Subject: [railML3] Extension methods
Posted by Thomas Nygreen JBD on Fri, 11 Jan 2019 16:01:58 GMT
View Forum Message <> Reply to Message

Dear all,

Apologies for flooding the forum recently, but these are exciting times and there are so many interesting things to discuss.

On page 9 of Christian's slides provided in the modelling patterns thread the "any" extension points are presented. We also know these from railML 2. The effect of these extension points is the flexibility that when a use case arises that is not fully covered by railML, one can create new attributes and elements that can be used at any of these extension points.

In the same thread, Jörg and I briefly touched upon differences between subschemas in the availability of these extension points, but the topic I want to raise here is how these extension points are used.

The complete flexibility of these extension points is both a strength and a weakness. They make it possible to define a new global attribute that can be used on all elements with the anyAttribute extension point. The downside is that it is not possible (with plain XSD 1.0) to restrict where an extension attribute or element can be placed. If needed, this has to be done by formulating rules that are checked through separate routines. One relevant standard for this is Schematron.

The discussion on polymorphism inside railML got me thinking about extensions through polymorphism. In an extension schema it is fully possible to extend a railML complexType to add new attributes. Then, the xsi:type of the standard railML element can be set to this new extended type to include the new attributes. An example:

In railML2.4nor, which uses the extension points provided in railML 2.4, there is a new attribute nor:mounted (I will use the nor: prefix for this extension namespace). This attribute is intended as an extension for the railML element <signal> to describe if the signal is mounted on a pole or gantry. However, there is no technical barrier available to restrict where this attribute is used, so using it on any other element with the anyAttribute extension point will not cause any validation errors. This is not necessarily a problem if we assume that users of this extension apply common sense and read the extension documentation. A use case can also appear where it is useful to use the same attribute on other elements as well. Also, this approach makes it easy to combine multiple extension schemas that were developed independently.

An alternative implementation would be to create a new complexType (e.g. nor:signal) that extends tSignal and adds new attributes such as mounted. To use these new attributes the type of the <signal> element must be recasted by setting it explicitly to the extended type: <signal xsi:type="nor:signal" ... mounted="pole"/>. The strength of this approach is that it does control where the extensions can be used. In this example the mounted attribute can only be used on the signal element and only when xsi:type of that signal is set to nor:signal. The weaknesses are that each element type has to be extended to apply the same attribute to multiple elements, and that it is more complex to combine extensions. If two different extensions are used together, and they both provide a type extending the signal element, one must choose one of the types, or one schema must base its extension on the other.

It is also possible to combine the two approaches. This reduces the problem of colliding extensions. For technical reasons it is however not possible to add new subelements in an extended type if the original element type contains the any element extension point. For the same technical reasons this problem does not apply to attributes.

The extensions I know (TPS and railML2.4nor) both use the provided extension points and no recasting of elements. For railML2.4nor we have discussed but not yet implemented an extra layer of validation in Schematron.

What are the opinions on extending by recasting (i.e. xsi:type)? Should this be allowed, or even encouraged?

Subject: Re: [railML3] Extension methods
Posted by Milan Wölke on Thu, 09 Dec 2021 15:58:08 GMT
View Forum Message <> Reply to Message

Hi Thomas,

I meant to reply to this for a long time already. And after you brought it up again during the conference in Sweden, I feel I should add my position on this.
From my point of view we should switch over to the xsi:type based approach of extending existing railML modelling in railML 3. My reasoning would be that it is otherwise impossible to generate code for an extension. This may actually be fine for minimalistic extensions, but for anything that is a bit more complex, I think that it's a necessity. Even when not using code generation it has advantages such as the fact that like this documents can be validated.
I do not agree with the weakness you see with this approach regarding the fact that you need to specify it more than once like this. It is true, you need to specify it for all the places you need an extension. But you also need to implement it as an exporter and an importer at the same places anyway. Like this at least you can be sure that it either is specified or not.

JM2C

Best regards, Milan

Subject: Re: [railML3] Extension methods
Posted by on Fri, 18 Mar 2022 15:09:20 GMT
View Forum Message <> Reply to Message

Dear coordinators,

since you did ask for, I want to give you an explicit reply on this topic. All in all, we do not have objections against the 'new' (more explicit) approach on extensions, otherwise we would have objected when Thomas wrote about it. But I must confess I am also not happy about that.

I welcome the advantage to be able to explicitly define the place (target element) of the extension. The advantage of clearness and less misunderstanding surely outweighs the possible disadvantages (the need to repeat it for each instance).

I am not happy about the raised complexity, difficulty and formalism which comes along with this. We already see a raising "access barrier" against railML because of its raising complexity especially for "outsiders". Extensions are at least a bit a cure against such concerns. If extensions will now become also more complex and difficult to use by "polymorphism" (to express it rather friendly) or "hack-casting of elements" (to express it rather conservative), we fear that concerns against railML in general will raise even more.

However, as I wrote before: All in all, we do not want to veto. RailML is already so much complex that you can probably only use it if you are a railway-maniac and a computer-freak the same time so that the increase in complexity may be regarded as relatively low.

Best regards,
Dirk.

Subject: Re: [railML3] Extension methods
Posted by Milan Wölke on Thu, 24 Mar 2022 13:31:57 GMT
View Forum Message <> Reply to Message

Hi Dirk,

Thank you for your response and input on this. Also thanks for you continued support of railML.
I agree with you in that the access barrier to railML is getting higher due to the increased complexity of the standard which follows the increasingly complex and diverse requirements railML needs to and aims to fulfill. I also agree that the proposed new way of specifying extensions can be a bit more difficult, especially if one wants to add a rather complex extensions, e.g. a custom extended substructure of an existing railML element. Simple extensions such as an added attribute will not be affected by this change. They will actually still work, for those who prefer it that way.
As I wrote before, I do support this change to extending by subclassing of railML elements, like I understand you also do. However, I think you raised a valid point here. Therefore I would propose to write a detailed step by step tutorial, that explains how exactly to go about when creating a complex railML extension. That could be done after the release of railML 3.2 next month. If possible I would like to ask you to volunteer for reviewing such tutorial. Would you be interested in this? Also do you think that the programming of reading/writing programs should also be included in such tutorial for one or two major programming languages?

Best regards, Milan

Subject: Re: [railML3] Extension methods

Posted by Jörgen Strandberg on Fri, 29 Apr 2022 16:45:45 GMT
View Forum Message <> Reply to Message

Hi,

I have too found it hard to implement a solution for our non-railml data. But I am not sure that it's a bad thing that handling non-standard data is tedious or discouraging.

Could even some new use cases be defined out of the needs for the additional data you indirectly speak of?

A technical question related to the solution where xsi:type is used to define a subclass of a standard type:
Will that not pose a requirement on standard-compliant tools to be able to load foreign namespaces dynamically, to be able to access the standard attribute values?

Subject: Re: [railML3] Extension methods
Posted by Milan Wölke on Mon, 30 May 2022 17:57:53 GMT
View Forum Message <> Reply to Message

Hi all,

As promised I created a wiki article on how to create extensions for railML 3.2. It demonstrates how to extend existing standard railML types and also provides examples on how to use such extended railML elements by use of xsi:type. Please take a look at https://wiki3.railml.org/wiki/Dev:Using_xsi:type and let me know what you think.

Regarding the technical question of @Jörgen; As per our experience it will not do that. Of course if you want to validate the custom extensions you would need to load the xsds, but for simple reading of the standard railML content that should not be necessary.

Best regards, Milan