket. The URL can also be a universal link URL, which automatically redirects to the correct ticketing platform, depending on used OS and context.

7 Additional differences to legacy EU-Spirit

7.1 Joining of trains

The legacy EU-Spirit system is able to “join” trains. At border exchange points the RCC looks, whether the exchange point is reached and left by the same service. If yes, currently the RCC joins the two corresponding legs into one leg. This will not be done anymore!
In future, the RCC will still look, whether a border exchange point is reached and left by the same service. However, the RCC will not join the corresponding legs into one. Instead it will generate between the two corresponding legs an additional *TransferLeg* with a *TransferMode* set to *remainInVehicle*. Additionally the RCC will not count a transfer for this in *TripStructure.Transfers*.

8 Future meta-data mechanisms

Currently the needed meta-data (specifically city names and exchange points) is provided by each partner, collected, and distributed in a separate data process, which is not part of the legacy interface used for EU-Spirit. Using OJP in future enables automatic mechanisms for gathering this meta-data.

8.1 Exchange points

Exchange points are stop places or stop points where it is possible to change from one system to another. An automatic mechanism for defining these points could do the following:

- Gather all stop places from all EU-Spirit systems.
- Automatically identify which regional stops match which stops in the long distance system. The found matches are transition points.
- Automatically identify which regional stops match which regional stops from neighboring regions. The found matches are regional transition points.

A detailed concept will contain a certain complexity and might reveal additional obstacles or requirements.

8.2 City name list

In order to do location identification in a distributed system, it is required to conclude from the user’s input which regional system is responsible for the actual identification. Therefore a collection of names for cities and town can be used. For each name the responsible system is listed. An automatic mechanism for gathering there lists could do the following:

- Gather all topographic places from all EU-Spirit systems.
- Generate a complete list which is used for matching the user’s input.

8.3 Collection of all places

Instead of collecting city names, the RCC could also collect all locations from all systems. Location identification would be then done completely inside the RCC. This solution would have the best performance.

9 Implementation Concept

Experiences with distributed system in general and specifically with EU-Spirit showed that a long testing phase is necessary. Interpretations of specifications are different and lead to different implementations. Therefore the main goal of the implementation concept is to start combining systems as early as possible. Instead of a long implementation phase without interaction between the partners, we like to have communication systems in an early phase.
This leads roughly to the following steps:

- April 2019: OJP profile for EU-Spirit defined for beginning implementation.
- May 2019 until December 2019: First implementation phase. Each partner should implement its systems as far as possible. In this phase the programmers will work full-time.
- January 2020 (perhaps earlier): Building up a test system with RCC, passive servers and active servers of all partners. All participating systems should be as far implemented as it is needed for this first system integration.
- January 2020 until December 2020: Technical testing, error analysis, error correction, concept clarification, implementation changes, implementation completion will have to take place. In the end all parts of the system shall be ready for production. In this phase all partners will need to respond fast to input from other partners. Most times no full-time programmers will be needed.
- January 2021: Take system into production.

10 Excerpt of specification details for EU-Spirit

10.1 MultiPointTrip

0) The principal algorithm how the RCC queries passive systems and how passive systems should behave is not changed. Some clarification how *MultiPointTrip* will be used within this unchanged algorithm is given below.
1) The RCC uses *Trip* for requesting complete itineraries from one single passive system, i.e. with both origin and destination lying in the same region.
2) The RCC uses *MultiPointTrip* for requesting partial itineraries from passive systems, i.e. at least one side consists of exchange points.
3) In a 3-system-scenario (e.g. Berlin/flight/Denmark, or Scania/Sweden/Stockholm) the long distance system will be queried with *MultiPointTrip* with the policy filter *MultiPointType* set to "*anyPoint*" Origin and destination then consist of exchange points. Depending on the request direction (arrival departure), either all origins (for departure) or all destinations (for arrival) will have set a *DepArrTime*. The respective other side will have relative(!) times set in the *TimeAllowance*.
   The queried long distance system should return computed partial trips. Please remember, that it is not

necessary to compute itineraries to all exchange points, but rather to compute itineraries which are overall "optimal", including the travel times to and from exchange points.
A special case can occur, if either real origin or real destination location lies within a long distance system. The respective long distance system is then queried with origin (or destination) being the real location and the other side being exchange points. Everything else stays as described above.

4) In a 2-system scenario (e.g. Scania/Denmark, or Rheinland-Pfalz/Luxemburg) one of the regional systems is queried first. For departure searches the origin system is queried first, for arrival searches the destination system.
In case of a departure search this first request is a *MultiPointTrip* where the origin location is a real location with an absolute(!) time in *DepArrTime*. The destination locations are exchange points with relative times in *TimeAllowance*. The result should be partial trips.
In case of an arrival search this first request is a *MultiPointTrip* where the destination location is a real location with an absolute(!) time in *DepArrTime*. The origin locations are exchange points with relative times in *TimeAllowance*. The result should be partial trips.
5) For computing connecting trips (in order to complete a long distance trip or the trips computed by the first 2-system-scenario request) the RCC queries a passive server with *MultiPointTrip*. Either destination (origin region) or origin (destination region) consists of one exchange point with an absolute (!) time set in *DepArrTime*. The other side will be a real location with no given time.
The queried system should return computed partial trips.

10.2 LocationInformation

0) The principal algorithm how the RODI works and what it returns is not changed. However, some clarification how *LocationInformation* will be used is given below. The RODI can either be queried either for locations in one single region, or for locations in several regions (city name request). In the latter case, the RODI in the first step tries to identify the meant city and deduce the meant region from this. When in question, the RODI cannot decide and will return city name results as a first step.
1) When the RODI gets a city name request it computes a list of matching city names together with their respective providers and returns it.
2) A city name request to the RODI is similar to a normal *LocationInformation* request. *InitialInput* has to be filled and the system filter (see 6.1) set to be either empty (or missing) or to a list of systems. If the system filter contains exactly one system, the *LocationInformation* request will be forwarded directly to the corresponding system.
3) City name results are topographic places and marked with a flag to be city name results (see **Fehler! erweisquelle konnte nicht gefunden werden.**). They will also carry the system ID the place belongs to (see **Fehler! Verweisquelle konnte nicht gefunden werden.**).
4) In order to use a city name result for retrieving actual location, an active server should generate a new *LocationInformation* request. *InitialInput* should be re-used from the previous request, adding *GeoPosition* from the respective city name result. As a system filter, the system ID of the city name result should be set.

11 OJP profile “simple”

Apart from the distributed journey planning use-case, OJP can also be used as an API for third-party applications to access and use a journey planning system. In order to build such a stand-alone OJP server, many of the above mentioned topics are not relevant and can therefore be ignored during implementation. Some services or features of OJP are not used in that case and some background knowledge is not needed.

However, if an OJP server fulfills the EU-Spirit profile, it can also be used for the third-party use-case. If only the simple profile is implemented, the resulting service cannot be used for distributed journey planning.

11.1 Chapters relevant for simple profile

- Chapter 1.1 contains links to OJP related information.
- Chapters 2.1 and 2.4 explain terms and general principles for OJP services.
- Chapter 3.1 presents an overview of the OJP services.
- The detailed definition of needed features for each OJP service from chapter 4 is in parts also used for the simple profile. Some chapters dealing with the passive servers define the requirements for an OJP server fulfilling the simple profile. OJP services dealing with distributed journey planning are left aside. The relevant chapters for the simple profile are therefore 4.1.2, 4.2.2, 4.3.2, and 4.4.2.

12 ToDo

- Typical sequence of calls (as table).

13 Wishes for OJP v1.1 (and for TRIAS v1.3)

- Extension for OJP v2.0: Take over the section definition from TRIAS, in order to allow changes of service parameters through a leg.
- Extension for OJP v1.1: Define a field in *LegBoard*/*LegAlight* which consumes an arbitrary identifier of the service (usually not identical to the *JourneyRef*) at exactly that place (e.g. the train number). This shall be used for the search controller in EU-Spirit being able to recognize the usage of the same train by two adjacent systems. Alternatively, two such fields (for origin and destination) within *TripStructure* could be defined.
- Change of documentation for OJP v1.1: The field *FareProductStructure.FareProductBookingGroup.SaleUrl* should be declared in the documentation, to contain an URL pointing to a fully responsive webview, which can be used within any container. The webview itself should contain further ticketing information and/or the possibility to sell the ticket. The URL can also be a universal link URL, which automatically redirects to the correct ticketing platform, depending on used OS and context.
- Extension for OJP v1.1: Add waiting times as a *nonNegativeInteger* for exchange points in *ExchangePointResultStructure*.
- Extension for OJP v1.1: Add the possibility of a “silent” fare product. An additional boolean would define, that a specific fare product is not intended to be shown to the end user. Purpose would be to transport “incomplete” fare products, which have to be completed by other systems first.
- Extension for OJP v1.1: Add a system filter for *LocationInformation* in *InitialInput*.(Not in *PlaceParamStructure*, because the system filter is only used in initial requests.)

- Extension for OJP v1.1: Add a flag to *PlaceStructure*, whether the place is a city name result or not. Additionally it contains the system ID of the corresponding system, which is needed for subsequent requests after getting city name results from the RODI.