Subject: Re: railML 3.x: Data Modelling Patterns
Posted by on Wed, 14 Nov 2018 15:05:26 GMT
View Forum Message <> Reply to Message

Dear all,

In anticipation of the next TT developer meeting I have dealt with the above document and would like to document the points relevant to me in the following.
Christian, please don't see it as a negative comment ;-), with all not mentioned points I agree and I generally welcome that there are defined modeling principles.

I see a general problem: The TT data cannot be easily classified into microscopic or macroscopic. This is probably due to the fact that the TT objects are not hierarchically composed, but rather networked on the same level (m:n relations). Example: I have the option to specify a departure time to the minute or to the second or to omit it at unimportant stations, but from my point of view it is rather a question of vagueness than of macroscopic / microscopic. In all cases, the same data type is used (xs:time), which is always processed the same way by the reading system.
The only obvious differentiation according to microscopic/macroscopic results from the dependencies to the railML3 infrastructure.

Comments on the individual chapters:

Model hierarchy and inheritance (slides 3-6)

Do the requirements for container elements also apply if the container is a subelement (part) of another object or only at the top level?
Otherwise I see the requirements already fulfilled in railML2.x-TT:
- There is a general base element tElementWithNameAndId for all TT objects
- Top-level containers contain only elements of type

Element vs. Attributes (Slide 14)

The mere presence of several (countable) attributes is not a criterion for choosing element vs. attribute. With this reason one could also model a coordinate (x, y) as a countable attribute with the help of two elements. More important seems to be the existence of further information about the respective attribute (validForDirecton="both" in the example).

Unknown information and Default-Values (slide 15/16)

I understand the intended rule to mean that the absence of an attribute should always be interpreted as an "unknown value" in the future. I think this rule is too strict and has several disadvantages.
Currently, a missing attribute can have the following meanings:

1) Value unknown
2) Value intentionally absent. Example: arrival / departure (time) for beginning / ending trains at first / last station
3) Does not apply in context. Example: <stopDescription>.onOff only makes sense when held commercially, otherwise the meaning is "does not apply". However, this value does not currently exist for "onOff
4) Static default value is to be used. Example: <ocpTT>.trainReverse="false" applies in 99% of cases and can therefore be regarded as the default value.
5) Context-dependent default value must be used. Example: <stopDescription>.stopOnRequest is implicitly always true for non-commercial holds and therefore does not have to be explicitly specified.

Possible consequences:

1) is still allowed
2) and 3) could only be eliminated by changing the modelling. I see there primarily the replacement of attributes by elements, since a missing element can still represent both "value unknown" and "value intentionally not present". It is questionable whether this makes the schema more understandable.
4) and 5) These actual default values can certainly always be specified explicitly in the future, but in my opinion this would lead to an insane enlargement of the files. At least the (more frequent) static default values (4.) should be preserved.

Boolean information (slide 17)

It seems to me that the case "Mandatory Boolean with true|false as valid values" is not intended. What is the reason for this limitation?
In general, I would look at boolean values as general attributes and not define them as a special case. Option 2 therefore contradicts the statement "Mandatory elements can never be unknown" (slide 15).
In addition, the question arises in which cases a boolean value will be used in the future and when an enumeration with 2 elements will be used. In my opinion, this decision has so far been decided more according to semantic aspects.

References (slide 21)

As I said before, I still don't have the idea to divide the TT-world into macroscopic / microspcopic, but so far option 1 is used (macroscopic refers to microscopic) more frequently, which I still think makes sense. Example: A <train> references its operating points (<ocpTT>) and not vice versa.

Multiplicity (slide 24)

I generally agree with the statements, whereby in special cases there may be container elements with a minimum of 2 elements, e.g. there is currently a <Path> element with minOccurence="2" in railML2.x.

What is missing from my point of view

Definition of container elements in xml/Uml.

Currently most containers are defined in railML using xs:sequences. Is there a "ban" on other structuring elements such as "xs:choice" or "xs:all" or their equivalents in UML by railLM3-IS?

Use of polymorphism

Inheritance of elements is currently already used in the schema, but rather by defining "template elements" with certain general attributes. However, there are hardly any cases in TT where the schema requires an (abstract) base class, and alternatively different classes derived from it can be used. This would be conceivable, for example, when mapping hold information with an abstract element <ocpTT> with some common attributes and derived subclasses <ocpTTPass>, <ocpTTCommercialStop> or <ocpTTOperationalStop> with additional individual attributes. Is there any rejection or encouragement on the part of railML3-IS?

Best regards
Christian Rößiger

--
iRFP e. K. · Institut für Regional- und Fernverkehrsplanung
Hochschulstr. 45, 01069 Dresden
Tel. +49 351 4706819 · Fax. +49 351 4768190 · www.irfp.de
Registergericht: Amtsgericht Dresden, HRA 9347