Subject: railML 3.2: Additional information for travel paths in a macroscopic netElement
Posted by Thomas Langkamm on Mon, 06 Dec 2021 09:27:36 GMT
View Forum Message <> Reply to Message

In railML 2.5 we introduced a new "relation" element to ocps (Adding turning resistances to the <ocp>, https://trac.railml.org/ticket/413). We had some discussions in the IS SCTP group on how to integrate this in railML 3 (which is structurally a bit different from 2.5 there), or if we need this at all.

Here is my proposed solution.

The typical scenario would be that some software works on a mesoscopic or macroscopic version of the network, without detailed knowledge of the microscopic network. We often need more information than the connectivity (which is covered by netRelations) namely information about changes of direction (tracking vehicle position&direction in a train formation) and possibly more information that can be used for routing.

Before I propose a model, I would like to give some background information.

Change of direction (COD)
==========
CODs are considered to be fundamental information in many contexts. This information comes in 2 flavors: (1) Orientation changes (do we reverse driving direction, and are front/back vehicles exchanged?), and (2) the exact number of CODs. An orientation change occurs if we have an odd number of CODs, while the orientation remains the same if that number is even. A few examples where we need one of this information:

** Passenger information systems typically need to know about orientation changes, because the order of the vehicles is reversed. ("1st class in sections A and B today.")
** A duty scheduling system, however, will need to know the precise number of CODs, as each cabin change takes time or needs an additional driver -- we may have 2 alternatives for a train movement, for example one that involves 1 COD and another involving 3 CODs.
** Vehicle dispatch or rostering often only need the same information -- for example the locomotive must be in front of the passenger railcars. However, there are situations where the number of CODs is important as well: Say we have railcars with driver cabins on both side and the cabin on one side has a minor technical issue such that it can't be used to in front, but it can be used as last railcar in a train for example. In this case we need to know if any CODs occur even if the orientation remains (even number of CODs).

These software systems will often operate on a mesoscopic/macroscopic model, i.e. without a full network model, so they can't calculate the CODs on their own. The travel path in a net element may be a black box and hidden from these software systems. We need to provide more information in the infrastructure.

How could we integrate CODs in a macroscopic railML 3.2 model?

Even though we already have a "relation" between a pair of net elements in railML 3 (netRelation), this element is not sufficient to store this information because we need the context of a 3rd net element. Consider the following track plan for example. C is a station with a one siding track where trains could be parked.

With a microscopic netElement structure as follows:

This could have a macroscopic netElement structure as follows:

(Obviously we could split C in 2 net elements, which might be a better design. However, let's stick with one net element because the type of problems that we will see might arise in more complex models where we don't have the luxury to split net elements.)
Note that all netRelations in the macroscopic model are navigable in both directions, and we can navigate between C and all other net elements without changing orientation. This of course changes if we consider triples of net elements:

* From A to B via C (A-C-B) we always have a COD. The route may or may not involve shunting, depending on whether we use the switch on the left or the siding track to the right of C.
* For A-C-E we have a direct path (no change of direction or shunting. But we could also go via the top platform and then use2 CODs to get to E.
* A-C-D is similar to A-C-E (even though the platforms are reversed for the direct and indirect connection).
* For B-C-D we have 2 CODs and possibly shunting. Thus, the orientation of the train remains identical but a driver would have to do 2 cabin changes.
* B-C-E has only a direct connection. (We could go via the top platform in C, but then we have a circle in the path as we pass the bottom platform twice. IMO an internal path must not contain a circle.)
* D-C-E has a change of orientation and either 1 COD or 3 CODs.

Assuming that the model allows no CODs within a net element (which IMO is a reasonable assumption if we want to track CODs at all), we have a well defined information if a train changes direction or not when moving from X to Z via Y that does not depend on the travel route within these net elements. The _number_ of CODs (number of cabin changes) may depend on the travel route within the net elements -- but we do know if that number is even or odd regardless of the travel route.

Therefore I would suggest to use an extension of what we have in railML 2.5 where we use "changeDrivingDirection". I believe we should be able to add more information, namely if the

orientation is changed (changeDrivingDirection=true could also imply 2 CODs resulting in no orientation change) as well as the precise number of CODs (optional). I also believe that the "type" attribute should be an enumeration, that is, we have true/false for each of the proposed types. (requiresShunting, changeDrivingDirection and crossesContraflowTraffic could each independently be true or false.)

We add an object "travelPathInformation" (or maybe a better name? I don't want to use "route" as this would be ambiguous) that refers to 3 netElements: A start X, an intermediary Y and an endpoint Z. (Alternatively we could reference the netRelations linking X/Y and Y/Z in this order.) It describes the internal path for a train coming from X and moving to Z, within the net element Y. There can be more than one travelPathInformation for a sequence X-Y-Z if there is more than one internal path a train could take.

The object has the following attributes (besides the usual attributes like ID, designators and such):
* directed: boolean (mandatory). If true, there might be another travelInformation element for the route C-B-A. If false, the information here can be used for A-B-C as well as for C-B-A.
* distance: numeric. Travel distance for a train using this route (meters travelled on track which is well defined).
* lengthRestriction: numeric. Maximum length of the train for this route (for any change of direction we may have a length limit).
* changesOrientation: boolean. If true, we have a net change of orientation (last railcar coming from A moving to B is now the first railcar on the way to C).
* numberOfDirectionChanges: int. Describes how often the train changes direction.
* requiresShunting: boolean. (Same as railML 2.5.)
* crossesContraflowTraffic: boolean. (Same as railML 2.5.)

Here is how this could look for the travelPathInformations within C (listing all objects starting with A and ending in B, D or E):

<travelPathInformation netElement="neC" start="neA" destination="neB" directed="false" distance="2000" lengthRestriction="200" changesOrientation="true" numberOfDirectionChanges="1" requiresShunting="false" crossesContraFlowTraffic="false"/>
<travelPathInformation netElement="neC" start="neA" destination="neB" directed="false" distance="2400" lengthRestriction="300" changesOrientation="true" numberOfDirectionChanges="1" requiresShunting="true" crossesContraFlowTraffic="false"/>
<travelPathInformation netElement="neC" start="neA" destination="neE" directed="false" distance="3000" lengthRestriction="200" changesOrientation="false" numberOfDirectionChanges="0" requiresShunting="false" crossesContraFlowTraffic="false"/>
<travelPathInformation netElement="neC" start="neA" destination="neE" directed="false" distance="3400" lengthRestriction="200" changesOrientation="false" numberOfDirectionChanges="2" requiresShunting="true" crossesContraFlowTraffic="false"/>
<travelPathInformation netElement="neC" start="neA" destination="neD" directed="false" distance="2500" lengthRestriction="200" changesOrientation="false" numberOfDirectionChanges="0" requiresShunting="false" crossesContraFlowTraffic="false"/>
<travelPathInformation netElement="neC" start="neA" destination="neD" directed="false" distance="2900" lengthRestriction="200" changesOrientation="false" numberOfDirectionChanges="2" requiresShunting="true" crossesContraFlowTraffic="false"/>

(If we make this a child of netElement neC, then the netElement="neC" reference is not necessary.)

Routing
=======
This was given as the main application for the railML 2.5 addition. Basically, we want to determine the route(s) of a train journey where start and endpoint are known but there are several possible routes. However, routing happens on many levels and might need further information. If we want to plan train journeys (even in an early planning stage), I would expect that we need more detailled information than just a "turning resistance" in the network model. And I'm not sure if the infrastructure model is the right place to handle these informations.

As far as I'm concerned I wouldn't include "averageDelayTime" (from railML 2.5), because the time may be highly dependent on other factors (like length and type of the train, if we have a change of direction and the driver has to change cabins). Also "delay" assumes that there is some basis to calculate the delay on, and it's not clear how this basis is defined.

If we want to allow some form of routing preference, I suggest to use a numeric attribute "weight", or possibly an array of weights (weight/id combination), allowing to give a preference that would be optimizer friendly. (In railML we have "priority", but that's an integer between 1 and 255 which might be a problem for optimizers.) This allows to pass some data to an optimizer and use the available paths within that net element in a routing algorithm. The precise meaning of the weight would have to be described on a per-case bases.

This model allows us to migrate a railML 2.5 schema to 3.2, by using 2 weights instead of "averageDelayTime" and "priority". This is how we could implement this:

<travelPathInformation netElement="neC" start="neA" destination="neB" directed="false" distance="2000" lengthRestriction="200" changesOrientation="true" numberOfDirectionChanges="1" requiresShunting="false" crossesContraFlowTraffic="false">
<weight id=" delayTime" value="1.5"/>
<weight id="priority" value="1"/>
</travelPathInformation>
<travelPathInformation netElement="neC" start="neA" destination="neB" directed="false" distance="2400" lengthRestriction="300" changesOrientation="true" numberOfDirectionChanges="1" requiresShunting="true" crossesContraFlowTraffic="false">
<weight id="delayTime" value="6.0"/>
<weight id=" priority" value="20"/>
</travelPathInformation>