Subject: constraints for OperatingPeriod
Posted by Andreas Tanner on Tue, 25 Sep 2012 13:00:28 GMT
View Forum Message <> Reply to Message

The operatingPeriod type currently is modelled quite flexibly to accommodate different use cases. However, the standard does not define which combinations of attributes can meaningfully be used together. A stricter definition would spare a a lot of discussions between different users of the standard. Here is a suggestion:

***
The OperatingPeriod element can be used for three different use cases.

- the calendar based operating period:
-- bitmask and startDate are mandatory,
-- endDate, operatingDay, specialService are not allowed. [endDate is not allowed since it would be redundant with startDate + bitmask length]

- standard week operating period:
-- operatingDay is mandatory (at least one),
-- specialService optional
-- startDate, endDate are optional and if used, both must be given
-- bitmask is not allowed

- abstract operating period
-- name or code are mandatory
-- bitmask, operatingDay, specialService are not allowed.

Always allowed, and optional if not declared otherwise, are name, additionalName, code, description, timetablePeriodRef, xml:lang, dayOffset

***

For railML 3.0, I would suggest to model the three cases as distinct types CalendarBasedOperatingPeriod, etc, all derived from base class OperatingPeriod.

Best regards
--Andreas Tanner.

Subject: Re: constraints for OperatingPeriod
Posted by on Tue, 02 Oct 2012 15:34:48 GMT
View Forum Message <> Reply to Message

Dear Andreas,

when the current operatingPeriod type was defined, the aim was to

explicitly allow a 'describing' definition and a bitMask redundantly. The background is that there are many 'describing' expressions for the same bitmask. For instance: "Mo-Fr, 01.08-31.08 only" and "[Sa+So], not until 31.07 and not from 01.09." lead to the same bitmask.

In some cases,
- we do want to have the bitmask for easier and clearer reading of the real operating days,
- we do want to have the 'describing' expressions to exactly reconstruct the input of the user, meaning to 'show' the contents of the bitmask to the user in a kind which is familiar for him and which reflects the input. If the user did input "Mo-Fr[S]", he should not be shown "W[Sa]" and vice versa.

Please have a look at the examples in
www.irfp.de/download/railml_beispiel_verkehrstage.pdf

This means that
> - the calendar based operating period:
> -- bitmask and startDate are mandatory,
> -- endDate, operatingDay, specialService are not allowed.
would not fit to the current requirements.

Concerning a 'third' combination
> - abstract operating period
> -- name or code are mandatory
> -- bitmask, operatingDay, specialService are not allowed.

I would welcome such a possibility. But, from my side we need a definition whether different 'abstract operating periods' are disjunctive or not. Either we define a matrix or, at least, we define that different 'abstract operating periods' _always_ have to be disjunctive.

Additionally, I would prefer to allow an abstract operating period to refer to a 'real' operating period. In my opinion, any abstract operating period earlier or later becomes real. So, we can have a timetable which was planned with abstract operating periods (in the first instance) but in the meantime, the user does know which real dates are assigned to which abstract operating period.

So, an abstract operating period can have
- either a bitmask (at least)
- or a reference to another (non-abstract) operating period.

From these suggestions, I would prefer
- 'abstract operating periods' always have to be disjunctive
- 'abstract operating periods' can optionally reference another

(non-abstract) operating period.

Best regards,
Dirk.

Subject: Re: constraints for OperatingPeriod
Posted by Andreas Tanner on Mon, 15 Oct 2012 07:35:28 GMT
View Forum Message <> Reply to Message

Am 02.10.2012 17:34, schrieb Dirk Bräuer:
> Dear Andreas,
>
> when the current operatingPeriod type was defined, the aim was to
> explicitly allow a 'describing' definition and a bitMask redundantly.
> The background is that there are many 'describing' expressions for the
> same bitmask. For instance: "Mo-Fr, 01.08-31.08 only" and "[Sa+So], not
> until 31.07 and not from 01.09." lead to the same bitmask.
>

This is two different issues:
1) Redundancy
2) "describing" expressions.
With the latter, I agree, but I intended them to be covered in the "standard week" case. It does include the specialService etc. constructions. It should be named more appropriately, though, maybe "rule based" or something like that.

With the redundancy, I do have a problem. It does allow to be lenient when /writing/ railML, but the costs incur at the import side. For serious import software, one has to
- extend the customer-specific specification as to disallow inconsistencies between bitmask and rule
- code a check of the resulting precondition
- add a test case for the software.

So please, let us abstain from casually sprinkle some redundancy through the standard for "easer and clearer reading".

>
> ---
> Concerning a 'third' combination
>> - abstract operating period
>> -- name or code are mandatory
>> -- bitmask, operatingDay, specialService are not allowed.
>
> I would welcome such a possibility. But, from my side we need a
> definition whether different 'abstract operating periods' are

> disjunctive or not. Either we define a matrix or, at least, we define
> that different 'abstract operating periods' _always_ have to be
> disjunctive.

This is an important point. In http://www.irfp.de/download/railml_doku_beispiele.pdf, you write on the issue of /keys/ for a train. In another thread in this forum, delay infos are discussed. There, for instance, one may want to state that "that instance of the 12345 train with operating period "mo-fr except holiday" that runs on oct 13, is delayed". So we need even more than an understanding just of disjoint operating period, namely, a concept of unrolling rule-based operating periods onto a calendar, and identify "instances" within a period (which is easy once we have bitmasks /resulting from unrolling/).

>
> Additionally, I would prefer to allow an abstract operating period to
> refer to a 'real' operating period. In my opinion, any abstract
> operating period earlier or later becomes real.

Here I understand that you aim at the mapping of different stages in the planning process. I agree that this is important. I think that this topic would need more analysis.

Maybe there could be a slot at the upcoming railML conference to discuss these issues?

Best, --Andreas.

Subject: Re: constraints for OperatingPeriod
Posted by on Wed, 17 Oct 2012 18:11:35 GMT
View Forum Message <> Reply to Message

Dear Andreas,

> With the redundancy, I do have a problem. It does allow to be lenient
> when /writing/ railML, but the costs incur at the import side. For
> serious import software, one has to
> - extend the customer-specific specification as to disallow
> inconsistencies between bitmask and rule
> - code a check of the resulting precondition
> - add a test case for the software.

I am aware the problems of redundancies when importing. I do not at all defend them in general.

You do not need to "disallow inconsistencies between bitmask and rule" when importing. You can describe which elements and attributes of RailML are relevant for your software and which are not. (From my opinion, you have to create an own Interface Documentation for a serious import software.) There you can consider either the bit mask or the rule to be relevant and to be exclusive alternatives for your software. If you feel necessary to close all possible inconsistencies you can do, but this is not requested by RailML and probably not completely possible. (For instance, when importing a timetable, you will probably not import all infrastructure and all vehicle data in detail just to check whether there are inconsistencies in it...)

Please consider:
When writing a RailML file, the software does normally not know for which purpose the RailML file will be used. It has to create a RailML file which is most possibly general.
When reading a RailML file, the software can exactly know the requirements of the target system and therefore can decide which elements and attributes are relevant and which additional rules apply. From my opinion, there always will be additional (semantical) rules which are out of the scope of RailML.

RailML does not implicitly avoid all problems that may occur. Rather, RailML gives you a basis not to "reinvent the wheel" again and again each time you need an interface to/from another software.

> So please, let us abstain from casually sprinkle some redundancy through
> the standard for "easer and clearer reading".

Well, ok, but in practice we have to be aware that
- if we always implement the "most scientific, exact but most complicated way" to describe a case only, to totally avoid redundancies, we risk that RailML will not be accepted by other software developers because of being too much difficult. It should not be our aim to create a RailML which is free from redundancies but nobody will use.
- sometimes we have to implement a 'short' solution possibly before we ourselves are aware the more general solution. Then, later, when needing the more general solution, we cannot delete the 'short' solution immediately because of the rules of downward compatibility. So, we have a redundancy at least temporarily.

> a concept of unrolling rule-based operating periods onto a calendar,

I agree. This should have been the bit mask.

> and identify "instances" within a period (which is easy once we have
> bitmasks /resulting from unrolling/).

I have some problem with these 'actual' (running) information in a schema called 'timetable'. I think the term 'timetable' implements that it is about planning only, not about 'actual' (running) information.

But, this should not mean that such information should not be possible in RailML. I am not sure whether it is possible to describe the aspects of 'actual' (running) information in a proper way with current RailML. We have some information about that but they are more dead bodies from RailML1...

Anyway, I agree with you: We would need a possibility to identify "instances" within a period to describe 'actual' information additionally to 'timetable' information - either in 'timetable' schema or somewhere else.

>> Additionally, I would prefer to allow an abstract operating period to
>> refer to a 'real' operating period. In my opinion, any abstract
>> operating period earlier or later becomes real.
>
> Here I understand that you aim at the mapping of different stages in the
> planning process.

Yes. I exactly "aimed" on

> a concept of unrolling rule-based operating periods onto a calendar

So far, this should have been the bit mask. But again: RailML is not perfect nor complete...

Best regards,
Dirk.

Subject: Re: constraints for OperatingPeriod
Posted by Susanne Wunsch railML on Fri, 09 Nov 2012 09:27:47 GMT
View Forum Message <> Reply to Message

Hi Dirk, Andreas and others,

Dirk Bräuer <dirk.braeuer@irfp.de> writes:
>> With the redundancy, I do have a problem. It does allow to be
>> lenient when /writing/ railML, but the costs incur at the import
>> side. For serious import software, one has to
>> - extend the customer-specific specification as to disallow
>> inconsistencies between bitmask and rule
>> - code a check of the resulting precondition
>> - add a test case for the software.
>

> Please consider:
> When writing a RailML file, the software does normally not know for
> which purpose the RailML file will be used. It has to create a RailML
> file which is most possibly general.
> When reading a RailML file, the software can exactly know the
> requirements of the target system and therefore can decide which
> elements and attributes are relevant and which additional rules
> apply. From my opinion, there always will be additional (semantical)
> rules which are out of the scope of RailML.

Thank you, Dirk, for the above described explanations.

Andreas, are you convinced by them?

How about the wanted re-structuring of 'operatingPeriod' for the next major release? Can we drop it? Or do you propose another easier to implement/validate/understand structure?

> Anyway, I agree with you: We would need a possibility to identify
> "instances" within a period to describe 'actual' information
> additionally to 'timetable' information - either in 'timetable'
> schema or somewhere else.
>
>>> Additionally, I would prefer to allow an abstract operating period to
>>> refer to a 'real' operating period. In my opinion, any abstract
>>> operating period earlier or later becomes real.
>>
>> Here I understand that you aim at the mapping of different stages in
>> the planning process.
>
> Yes. I exactly "aimed" on
>
>> a concept of unrolling rule-based operating periods onto a calendar
>
> So far, this should have been the bit mask.

I filed a ticket for the issue of an "abstract operating period":

http://trac.assembla.com/railML/ticket/187

Kind regards...
Susanne

--
Susanne Wunsch
Schema Coordinator: railML.common

Subject: Re: constraints for OperatingPeriod
Posted by Andreas Tanner on Fri, 09 Nov 2012 09:50:47 GMT
View Forum Message <> Reply to Message

Dear all,
as far as railML 2.x is concerned, my suggestion was just to enhance the documentation. The discussion on redundancy has some philosophical traits and therefore, absolute truth cannot be expected to be found.

For railML 3.0, however, I would like to keep the discussion open and let us elaborate a model of validity with a more formally defined semantics that allows to address specific instances of trains from a timetable.

Best, Andreas.

Am 09.11.2012 10:27, schrieb Susanne Wunsch:
> Hi Dirk, Andreas and others,
>
> Dirk Bräuer <dirk.braeuer@irfp.de> writes:
>>> With the redundancy, I do have a problem. It does allow to be
>>> lenient when /writing/ railML, but the costs incur at the import
>>> side. For serious import software, one has to
>>> - extend the customer-specific specification as to disallow
>>> inconsistencies between bitmask and rule
>>> - code a check of the resulting precondition
>>> - add a test case for the software.
>>
>> Please consider:
>> When writing a RailML file, the software does normally not know for
>> which purpose the RailML file will be used. It has to create a RailML
>> file which is most possibly general.
>> When reading a RailML file, the software can exactly know the
>> requirements of the target system and therefore can decide which
>> elements and attributes are relevant and which additional rules
>> apply. From my opinion, there always will be additional (semantical)
>> rules which are out of the scope of RailML.
>
> Thank you, Dirk, for the above described explanations.
>
> Andreas, are you convinced by them?
>
> How about the wanted re-structuring of 'operatingPeriod' for the next
> major release? Can we drop it? Or do you propose another easier to
> implement/validate/understand structure?
>
>> Anyway, I agree with you: We would need a possibility to identify
>> "instances" within a period to describe 'actual' information
>> additionally to 'timetable' information - either in 'timetable'

>> schema or somewhere else.
>>
>>>> Additionally, I would prefer to allow an abstract operating period to
>>>> refer to a 'real' operating period. In my opinion, any abstract
>>>> operating period earlier or later becomes real.
>>>
>>> Here I understand that you aim at the mapping of different stages in
>>> the planning process.
>>
>> Yes. I exactly "aimed" on
>>
>>> a concept of unrolling rule-based operating periods onto a calendar
>>
>> So far, this should have been the bit mask.
>
> I filed a ticket for the issue of an "abstract operating period":
>
> http://trac.assembla.com/railML/ticket/187
>
> Kind regards...
> Susanne
>

Subject: Re: constraints for OperatingPeriod
Posted by Susanne Wunsch railML on Fri, 09 Nov 2012 14:51:28 GMT
View Forum Message <> Reply to Message

Dear Andreas and others,

Andreas Tanner <ata@ivu.de> writes:

> as far as railML 2.x is concerned, my suggestion was just to enhance
> the documentation.

Would you be so kind as to enhance the wiki documentation by the findings of this thread yourselves? If you need any assistance do not hesitate to ask for.

http://www.wiki.railml.org/index.php?title=TT:operatingPerio d

> The discussion on redundancy has some philosophical traits and
> therefore, absolute truth cannot be expected to be found.

That's something we should be aware of every time we develop railML.

> For railML 3.0, however, I would like to keep the discussion open and
> let us elaborate a model of validity with a more formally defined

> semantics that allows to address specific instances of trains from a
> timetable.

That is somewhat covered by the below mentioned Trac ticket.

> Am 09.11.2012 10:27, schrieb Susanne Wunsch:
>> I filed a ticket for the issue of an "abstract operating period":
>>
>> http://trac.assembla.com/railML/ticket/187

Kind regards...
Susanne

--
Susanne Wunsch
Schema Coordinator: railML.common

Subject: Re: constraints for OperatingPeriod
Posted by Joachim Rubröder railML on Wed, 06 Feb 2013 15:46:15 GMT
View Forum Message <> Reply to Message

Dear all,
I totally agree about the different kinds of operatingPeriods and the need for an 'abstract period'.

The second important point to consider is the operatingPeriod in the different stages of the planning process. Evolving from an 'abstract period' to a concrete 'bitmask'.

> Andreas Tanner <ata@ivu.de> writes:
>
> as far as railML 2.x is concerned, my suggestion was just to enhance
> the documentation.
>
> For railML 3.0, however, I would like to keep the discussion open and
> let us elaborate a model of validity with a more formally defined
> semantics that allows to address specific instances of trains from a
> timetable.

Many thanks for the discussion in this thread. I will be glad to implement such a model of validity in a future version of railML.

I therefore switched the ticket to railML 3.0:
http://trac.assembla.com/railML/ticket/187

Kind regards,
Joachim

Joachim RubrÃ¶der
Schema Coordinator: railML.timetable

--
----== posted via PHP Headliner ==----