Integration Architecture for Driver Assistance Systems

1 Introduction

This document describes in broad terms the overall architecture planned for Driver Assistance Systems integration within the existing Finnish Transport Agency rail traffic management infrastructure.

Rather than being a normative specification, this document is intended to function as a general description of the systems, concepts and technologies involved. Exact details will be determined when DAS implementation has advanced to the appropriate stage, and extensive coordination with the FTA and its suppliers is required during the implementation phase.

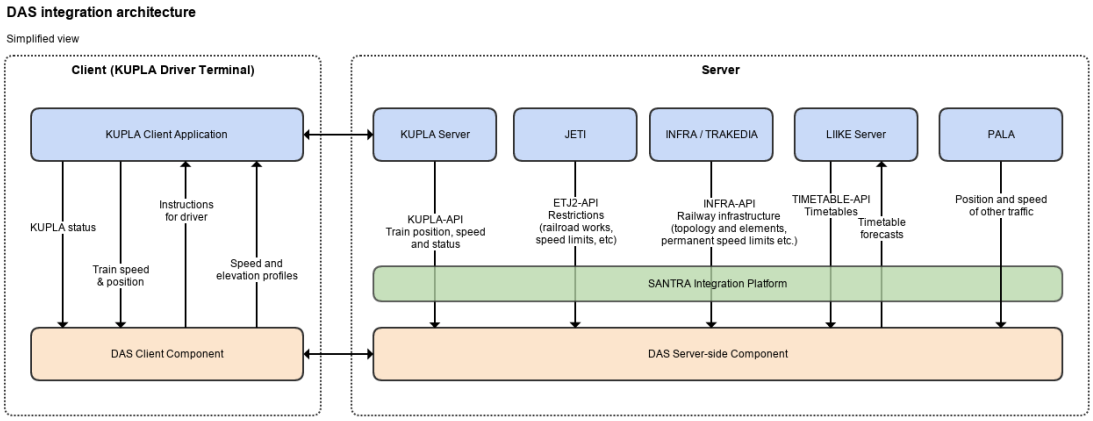
This document consists of three parts:

1. overview of the relevant systems, interfaces and principles
2. description of the integration mechanisms within the KUPLA driver terminal system
3. description of the server-side integration mechanisms

1.1 Version History

- 2015-06-01 - Initial version
- 2015-11-19
    - Additions to KUPLA/DAS Client status interface
    - Explicit mention of the use of the SANTRA integration interface on the server side
    - Clarification of railway addresses
    - Fleshed out server-side integration interfaces
    - Added PALA as an alternative source of traffic data
    - Elevation data is now included in the railway infrastructure
    - DAS/KUPLA client activation sequence clarified
    - General reorganization

1.2 Overview



The primary actors and their relationships in the DAS integration scenario are presented in the figure above. Blue rectangles represent existing systems, orange rectangles parts of the DAS system.

- The **KUPLA client application** runs on the KUPLA driver terminal and provides the train driver with timetable, limit/restriction and other information.
- The **DAS client component** provides the KUPLA client application with driving instructions. It can receive train speed and position information from the KUPLA application if needed.
- **KUPLA Server** communicates with all KUPLA clients and can optionally provide the DAS server with train status information
- **JETI** is a system for managing operational restrictions and temporary limits caused by railroad works and other events, and provides these to the DAS Server.
- **TRAKEDIA** is responsible for maintaining the permanent railway infrastructure database, including tracks, lines, switches, signals and all other rail elements.
- **LIIKE** as a whole is the system for traffic control, capacity reservations and timetable management. Its server component provides the DAS server with timetable data and receives from it forecasts of the trains' arrival times.
- **PALA** provides a network-wide view of traffic, basically the same information as KUPLA server, but optionally including other traffic
- The **DAS Server-side component** is the part of the DAS system responsible for providing timetable forecasts to traffic control and any required communication with the DAS client component.

Note that the figure is by necessity simplified, and in reality the existing server-side architecture is significantly more complex.

Arrows in the diagram represent information flows. Integration between components is typically performed using RESTful HTTP API calls, and the initiating party is normally the one requiring the requested information. If considering purely API calls, the arrows would typically be reversed.

Also note that the diagram describes only the systems ultimately responsible for the given information, and in practice multiple interfaces are often consolidated into a single endpoint.

Server-side interfaces are expected to be routed through the SANTRA integration platform. SANTRA provides http-based proxying, validation, filtering and transformation functionality and has been widely used to implement systems integration between the FTA and train operators. The presence of SANTRA does not affect the implementation details of the interfaces in any major way.

1.2.1 Process overview

This section broadly outlines the sequence of events from a train's departure to its arrival. Communication failures etc. are not accounted for.

From the KUPLA driver terminal's point of view:

1. The driver prepares the train for departure.
    a. Driver starts the KUPLA application and retrieves the timetable and restriction list for the train.
    b. The DAS client component is activated via a HTTP request by the KUPLA software
    c. Driver instructs KUPLA to enter "drive mode", i.e. to start sending tracking information to KUPLA Server
        - At this point, it is presumed that the DAS system is ready to offer instructions to the driver
2. The train departs.
    a. The KUPLA software transmits health and positioning information to the KUPLA server, and receives any new or updated restrictions or timetable changes.
        - New information is presented to the driver, who is required to confirm it; this confirmation is transmitted to the server
    b. The DAS client periodically receives positioning information from the device's positioning system or from the KUPLA software (transmitting to its server component if necessary)
    c. The DAS client, based on the positioning information and any other information it has received from its own server, presents the driver with instructions
    d. The preceding three steps are repeated as long as the train runs.

3. The train arrives at its destination.

    a. The KUPLA software exits drive mode and the DAS client component is unactivated.

From the server-side point of view:

1. A request is received by KUPLA Server from a KUPLA terminal for timetable and restriction data, which is fulfilled
    - If requested, the same data is also provided to the DAS
2. A message is received by KUPLA Server that a train is about to depart (i.e. KUPLA enters drive mode)
    - This is relayed to traffic control and other relevant parties (e.g. DAS)
    - The KUPLA server starts receiving periodical status and positioning messages from the KUPLA client
3. While the train is running, LIIKE Server monitors the train's position and speed (using data from various sources) and presents this data to a traffic controller (TC)
    a. LIIKE Server additionally periodically retrieves timetable forecast data from the DAS server and presents it to the TC
    b. The TC may manually adjust or correct the train's timetable based on the forecast or for any other reason
    c. Any adjusted timetables are provided to the DAS when requested
    d. In case of a new restriction (from JETI) or timetable change (from LIIKE), the KUPLA server relays this information to the KUPLA client
    e. The preceding four steps are repeated as necessary while the train is running
4. The train arrives at its destination.
    - The KUPLA software informs KUPLA server that the train is leaving drive mode.
    - This information is relayed to traffic control and other relevant parties (e.g. DAS)

1.2.2 Railway addresses

This document refers at various times to **railway addresses**. There are two forms:

- line kilometer -based addresses, which are unique to the entire network
    - they have the format of (AAAA) BBBB+CCCC where A is the railway line number, B is the identifier of a kilometer marker, and C is the distance to that marker in the *ascending direction* (i.e. when traveling towards the kilometer marker having a numerically larger identifier)
    - railway infrastructure modeling is based on kilometer markers
- positioning marker -based addresses
    - they have the format of BBBB+CCCC where B is the identifier of the last bypassed positioning marker, and C is the distance to that next marker *in the direction of travel*

- for example, given a distance of 1150 metres between two positioning markers 123 and 124, in the *ascending direction* a position might be 123+0400, while in the *descending direction* the same position would be 124+0750.
- they do not include line numbers
- unlike kilometer-based markers, positioning markers are visible to the train drivers and the KUPLA client therefore uses them
- positioning and kilometer marker locations and identifiers are usually, but not always, similar

The infrastructure API will provide positions in both formats.

2 Integration within the KUPLA driver terminal system

2.1 Overview of the KUPLA driver information system

The KUPLA system, scheduled for initial deployment in Q2/2015, consists of a touchscreen tablet device and accompanying software. Use of the system is required of all Finnish train operators, with the hardware supplied and owned by the operators themselves and the software by the Finnish Transport Agency. The purpose of the system is to provide information that currently is disseminated by paper printouts, such as train-specific timetables and notifications of railroad works and other operational restrictions. The aim is to lessen the workload of traffic management personnel, facilitate information flow between traffic management and train drivers, and to allow for more efficient usage of railway capacity.

In addition of the terminal device and its software, the family of systems KUPLA belongs to includes various other components, such as railway infrastructure data management systems (TRAKEDIA), monitoring software for traffic control (LIIKE and LOKI) and operational restriction management software (JETI). These other systems are all owned by the FTA and operated by it or its subcontractors and integrate with the KUPLA terminal solely via its server-side counterpart, KUPLA Server.

The initial operational version of the KUPLA terminal software provides the train driver with a real-time view into a variety of information. This includes the train's timetable and any restrictions such as speed limits, power draw limitations and abnormal track allocations superimposed on a rudimentary view of the railway infrastructure. The information shown is chosen based on the train's position and speed, which is tracked via a GPS receiver and transmitted in real time to traffic control along with various other data. Any new traffic restrictions on the train's path are automatically delivered to the KUPLA system in real time. Offline operation in areas of poor network and/or GPS coverage is also supported to a limited degree.

The terminal device KUPLA runs on is an ordinary Windows 8 (x86, not RT) tablet computer. Installation of other applications besides the KUPLA system itself is permitted, but use of the KUPLA software is mandatory

when the train is being driven. KUPLA itself enforces this and does not allow interaction with other applications while the train is underway. Any DAS software installed on the terminal must therefore coordinate with the KUPLA software for information display and user interaction purposes.

Specifications for the KUPLA terminal device are as follows:

- Device type: touchscreen tablet computer, x86 architecture, 10" screen size or larger, 720p display or better, reasonable CPU, memory and storage capacity
- Operating system: Microsoft Windows 8 or later (x86)
- Communications: Internet connection via 3G and/or LTE data and/or wireless LAN, connections provided by the train operator
    - Connection coverage is required to exceed 95% of the entire railway network
- Positioning: integrated or external GPS/GLONASS/GALILEO receivers optionally supported by accelerometers, tachometers etc. provided by the train operator
    - All positioning information is delivered to installed software via an NMEA 0183 serial connection and/or the Windows Location API
    - Positioning coverage is required to exceed 95% of the entire railway network
    - Positioning options within tunnels are being investigated

2.2 Basic principles of operation

The DAS sofware must be installed in the KUPLA device as a standalone application or service. Ensuring that the software is running and enabled when needed is the responsibility of the DAS supplier and the train operator.

The only user interface permitted to be displayed when the train is underway is that of the KUPLA software. The DAS software must not attempt to display a UI of its own, nor otherwise interfere in the operation of the KUPLA software.

The DAS software must not attempt to gain exclusive access to system resources such as human interface devices, positioning devices or network connections.

The DAS software must not consume system resources (CPU, memory, storage, network bandwidth etc.) to the degree that the operation of the KUPLA software is visibly hindered.

On the KUPLA device, up to four interfaces are present:

1. Provided by the KUPLA software:

a. for requesting current train information and position data

b. for requesting KUPLA to show a specific instruction to the driver

c. for providing a view to the driver of the elevation and speed profiles of the train's route

2. Optionally provided by the DAS client component:

a. for receiving status events related to the lifecycle of the KUPLA software

The DAS client component may, if necessary, provide a HTTP interface of its own to receive status messages from the KUPLA software.
The services are provided as HTTP endpoints bound to the local loopback interface (127.0.0.1 or ::1). It is the responsibility of the DAS software to establish and manage connections to these endpoints. The exact ports and request URIs of the endpoints are identical across all KUPLA installations, but nevertheless should be configurable within the DAS software.
Each HTTP request will result in one item of information being exchanged. Keep-alive may be used to avoid always opening a new connection. Regardless of keep-alive being used or not, the DAS software must not make more than one request per second to each endpoint.
The data transferred over these connections is in the JSON format, content type application/json, in the UTF-8 encoding.

2.3 Positioning interface

Each HTTP GET request to this interface results in a reply containing a JSON data structure as its body that includes

- train number
- train position as provided by the GPS receiver (coordinate system TBD)
- precision estimate of indicated position (format and values TBD)
- current train speed in km/h
- direction of travel (degrees)
- estimated railway address (positioning marker -based) and direction of travel in relation to the track definition
    - Direction is ascending (identifier of the next marker is numerically larger than the preceding marker) or descending (v.v.)
    - Address is a linear estimation and can have an error in the tens of meters in addition to any errors in the GPS position

- identifier of previous station on the line (if any)
- identifier of next station on the line (if any)

The request will always return a JSON data structure, even if positioning information is not available. Returned data in such cases is TBD.
The DAS software is not obligated to use this interface, if it prefers to receive position data directly from the positioning system in the device and provided such direct reception does not interfere with the KUPLA software.
This interface may be called by the DAS software as often as required, within reason. The KUPLA software may cache some or all of the information (e.g. updating positioning data in 5-second intervals even if the interface is called every second).

2.4 Driver instruction interface

Each HTTP POST request to this interface must include as its body a JSON data structure containing

- current target speed the driver should accelerate/decelerate to, in km/h
- type of instruction for the driver (one of: accelerate, maintain speed, let roll (ie. apply neither acceleration nor brakes), brake, clear)
    - exact values TBD
- optionally: estimated time until next instruction

The KUPLA software displays the last received target speed and instruction to the driver until it receives another. It will automatically count down the estimated time to the next instruction if such information is provided by the DAS. If the time runs out before a new instruction is relayed to the KUPLA software, the last received instruction will remain visible without the timeout counter.
If the instruction type is "clear", the last instruction received is removed from view and no new instruction is shown.
If the received JSON data structure is valid, the KUPLA sofware will respond with status code 204 (No Content). If it is malformed or otherwise erroneous, the response code is 400 (Bad Request) with an optional error message in the response body.

2.5 Elevation and speed profile interface

Each HTTP POST request to this interface must include as its body a JSON data structure containing one or more elements that represent points on the track. (*Note*: not to be confused with the points of a railroad switch)

For each point, information to be provided must consist of

- railway address corresponding to the point (see the positioning interface above for a description)
- elevation at the point described by the railway address (whether relative or absolute TBD)
- *optionally* optimal train speed at the point, if available

This interface may be called as often as necessary (within reason), and it is not mandatory. The KUPLA software will use the provided information to display to the driver the elevation and speed profiles of the train's route.
If the received JSON data structure is valid, the KUPLA sofware will respond with status code 204 (No Content). If it is malformed or otherwise erroneous, the response code is 400 (Bad Request) with an optional error message in the response body.

2.6 KUPLA client status interface

This optional interface is provided by the DAS client component, and will accept HTTP POST requests relaying the status of the KUPLA terminal software. The payload for this interface is a JSON data structure consisting of the status of the KUPLA application, one of the following:

- application started
    - as additional payload this message also includes the KUPLA registration key, a 16-decimal hexadecimal string, which uniquely identifies this particular KUPLA installation
- the driver has selected a train
    - as additional payload this message also includes the train number and date
- application enters drive mode
- application leaves drive mode
- application is about to exit

If the received JSON data structure is valid, the DAS sofware must respond with status code 204 (No Content). If it is malformed or otherwise erroneous, the response code is 400 (Bad Request) with an optional error message in the response body.
It is the responsibility of the DAS client component to react to (or ignore) these messages as necessary.

2.7 Server-side Integration

The server side integration consists of the following interfaces:

1. Timetable data, provided via TIMETABLE-API

2. Timetable forecast, provided by the DAS
3. Railway infrastructure data and permanent speed limits, provided via INFRA-API
4. Restrictions and temporary speed limits, provided via ETJ2-API
5. Train status information, provided via KUPLA-API
6. General traffic speed, status and position information provided by PALA

Each of these interfaces is described separately.
In addition to these interfaces it is assumed the DAS system will also retrieve engine characteristics and other such information from some other source outside the scope of this document.
Suggested Flow of Operation
This is a suggested outline for the DAS server-side component for querying the various APIs:

1. Query INFRA-API to get the railway network topology for the train departure date
    - poll periodically (e.g. every 10 minutes) revisions-endpoint to see when the data is potentially changed
    - dump previous data and get new data, or
    - conditionally (with ETag) query for new data and parse when changed
2. Query TIMETABLE-API to get the timetable for a train
    - poll periodically for changes
3. Query ETJ2-API to get the temporary speed limits for e.g. the whole travel dates of a train
    - poll periodically (e.g. every 10 minutes) revisions-endpoint to see when the data is potentially changed
    - dump previous data and get new data, or
    - conditionally (with ETag) query for new data and parse when changed
4. Use INFRA data to
    - determine the route for the train (that is, [Ratakmvali])
    - determine the elevation profile for the route
    - to map permanent speed limits to the route
5. Use ETJ2 data to map temporary speed limits to the route
6. Use PALA-API to get the latest locations of all the running trains (if necessary)
7. Get new data as often as needed, e.g. every 30 seconds
8. Query FORECAST-API to get the current forecasts for a train

3 ETJ2-API

Provides temporary speed restriction data via a REST-like XML interface. At least JSON and GeoJSON also provided, possibly GML at some point. WSDL2 descriptor may be provided later.

Versioned, that is, *v1* in the following denotes the selected (by the API client) version to use.All responses redirect to another location, whose response can be cached eternally. The data in the system changes only in large batches. System change can be observed by polling the following URL:https://<address>/v1/revisions.xmlData transfer may be reduced by issuing a request with a If-None-Match -header containing the previous ETag value. A HTTP 304 response indicates that the data has not changed. This applies to the *revisions* URL as well as all other URLs whose response contains an ETag header.

3.1 Interface descriptions

- https://<address>/v1/
- https://<address>/v1.wadl

3.2 XML Schema definition for the response

- https://<address>/v1/ennakkoilmoitukset.xsd

3.3 API methods

- https://<address>/v1/ennakkoilmoitukset.xml?typeNames=nopeusrajoitus&time=<interval>

where *interval* is the relevant interval (e.g. a month, or the travel time of a train).

3.4 Examples

- https://<address>/v1/ennakkoilmoitukset.xml?typeNames=nopeusrajoitus&time=2015-11-01T00:00:00Z/2015-11-30T23:59:59Z

3.5 Response

Contains at least the following information

- tunniste: string *(identifier of restriction)*
- voimassa: yyyy-MM-ddTHH:mm:ssZ/yyyy-MM-ddTHH:mm:ssZ *(validity: start and end)*
- nopeusrajoitus: decimal *(in km/h) (speed limit)*
- vaikutusalue: [{ sijaintivali, [raiteet.tunniste] }] *(area of effect: list of track segments, see below)*

where each *sijaintivali* contains:

- ratakmvali: (ratanumero) 0+0000 -> 0+0000 *(pair (start/end) of railway addresses in kilometer marker format)*
- pmvali_nouseva: 0+0000 -> 0+0000 *(pair of railway addresses in positioning marker format, ascending)*
- pmvali_laskeva: 0+0000 -> 0+0000 *(pair of railway addresses in positioning marker format, descending)*

Note that while *ratakmvali* is unique within the whole railway network, *pmvali*-locations are not.

4 Infra-API

Provides railway infrastructure data via a REST-like XML interface. At least JSON and GeoJSON also provided, possibly GML at some point.

WSDL2 descriptor may be provided later.

Versioned, that is, *v1* in the following denotes the selected (by the API client) version to use.All responses redirect to another location, whose response can be cached eternally. The data in the system changes only in large batches. System change can be observed by polling the following URL:https://<address>/v1/revisions.xmlData transfer may be reduced by issuing a request with a If-None-Match -header containing the previous ETag value. A HTTP 304 response indicates that the data has not changed. This applies to the *revisions* URL as well as all other URLs whose response contains an ETag header.

4.1 Interface descriptions

- https://<address>/v1/
- https://<address>/v1.wadl

4.2 XML Schema definitions for the responses

- https://<address>/v1/liikennepaikat.xsd
- https://<address>/v1/linjavaihteet.xsd
- https://<address>/v1/radat.xsd
- https://<address>/v1/raiteet.xsd
- https://<address>/v1/nopeusrajoitusalueet.xsd

4.3 API methods

Currently querying for intervals is not supported and the instant is given as a point-interval (UTC-midnight).

- https://<address>/v1/liikennepaikat.xml?time=<interval>
- https://<address>/v1/linjavaihteet.xml?time=<interval>
- https://<address>/v1/radat.xml?time=<interval>
- https://<address>/v1/raiteet.xml?time=<interval>
- https://<address>/v1/nopeusrajoitusalueet.xml?time=<interval>

4.4 Examples

- https://<address>/v1/liikennepaikat.xml?time=2015-11-02T00:00:00Z/2015-11-02T00:00:00Z

4.5 Responses

Contain at least the following information:

liikennepaikat: (*stations*)

- tunniste: string (*id*)
- nimi: string (*name)*
- sijainti (*position, see below*)

linjavaihteet: (*line switches*)

- tunniste: string *(id)*
- nimi: string *(name)*
- sijainti *(position, see below)*

radat: (*lines*)

- ratanumero: String *(line number)*
- kilometrimerkit: [{ratakm, pituus}] *(integer, decimal) (list of kilometer markers, for each: {identifier, length})*
- geometria: 4D-Multiline *(x, y, z, sijainti) (ETRS-TM35FIN, N2000) (track geometry: a list of vertices, for each vertex x, y, z coordinates and position (see below))*

raiteet: (*tracks*)

- tunniste: string *(track id (synthetic))*

- tunnus: string *(optional)* (track commercial id)
- linjaraidetunnus: string *(optional) (mainline track descriptor, one of a limited range, e.g. P for "northern", E for "southern", etc. A full list can be made available)*

nopeusrajoitusalueet: (*speed limits*)

- vaikutusalue: [{ sijaintivali, [raiteet.tunniste] }] *(speed limit bounds: a list of: a pair of positions (see below), a list of track identifiers)*
- nopeusrajoitus: [{ tonnage, decimal }] *(limitation; in km/h per tonnage limit)*

where *sijainti* contains:

- ratakm: (ratanumero) 0+0000 *(railway address in kilometer marker format)*
- pm_nouseva: 0+0000 *(railway address in positioning marker format, ascending)*
- pm_laskeva: 0+0000 *(railway address in positioning marker format, descending)*

and each *sijaintivali* contains:

- ratakmvali: (ratanumero) 0+0000 -> 0+0000 *(a pair of railway addresses in kilometer marker format)*
- pmvali_nouseva: 0+0000 -> 0+0000 *(a pair of railway addresses in positioning marker format, ascending)*
- pmvali_laskeva: 0+0000 -> 0+0000 *(a pair of railway addresses in positioning marker format, descending)*

Note that while ratakm-locations are unique within the whole railway network, pm-locations are not.

5 Kupla-API (server)

Provides train position, speed and status information via a REST-like XML interface. At least JSON and GeoJSON also provided, possibly GML at some point.

WSDL2 descriptor may be provided later.

Versioned, that is, *v1* in the following denotes the selected (by the API client) version to use.

5.1 Interface descriptions

- https://<address>/v1/
- https://<address>/v1.wadl

5.2 XML Schema definition for the response

- https://<address>/v1/junat.xsd

5.3 API methods

- https://<address>/v1/junat.xml?time=<interval>

where *interval* is a time filter for train status information.

5.4 Examples

- https://<address>/v1/junat.xml?time=2015-11-02T03:00:00Z/2015-11-02T03:30:00Z

5.5 Response

Contains at least the following information

- junanumero: string *(train number)*
- lahtopaiva: yyyy-MM-dd *(departure date)*
- operaattori: string *(train operator)*
- aikaleima: yyyy-MM-ddTHH:mm:ssZ *(timestamp of positioning data)*
- ajotila: ajo_aloitettu | ajossa | ei_ajossa (*mode: drive mode starting | drive mode active | drive mode inactive*)
- sijainti: [x,y] *(optional) (ETRS-TM35FIN) (coordinate position)*
- tarkkuus: decimal *(optional) (in metres) (positioning accuracy)*
- nopeus: decimal *(optional) (in km/h) (current speed)*

6 Pala

Provides position and speed information for all traffic via a REST-like XML interface. At least JSON and GeoJSON also provided, possibly GML at some point. WSDL2 descriptor may be provided later.

Versioned, that is, *v1* in the following denotes the selected (by the API client) version to use.Currently Pala is only acting as a proxy to Kupla-API, but in the future it will calculate more accurate location data, and can also provide some form of a push-based service.

6.1 Interface descriptions

- https://<address>/v1/
- https://<address>/v1.wadl

6.2 XML Schema definition for the response

- https://<address>/v1/yksikot.xsd

6.3 API methods

- https://<address>/v1/yksikot.xml

6.4 Examples

- https://<address>/v1/yksikot.xml

6.5 Response

Contains at least the following information

- junanumero: string (*train number)*
- lahtopaiva: yyyy-MM-dd (*departure date)*
- operaattori: string *(train operator)*
- aikaleima: yyyy-MM-ddTHH:mm:ssZ *(timestamp of response)*
- nopeus: decimal *(optional) (in km/h) (current speed)*
- sijainti *(optional) (position, see below)*

where *sijainti* contains:

- koordinaatti: [x,y] *(ETRS-TM35FIN) (coordinate position)*
- ratakm: (ratanumero) 0+0000 *(railway address in line kilometers)*

- pm_nouseva: 0+0000 *(positioning marker number and distance, in the ascending direction)*
- pm_laskeva: 0+0000 *(positioning marker number and distance, in the descending direction)*

Note that while *ratakm* is unique within the whole railway network, *pm-locations* are not.

7 Timetable-API

Provides timetable data via a REST-like XML interface. WSDL2 descriptor may be provided later.

This interface must be called periodically by the DAS, to receive any manual adjustments performed by traffic control. It is intended to perform well even if called often.

Versioned, that is, *v1* in the following denotes the selected (by the API client) version to use.

7.1 Interface descriptions

- https://<address>/v1/
- https://<address>/v1.wadl

7.2 XML Schema definition for the response

- https://<address>/v1/trains.xsd
- https://<address>/v1/timetable.xsd
- https://<address>/v1/composition.xsd

7.3 API methods

Regarding departureDate parameters: API Allows any date between +10 or -2 date from current date (EET) in format yyyy-mm-dd

- https://<address>/v1/trains?departureDate=<departure_date>
    - departure_date: Date of trains first departure.
    - Example: https://<address>/v1/trains?departureDate=2015-11-31

  - Returns: list of train numbers scheduled to departure from first station on given date. (http://rata.digitraffic.fi/api/v1/doc/index.html#Junavastaus can be used as a basis)

- https://<address>/v1/timetable?departureDate=<departure_date>&trainNumber=<train_number>,<train_number2>...
  - departure_date: Date of trains first departure in format yyyy-mm-dd.
  - train_number: list of train numbers, allows 1-n items in the list (e.g. IC60 would be just 60)
  - Example: https://<address>/v1/timetable?departureDate=departureDate=2015-11-31&trainNumber=60
  - Returns Timetable element for each train queried in one response, containing
    - Train number (e.g. 60)
    - Type (e.g. IC)
    - Maximum speed (e.g. 120) as km/h
    - basic train information
    - http://rata.digitraffic.fi/api/v1/doc/index.html#Junavastaus
    - arrival time ("arrive no earlier than")
    - departure time ("leave no later than")
- https://<address>/v1/composition?departureDate=<departure_date>&trainNumber=<train_number>,<train_number2>...
  - departure_date: Date of trains first departure in format yyyy-mm-dd.
  - train_number: list of train numbers, allows 1-n items in the list (e.g. IC60 would be just 60)
  - Example: https://<address>/v1/composition?departureDate=departureDate=2015-11-31&trainNumber=60
  - Returns train composition data for each queried train. Composition is given per route-section
    - http://rata.digitraffic.fi/api/v1/doc/index.html#Kokoonpanovastaus
    - braking capability
    - weight

8 Timetable Forecasts

Interface provided by the DAS server for supplying timetable forecast data. REST-like XML interface. WSDL2 descriptor may be provided later.

Versioned, that is, *v1* in the following denotes the selected (by the API client) version to use.

8.1 Interface descriptions

- https://<address>/v1/
- https://<address>/v1.wadl

8.2 XML Schema definition for the response

- https://<address>/v1/forecasts.xsd

8.3 API methods

- https://<address>/v1/forecasts.xml?trainNumber=<list of train numbers>

8.4 Examples

- https://<address>/v1/forecasts.xml?trainNumber=42,43,44

8.5 Response

Contains at least the following information

- address: string (*railway address of the point*)
- identifier: string (*station identifier if the point represents a station or other such location*)
- economical: timestamp (*time of arrival or bypass (as appropriate) when running as economically as possible, i.e. with the least energy expenditure*)
- fastest: timestamp (*time of arrival or bypass when running as fast as possible, i.e. without regard to energy expenditure, but within safety constraints*)