INT
Graph Model Concepts

2016 // Matthijs Hamers // INT Graph Model Concepts // v1.9

Table of Contents

Inhoud

INT Graph model concepts

The goal of the Intrabel Network Topology project (INT) is to model the Belgian railway infrastructure as a *connected* graph.

The INT graph model:

- uses **nodes** to model physical as well as virtual infrastructure information;
- connects these nodes using directed and undirected types of **edges**
- defines several **groups** and **spatial clusters** of nodes to model various macroscopic traffic management concepts

These concepts are explained in more detail in the following chapters.

1. Nodes

1.1 Nodes, Ports and Edges

In the INT graph model, all elements of the railway network are modelled as nodes. Some nodes represent *physical* infrastructure information such as switches, signals, speedSignals, etc., while others represent *virtual* information such as cluster borders or points that are important for the planning or real time follow-up of trains.

Different nodes can be associated to each other by creating explicit links between them, called Edges (Figure 1).

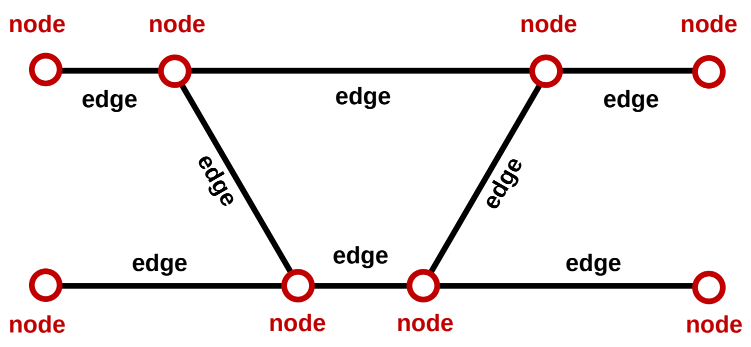


Figure 1

Some nodes are only connected to a single edge, while others may require connections with two, three or four edges – depending on the type of node. For that reason, each type of node has a specific number of connectors, called *ports.*

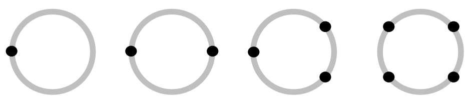


Figure 2

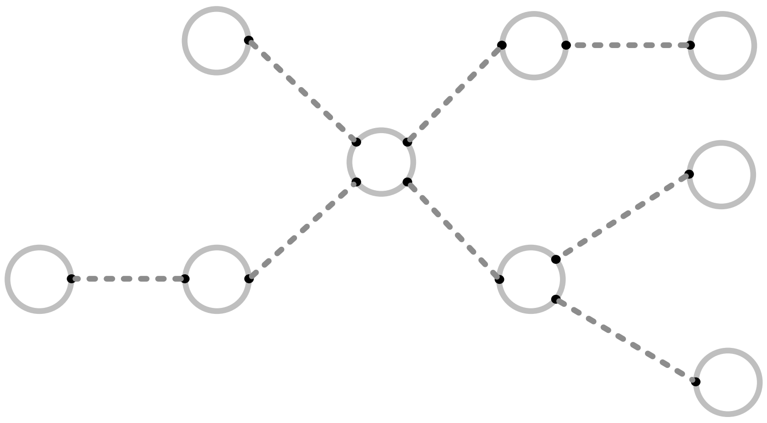


Figure 3

Each such a port can be considered a required connection of that node with an edge, allowing us to easily define some basic rules to check the coherency of the graph.

Also, note that it is possible that a certain node has *two* ports, even though it is linked to only *one* edge. In this case *both* ports are connected to the same edge and their purpose is to give the node a *sense of direction* – an **orientation** - on the edge. These nodes are then called ***inner nodes*** for that type of edge (see the blue node in Figure 4).

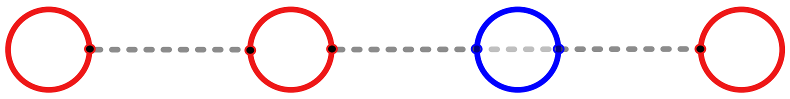


Figure 4

If the ports of a node are connected to different edges, or if the node only has one port, that node is an ***outer node*** for that type of edge (red nodes in Figure 4).

All INT nodes have three common properties:

Id (required)
A unique and stable **INT** identifier. Not to be confused with the optional **iuid** on some specific node types.

validFromDate (required)
validToDate (required)
A node can have multiple versions. Each version has a *valid from* and *valid to* date (closed interval). Multiple versions of a single node can never have overlapping lifecycles.

1.2 switch

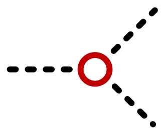


Figure 5

switch nodes correspond to the physical switches in the grids. These nodes are always located part of a grid[1].

A switch has three ports, which allow connection with three edges: one for the point and one for each of the two branches of the switch (Figure 6).

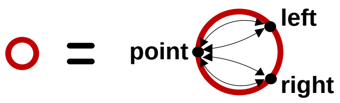


Figure 6

Note: even though it is impossible for a train to pass a switch from its left port to its right port (arrows in Figure 6), this kind of constraints are not defined on the level of nodes in INT, but by defining a layer of directed edges that describe the use of the infrastructure, see routeEdges.

[1] A Grid (also: grill) is a group of related switches that allow trains to change track. The switches inside the grid are always protected by managed stop signals on each track that grants access to the grid, to prevent two trains of using incompatible itineraries at the same time. See §3.2.1 for more details on Grids

A switch has some specific properties:

- **name (required)**

  The name of the switch; usually matches the name of the physical switch engine.

- **type (required):** normal / halfEnglishLeft / halfEnglishRight / english

  Indicates the type of switch that is being modelled by this node

- **iuid (optional)**

  In the future, each INT switch will have a reference to the corresponding INDI element. This property will become required as soon as the roll-out of this data source is finalized.

- **deviatingPort:** left / right / left+right

  Indicates which port of the switch node corresponds to the (physically) deviating branch of the switch.

  The possible values are:
  - **left**
  - **right**
  - **left+right** (in case of a symmetrical switch)

English and ***half English*** switches are modelled as a combination of two switch nodes - one for each physical switch engine - with a zero-length edge between them.

Example 1:
Figure 7 illustrates an English switch:

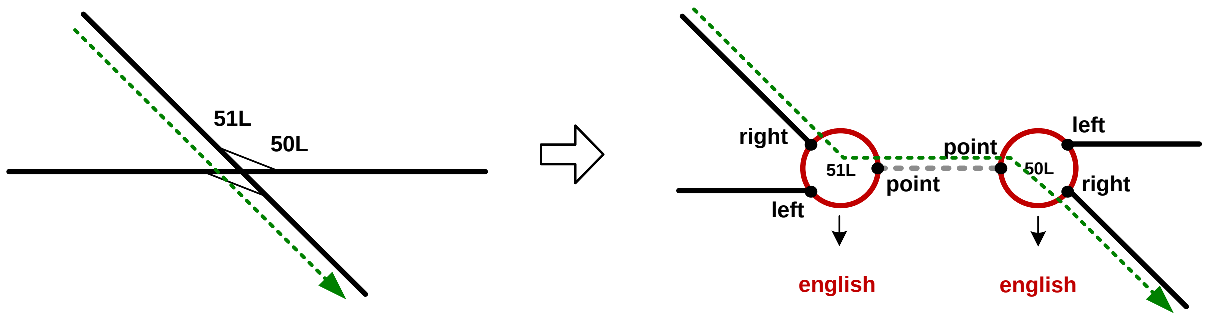


Figure 7

- both nodes 51L and 50L have a type **english** and their *point*-ports are connected to each other

Example 2:
Figure 8 illustrates a **half** English switch:

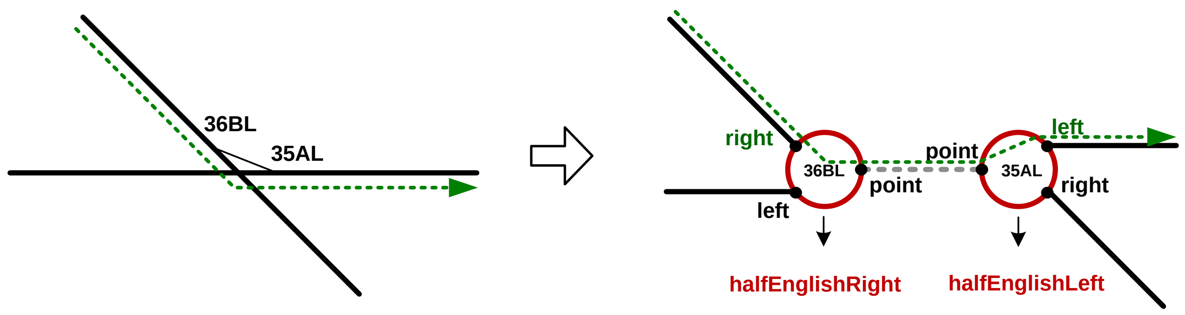


Figure 8

- switch 36BL has type ***halfEnglishRight***, because its ***right*** port is part of the deviating route
- switch 35AL has type ***halfEnglishLeft*** because its ***left*** port is part of the deviating route

Using two nodes allows us to model a shared stop signal (often used in shunting yards) like this:

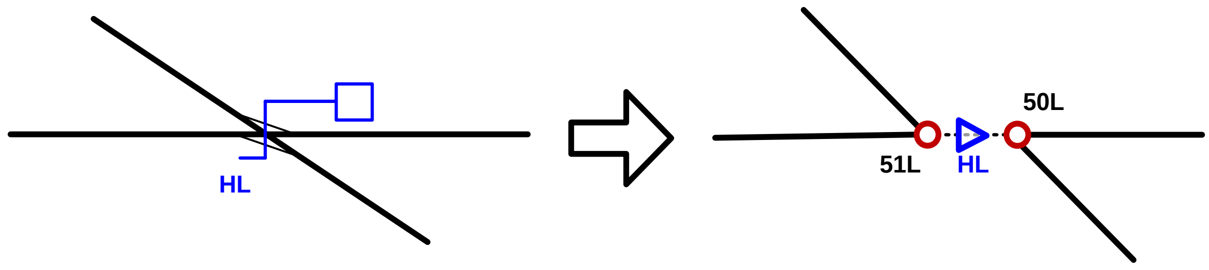


Figure 9

1.3 crossing

crossing nodes correspond to the physical crossings in the grids. They are always part of a grid.

Unlike English switches, crossings are modelled as a single node with four ports:

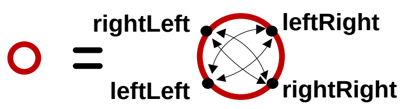


Figure 10

A crossing has two extra properties:

- **name (required)**

  Name of the crossing. In case of a crossing with moveable heart, the name matches the name of the physical engine.

- **iuid (optional)**

  In the future, each INT crossing will have a reference to the corresponding INDI element. This property will become required as soon as the roll-out of this data source is finalized.

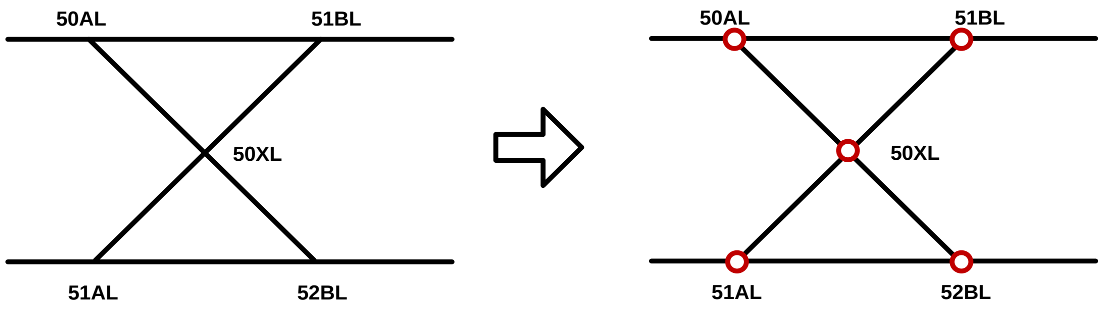


Figure 11

INT does not make a distinction between normal crossings and crossings with moveable hearts; both are modelled as a single node of the crossing.

1.4 deadEnd

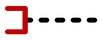


Figure 12

Some tracks in stations or shunting yards stop at a “dead end”. The corresponding deadEnd nodes have only one port and are always part of an interGrid.

A deadEnd node has one specific property:

- **iuid (optional)**
  In the future, each deadEnd node should have a reference to the corresponding INDI element. This property will become required as soon as the roll-out of this data source is finalized.

1.5 endOfModel

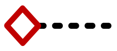


Figure 13

endOfModel nodes mark the end of an Edge in the graph, not because the physical track stops at that point, but because the track exceeds the modelled area of the Graph.

An endOfModel node can be used to model a frontier or an entrance to one or more private (unmodelled) railway sites. In that case, the endOfModel can have an unordered list of references to the exterior locations it gives access to[2].

See §3.2.3.6 for more information on external locations.

Similar to deadEnd nodes, endOfModel nodes have only one port and are always located outside a Grid.

An endOfModel node has two specific properties:

- **subLocations (optional)**[3]
  an unordered list of id’s of all subLocations this endOfModel node gives access to

- **iuid (optional)**
  In the future, each endOfModel node should have a reference to the corresponding INDI element. This property will become required as soon as the roll-out of this data source is finalized.

[2] 14/09/2015: This link between endOfModel and location is not yet implemented
[3] 14/09/2015: not yet implemented

1.6 border

border nodes are a special kind of *virtual* nodes that are used to delimit spatial clusters on a microscopic level.

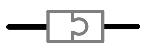


Figure 14 border

Each border node has two ports, ***1*** and ***2,*** to indicate its orientation on the trackEdge:

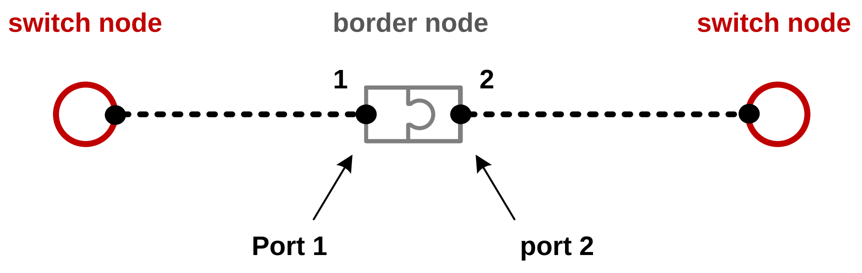


Figure 15

Border nodes represent the exact boundaries of a spatial cluster, allowing us to define each cluster as a sub-graph of all delineated nodes.

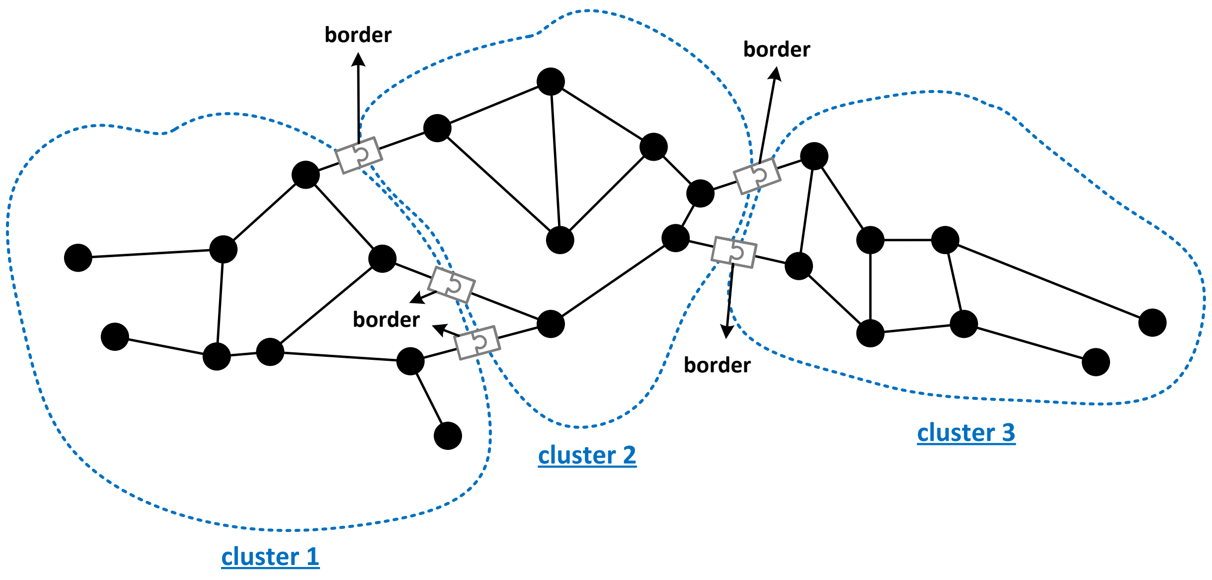


Figure 16

Since a border node is always part of both adjacent clusters, they also allowing us to easily link neighbouring clusters.

A border node has one or more[4] *functions*, depending on the types of Clusters it delineates:

- **gigBorder**
- **serBorder**
- **locationBorder**

Each function is described below. Also, see Chapter 3 for more details on *Clusters*.

1.6.1 gigBorder function

Borders with a *gigBorder* function delimit gig (*grid/intergrid)* clusters. For more information on these clusters, see §3.2.

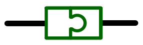


Figure 17 green gigBorder

The *gigBorder* function adds one attribute to the border node:

- **ebpName (optional):**

  Name of the track in the EBP interlocking system

1.6.2 serBorder function

Borders with a serBorder function delimit *SER* clusters. For more information on these clusters, see §3.2.

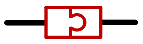


Figure 18 red serBorder

The serBorder function doesn't add any additional attributes to the node.

note: a border with a serBorder function always has a gigBorder function as well, because interGrids are split on the edges of an SER zone. See §3.2.2 for more information on SER zones.

[4] A single Border node can have multiple functions if it represents the boundary of multiple types of clusters

1.6.3 locationBorder function

Borders with a *locationBorder function* are also called *location borders.* They outline *location* clusters, which are important macroscopic INT entities that form the link between the microscopic and macroscopic graph model. The location borders between two locations contain extra macroscopic information about the connection between both clusters. For more information on Locations, see §3.1.

Figure 19 blue locationBorder

A locationBorder function adds four attributes to the border node:

- **lineId (optional)**

  A reference to the macroscopic a371:line that corresponds to the connection between both locations

  Only available if the border is positioned on a trackEdge that is uniquely linked to one line:
  - *if the trackEdge is linked to a a371:trackSection that is linked to a line;*
  - *or if the line can be deduced by combining the associated ptcarId references of the locations with the a371:ptcarByLine information*

- **trackId (optional)**

  A reference to the a371:track of the a371:line

  Only available if the border has a lineId and if it is positioned on a trackEdge that is uniquely linked to a specific line track; or if the associated lineSection only has one track

- **lineSectionId (optional)**

  A reference to the macroscopic a371:lineSection that corresponds to the connection between both locations

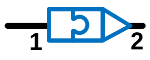
  Only available if the border has a lineId

- **normalDrivingDirection (optional):** up / down / up+down

  This attribute indicates the normal (or preferred) driving direction on the given line and track. There are three possible values:

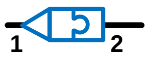
  - up

    the normal driving direction of the given track is equal to the orientation of this border (and its trackEdge): from port ***1*** to port ***2***

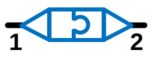
  - down:

    the normal driving direction of the given Track is opposite to the orientation of this border (and its trackEdge): from port ***2*** to port ***1***

  - up+down:

    Both directions are considered as normal driving direction[5].

[5] In this case the Track should be equipped with normal side StopSignals in both directions

1.7 StopSignal

StopSignal nodes represent the different types of railway stop signals.

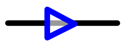


Figure 20

Each signal node has two ports: ***front*** and ***rear***[6]***.*** These ports define the orientation of the signal on the track.

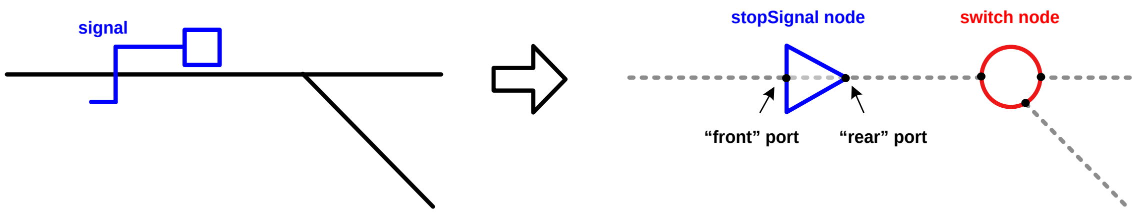


Figure 21

StopSignal nodes have several properties:

- **name**
- **type:** large / simple / small / fictive / beacon[7]

  Indicates the physical type of stop signal.

  <u>*large, simple* and *small* type signals</u>: are the three conventional types of lateral signalling;

  <u>*beacon:*</u> will be used for ETCS, TVM or TBL cab signalling;

  <u>*fictive* signals</u>: are sometimes used to match routing data from different data sources, or to indicate the exact position for realtime "*train position messages"* on places where no physical signal is present.

- **regime (optional)**: normal / wrong

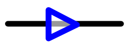
  Specifies whether the signal is to be used for the *normal* or *wrong* driving direction

- **control:** managed / automatic / panel **(optional)**

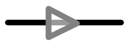
  optional: only present for stopSignal types large, simple and small.

  Managed stop signals are controlled by an operator in a signalling box. These signals are always closed, until a route is set by the operator.

[6] Not yet implemented as such (28/06/2016): currently the ports of managed stopSignals are still called "track" and "grid", where "track" equals "front". The ports of automatic stopSignals are still called "1" and "2", where "1" equals "front"
[7] Not yet implemented (13/08/2014)



Automatic stop signals are open by default and close automatically based on the track occupation.



Panels are stop signals that are always closed and usually mark the end of a shunting track.

- **ptrefId (optional)**

  Some stop signals act as a "***ptref***", which means that they are used as a reference points for *train position messages*. In this case, a reference to the corresponding a371:ptref element is added.

  Used for compatibility with the legacy applications

- **blockSignalId (optional)**

  Stop signals with ***control*** *= automatic* have a reference to the corresponding a371:blockSignal element.

  Used for compatibility with the legacy applications

- **iuid (optional)**

  A reference to the corresponding signal in the INDI datasource;

1.8 speedSignal

speedSignal nodes represent the lateral speed panels along on the railway tracks.

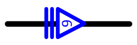


Figure 22

INT only contains speed signals that induce a speed restriction at a specific location, such as *origin panels*, *reference* panels and *end-of-zone* panels of *type bc*; speed *announcement* panels and *end-of-zone panels type ac* are not present since they are of no use for traffic management applications. (see Figure 23)

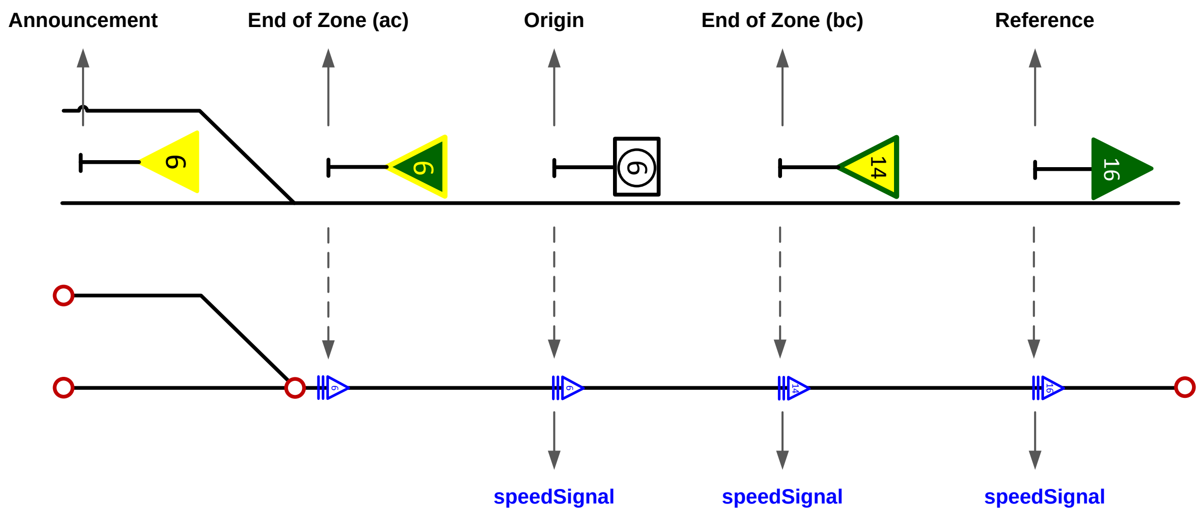


Figure 23

speedSignals are regime independent directed nodes node with two ports: **front** and **rear**.

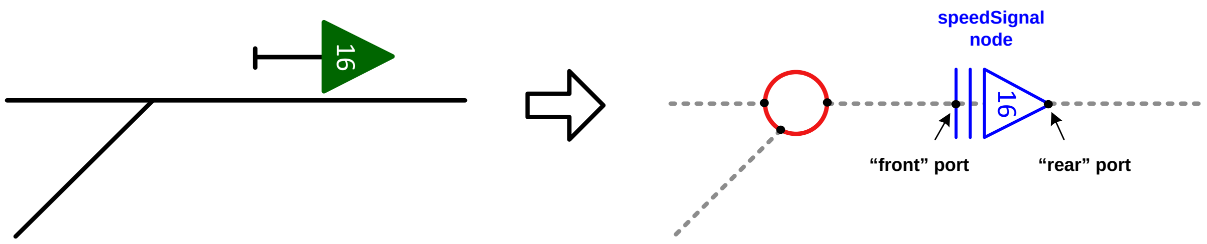


Figure 24

A speedSignal node has the following properties:

- **name**

  (a371) name of the speed signal

- **type:** panel / temporarySpeedLimit / fictive:

  - **panel**

    covers all physical speed signal panels

  - **temporarySpeedLimit**

    Usually a temporary speed limit (tsl) is not an infrastructure asset, but rather a client side operational status on the infrastructure. On rare occasions however, tsl's can be introduced for very long term maintenance or construction of the railway infrastructure.

  - **fictive**

    Fictive speedSignals can be used in rare occasions to insert a virtual speed restriction that is not prescribed by physical signalling, but is nevertheless required according to the regulations.

- **speedRestrictions**

  A speedSignal contains a list of one or more **speedRestriction** items.

  Each *speedRestriction* is a pair of **speedMax + speedProfile** properties:

  - **speedMax** [km/h]:

    The maximum allowed speed for the given profile, starting from this node

  - **profile:** base / freight / highSpeed / liquidMetal

    Indicates for which train profile the given speedMax is valid. The value *base* is used to indicate *all* trains.

Remarks:

If a single infrastructure element contains *multiple* speed panels in the same direction, a single speedSignal with multiple speedRestrictions is created (Figure 25)

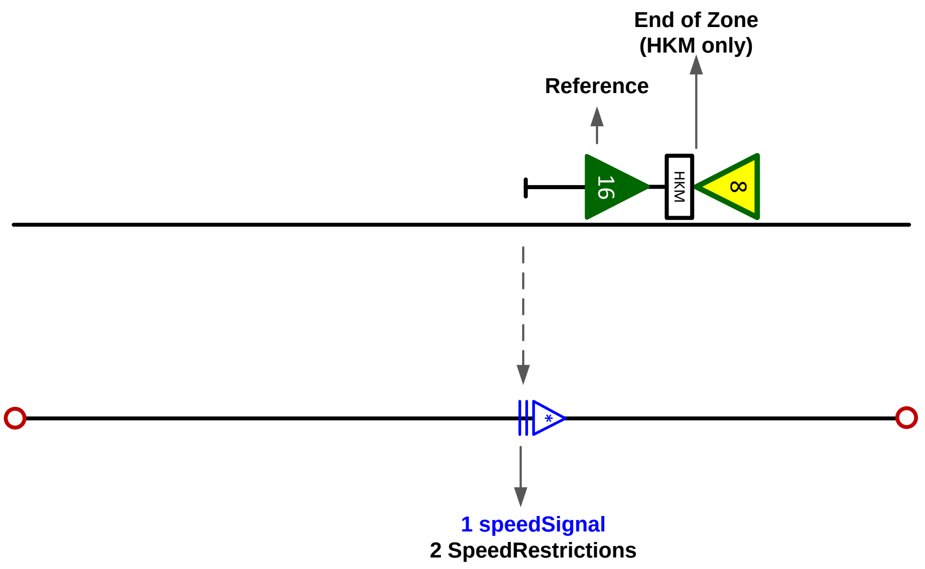


Figure 25

The resulting speedSignal node in Figure 25 will have two speedRestrictions:

- **profile** = base, **speedMax** = 160 km/h
- **profile** = freight, **speedMax** = 80 km/h

However, if a single infrastructure element contains *multiple* speed panels in opposing directions, two speedSignal nodes will be created in INT, each with its own speedRestriction(s).

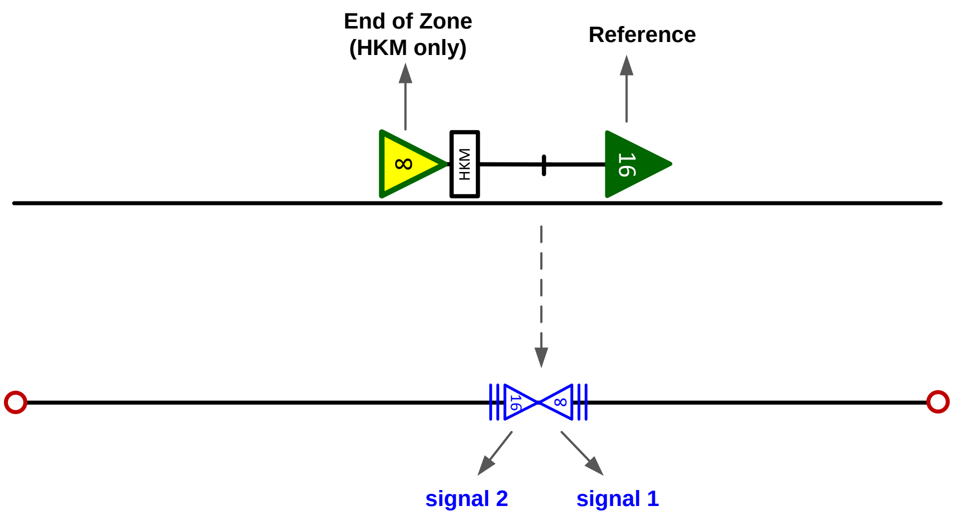


Figure 26

The two corresponding speedSignal nodes in Figure 25 each have their own list of speedRestrictions:

Signal 1:
- **profile** = freight, **speedMax** = 80 km/h

Signal 2:
- **profile** = base, **speedMax** = 160 km/h

1.9 LocationPoint

locationPoints are virtual nodes that represent points on the railway network that are of interest for train planning and traffic management. These nodes allow us to link macro/mesoscopic train timetables to the microscopic infrastructure model and vice versa.

locationPoints are used to create ***location*** clusters and ***subLocation*** groups, see §3.1 for more information on these concepts.

In the course of this document, locationPoints are visualized as green dotted circles, as shown in Figure 27.

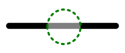


Figure 27

However, if the locationPoint implements one or more functions, this shape will change accordingly: each function adds a visual detail to the base shape:

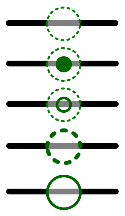


Figure 28

A locationPoint has two extra node attributes:

- **subLocationId**

  A reference to its subLocation. A subLocation is a group of locationPoints belonging to the same a371:ptdes or a371:ptcar.

  subLocation groups are described in more detail in §3.1.1.

- **functions:**

  A **list** of zero or more functions. There are four possible functions:
  - ***timingLocation***
  - ***timingPtcar***
  - ***operation***
  - ***local***

  These functions are described in detail below.

1.9.1 timingLocation function

A locationPoint has a *timingLocation* function if and only if that locationPoint represents the passage through a location cluster.

The usage of a *timingLocation* function will be explained in more detail in chapter §3.1

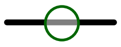


Figure 29

The shape in Figure 29 represents a *timingLocation* point: the solid outer circle indicates its timingLocation function.

A locationPoint with a timingLocation function is also called a timing location point

1.9.2 timingPtcar function

A locationPoint has a **timingPtcar** function if and only if that locationPoint represents the passage through a a371:ptcar[8].

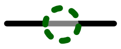


Figure 30

The shape in Figure 35 represents a timingPtcar point: the bold outer circle line indicates its timingPtcar function.

A locationPoint with a timingPtcar function is also called a timing ptcar Point

In practice however, a locationPoint with a *timingPtcar* function should also have a *timingLocation* function. Therefore, the shape in Figure 31 will be the more common representation: the solid, bold outer circle represent both timing functions.

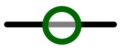


Figure 31

[8] In a371, a ptcar (point caractéristique) is an imaginary intersection of a macroscopic location (e.g. station, bifurcation, shunting yard, frontier, …) with all lines passing at that location.

TimingPtcar points exist mainly for compatibility with the legacy applications:

- *timingPtcar* points correspond to the intersections of a371:ptcar with the microscopic graph model, based on the macroscopic mileage of that ptcar on a line. (See Figure 32)
- many planning and traffic management applications use macroscopic timetables, based on a371:ptcar. *timingPtcar* functions are used when transforming these macroscopic timetables into microscopic paths in the graph model.

For example:

The bifurcation "Y" at the top of Figure 32 is a a371:ptcar. According to macroscopic information of this a371:ptcar , the bifurcation is positioned on km 145 on a line L100 and is the beginning of a new line L200.

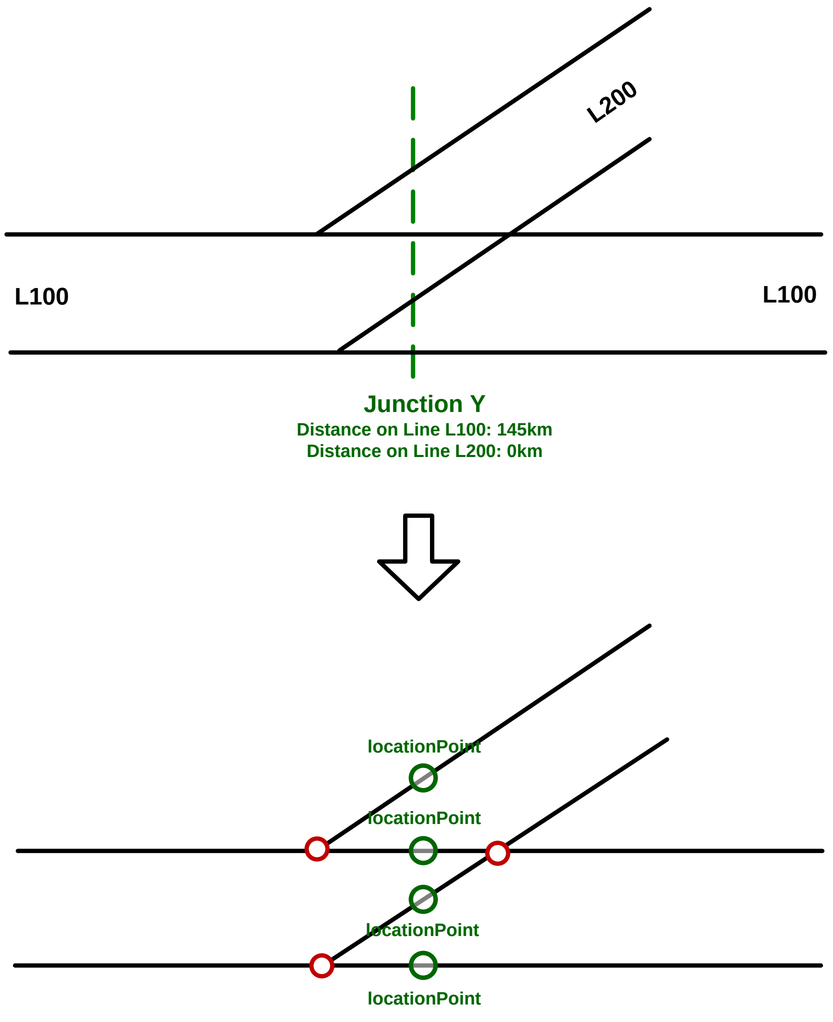


Figure 32

At the bottom of Figure 32 we see the corresponding INT graph with four locationPoints with functions *timingPtcar* and timingLocation: one for each intersection between the microscopic graph model and an imaginary line representing the macroscopic position of the a371:ptcar.

If a train, according to its timetable, has a passage in bifurcation Y on Line L100 at a certain time, we now know exactly where that train will be in the graph model.

1.9.3 operation function

A locationPoint with *an* ***operation*** function represents a specific point on a track where trains can perform some kind of operation. For example: an operation could be a halt, a departure, a service stop or a change of front on a track in a station, halting place or shunting yard, ..

A locationPoint with an operation function is also called an operation point

Operation points can be used by planning or traffic management systems: when creating or modifying a timetable for a train, these locationPoints can be used to specify the required points of operation, such as the origin, the destination or the stops of the train.

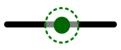


Figure 33

The shape in Figure 33 represents an operation point: the dot in the center indicates its operation function.

A locationPoint can have both a *timingPtcar*, a *timingLocation* and an *operation* function:

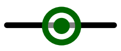


Figure 34

The operation function adds three additional attributes to a locationPoint:

- **trackSectionId**

  A reference to the corresponding a371:trackSection

- **nameDutch**

  The commercial Dutch name of the operation point

- **nameFrench**

  The commercial French name of the operation point

The example graph in Figure 35 has six o*peration points*:

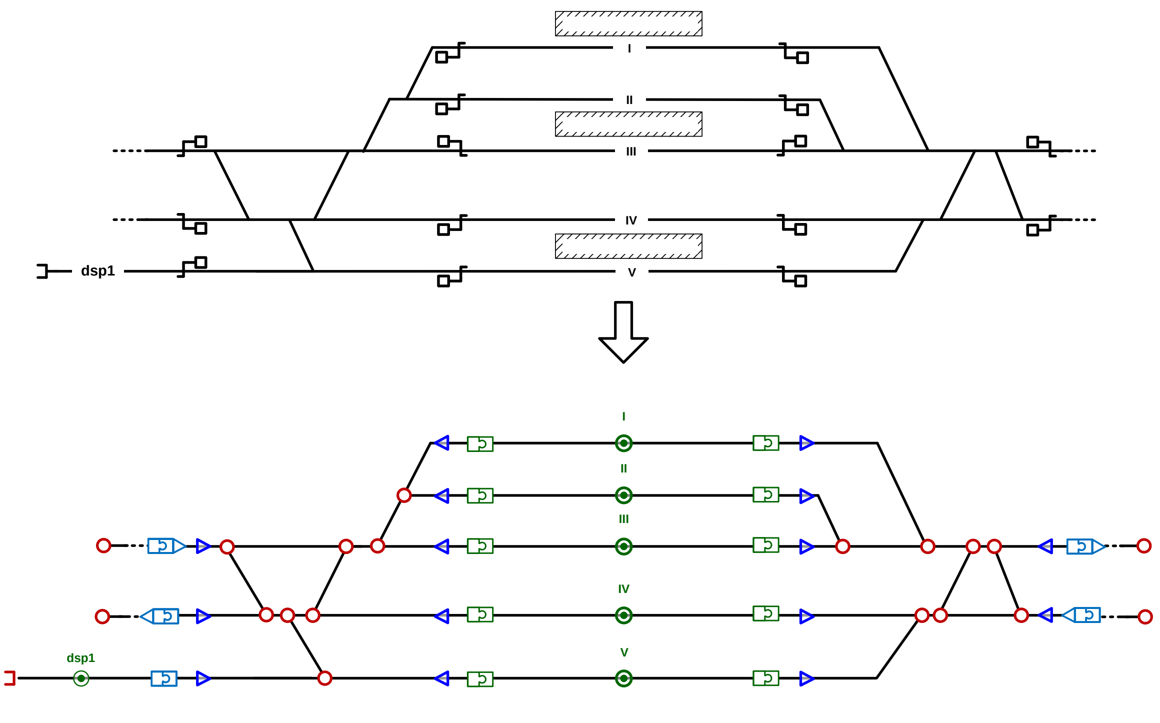


Figure 35

- 5 locationPoints with the names **I, II, III, IV** and **V** represent the different tracks in the station.

  Note: these locationPoints also have a timingLocation and timingPtcar function because they represent the passage through the location cluster and the a371:ptcar.

- 1 locationPoint with name **dsp1** represents a dead end track that can be used for shunting

  Note: this locationPoint also has a timingLocation function, but no timingPtcar function because it does not represent the passage through the a371:ptcar.

1.9.4 local function

This function exist for technical and historical reasons only: the main purpose the *local* function is to tell client systems that although this node is part of a location cluster, and although it has exactly the same local infrastructure information as an *operation point*, this is a point on the network that does not actually allow any train operations in reality.

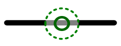


Figure 36

The shape in Figure 36 represents a locationPoint with a *local* function: the unfilled dot in the center indicates the function.

A *local* function adds three[9] attributes to a locationPoint node:

- **trackSectionId**

  A reference to the corresponding a371:trackSection

- **nameDutch**

  The commercial Dutch name of the *local* locationPoint

- **nameFrench**

  The commercial French name of the *local* locationPoint

Note: in practice, a locationPoint with the local function will always have a *timingPtcar* and *timingLocation* function as well, because they are used only in very specific cases, see the example below.

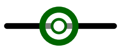


Figure 37

[9] The *local* function has the same three attributes as the *operation* function

For example

In Figure 38 we see a location with a single *siding track,* which can be used to park a train temporarily while another train passes on the main track.

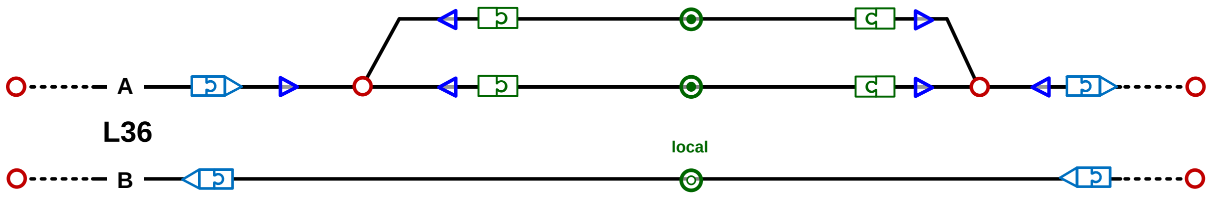


Figure 38

Although there is only one siding, which is positioned on Track A of the Line, this location and its corresponding a371:ptcar are macroscopically positioned on *both* tracks of line L36. As a result, we have *three* locationPoints with functions timingLocation and timingPtcar.

Additionally, we need to add the following functions:

- An *operation* function on the top two locationPoints on (track A); to allow a planning system to choose between the straight track and the siding track
- For the bottom locationPoint (track B), the two existing *timing* functions would normally be sufficient, because there is no operation possible here. However, because there is local a371:trackSection information present (and necessary for some clients), we must add a *local* function to the node. This function may be ignored by most clients.

1.10 routePoints

routePoints are nodes that are specifically used by routeEdges in grids to provide additional routing details, such as information on liberation and sectioning. See §2.3 for more information on routeEdges, or §3.2.1.1.1 for more information on grids.

Properties of a routePoint:

- **name**

  Short name of the routePoint

- **iuid (optional)**

  In the future, each INT routePoint will have a reference to the corresponding INDI element. This property will become required as soon as the roll-out of this data source is finalized.

- **functions**

  A routePoint can have zero[10], one or two functions:

  - **sectioningPoint**
  - **liberationPoint**

  Each function is described in more detail below

1.10.1 sectioningPoint function

A routePoint with a sectioning function corresponds to a virtual infrastructure element that can be used by routeEdges to differentiate multiple alternative *route variants* of the same route.

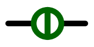


Figure 39 sectioningPoint

[10] INT does not use all routePoint functions that are provided by the a370/INDI data source. Therefore, an int:routePoint can have zero functions

For example:

In Figure 40, there are two possible route variants for the route from stopSignal HL to stopSignal SL:

- HL → SL via sectioningPoint **1L**
- HL → SL via sectioningPoint **2L**

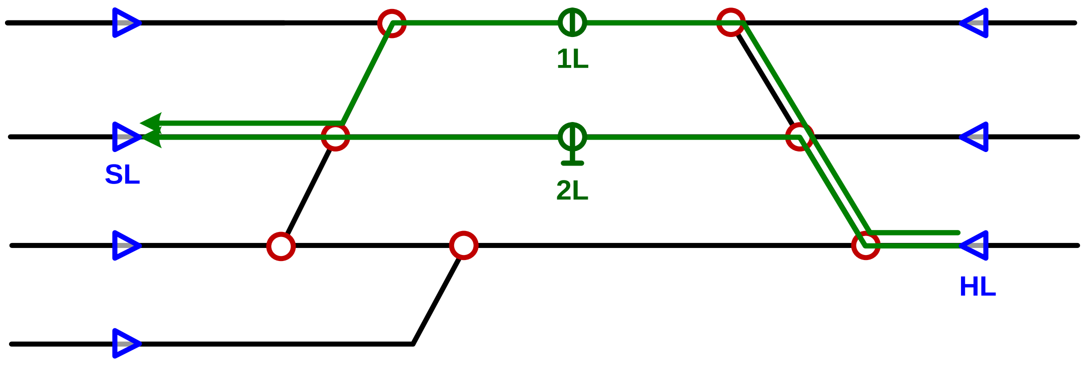


Figure 40

Client applications such as the planning tools can use these sectioningPoints to specify which route variant a train should use; for example to avoid conflicts with other planned trains in a grid or to avoid parts of the grid that are under maintenance.

Realtime/ARS systems can use these sectioningPoints to communicate to the interlocking system which route variant should be set.

1.10.2 liberationPoint function

routePoints with function liberationPoint correspond to physical infrastructure elements for route liberation in a grid, such as axle counters or pedals. Their purpose is to detect when an activated route (or part of a route) is fully traversed by the train, so that the interlocking system can liberate that route (full liberation) or part of that route (partial liberation) as soon as possible.

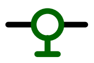


Figure 41 liberationPoint

Liberation points for full liberation are positioned at or near the managed signal at the end of a route, while liberation points for a partial liberation can be placed on inner trackEdges of the grid.

A routePoint can serve as a sectioningPoint and liberationPoint (for partial liberation) at the same time.

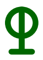


Figure 42 sectioningPoint + liberationPoint

For example:

Once the left train in Figure 43 fully passed liberationPoint **2L** on the green route, the first part is liberated. This way, the first two switches can already be reused to set a route for another train.

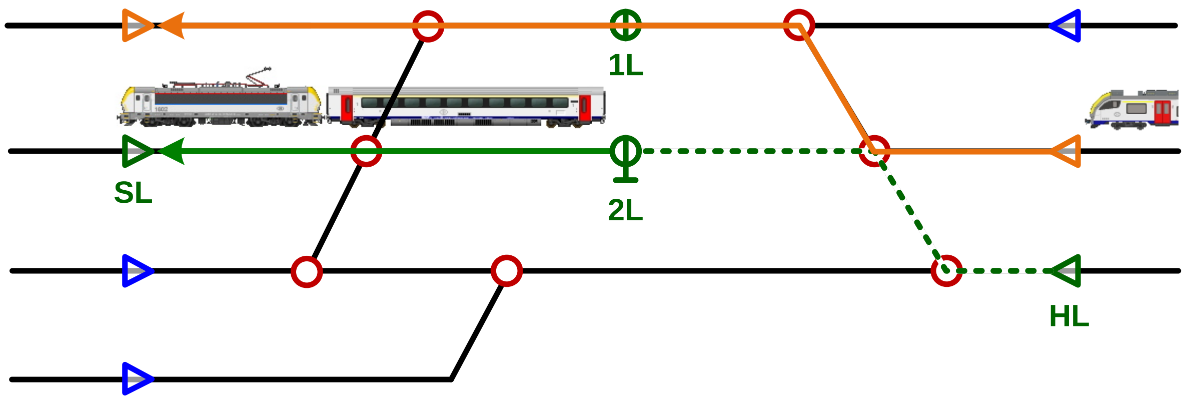


Figure 43

1.11 milestonePoint[11]

milestonePoints represent the microscopic position of a milestone of a line on a trackEdge.

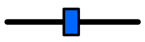


Figure 44

Multiple milestonePoints for the same milestone are bundled in a milestone group, see §3.1.3

Properties of a milestonePoint:

- **milestoneId**

  a reference to the milestoneGroup

1.12 levelCrossingPoint[12]

levelCrossingPoints represent the microscopic position of a level crossing[13] on a trackEdge.

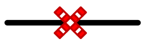


Figure 45

One or more levelCrossingPoints for the same level crossing (on different trackEdges) are bundled in a levelCrossing group.

Properties of a levelCrossingPoint:

- **levelCrossingId**

  a reference to the levelCrossing group

[11] 29/08/2016: milestonePoints are not yet present in INT
[12] 29/08/2016: levelCrossingPoints are not yet present in INT
[13] A level crossing is an intersection between a railway line and a road at the same level, as opposed to bridges or tunnels

1.13 Gradients

The railway network is far from a flat surface; it contains many slopes that could impact the speed, the acceleration and the breaking capacity of trains. In order to allow accurate train runtime calculations, the INT model needs to incorporate the height profile of the railway network.

This is done by adding special nodes that indicate - or repeat - the position and angle of each gradient on the tracks (Figure 47).

Figure 46 - gradient node

A *gradient* node has one additional property:

- **Value [mm/m]**

  Indicates the slope value in millimeters per meter, starting at the node's position in the "up"-direction of its trackEdge (see §2.2)

  A **positive** value corresponds to an **inclination**, a **negative** value to a **declination**

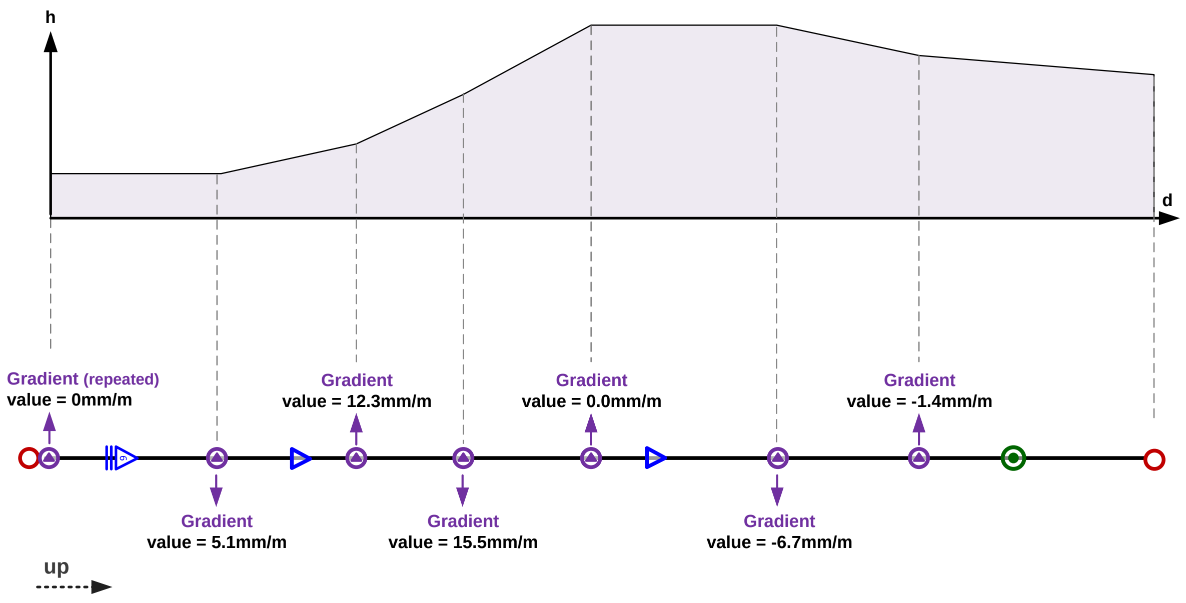


Figure 47

Notes:

- A gradient node is always valid in both directions of the track. However, note that the gradient value is **only** given for the **up**-direction of the trackEdge. When encountering a gradient node in the down-direction on the trackEdge[14], you should thus take the value of the next gradient node in that direction and invert it (multiplied by -1)

- To prevent clients from having to search backwards in the graph model to know the current effective gradient on any given trackEdge, all trackEdges that are uniquely linked to a line repeat the active gradient value at the beginning (0 meter) of that edge.

1.14 Curves

Although all significant curves on the railway network are always protected by speedSignals that reduce the approach speed of trains to a safe limit, curves do cause more friction on the wheels and therefore impact the acceleration and breaking capacity of trains. Curve nodes indicate the exact position and radius of a curve on a track.

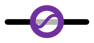


Figure 48 – curve node

A *curve* node has one additional property:

- **value [m]**

  Contains the absolute value of the radius of the curve in meter valid from the node's position in the "up"-direction of its trackEdge (see §2.2)

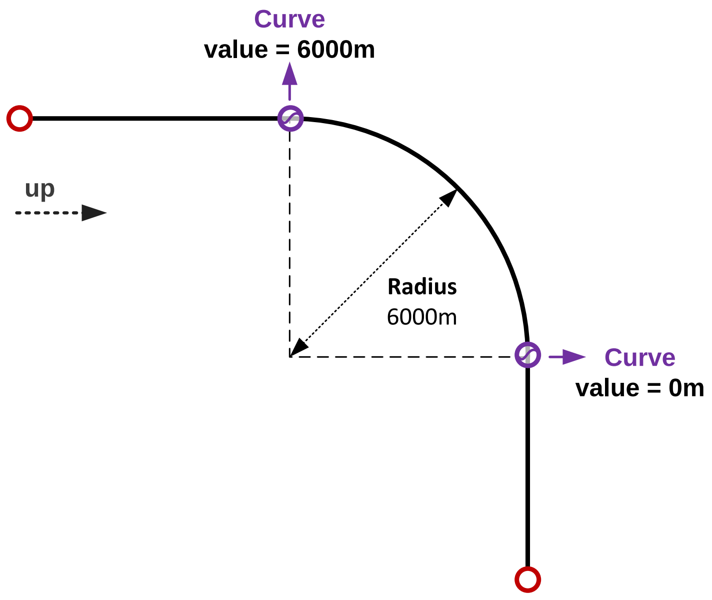


Figure 49

[14] Or at the "start" of a trackEdge in the down-direction

Notes:

- A curve node with a radius value of **zero** means no curve; the track goes straight again

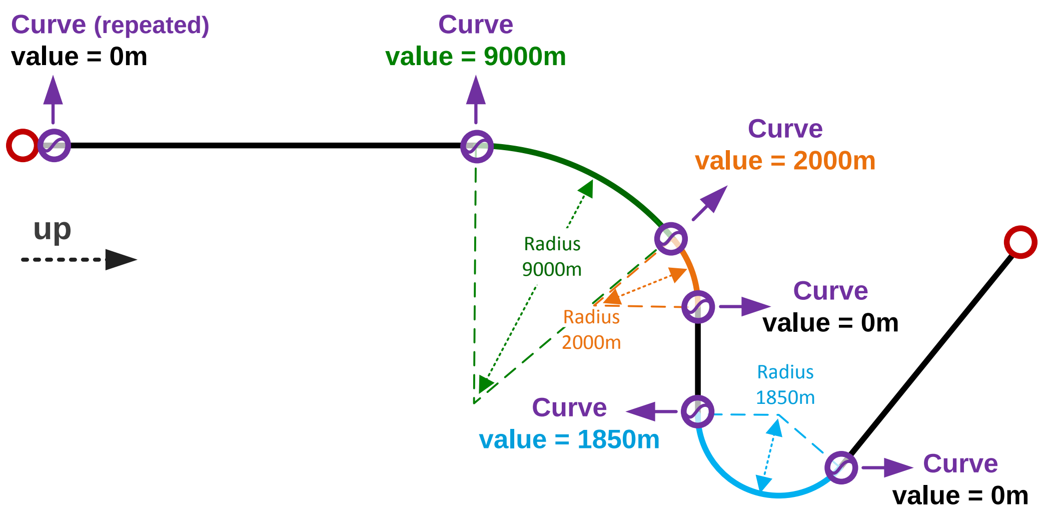


Figure 50

- A curve node is always valid in both directions of the track. However, note that the curve changes are only given for the **up**-direction of the trackEdge. When encountering a curve node in the **down**-direction of a trackEdge[15], you should take the value of the **next** curve in that direction.

- To prevent clients from having to search backwards in the graph model to know the current effective curve radius on any given trackEdge, all trackEdges that are uniquely linked to a line repeat the active curve value at the beginning (0 meter) of that edge.

[15] Or at the "start" of a trackEdge in the down-direction

2. Edges

2.1 General

In Chapter 1 we have seen that the INT model creates graph nodes for all physical infrastructure and virtual elements. Now we still need to specify the *relation* between these nodes. This is done by defining edges.

Each INT edge:

- Has a specific ***type***
- Is either **directed** or ***undirected***
- Has exactly two ***outer nodes***
- Can have optional ***inner nodes***

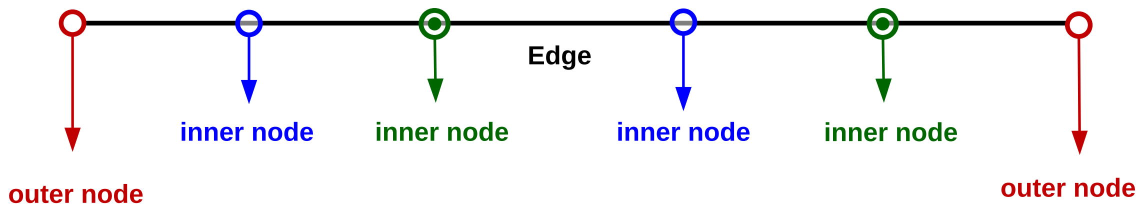


Figure 51

2.1.1 Type and Direction

The **type** of the edge tells us:

- which types of nodes are allowed to be *outer nodes*
- what kind of ***relationship*** is being modelled between the ***outer nodes*** of the edge
- which are the possible *inner nodes* for this specific edge
- whether the edge - and therefore the relationship - is ***directed*** or ***undirected***

INT has two types of edges: *trackEdges and routeEdges*. These will be discussed in detail below.

2.1.2 Outer nodes and inner nodes

An edge is always delimited by exactly two nodes, which are then called ***outer nodes*** for that type of edge. Outer nodes are the *decision points* for the graph built with that type of edges, or in other words: outer nodes connect multiple edges of the same type (see Figure 52)

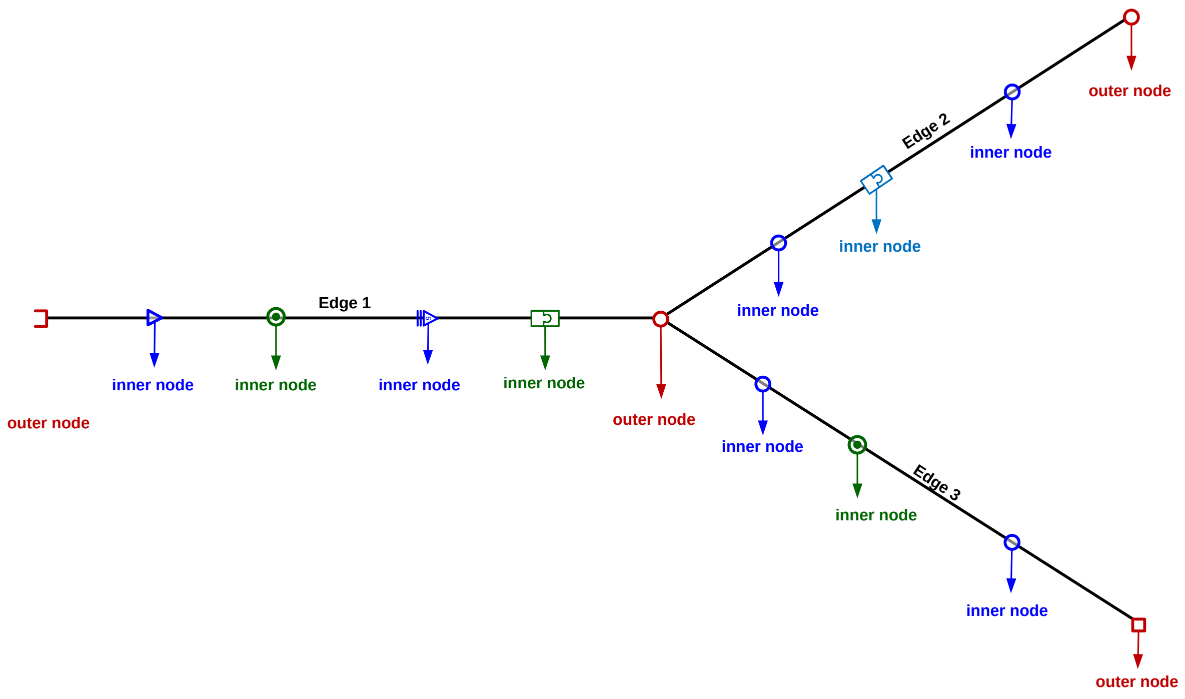


Figure 52

Again depending on its type, an edge can also have zero or more ***inner nodes*****.** These *inner nodes* contain *relevant information* for that type of edge, but they are *not decision points* in the corresponding graph. Therefore, they are placed between the two *outer nodes,* in a specific sequence and at a specific distance along the edge.

Note that because *outer nodes* and *inner nodes* are *edge specific*, any type of node (as seen in chapter §1) could be used as *outer nodes* and or *inner nodes* in various types of edges.
e.g: see Figure 53.

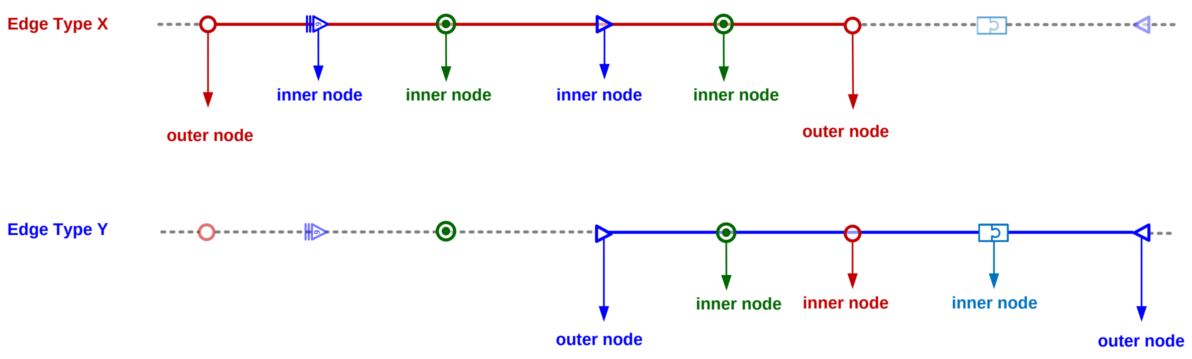

Edge Type X
inner node
inner node
inner node
inner node
outer node
outer node
Edge Type Y
inner node
inner node
inner node
outer node
outer node

Figure 53

2.2 Track Edges

2.2.1 Definition

The most important edges in the INT model are the ***trackEdges***. As the name states, they represent the physical *tracks* of the railway network and are the basis of the microscopic and mesoscopic graph model.

trackEdges are *undirected*[16] edges because they only represent the *existence* of infrastructure and are unaware of possible routing constraints, which are handled by a different set of edges.

2.2.2 Outer nodes of a trackEdge

The **outer nodes** of trackEdges are either:

- *physical* decision or end points, such as **switch** nodes, **crossing** nodes or **deadEnd** nodes:

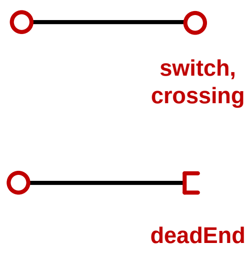


Figure 54

- *virtual end points, such as endOfModel nodes*

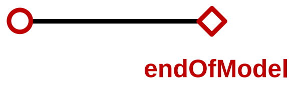


Figure 55

- *(managed) stopSignal nodes, when one or both sides of the trackEdge are part of a mesoscopic*[17] or *hybrid* Grid:

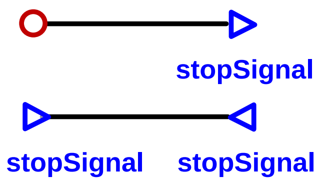


Figure 56

[16] trackEdges are undirected, but do have an orientation, see §2.2.4
[17] See §3.2.1.1.1.1 for more information on mesoscopic grids

Example 1:

Figure 57 shows an example infrastructure schema:

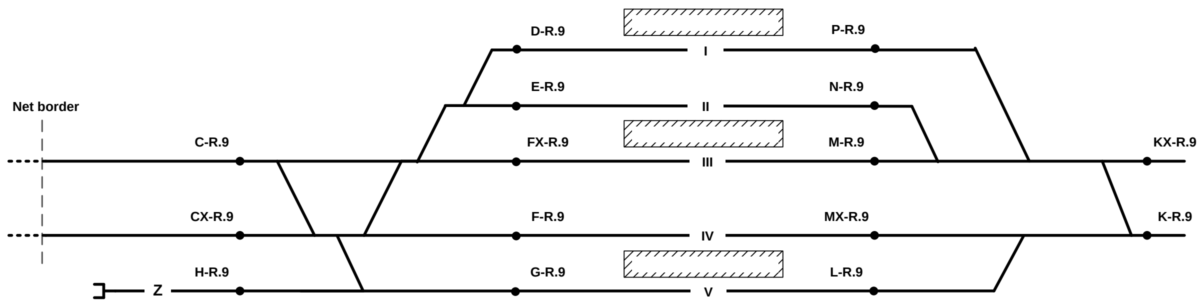


Figure 57

The corresponding *microscopic* trackEdge graph in Figure 58 is made out of **22** trackEdges:

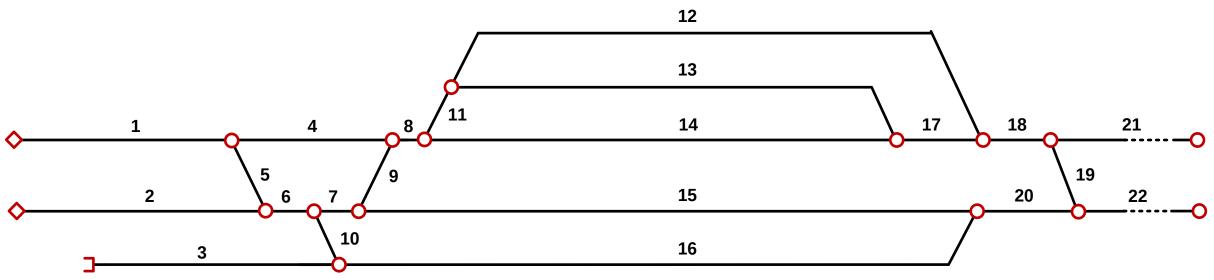


Figure 58

Example 2:

If we would only have *mesoscopic* detail in the left grid of Figure 58, the trackEdge graph would have **25** (mesoscopic) trackEdges[18]:

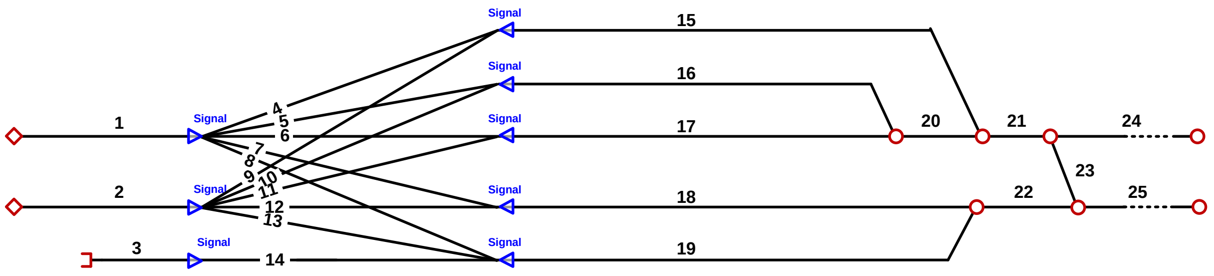


Figure 59

[18] See §3.2.1.1.1.1 for more information on mesoscopic grids

2.2.3 Inner nodes of a trackEdge

Since trackEdges represent the physical railway tracks of our network, they should connect every possible INT node (as seen in chapter 1) to each other. Therefore, with exception of the already mentioned outer nodes, all other physical and virtual INT nodes are **inner nodes** of trackEdges.

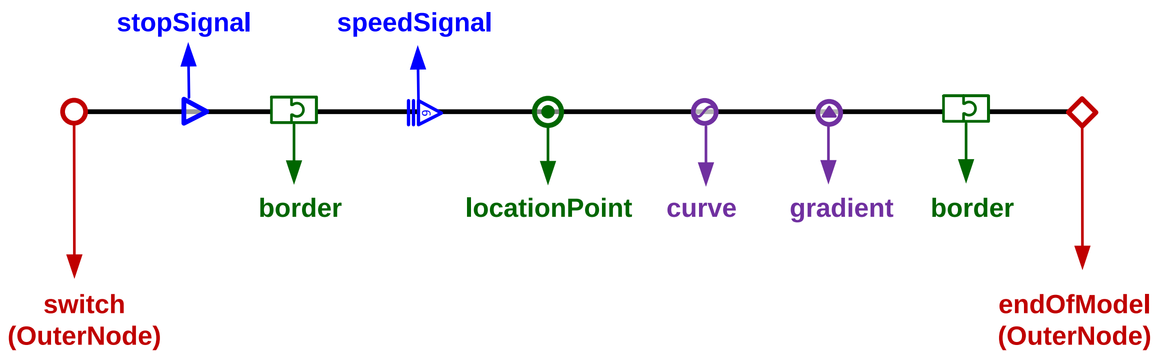


Figure 60

Example: Figure 61 shows the same infrastructure as Figure 57 with inner nodes.

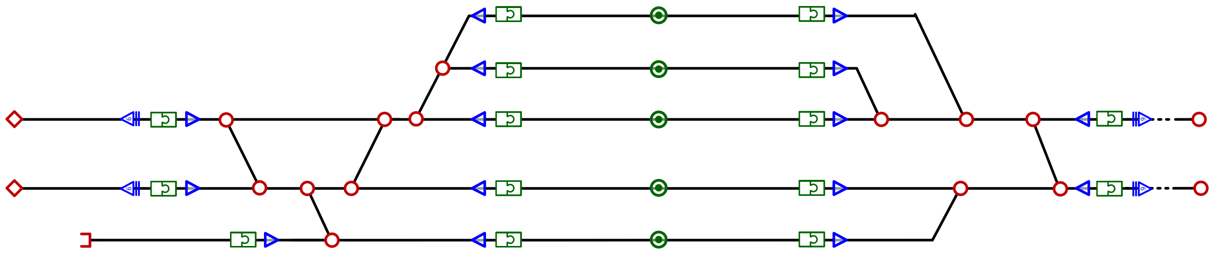
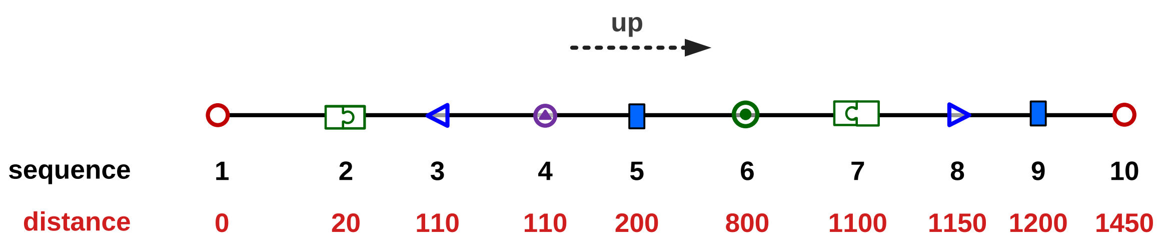


Figure 61

2.2.4 Position of a node on a trackEdge (nodeByTrackEdge)

Even though trackEdges are *undirected* graph edges (meaning that they are valid in both directions), they do have an *orientation:* trackEdges are oriented along the ascending distance of its nodes along the referenced line or location.

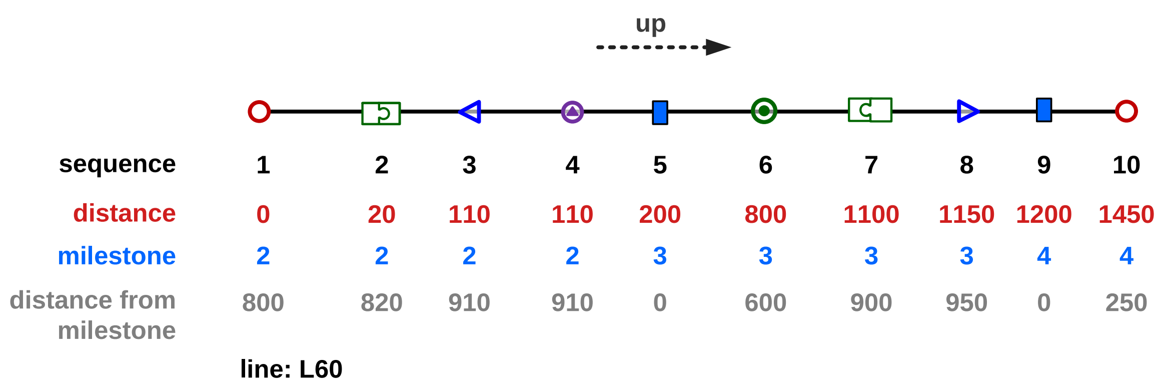
Each node has an explicit sequence number and a relative distance (in meters) on its trackEdge:



sequence	1	2	3	4	5	6	7	8	9	10
distance	0	20	110	110	200	800	1100	1150	1200	1450

Figure 62

Additionally, if the trackEdge is linked to exactly one line, each nodeByTrackEdge will also indicate its referenced milestone and its distance (in meters) from the previous milestone[19] on that line.



sequence	1	2	3	4	5	6	7	8	9	10
distance	0	20	110	110	200	800	1100	1150	1200	1450
milestone	2	2	2	2	3	3	3	3	4	4
distance from milestone	800	820	910	910	0	600	900	950	0	250

line: L60

Figure 63

Notes:

- The order of nodes on the trackEdges should always be derived using the sequenceNumber of the nodes; several nodes may have the exact same relative distance on the trackEdge, but they can never have the same sequenceNumber. See nodes 3 and 4 in Figure 62.
- If a node property, such as the normalDrivingDirection on border nodes, indicate a value 'up', this means: according to the up-orientation of its trackEdge.

[19] Note: it is possible that certain nodes are positioned *before* the first milestone of the line; in that case, they will have a negative distanceFromMilestone from that milestone.

2.2.5 Properties of a trackEdge

A ***trackEdge*** has several properties:

- **id**

  unique ID of the trackEdge

- **validFromDate / validToDate**

  validity period of the trackEdge version

- **fromNodeId / toNodeId**

  an explicit reference to the outer Nodes of this trackEdge

- **distanceLineId (optional)**

  Optional reference to a Line: if the trackEdge is uniquely linked to a single ***Line***, the positions of the nodes on the trackEdge are expressed as a distance from the referenced milestone on this Line (see milestoneId in nodeByTrackEdge).

- **trackId (optional)**

  A reference to the Line Track; only present if the ***distanceLineId*** property is available and if the trackEdge is uniquely linked to a single track of that line.

- **nodesBytrackEdge**

  An ordered list of references to the nodes on this trackEdge, in ascending order of distance on the distanceLine or from the distanceLocation.

  Each **nodeByTrackEdge** has following properties:

  - **nodeId**
    A reference to the Node

  - **trackEdgeId**
    A reference to the trackEdge

  - **validFromDate/validToDate**
    validity period of the nodeByTrackEdge version

  - **inPort / outPort**
    to make the orientation of the node on the trackEdge explicit, the order in which the node is visited by the trackEdge is given; the trackEdge first enters the ***inPort***, then the ***outPort***

- **distance**
  the relative distance of the node in [meters] from the start of the trackEdge

- **distanceMilestoneId / distanceFromMilestone (optional)**
  if the trackEdge is uniquely linked to a single Line, the positions of all nodes on the trackEdge are expressed as a distance in [meters] (**distanceFromMilestone**) from the reference milestone (**distanceMilestoneId**) on this Line (see **distanceLineId**).

  distanceFromMilestone is usually a positive value, but will contain *negative* values for all elements which are positioned *before* the first milestone of a line.

- **sequenceNumber**
  the sequence number of the node on the trackEdge ; to make the sequence of nodes on the trackEdge explicit, even when multiple nodes have the same ***distance***.

2.3 routeEdges

2.3.1 Definition

As explained in §2.2, the purpose of ***trackEdges*** is to model our microscopic and mesoscopic infrastructure graph by connecting all physical and virtual nodes. Although the trackEdge graph would allow us to calculate each *possible* path between nodes, it does not tell us how trains are *allowed to* ride.

A train path through a grill or on a track is only allowed when the signalling equipment allows us to use a certain route. Likewise, the allowed speed on such a route is defined by signalling equipment, such as speedSignals and speed indications on stopSignals; not by physical specifications of switches and tracks themselves. To model this, we need another type of edges: ***routeEdges***.

routeEdges are ***directed** edges* that connect consecutive managed **stopSignals**. They are positioned *on top of* the trackEdge graph. Each routeEdge corresponds to an *allowed* routing variant inside a grid, or an allowed route in open track.

2.3.2 Outer nodes

In general, the outer nodes of a routeEdge are consecutive *managed stopSignals*[20] because these are the *decision points* for the routing graph:

- routeEdges inside a grid will have facing stopSignal nodes (Figure 64),

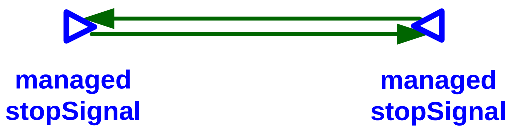


Figure 64

- routeEdges between two grids will have opposite stopSignals as outer nodes (Figure 65).

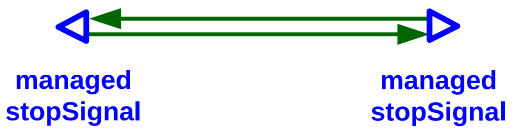


Figure 65

- Other possible outer nodes are physical end points such as a deadEnd;

[20] 2013/09/25: automatic stopSignals with a *PTREF function* are also outer nodes for routeEdges, because they act like managed stopSignals in fictive grids.

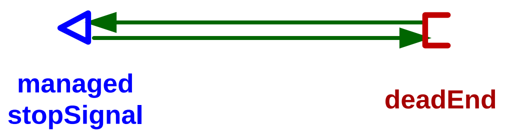


Figure 66

- or virtual end points, such as endOfModel nodes;

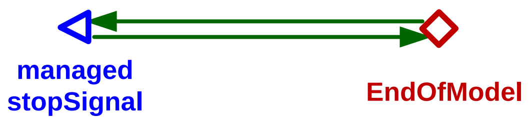


Figure 67

2.3.3 Inner nodes

Unlike trackEdges, routeEdges *do not* contain all INT nodes as inner nodes; only nodes that are relevant for train *navigation* in the *direction* of the given routeEdge, are added:

- switches
- crossings
- stopSignals (control = automatic), <u>valid in the direction of the routeEdge</u>
- routePoints
- speedSignals, <u>valid in the direction of the routeEdge</u>
- locationPoints
- border nodes with function locationBorder

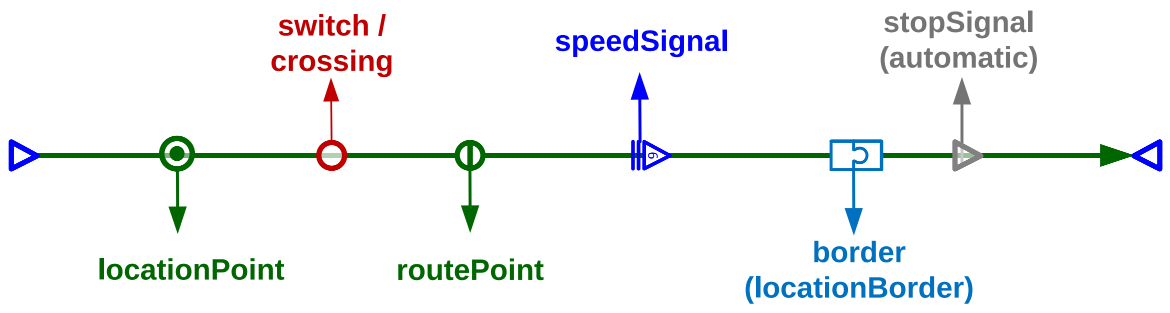


Figure 68

For example:

Figure 69 shows *a trackEdge graph* at the top and the corresponding *routeEdge graph* at the bottom.

The trackEdge graph contains 14 undirected edges. It has microscopic detail in the left grid (trackEdges 3,4,5,6), but only mesoscopic detail in the right grid (trackEdges 9,10,11,12).

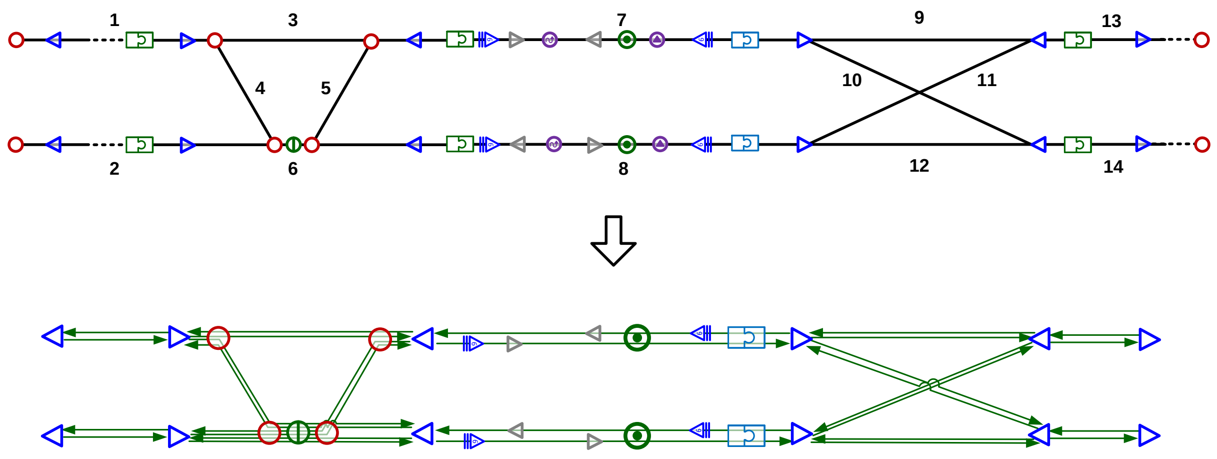
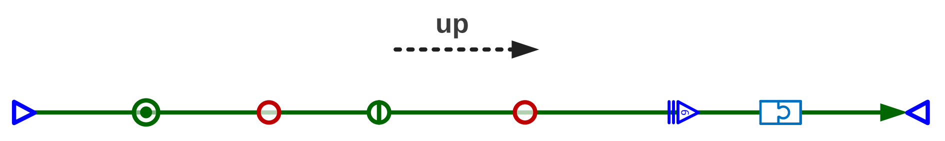


Figure 69

The routeEdge graph at the bottom of Figure 69 contains 28 directed routeEdges. The routeEdges in the right grid are also mesoscopic: they do not contain any switch or crossing nodes.

2.3.4 Nodes on a routeEdge (nodeByRouteEdge)

Each node(ByRouteEdge) has an explicit sequence number and a *relative* distance (in meters) on its routeEdge:



sequence	1	2	3	4	5	6	7	8
distance	0	25	50	110	120	400	450	450

Figure 70

Multiple nodes can have the exact same relative distance on the routeEdge, but never the same sequence number (see nodes 7 and 8 in Figure 70).

As illustrated in Figure 69, not all the nodes of the trackEdge graph are present as inner nodes in the routeEdge graph: some are irrelevant for navigation purposes or are not valid in the direction of the routeEdge.

There is however a tight link between a routeEdge and the underlying trackEdge(s) it makes use of: every node on a routeEdge tells us *for each of its ports* to which underlying trackEdge it is connected (Figure 71):

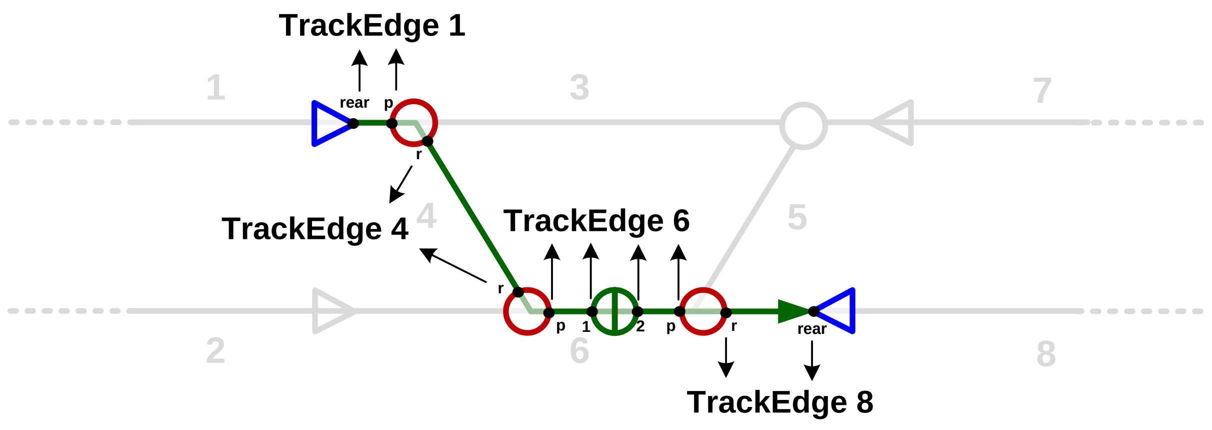


Figure 71

This way, a client system can easily retrieve more detailed infrastructure information (such as gradients and curves) on the underlying *trackEdges.*

2.3.5 Properties of a routeEdge

A ***routeEdge*** has several properties:

- **id**

  unique ID of the routeEdge

- **validFromDate / validToDate**

  validity period of the routeEdge version

- **fromNodeId / toNodeId**

  an explicit reference to the outer nodes of this routeEdge

- **routingId (optional)**

  An reference to the a371:route

  Only present if the routeEdge represents a microscopic route variant or a mesoscopic route inside a grid.

- **trackSectionId (optional)**

  A reference to the a371:trackSection;

  Only present if the routeEdge represents a route traversing an interGrid.

- **speedMax (optional)** [km/h]

  The maximum allowed speed on this routeEdge.

  Only present if the routeEdge represents a microscopic route variant or a mesoscopic route inside a Grid.

- **nodesByRouteEdge**

  An ordered list of references to the nodes on the routeEdge.

  Each **nodeByRouteEdge** has following properties:

  - **nodeId**
    A reference to the node

  - **routeEdgeId**
    A reference to the routeEdge

- **validFromDate/validToDate**
  validity period of the nodeByRouteEdge version

- **inPort / inTrackEdgeId / outPort / outTrackEdgeId**

  To make the link between the nodes on the routeEdge and the trackEdges explicit, each nodeByRouteEdge tells us exacly which port of the node is connected to which underlying trackEdge.

  The routeEdge always enters the ***inPort*** first, then the ***outPort.***

- **distance**
  the distance of the node in [meters] from the beginning of the RouteEdge

- **sequenceNumber**
  the sequence number of the node on the routeEdge ; to make the sequence of nodes on the routeEdge explicit, even when multiple nodes have the same ***distance***.

3. Groups and Clusters

The basic infrastructure model is built using various types of nodes which are then connected to each other using Edges. However, some higher-level infrastructure elements and logical concepts cannot be modelled using just nodes and edges: they require other types of relations, called **Groups** and **Clusters**.

A **Group** is an unordered collection of elements that belong together and represent a certain group entity.

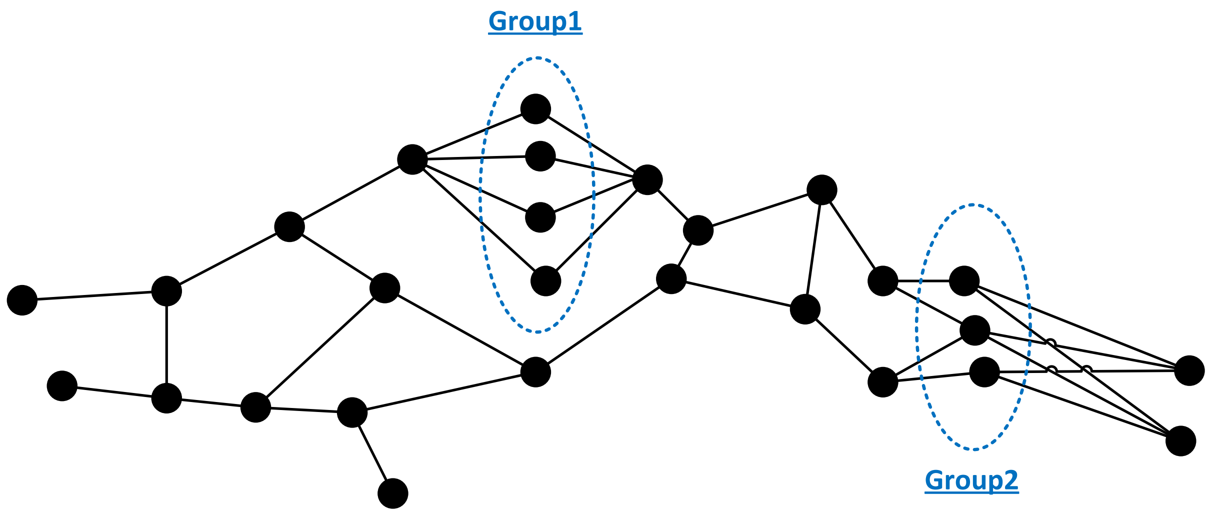


Figure 72

A **Cluster** is a spatial group of connected nodes, delimited by ***border*** nodes, resulting in a connected sub-graph of the full trackEdge graph. See §1.6 for more information on border nodes.

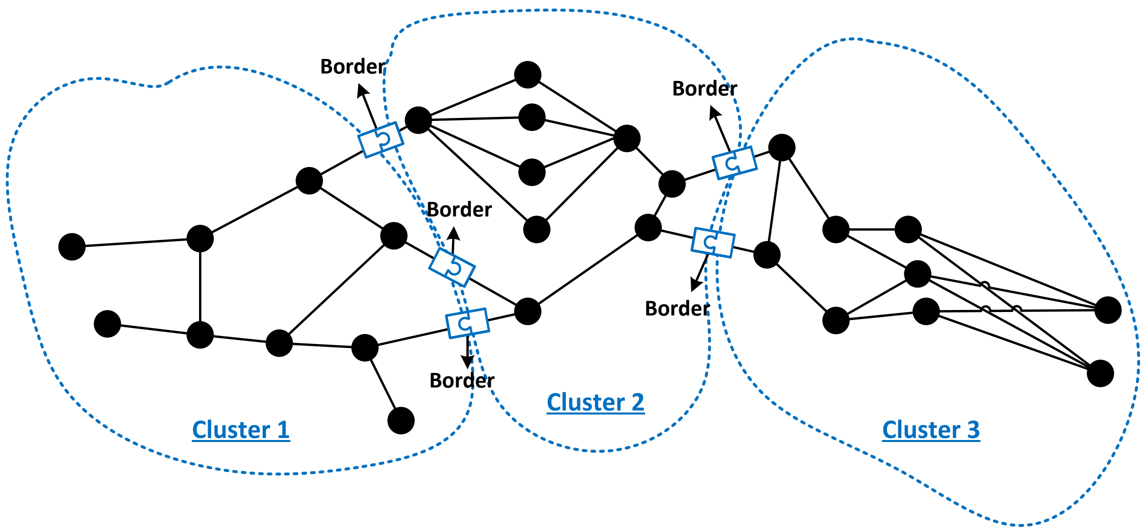


Figure 73

INT has three types of groups:

- subLocation
- milestone[21]
- levelCrossing[22]

And three types of spatial Clusters:

- Grid and InterGrid (short: GIG)
- SER-zone
- location

This chapter describes these various types of Groups and Clusters in more detail.

3.1 Groups

3.1.1 subLocation

subLocations are *railway stations*, *halting places*, *shunting tracks*, *shunting yards*, *bifurcations*, or *any* other site on the network where a train can perform an operation. subLocations are modelled as **groups** (not clusters) of one or more related locationPoint nodes. See §1.9 for more information on locationPoints.

int:subLocation groups corresponds to **a371:ptdes** and/or **a371:ptcar**

The INT system automatically creates a subLocation:

- for each a371:ptdes
- for each a371:ptcar that has *zero* a371:ptdes
- for each a371:ptcar of type *Bifurcation* or *Grill*[23]

Note: each int:subLocation group has a reference to the corresponding a371:ptdes and/or a371:ptcar. If an a371:ptcar has more than one a371:ptdes, multiple int:subLocations will refer to the same ptcarId

Each subLocation has a *type*[24] to indicate the type of railway site it represents:

- station
- stop
- bifurcation
- grid
- shuntingYard
- deadEnd
- frontier

[21] 23/04/2015: not yet implemented
[22] 15/12/2014: not yet implemented
[23] A a371:ptcar of type Bifurcation or Grill with an associated a371:ptbed will result in two int:subLocations
[24] 26/03/2015: Type is not yet implemented

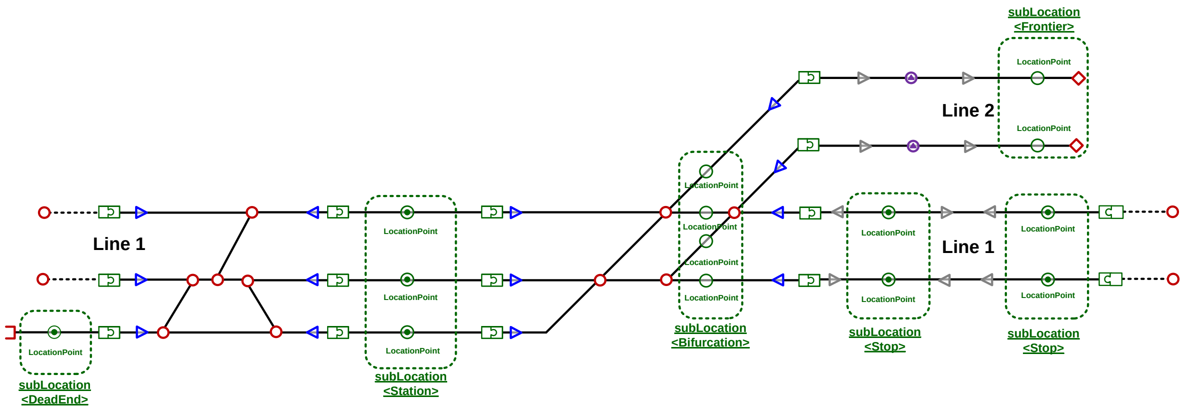

subLocation <Frontier>
LocationPoint
LocationPoint
Line 2
LocationPoint
LocationPoint
LocationPoint
LocationPoint
Line 1
LocationPoint
LocationPoint
LocationPoint
LocationPoint
LocationPoint
LocationPoint
LocationPoint
Line 1
LocationPoint
subLocation <Bifurcation>
subLocation <Stop>
subLocation <Stop>
LocationPoint
subLocation <DeadEnd>
LocationPoint
subLocation <Station>

Figure 74

The example Graph in Figure 74 has **six** subLocations, with different types:

subLocation<station>
This passenger station has three tracks. The corresponding subLocation is a group of three locationPoints, with both an operation and a timingPtcar/timingLocation function.

subLocation <deadEnd>
This dead end shunting track is a subLocation group with only one locationPoint. This locationPoint has an operation function, but no timingPtcar function since it does not represent a a371:ptcar (in this example).

subLocation<bifurcation>
The grid at the right of the Station is a Bifurcation because here **Line 2** branches of **Line 1**. The corresponding subLocation is a group of four locationPoints, which have timingPtcar/timingLocation functions, but no operation functions because they do not allow any train operations other than a passage.

Note: if this location would have had the *same* incoming and outgoing Lines at *both sides*, the location type would have been *Grid* instead of *Bifurcation*.

subLocation<stop>
The two passenger stops on Line 1 both have two tracks. The corresponding subLocation are groups of two locationPoints with both an operation and a timingPtcar/timingLocation function.

subLocation<frontier>
This location represents the net frontier. It has two locations with only a timingPtcar /timingLocation function.

A **subLocation** group has the following properties:

- **id**

  unique ID of the subLocation

- **validFromDate / validToDate**

  validity period of the subLocation version

- **shortNameDutch, shortNameFrench**

  short name of the subLocation, matching the short name of the a371:ptdes or a371:ptcar

- **longNameDutch, longNameFrench**

  long name of the subLocation, matching the short name of the a371:ptdes or a371:ptcar

- **locationId (optional)**[25]

  reference to the location *cluster* to which this subLocation *group* is assigned. See §3.2.3 for more information on locations.

- **ptdesId (optional)**

  reference to the corresponding a371:ptdes, if any.

  This property is **not** present if the subLocation corresponds to:
  - an a371:ptcar without any associated a371:ptdes
  - an a371:ptcar of type Bifurcation of Grill
    (because if such a a371:ptcar has one or more associated a371:ptdes, another specific subLocation will be created for each one of them)

- **ptcarId**

  reference to the corresponding a371:ptcar

[25] This property should become required in the future

3.1.2 levelCrossing[26]

A ***levelCrossing*** is a group of one or more microscopic ***levelCrossingPoint****s*. The Graph in Figure 75 has a three levelCrossings.

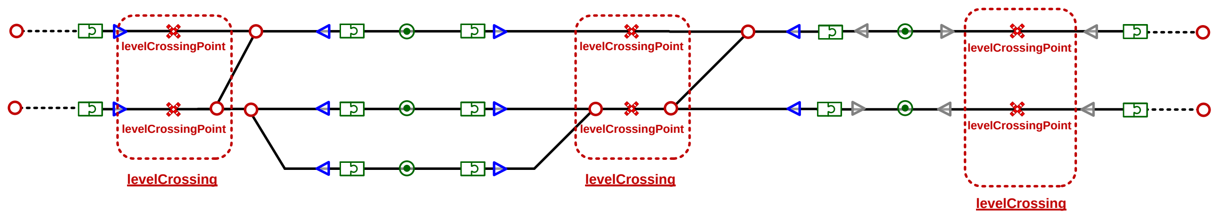


Figure 75

Each levelCrossing contains railway specific information (type, name, line, milestone, responsible signalling box) as well as general information (street, city, ..).

[26] 2013/10/23: Not yet developped in INT

3.1.3 milestone[27]

A ***milestone***[28] is a group of one or more microscopic ***milestonePoint***s. The Graph in Figure 76 five milestone groups: two on Line 1 and three on Line 2:

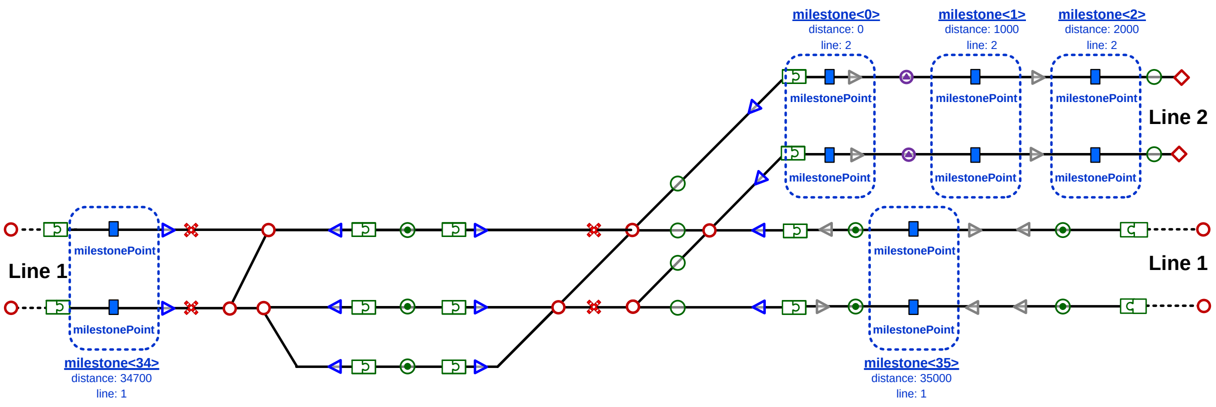


Figure 76

Milestones are used to express the distances of nodes as **distance points** on a line.

A **distance point** is the combination of:

- a reference to the a milestone[29] on the line;
- a distance in meters from that milestone

See ***nodeByTrackEdge*** in $2.2.4 for more detail.

A **milestone** group has the following properties:

- **id**

  unique ID of the milestone group

- **validFromDate / validToDate**

  validity period of the milestone version

- **lineId**

  id of the line on which the milestone is positioned

- **sequenceNumber**

  sequence number of the milestone on the line

[27] 2015/11/09: not fully developped in INT: milestone entities exists and are used in distance point references, but are not yet groups of microscopic milestonePoints
[28] int:milestone, not a371:milestone
[29] usually the *previous* milestone on the line, except for elements positioned before the first milestone

- **distanceFromFirstMilestone**

  distance in meters from the first milestone

3.1.4 platform[30]

A platform is a group of platformSides

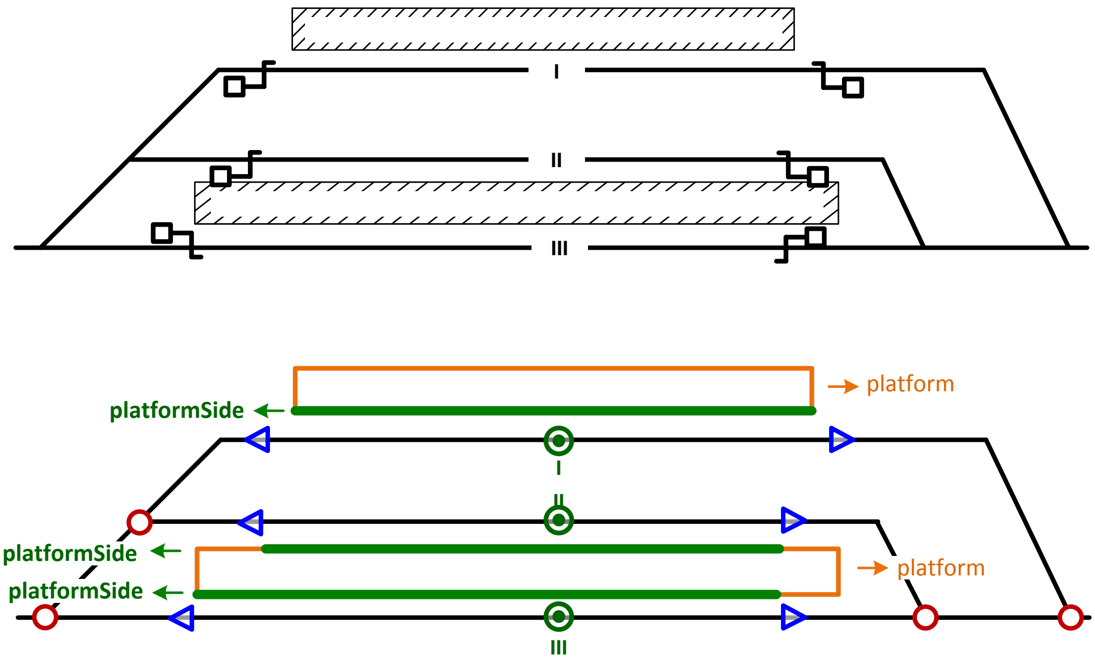


Figure 77

To do..

[30] 2016/09/07: not fully developped in INT: platform and platformSide entities exists and are linked to locationPoints, but are not yet groups of microscopic platformPoints

3.2 Clusters

3.2.1 Grids and InterGrids (GIG)

3.2.1.1 Definitions

A first type of cluster is a GIG (or Grid InterGrid). GIG clusters are used to assign all graph model nodes to either a Grid or an InterGrid.

3.2.1.1.1 Grid

In a railway network, a *grid* (or *grill*) is a group of related switches and crossings that allow trains to change track. A grid must always be *protected* by managed stop signals on each track that grant or prohibit access to the grid, to prevent two trains of using incompatible itineraries at the same time.

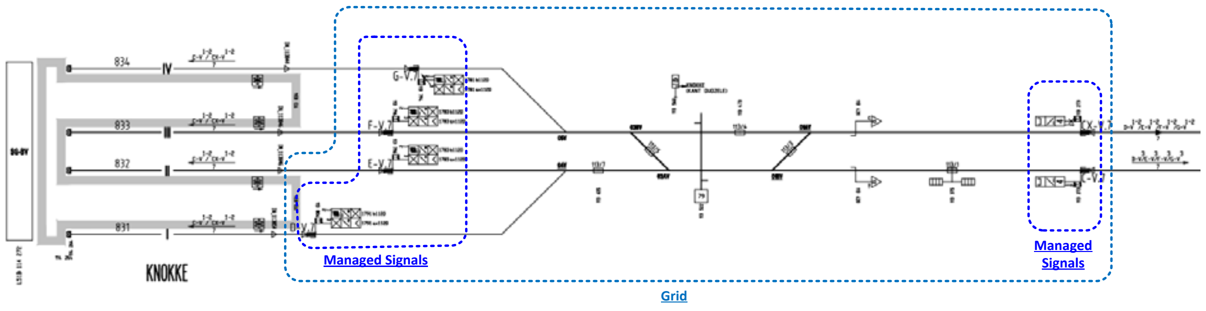
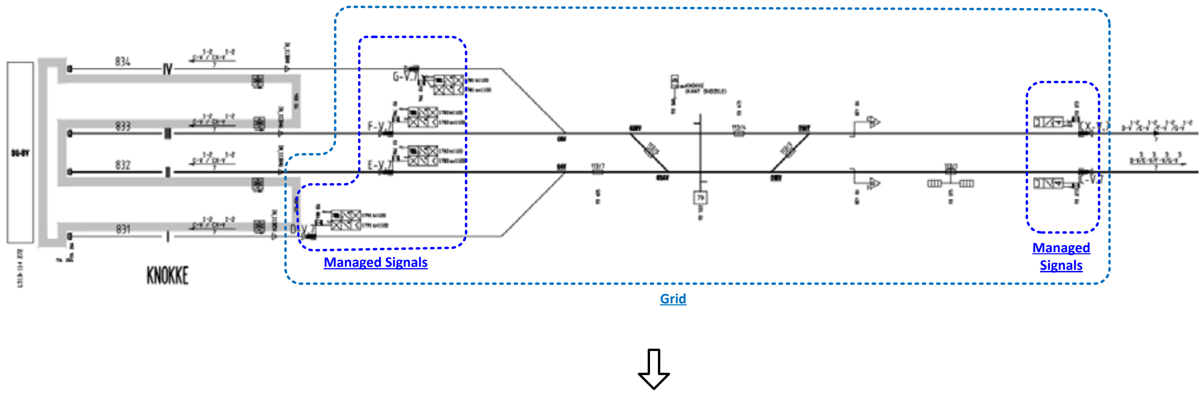


Figure 78

In INT, these grids are modelled as a **spatial cluster**: each **Grid cluster** is a sub-graph delimited by border nodes with a function ***GIGBorder*** (see §1.6.1). The GIG cluster contains all the nodes of that grid including the protective StopSignal and border nodes. (Figure 78)



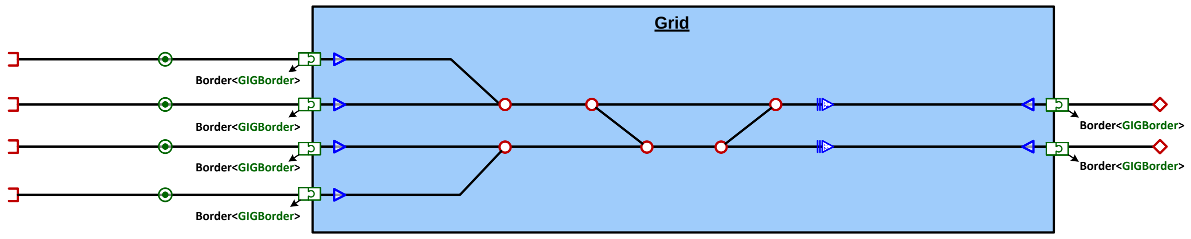


Figure 79

Depending on the available level of detail, a grid can be:

- *mesoscopic;*
- *microscopic*;
- *hybrid*

3.2.1.1.1.1 Mesoscopic grids

If the INT system did not receive any detailed infrastructure information (switches, crossings, routing variants, …) for a certain *grid* from its authentic data sources, a *mesoscopic* Grid is created by default.

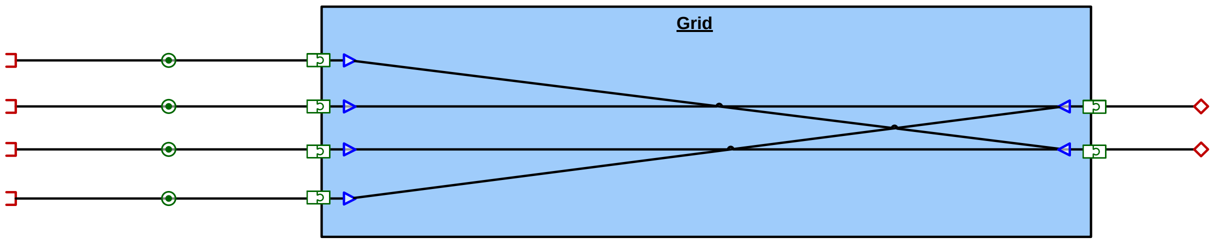


Figure 80

A mesoscopic Grid contains StopSignal nodes and can contain various *virtual* nodes, but lacks all other nodes that represent *physical* infrastructure elements such as switches and crossings. The mesoscopic trackEdges and mesoscopic routeEdges are created by interpreting *allowed* routings inside the Grid, based on the mesoscopic a371:routings.

For example: if we know there is an allowed *Routing* from signal A to signal D, a *mesoscopic trackEdge* and a *mesoscopic routeEdge* will be created from signal A to signal D. The same goes for every known Routing for that Grid. The result is a mesoscopic Grid such as the graph in Figure 80.

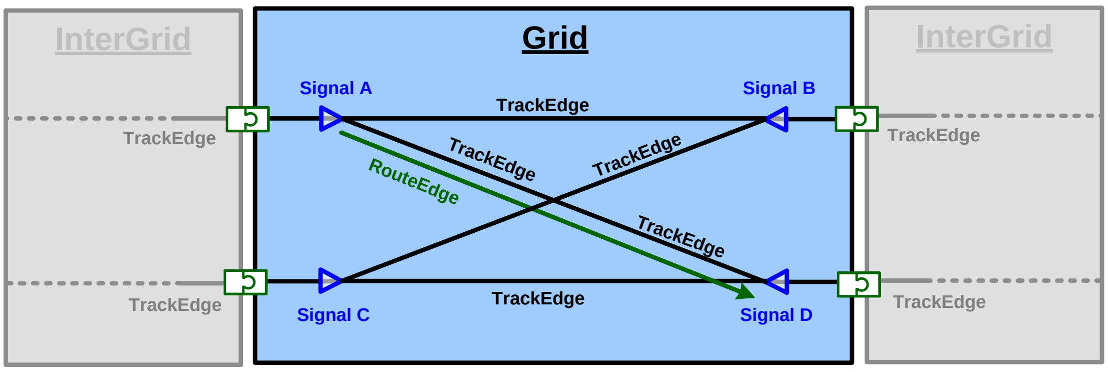


Figure 81

Each mesoscopic trackEdge tells us that there is a *physical possibility* for a train to ride from signal A to signal D, but it does not tell us exactly how.

Each mesoscopic routeEdge tells us that it is *allowed* for a train to ride from signal A to signal D using the underlying mesoscopic trackEdge mentioned above.

Because all these Routings are known and maintained in the a371 database itself, all INT Grids are *mesoscopic* by default. However, when there is an infrastructure data source available that feeds us more detailed infrastructure data, a mesoscopic grid can be upgraded to a *microscopic* Grid.

3.2.1.1.1.2 Microscopic grids

If INT received more detailed infrastructure information from a data source for a certain part of the network, *microscopic Grids* can be created. Unlike mesoscopic grids, these grids contain nodes for all *switches, crossings* and other available infrastructure elements such as *RoutePoints.* They also contain microscopic routeEdges, which still go from signal to signal, but now enumerate all relevant nodes along its path.

Currently, INT uses the A370/Graffiti database as authentic datasource for microscopic information. This system provides detailed route and infrastructure information for all EBP[31] zones.

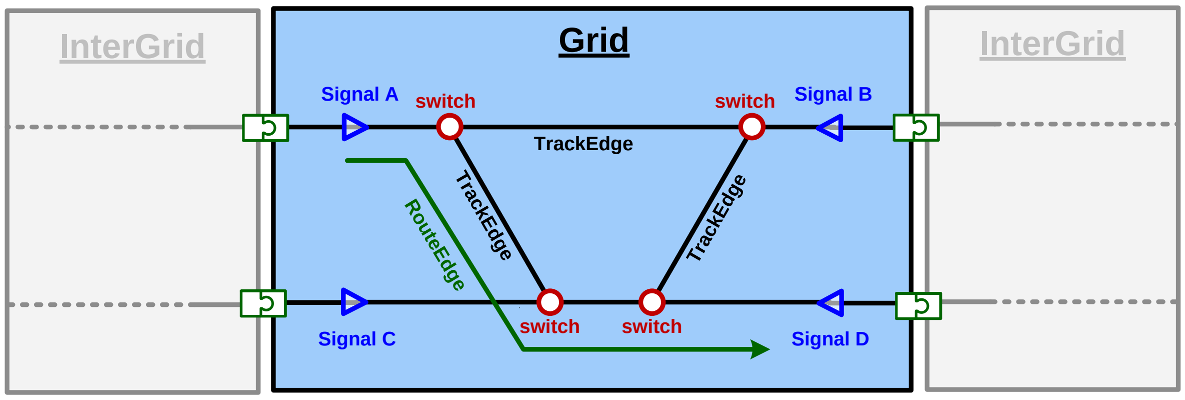


Figure 82

Example: the Graffiti/A370 infrastructure data source provided all possible microscopic routes for the grid in Figure 81. Each route enumerates all the elements (stop signals, switches, crossings…) along the route from one border to another border[32]. The construction of the microscopic INT Grid is similar to the description in 3.2.1.1.1.1, but now we also create a *node* for each encountered infrastructure element and create trackEdges for each piece of track between the switches or crossings.

Note: one of the main goals of the INT project is to always use authentic infrastructure data sources for all elements that are not home to traffic management. There are two important reasons for this: to prevent redundant infrastructure encoding and to be able to fix errors on the lowest possible layer. Therefore, in the near future the INT system will evolve to newer data sources such as InfraRef or INDI as soon as they become available.

3.2.1.1.1.3 Hybrid Grids

As mentioned, the INT system is *mesoscopic* by default and adds *microscopic* details when and where possible. However, when microscopic data is only partially available for a certain grid, or if this data is incoherent with the *default mesoscopic* information, a *hybrid* grid will be created to assure as much detail as possible without loss of information.

A hybrid grid contains microscopic as well as mesoscopic trackEdges and routeEdges.

[31] Regions of the railway network controlled by an EBP ("elekronische bedieningspost") interlocking system
[32] More precisely: from one EBP *track* to another EBP track, outside the grid

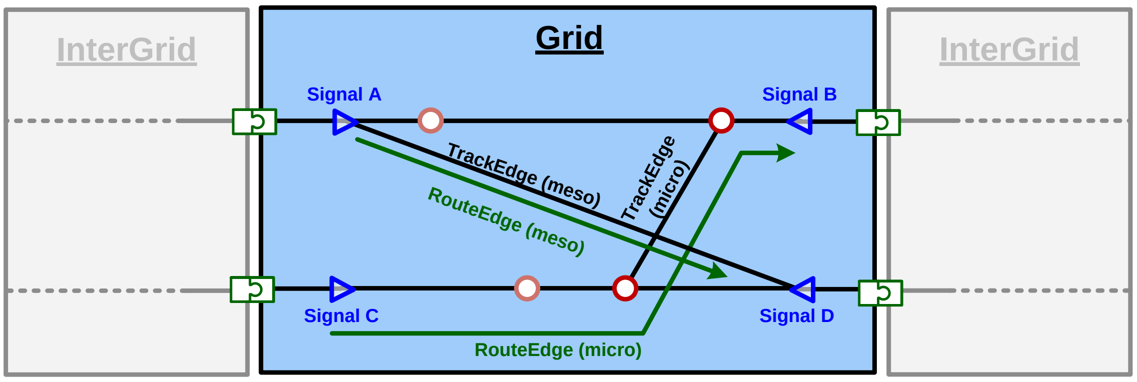


Figure 83

For example, the Grid in Figure 82 is the same grid as in Figure 81, only here suppose that the data source did not provide any detailed routes from signal A to signal D and vice versa. The resulting hybrid Grid has microscopic nodes and edges where possible, but contains mesoscopic track and route edges for each missing route.

3.2.1.1.2 InterGrid

In INT, an InterGrid is the collection of all physical and virtual elements between one[33] or two Grids. They contain all the elements of the railway tracks in open lines, stations, stops, shunting yards, and other railway sites.

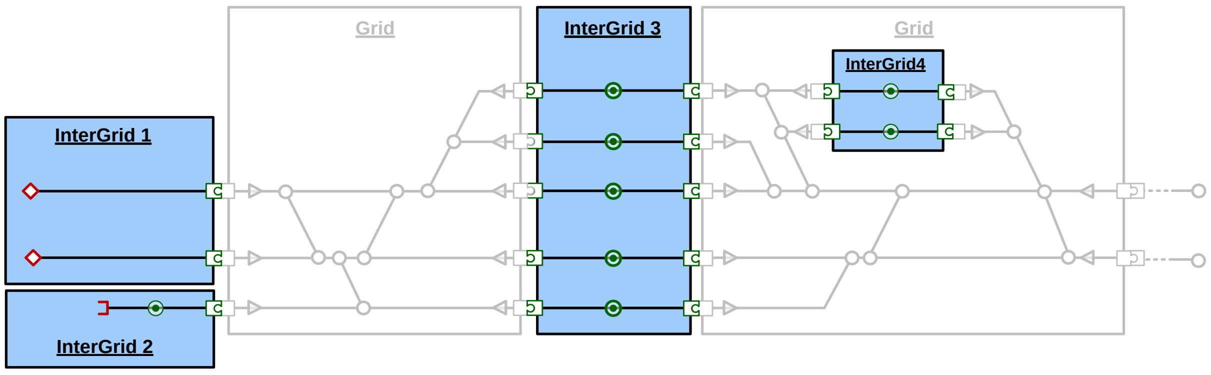


Figure 84

We do not make a distinction between *mesoscopic* and *microscopic* InterGrids because the nodes and (partial) edges inside are equal in both cases. However, it is possible that an InterGrid is connected to a microscopic Grid on one side, but to a mesoscopic Grid on the other side (Figure 84).

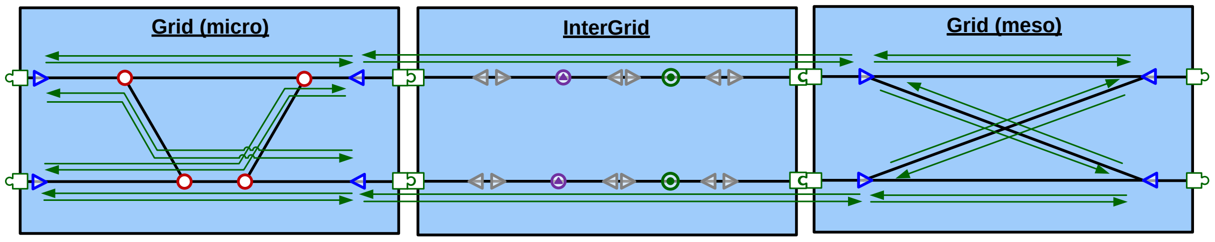


Figure 85

[33] If all tracks inside an interGrid are deadEnd or endOfModel tracks, it only has one adjacent Grid. The same goes for inner shunting tracks inside a grid.

3.2.1.2 Usage

Splitting up the infrastructure graph into GIG-blocks has several advantages:

1. **Data maintenance**

   GIG blocks are often used to locate and validate the huge amount of elements of the full topology. The INT application contains several reports that check the quality and coherency of the infrastructure graph on the import side (data sources) as well as on the export side (release). These reports often indicate in which Grid or InterGrid an issue occurs, so that the INT operator can easily locate and open the corresponding view.

2. **Data sources**

   Grid blocks are used to import and match the infrastructure data coming from the authentic data sources. See 3.2.1.1.1

3. **INT Views**

   Grids and InterGrids are the building blocks for composing the graphical infrastructure diagrams, called *INT views*.

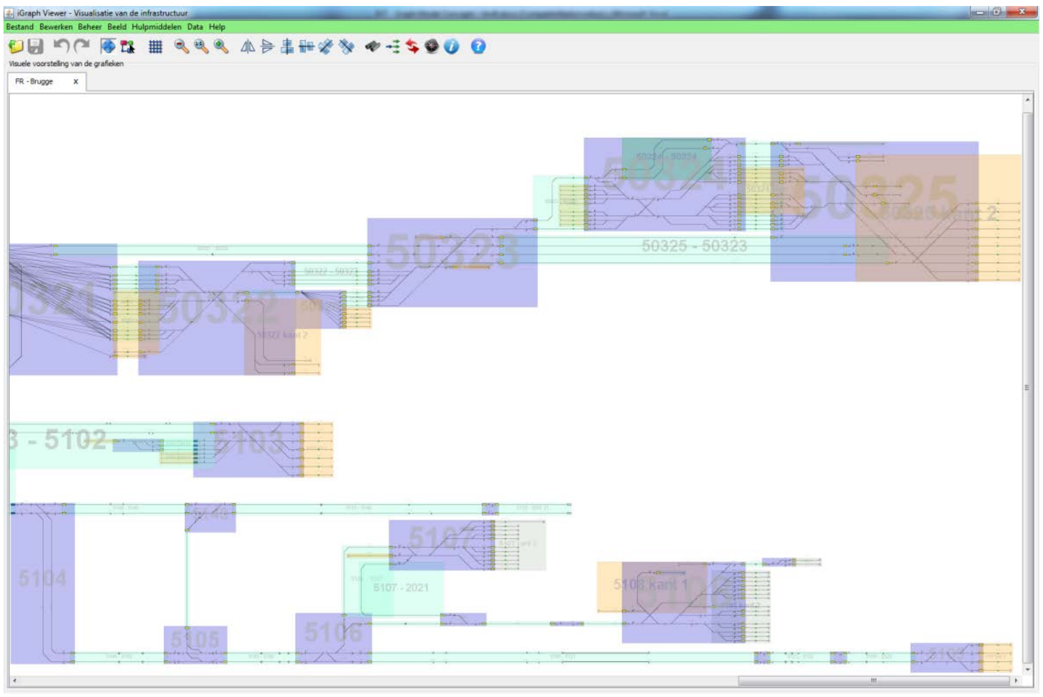

iGraph Viewer - Visualisatie van de infrastructuur
Bestand Bewerken Beheer Beeld Hulpmiddelen Data Help
Visuele voorstelling van de grafieken
FR - Brugge
50323
50325
50325 - 50323
50322
5102
5103
5104
5105
5106
5107
5107 - 2021
5108 Kanl 1

Figure 86

3.2.2 SER zone

3.2.2.1 Definitions

SER

An **SER** is a French acronym for '**S**ystème **E**lectronique d'aide à la **R**égulation' : a series of support systems for train traffic management.

SER zone

for traffic management purposes, the railway network is logically divided in different zones. Each such a zone consists of one main signaling post and may have several smaller (satellite) signaling posts, all sharing the same SER tools.

3.2.2.2 Usage

In INT, these SER zones are modeled as *spatial clusters,* delimited by border nodes that have a **SER-border function**.

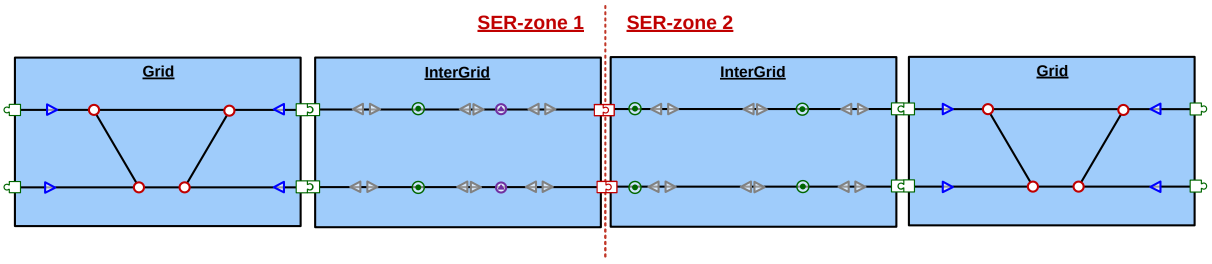


Figure 87

Note that the border between two SER zones is usually between two locations in open line. This border splits the InterGrid cluster too. The corresponding border nodes have at least two functions:

- GIG function: to indicate the border of both InterGrids
- SER function: to indicate the border of both SER zones.

Note: a SER border node is automatically generated by the INT system on a trackEdge when it contains two consecutive stopSignals belonging to two different locations of different SER zones. The border is then positioned in the middle between these two stopSignals.

3.2.3 locations

3.2.3.1 Definition

In §3.1.1, we already discussed *subLocation groups:* a subLocation group is a collection of locationPoints corresponding to the a371:ptdes and a371:ptcar. The main purpose of these groups is to allow the legacy traffic management applications to link these a371 concepts to the INT graph model concepts.

However, in order to link the *microscopic graph model* to the *macroscopic graph model*, we need a much more detailed definition of these places of interest.

A **location i**s a *spatial cluster* that contains **all nodes** that belong to one or more subLocation groups.

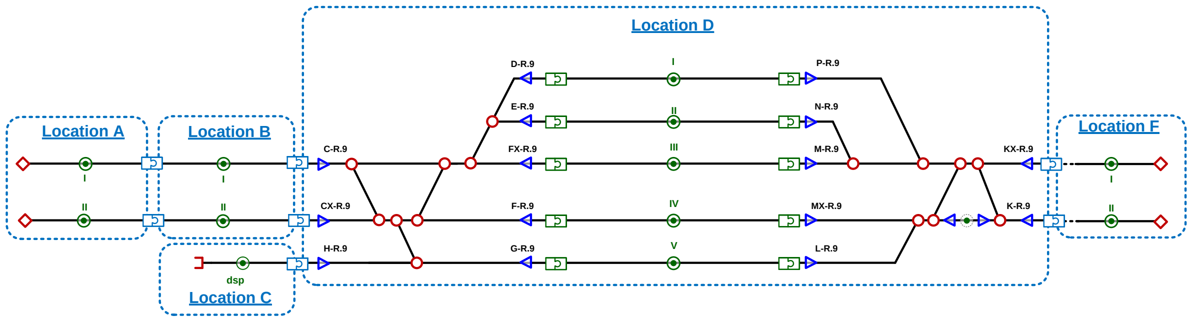


Figure 88

3.2.3.2 borders / locationBorder functions

Each location cluster is microscopically delimited by **border nodes** that have *locationBorder* functions (see §1.6.3) to indicate that they represent the boundaries of a location cluster, see the blue border nodes in Figure 88

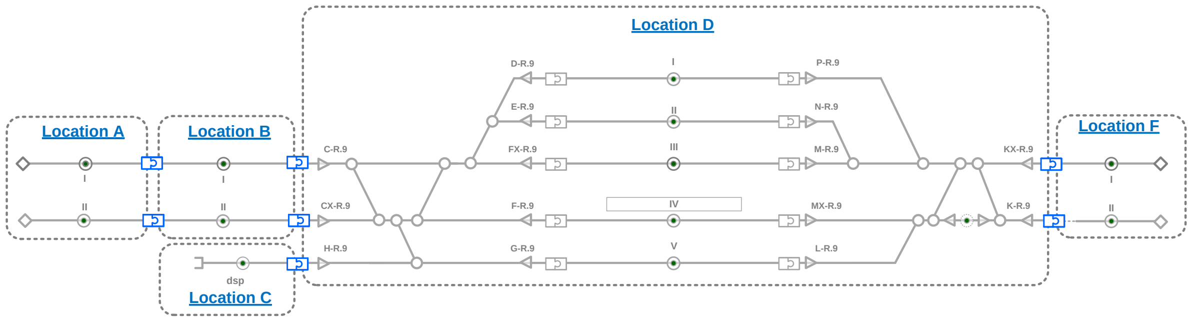


Figure 89

Unlike GIG and SER borders, location border functions are *manually* added by the INT operators; they manually define the boundaries of each cluster, based on a set of business rules. Often they will try to reuse existing border nodes that are already present for SER and/or GIG clusters, and simply *add* a locationBorder function to it. However, if no existing border is available yet, a new border node must be added.

border nodes with a locationBorder function are also called location borders.

3.2.3.3 locationPoints / timingLocation functions

Each **passage through** a location cluster, using the available routeEdges, must encounter **exactly one locationPoint** with a function **timingLocation** on its path from border to border.

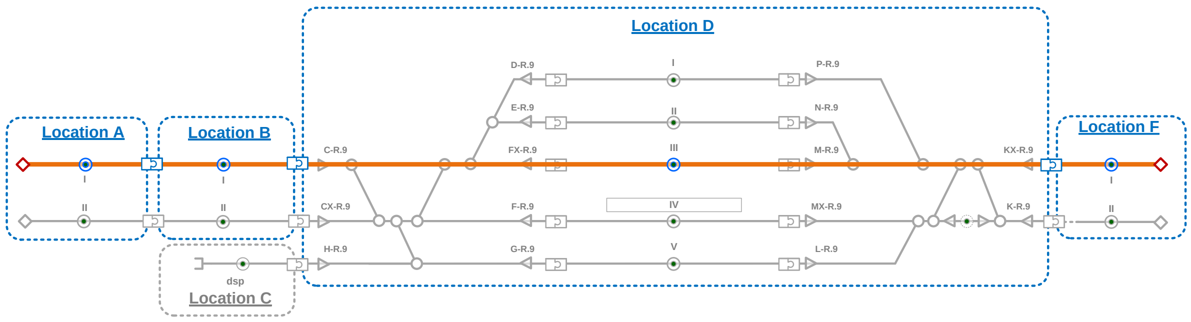


Figure 90

See §1.9.1 for more info on the locationPoints.

3.2.3.4 The link between subLocation and location

Although the relation between a location and a subLocation is often *one-to-one*, sometimes a location cluster can contain **multiple** subLocations; in that case, only one of its subLocation is considered to be the **main** - or 'most important' - subLocation and lends its name to the location.

One location can never contain multiple subLocations that are linked to different a371:ptcar, because the macroscopic models of a371 and INT must always remain compatible

Figure 90 shows the link between subLocation *groups* and their corresponding location *clusters*.

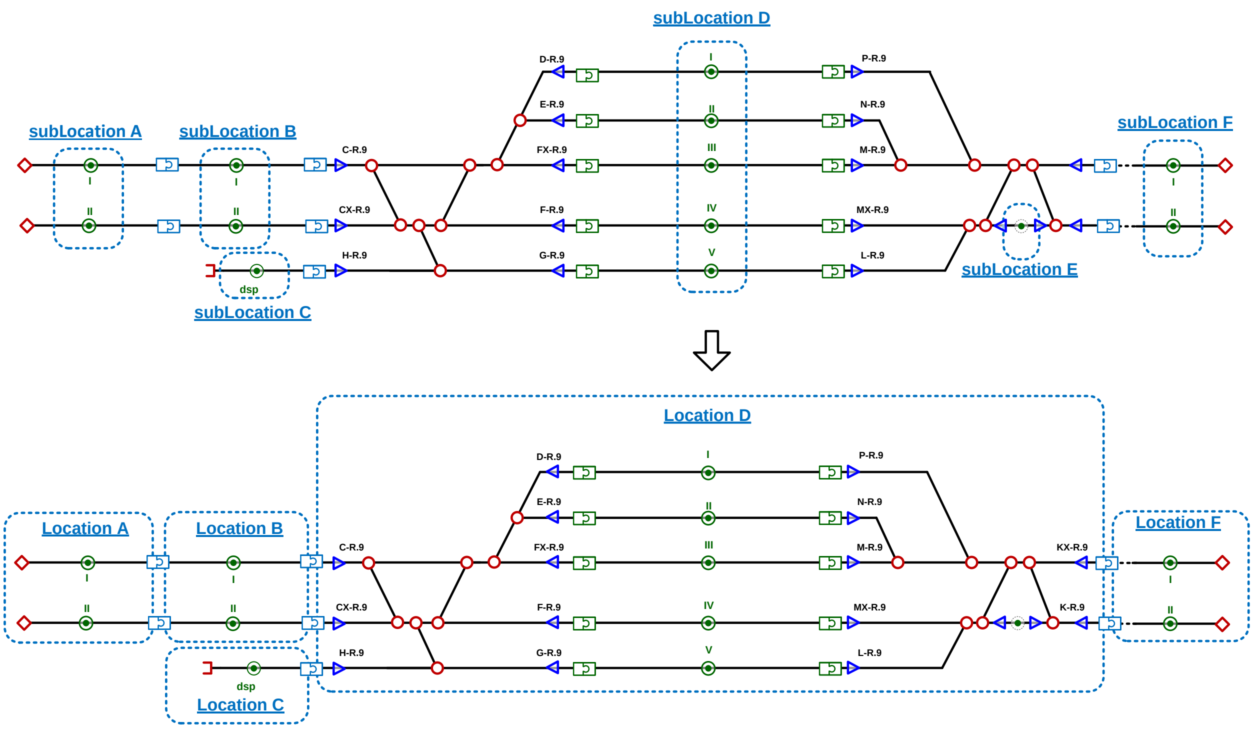


Figure 91

For locations A, B, C and F, the link between subLocation and location is one-to-one, but location D consists of **two** subLocations: subLocations D and subLocation E. subLocation D is the **main** subLocation because it is the most important one: it is a station while subLocation F is merely a shunting track[34], therefore the cluster is named after subLocation D.

These two subLocations D and F **cannot** be clustered in two separate location clusters like all the other subLocations, because that would break the rule that says: "*each passage through a location cluster, using routeEdges, must encounter exactly one locationPoint with a timingLocation function*".

[34] If multiple subLocations have the same functional importance, the number of locationPoints would win the tie

For example:

- If subLocation E would be clustered as a stand-alone location, it would be completely surrounded by location D. A train that rides from the right location border (LB1) to location E (via LB2) would pass though location D without encountering any locationPoints of location D (Figure 91):

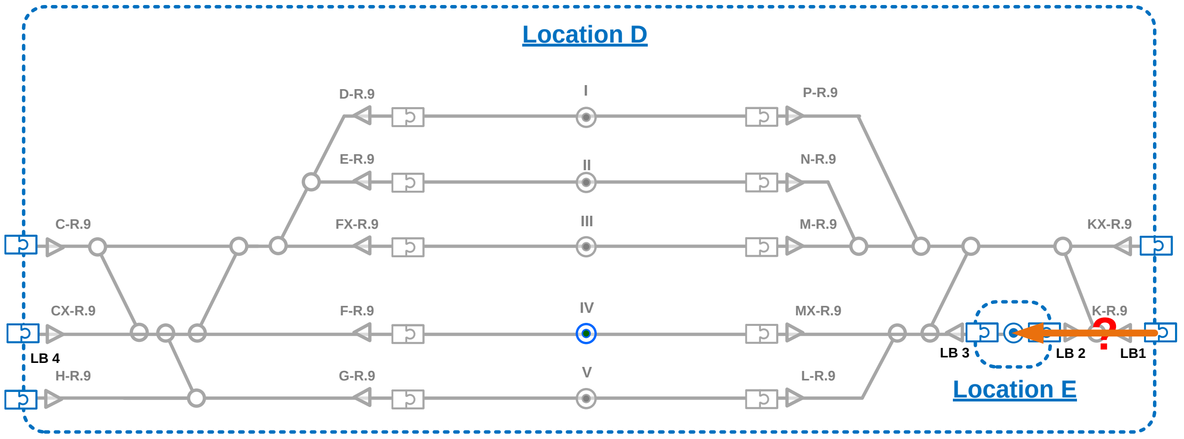


Figure 92

- To solve this, we cannot add extra locationPoints for location D either, because then a train could encounter multiple locationPoints for the same location D sequentially (Figure 92):

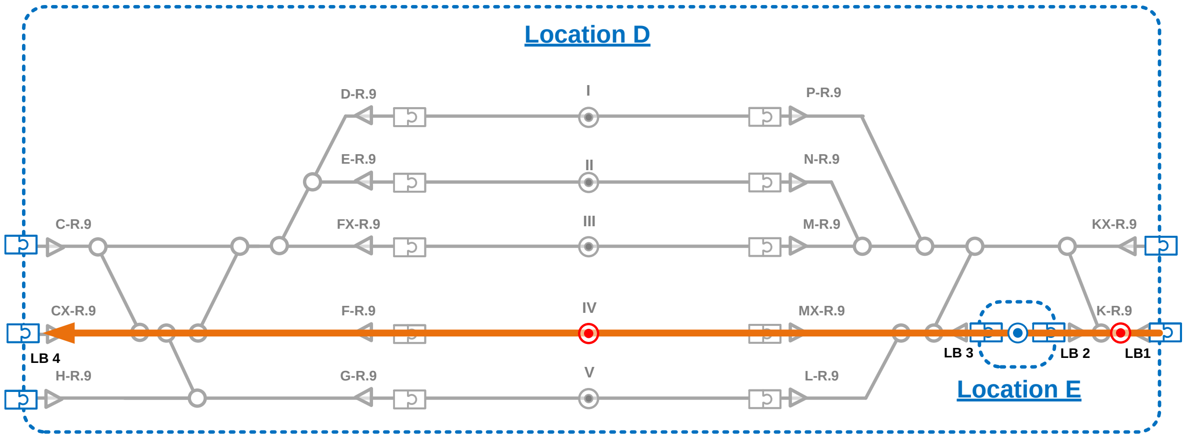


Figure 93

- Neither could we cluster it differently without relapsing to the same situation as the examples above (Figure 93)

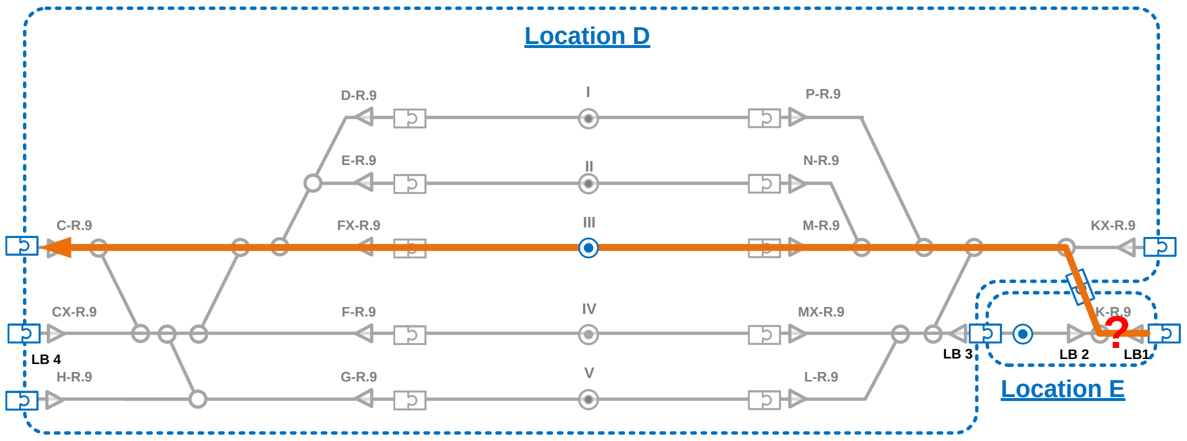


Figure 94

3.2.3.5 Properties

A location cluster has the following properties:

Id
Unique technical id of the location

validFromDate/validToDate
Validity period of the location version

longNameDutch
Long Dutch name of the location cluster, based on its main subLocation name

longNameFrench
Long French name of the location cluster, based on its main subLocation name

shortNameDutch
Short Dutch name of the location cluster, based on its main subLocation name

shortNameFrench
Short French name of the location cluster, based on its main subLocation name

symbolicName (max 10 characters)
Language independent symbolic abbreviation of the location cluster. The maximum of 10 characters is not a technical constraint, but rather a practical business rule. The symbolicName of a location cluster is *globally unique* and based on the symbolic name of the a371:ptcar. See §3.2.3.8 for more info on the naming convention.

ptcarId
Reference to the a371:ptcar that is associated with this location. This is automatically derived from its subLocation(s)

netId (optional)
UIC net code of the county the location is part of

3.2.3.6 External locations

Since the INT graph model does not cover foreign infrastructure, it doesn't have microscopic foreign location clusters either. Same goes for domestic private (unmodelled) shunting yards.

However, for each a371:ptcar or a371:ptdes that is located behind the 'end of the model', INT will create a corresponding subLocation[35] and a location entity. These "external" locations have all properties as mentioned above, but do not contain any nodes and are not delimited by border nodes; they simply tell client systems of their existence and allow them to map them to their matching a371 entities.

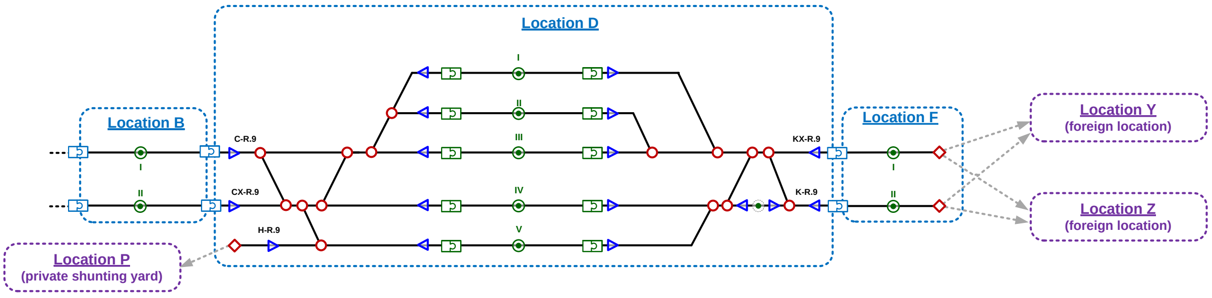


Figure 95

Each endOfModel node provides a list of references to all external (sub)locations it gives access to, see Figure 94[36].

These links are only references, not trackEdges or routeEdges: they merely inform client navigational systems where they should ***leave*** the graph model in order to reach a foreign location, or vice versa: where to enter the graph model when coming from an external location.

[35] Exception: foreign a371:ptcar are always located behind the end of the model and do not have a corresponding int:subLocation; only an int:location
[36] 27/06/2016: not yet implemented

3.2.3.7 Link between microscopic and macroscopic model

The macroscopic INT model uses more high level concepts to describe the railway infrastructure, such as locations, lines, lineSections and tracks.

Although not yet explicitly exported[37] as such, the macroscopic INT model can be considered a graph model too: the nodes are not individual switches or signals, but entire locations, and the edges between these nodes are possible macroscopic connections between these locations (using Lines and Tracks)

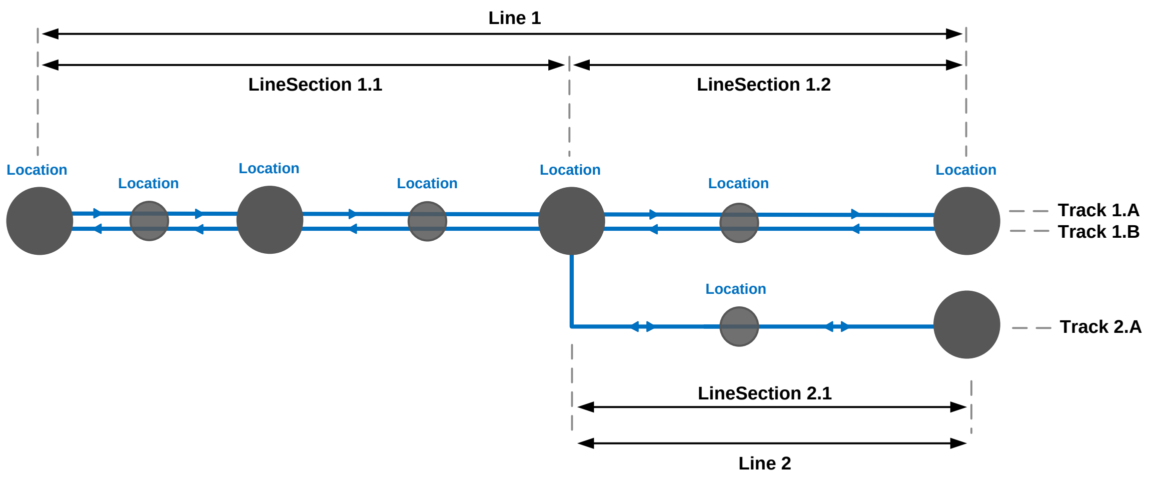


Figure 96

The main purpose of detailed location clustering as described in previous paragraphs is to create a macroscopic model in such a way that it is not only linked to, but even automatically derived from the microscopic graph model:

[37] 27/06/2016: in the near future, the macroscopic INT graph model will be exported explicitly, by adding locationConnectionTrack edges between locations.

Figure 96 visualizes the link between the microscopic and macroscopic model:

- Each location cluster represents a node in the macroscopic model
- A location border node between two location clusters represents a possible macroscopic connection between them.

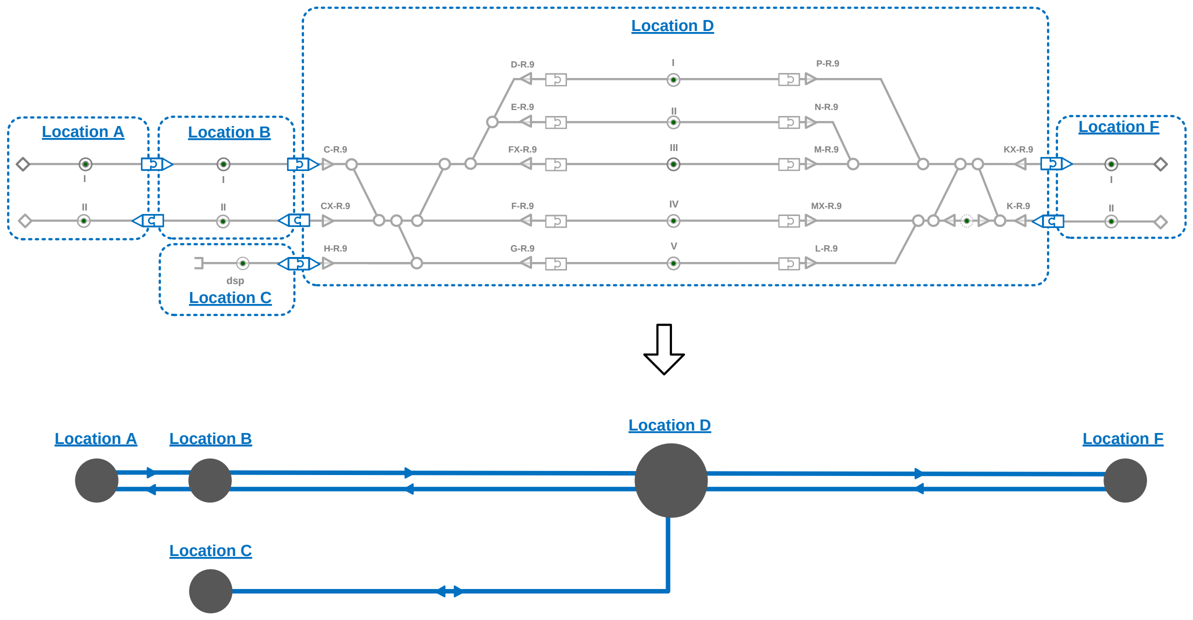


Figure 97

Macroscopic location connection tracks are not directed (they are valid in both ways), but they can have an indication of the normal driving sense if the corresponding location border node has a normalDrivingDirection property (see §1.6.3).

Note: since each location cluster contains a reference to its corresponding a371:ptcar, and because each location border has properties that reference the associated a371:line, a371:lineSection and a371:track, the macroscopic INT model remains compatible with the a371 macroscopic model.

3.2.3.8 Location symbolic name convention

Unlike subLocation groups, which are automatically derived from the a371:ptdes and a371:ptcar, location clusters are manually created and maintained by the INT operators, including the **name** properties.

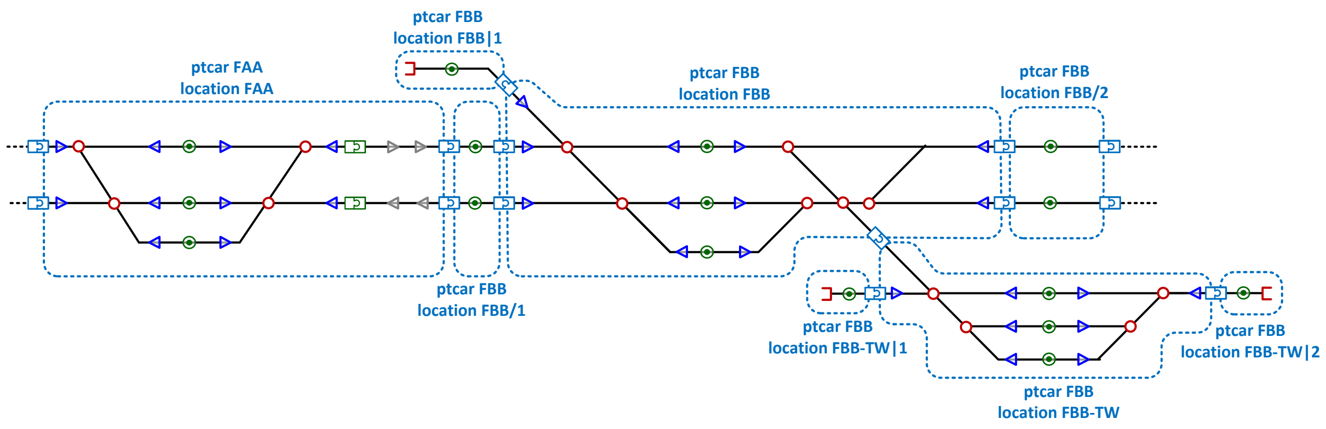


Figure 98

While the **long** and **short names** of a location can easily be taken from its **main** subLocation (see §3.2.3.4), the **symbolicName** must be uniquely defined by an INT Operator according to a set of business rules:

- Each **a371:ptcar** already has a short symbolic name (max 5 characters) which is unique and language independent. As a general rule, this symbolic name is used as **base prefix** for the symbolic name of the location.

  for example: the symbolic name of ptcar Brussel-Zuid / Bruxelles-Midi is "FBMZ"

- If there is only **one** location cluster associated with a certain ptcar:

  The **main location** (the one containing the most important subLocation) simply re-uses the symbolic name of the ptcar:

  location.symbolicName = ptcar.symbolicName

  e.g.: location FAA in Figure 97

- If one a371:ptcar corresponds to **multiple** location clusters:

    1. The **main location** (the one containing the most important subLocation) simply re-uses the symbolic name of the ptcar:

       location.symbolicName = ptcar.symbolicName

       e.g.: location FBB in Figure 97

    2. For each location **adjacent to** this main location, use its symbolic name and add a specific **suffix** based on the **usage** of the location and its **side** relative to its ptcar:

        - for **deadEnd** locations: add suffix **"|"** and its **side 1 or 2**

          e.g.: location FBB|1 in figure Figure 97

        - for **shunting track** locations (sas): add suffix **"/"** and its **side 1 or 2**

          e.g.: location FBB/1 in figure Figure 97

        - for **shunting yards** or other sites: add suffix **"–"** and **max 3 characters**

          e.g.: location FBB-TW in figure Figure 97

    3. If there are still other unnamed locations for this ptcar, reapply step 2 recursively for each location.

       e.g.: location FBB-TW|2 and FBB-TW|1

3.3 locationZone[38]

A **LocationZone** is a **group** of **related**[39] **locations clusters**, sharing the same a371:ptcar. Since locations are already clusters, a LocationZone is modelled as a group of clusters.

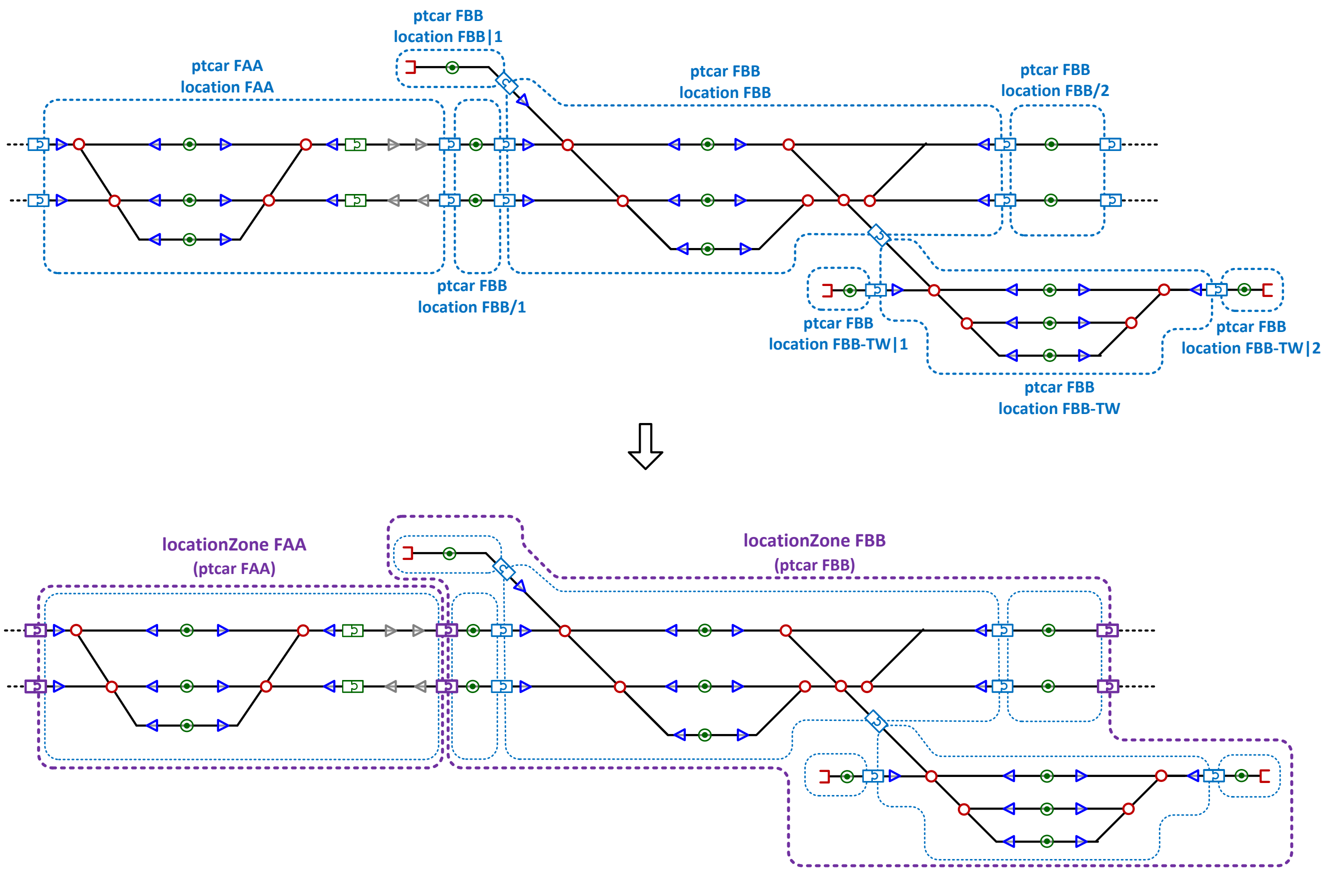


Figure 99

Some locationZones only contain a single location, while others will merge several clusters. In that case, the most important location in the LocationZone is indicated to be the ***main*** location.

[38] 17/10/2013: not yet implemented in INT
[39] Linked to the same a371:ptcar

3.4 line, lineSection, lineSectionTrack

3.4.1 Concepts

The macroscopic a371 infrastructure is modelled using **lines**, **lineSections** and **lineSectionTracks**.

A **line** is a railway connection between two macroscopic locations, such as stations or bifurcations. A line can have multiple tracks and can have several intermediate macroscopic locations.

A line is divided into **lineSection**s. Each lineSection is a piece of the Line with a constant number of tracks between two locations on which it is not possible to change track.

Each Line has at least one lineSection, but if some of the Line's intermediate locations have grids that allow trains to change between Line tracks, the Line is split into multiple lineSections. A lineSection has a number of **lineSectionTracks**: one for each its tracks.

Both lines, lineSections and lineSectionTracks are not directed, but oriented along the ascending milestones of the line.

For example:

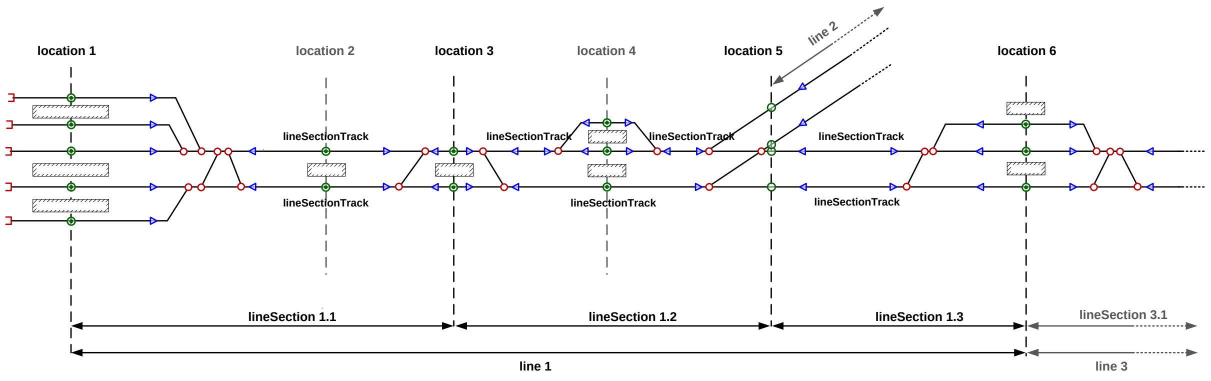


Figure 100

Line 1 in Figure 99 is a two track rail connection between **location 1** and **location 6.** It has with four intermediate locations:

- **locations 2**
- **locations 3**
- **locations 4**
- **locations 5**

Because **location 3** and **location 5** allow trains to change track, Line 1 has not one but three lineSections:

- **lineSection 1.1** between **location 1** and **location 3**
- **lineSection 1.2** between **location 3** and **location 5**
- **lineSection 1.3** between **location 5** and **location 6**

Each lineSection in the example has two lineSectionTracks: one for track A and one for track B of the Line.

3.4.2 Link between lines and graph model

To be able to link the macroscopic Line model with the microscopic trackEdge model, INT publishes a special kind of group: **nodesBylineSectionTrack**.

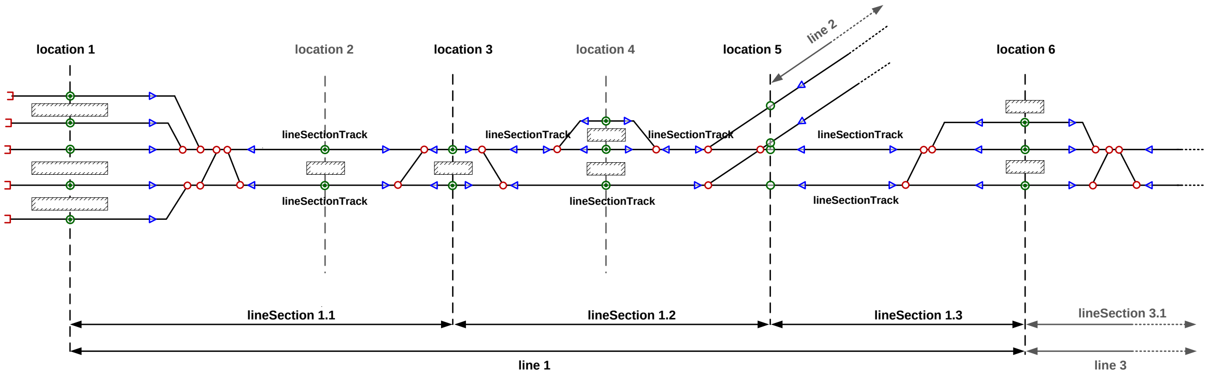


Figure 101

Each lineSectionTrack in the macroscopic line model has a corresponding nodesBylineSectionTrack group that sums up all sections of the open line track that are part of that lineSectionTrack

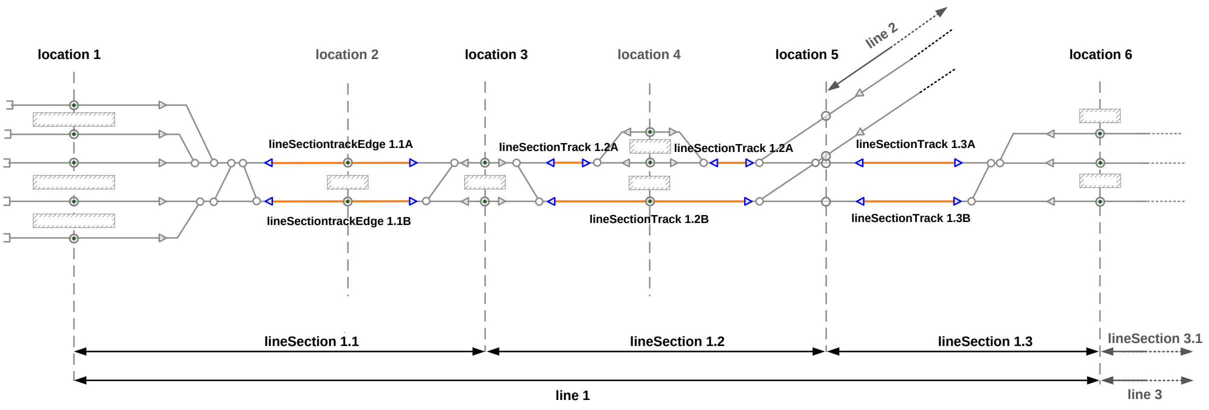


Figure 102

The group can contain the following nodes:

- stopSignal
- locationPoint
- border with serBorder function

These nodes are sorted in order of ascending line milestones and each ***nodeBylineSectionTrack*** indicates which of its ports are connected to which trackEdge.

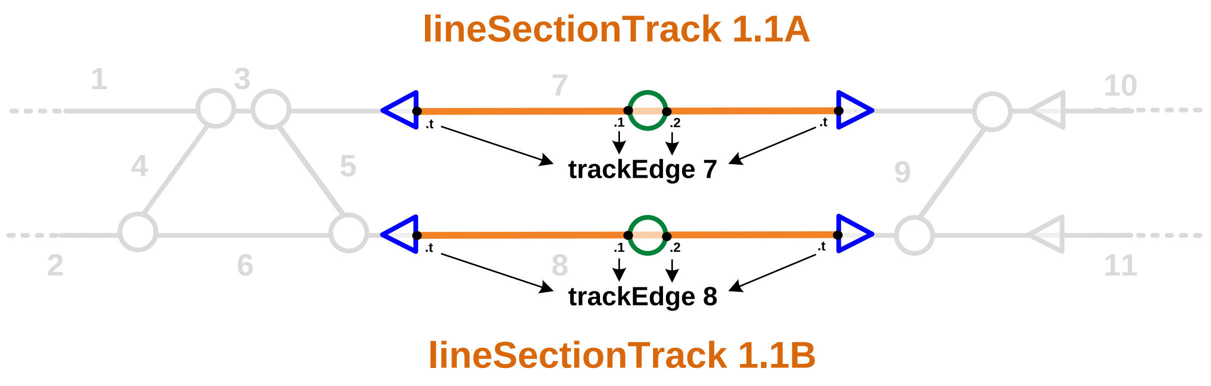


Figure 103