Subject: Explizite Kennzeichnung von gelöschten Zügen und Zugausfällen
Posted by christian.wermelinger on Tue, 26 Feb 2013 18:06:45 GMT
View Forum Message <> Reply to Message

Hallo zusammen

Aktuell mÃ¼ssen beim Import von ZÃ¼gen Annahmen getroffen werden. Hierzu ein Beispiel:
1. Initialer Import von Zug 1234 Ã¼ber RailML-Schnittstelle. Resultat: Zug 1234 ist im Zielsystem
vorhanden.
2. Anschliessend wird ein weiterer RailML-Import gestartet (Zug 1234 ist nicht mehr in der RailML-
Importdatei einhalten).
Was nun? Soll der Zug gelÃ¶scht oder als annulliert gekennzeichnet werden? RailML liefert
diesbezÃ¼glich keine expliziten Informationen.

Soweit mir bekannt ist, erlaubt es RailML nicht, ungÃ¼ltige ZÃ¼ge (welche z.B. zu einem frÃ¼heren
Zeitpunkt irrtÃ¼mlicherweise ins Zielsystem importiert wurden) explizit als 'inaktiv' (bzw. gelÃ¶scht)
zu kennzeichnen. Weiter vermisse ich die MÃ¶glichkeit, Teile eines Zuges (trainParts) explizit als
'(Teil-)Ausfall' zu kennzeichnen ohne das dafÃ¼r zwingend eine neue Verkehrsperiode angelegt
werden muss. In beiden FÃ¤llen wÃ¤re es zudem wÃ¼nschenswert, bei Bedarf eine BegrÃ¼ndung
erfassen zu kÃ¶nnen.

Folgend versuche ich einen mÃ¶glichen LÃ¶sungsansatz zu skizzieren.

```
<railml>
 <train>
  <!-- Use-Case: Annulation technischer Art (optional) -->
  <disabled timestamp="2013-02-25T10:07:52,553">
      <!-- optional -->
   <reason code="INVALID_EXPORT" label="UngÃ¼ltiger Export"
remarks="UngÃ¼ltiger
Zug. Wurde am 1.2.2013 irrtÃ¼mlicherwise exportiert.">
  </disabled>
 </train>

 <trainPart>
  <operatingPeriodRef ref="op_123"></operatingPeriodRef>
  <!-- Use-Case: Fachlich getriebener (Teil-)ausfall eines Zuges
(optional) -->
  <cancellation timestamp="2013-02-25T10:07:52,553">
```

```
  <!-- schrÃ¤nkt die in 'trainPart\operatingPeriodRef' referenzierte
Verkehrsperiode
weiter ein (Bsp. Teilausfall) -->
  <restriction>
   <!--xsd:choice: restrictionOperatingPeriod ODER restrictionTimeSpan-->
   <restrictionOperatingPeriod ref="op_456"></restrictionOperatingPeriod>
   <restrictionTimeSpan endDate="2013-06-01" startDate="2012-07-15"/>
  </restriction>
  <!-- optional -->
  <reason code="CONSTRUCTION_AREA" label="Baustelle" remarks="Baustelle
auf
Strecke A-Z im Zeitraum vom 1.6.2013 bis 15.7.2013">
 </cancellation>
   </trainPart>
</railml>
```

'disabled' und 'cancellation' wurden bewusst als Elemente modelliert, da
damit Erweiterbarkeit
gegeben ist. Dies ist z.B. beim heute verwendeten Attribut
'trainPart@processStatus' nicht der Fall
(abgesehen davon, dass die AufzÃ¤hlung meines Erachtens in sich nicht
konsistent ist).

Wie erwÃ¤hnt handelt es sich hierbei nur um ein Vorschlag.
VerbesserungsvorschlÃ¤ge und
ErgÃ¤nzungen sind sehr willkommen.

Viele GrÃ¼sse,
Chris

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

Posted by Andreas Tanner on Wed, 27 Feb 2013 10:18:31 GMT

View Forum Message <> Reply to Message

Hallo Christian,
interessanter Punkt. Da hat railML noch Lücken.

Ich denke, man sollte trennen, ob man

a) Informationen aus der dispositiven Welt, oder
b) Änderungen eines Planes

übertragen will
"Zugausfall" würde ich a) zuordnen und hier eine Lösung anstreben, die

auch Verspätungsinformationen, Ersatzzüge usw. umfasst, vgl. auch die Diskussion news://news.railml.org:119/k4uoii$m50$1@sifa.ivi.fhg.de.

b) wäre für uns ebenfalls interessant: das importierende System liest Fahrplaninformationen und erstellt eine darauf basierende Dienstplanung. Jetzt ändert sich der Fahrplan (Fahrten gelöscht, ergänzt, verschoben, Laufweg geändert, Formation geändert...) und entsprechend müssen die Dienstpläne angepasst werden, wobei man so wenig wie möglich neu machen will.

Gruß Andreas

Am 26.02.2013 19:06, schrieb Christian Wermelinger:
> Hallo zusammen
>
> Aktuell müssen beim Import von Zügen Annahmen getroffen werden. Hierzu
> ein Beispiel:
> 1. Initialer Import von Zug 1234 über RailML-Schnittstelle. Resultat: Zug
> 1234 ist im Zielsystem
> vorhanden.
> 2. Anschliessend wird ein weiterer RailML-Import gestartet (Zug 1234 ist
> nicht mehr in der RailML-
> Importdatei einhalten).
> Was nun? Soll der Zug gelöscht oder als annulliert gekennzeichnet werden?
> RailML liefert
> diesbezüglich keine expliziten Informationen.
>
> Soweit mir bekannt ist, erlaubt es RailML nicht, ungültige Züge (welche
> z.B. zu einem früheren
> Zeitpunkt irrtümlicherweise ins Zielsystem importiert wurden) explizit
> als 'inaktiv' (bzw. gelöscht)
> zu kennzeichnen. Weiter vermisse ich die Möglichkeit, Teile eines Zuges
> (trainParts) explizit als
> '(Teil-)Ausfall' zu kennzeichnen ohne das dafür zwingend eine neue
> Verkehrsperiode angelegt
> werden muss. In beiden Fällen wäre es zudem wünschenswert, bei Bedarf
> eine Begründung
> erfassen zu können.
>
> Folgend versuche ich einen möglichen Lösungsansatz zu skizzieren.
>
> <railml>
> <train>
> <!-- Use-Case: Annulation technischer Art (optional) -->
> <disabled timestamp="2013-02-25T10:07:52,553">
> <!-- optional -->
> <reason code="INVALID_EXPORT" label="Ungültiger Export"
> remarks="Ungültiger

```
>  Zug. Wurde am 1.2.2013 irrtümlicherwise exportiert.">
>    </disabled>
>   </train>
>
>   <trainPart>
>    <operatingPeriodRef ref="op_123"></operatingPeriodRef>
>    <!-- Use-Case: Fachlich getriebener (Teil-)ausfall eines Zuges
>  (optional) -->
>    <cancellation timestamp="2013-02-25T10:07:52,553">
>     <!-- schränkt die in 'trainPart\operatingPeriodRef' referenzierte
>  Verkehrsperiode
>  weiter ein (Bsp. Teilausfall) -->
>     <restriction>
>      <!--xsd:choice: restrictionOperatingPeriod ODER restrictionTimeSpan-->
>      <restrictionOperatingPeriod ref="op_456"></restrictionOperatingPeriod>
>      <restrictionTimeSpan endDate="2013-06-01" startDate="2012-07-15"/>
>     </restriction>
>     <!-- optional -->
>     <reason code="CONSTRUCTION_AREA" label="Baustelle" remarks="Baustelle
>  auf
>  Strecke A-Z im Zeitraum vom 1.6.2013 bis 15.7.2013">
>    </cancellation>
>       </trainPart>
>  </railml>
>
>  'disabled' und 'cancellation' wurden bewusst als Elemente modelliert, da
>  damit Erweiterbarkeit
>  gegeben ist. Dies ist z.B. beim heute verwendeten Attribut
>  'trainPart@processStatus' nicht der Fall
>  (abgesehen davon, dass die Aufzählung meines Erachtens in sich nicht
>  konsistent ist).
>
>  Wie erwähnt handelt es sich hierbei nur um ein Vorschlag.
>  Verbesserungsvorschläge und
>  Ergänzungen sind sehr willkommen.
>
>  Viele Grüsse,
>  Chris
>
```

Posted by on Tue, 12 Mar 2013 18:23:10 GMT
View Forum Message <> Reply to Message

Hallo Christian, Andreas und alle Mitlesenden,

mit den „Änderungsinformationen“ oder „Historie“ im Allgemeinen ist das

eine schon mehrfach andiskutierte Sache in RailML.

Bisher war die Philosophie in RailML so, dass eine RailML-Datei nur einen aktuellen Datenstand absolut und vollständig abbildet: Sie bezieht sich nicht auf einen früheren Datenstand. Wenn sich Daten geändert haben, musste man alles mit RailML neu übertragen, auch das, was sich nicht geändert hat.

Zum Ausfall gekennzeichnete Züge sind nun eine besondere Form von „Änderungsinformationen“: Die Datei bezieht sich damit zumindest formell auf einen früheren Stand, nämlich den, zu dem der Zug noch nicht ausfallen sollte. Das ganze wird deutlich, wenn man bedenkt, dass ein Zug mehrfach ein- und ausgelegt werden könnte, d. h. erst soll er fahren, dann wieder nicht, dann doch usw. Und das in unterschiedlicher Abfolge für jeden möglichen Verkehrstag des Zuges.

Es ist mir klar, dass man konkret den Ausfall eines Zuges nicht so „bürokratisch“ sehen muss: Einfach wäre es vielleicht, eine operatingPeriodRef bzw. Verkehrstage-Bitmaske zu ergänzen, die aussagt, wann der Zug abweichend von der eigentlichen (bisher einzigen) operatingPeriodRef _nicht_ verkehrt - also an denen er irgendwann in der Vergangenheit mal verkehren sollte und nach aktuellem Erkenntnisstand mal jemand entschieden hat, dass er nicht verkehren soll.

Ich möchte nur darauf hinweisen, dass wir damit ein kleines Fässchen aufmachen, bei dem sich vielleicht herausstellt, dass es so schnell keinen Boden hat: Wenn wir einmal mit „Änderungsinformationen“ anfangen, werden wir sie vmtl. so schnell nicht wieder los. Der Anwender kennt ja den Werdegang von RailML nicht und akzeptiert auch nicht unbedingt einen willkürlichen Zwischenstand. Es könnte daher sinnvoll sein „wenn schon, dann richtig“, also quasi bei _jedem_ Element in RailML (per Vererbung) solche Informationen wie „Datensatz/Element gültig von ... bis ...“ einzuführen. Dann kann man „genau genommen genau“ erkennen, ob ein Element - auch Zug, Zugteil usw. - noch „gültig“ ist und von wann bis wann das jemand mal so geplant hatte.

Auch diese vermeintliche „Ideallösung“ wäre u. U. noch nicht der Boden des Fasses, wenn man bedenkt, dass im Falle von Zügen eventuell noch wichtig sein könnte, auf welcher Planungsebene er ein- und ausgelegt wurde. War er schon beim Infrastrukturbetreiber bestellt und muss daher abbestellt werden? Oder war es „nur so eine Idee“, die zu keinem Zeitpunkt offiziellen Status erlangte? Und das nach Verkehrstagen unterschieden: Zug ist im Juli bis September bestellt worden, im Juli soll er immer noch fahren, im August ist er bereits abbestellt, im September ist er noch abzubestellen...

- - -
Letztendlich kann man es auch anders aufrollen: Bei Datenübertragung in

einem komplexen Prozess muss üblicher Weise irgendwo ein Änderungsabgleich stattfinden. Dies kann beim Export _vor_ RailML stattfinden - RailML überträgt dann Änderungsinformationen und bezieht sich auf einen früheren Export - oder es kann beim Import _nach_ RailML stattfinden, d. h. beim „mergen“ (=soll ein Anglizismus sein - bitte englisch „auslesen“) der RailML-Daten mit den aktuell im System vorhandenen Daten.

Es scheint, dass beide Wege gleichwertig sind. Die bisherige RailML-Philosophie war „keine Änderungsinformationen“, d. h. „mergen“ beim Einlesen. Das Einlesen ist ohnehin ein komplexerer Prozess als das Rausschreiben, wegen der zusätzlich notwendigen Datenintegritätsprüfungen.

Zu bedenken ist immer auch, dass u. U. verschiedene Datenquellen in Betracht kommen: Ein „mergen“ beim Einlesen kann auch Daten zusammenführen, die aus verschiedenen Quellen kommen. Ein Änderungsexport hingegen würde sich immer auf die zuvor aus dem eigenen System exportierten Daten beziehen, d. h. hier ist im Gesamtprozess kein Informationsgewinn möglich.

Wir (iRFP) haben uns dieser Lesart angepasst, d. h. wir können beim Einlesen „mergen“, können aber derzeit keinen Änderungsexport anbieten. Wir würden aus Aufwandsgründen in absehbarer Zeit nicht beides anbieten, zumal wie gesagt derzeit kein Mehrwert erkennbar ist. Insofern würde ich das von unserer Seite erst einmal zurückhaltend betrachten wollen.

Ich verstehe, dass die Situation vielleicht anders aussieht, wenn zwei nicht gleichwertige Systeme Daten austauschen - d. h. wenn beim Einlesen aus irgendwelchen Gründen weniger Prozesskapazität vorliegt als beim Rausschreiben. Das wäre dann aber vielleicht ein Fall, der nicht mehr im allgemeingültigen RailML auftauchen muss - eher ein individueller Aufsatz oder eine bilaterale Lösung.

Viele Grüße,
Dirk.

Subject: Re: Explizite Kennzeichnung von gelöschten Zügen und Zugausfällen
Posted by christian.wermelinger on Mon, 16 Sep 2013 15:09:36 GMT
View Forum Message <> Reply to Message

Hallo zusammen

Erstmal ein DankeschÃ¶n an Andreas und Dirk fÃ¼r den wertvollen Input.

Andreas Tanner wrote:
> Ich denke, man sollte trennen, ob man
> a) Informationen aus der dispositiven Welt, oder

Andreas' Vorschlag, Informationen aus der dispositiven Welt und PlanÃ¤nderungen konsequent zu trennen, erachte ich als sinnvoll. GrundsÃ¤tzlich muss ich jedoch Dirk beipflichten.

> Bisher war die Philosophie in RailML so, dass eine RailML-Datei nur einen
> aktuellen Datenstand absolut und vollstÃ¤ndig abbildet: Sie bezieht sich
> nicht auf einen frÃ¼heren Datenstand.

Meines Erachtens sollte an dieser Philosophie festgehalten werden.

> Ich mÃ¶chte nur darauf hinweisen, dass wir damit ein kleines FÃ¤sschen
> aufmachen, bei dem sich vielleicht herausstellt, dass es so schnell keinen

werden
> wir sie vmtl. so schnell nicht wieder los.

kleines FÃ¤sschen, sondern ein ziemlich grosses Fass aufmachen wÃ¼rden! Dirk ist ja bereits auf mÃ¶gliche Konsequenzen eingegangen.

> einzufÃ¼hren.

Von einer halbfertigen, nicht zu Ende gedachten LÃ¶sung profitiert schlussendlich niemand. Andererseits fÃ¼hrt der Einsatz zeitabhÃ¤ngiger Attribute zu einer stark erhÃ¶hten KomplexitÃ¤t und blÃ¤ht die Daten weiter auf. Dies bringt insbesondere dann keinen Mehrwert, wenn dieser KomplexitÃ¤tsgrad gar nicht gefordert ist (z.B. weil das Zielsystem gar keine Historie bzw. zeitabhÃ¤ngigen Attribute kennt oder benÃ¶tigt).

> Letztendlich kann man es auch anders aufrollen: Bei DatenÃ¼bertragung in

> stattfinden. Dies kann beim Export _vor_ RailML stattfinden - RailML

frÃ¼heren
> Export - oder es kann beim Import _nach_ RailML stattfinden, d. h. beim

der
> RailML-Daten mit den aktuell im System vorhandenen Daten.
>
> Es scheint, dass beide Wege gleichwertig sind. Die bisherige

> Einlesen.

Wie bereits erwÃ¤hnt, erachte ich es als sinvoll weiterhin an der

Wir handeln dies nun auch so. Beim Import von FahrplÃ¤nen wird der bereits
im System vorhandene Datenbestand mit den zu importierenden RailML-Daten
abgeglichen. FÃ¼r den Datenabgleich bieten wir folgende Import-Optionen an:
1. neue ZÃ¼ge importieren: ja/nein
2. bestehende ZÃ¼ge aktualisieren: ja/nein
3. nicht mehr vorhanden ZÃ¼ge lÃ¶schen: ja/nein

Damit sind fÃ¼r uns zumindest die wichtigsten Use-Cases im Zusammenhang
mit PlanÃ¤nderungen abgedeckt.

Viele GrÃ¼sse,
Christian

Dirk BrÃ¤uer wrote:
>
> Hallo Christian, Andreas und alle Mitlesenden,
>

Allgemeinen ist
das
> eine schon mehrfach andiskutierte Sache in RailML.
>
> Bisher war die Philosophie in RailML so, dass eine RailML-Datei nur einen
> aktuellen Datenstand absolut und vollstÃ¤ndig abbildet: Sie bezieht sich
> nicht auf einen frÃ¼heren Datenstand. Wenn sich Daten geÃ¤ndert haben,
> musste man alles mit RailML neu Ã¼bertragen, auch das, was sich nicht
> geÃ¤ndert hat.
>
> Zum Ausfall gekennzeichnete ZÃ¼ge sind nun eine besondere Form von

formell
> auf einen frÃ¼heren Stand, nÃ¤mlich den, zu dem der Zug noch nicht ausfallen

> sollte. Das ganze wird deutlich, wenn man bedenkt, dass ein Zug mehrfach
> ein- und ausgelegt werden kÃ¶nnte, d. h. erst soll er fahren, dann wieder
> nicht, dann doch usw. Und das in unterschiedlicher Abfolge fÃ¼r jeden
> mÃ¶glichen Verkehrstag des Zuges.
>
> Es ist mir klar, dass man konkret den Ausfall eines Zuges nicht so

> operatingPeriodRef bzw. Verkehrstage-Bitmaske zu ergÃ¤nzen, die aussagt,

> wann der Zug abweichend von der eigentlichen (bisher einzigen)
> operatingPeriodRef _nicht_ verkehrt - also an denen er irgendwann in der
> Vergangenheit mal verkehren sollte und nach aktuellem Erkenntnisstand mal
> jemand entschieden hat, dass er nicht verkehren soll.
>
> Ich mÃ¶chte nur darauf hinweisen, dass wir damit ein kleines FÃ¤sschen
> aufmachen, bei dem sich vielleicht herausstellt, dass es so schnell keinen

werden
> wir sie vmtl. so schnell nicht wieder los. Der Anwender kennt ja den
> Werdegang von RailML nicht und akzeptiert auch nicht unbedingt einen

Element

> jemand mal so geplant hatte.
>

Boden
des
> Fasses, wenn man bedenkt, dass im Falle von ZÃ¼gen eventuell noch wichtig
> sein kÃ¶nnte, auf welcher Planungsebene er ein- und ausgelegt wurde. War er
> schon beim Infrastrukturbetreiber bestellt und muss daher abbestellt

> offiziellen Status erlangte? Und das nach Verkehrstagen unterschieden: Zug
> ist im Juli bis September bestellt worden, im Juli soll er immer noch
> fahren, im August ist er bereits abbestellt, im September ist er noch
> abzubestellen...
>
> - - -
> Letztendlich kann man es auch anders aufrollen: Bei DatenÃ¼bertragung in

> stattfinden. Dies kann beim Export _vor_ RailML stattfinden - RailML

frÃ¼heren
> Export - oder es kann beim Import _nach_ RailML stattfinden, d. h. beim

der
> RailML-Daten mit den aktuell im System vorhandenen Daten.
>
> Es scheint, dass beide Wege gleichwertig sind. Die bisherige

> Einlesen. Das Einlesen ist ohnehin ein komplexerer Prozess als das
> Rausschreiben, wegen der zusÃ¤tzlich notwendigen DatenintegritÃ¤tsprÃ¼fungen.

>
> Zu bedenken ist immer auch, dass u. U. verschiedene Datenquellen in

> hingegen wÃ¼rde sich immer auf die zuvor aus dem eigenen System
> exportierten Daten beziehen, d. h. hier ist im Gesamtprozess kein
> Informationsgewinn mÃ¶glich.
>
> Wir (iRFP) haben uns dieser Lesart angepasst, d. h. wir kÃ¶nnen beim

anbieten.
> Wir wÃ¼rden aus AufwandsgrÃ¼nden in absehbarer Zeit nicht beides anbieten,
> zumal wie gesagt derzeit kein Mehrwert erkennbar ist. Insofern wÃ¼rde ich
> das von unserer Seite erst einmal zurÃ¼ckhaltend betrachten wollen.
>
> Ich verstehe, dass die Situation vielleicht anders aussieht, wenn zwei
> nicht gleichwertige Systeme Daten austauschen - d. h. wenn beim Einlesen
> aus irgendwelchen GrÃ¼nden weniger ProzesskapazitÃ¤t vorliegt als beim
> Rausschreiben. Das wÃ¤re dann aber vielleicht ein Fall, der nicht mehr im
> allgemeingÃ¼ltigen RailML auftauchen muss - eher ein individueller Aufsatz
> oder eine bilaterale LÃ¶sung.
>

> Dirk.
>
>

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

Posted by Andreas Tanner on Tue, 24 Sep 2013 09:00:57 GMT

View Forum Message <> Reply to Message

Hallo allerseits,

auf der vergangenen railML-Konferenz habe ich einen Vortrag zum Thema
gehalten:
http://documents.railml.org/events/slides/2013-09-18_ivu_tan ner-differentialdataexchange.pdf.

Ich würde gern mit einer Gruppe von Interessierten eine
Standarderweiterung erarbeiten, die die angefragten Anwendungsfälle abdeckt.

Gruß Andreas

Am 16.09.2013 17:09, schrieb Christian Wermelinger:
> Hallo zusammen
>
> Erstmal ein DankeschÃ¶n an Andreas und Dirk fÃ¼r den wertvollen Input.
>
> Andreas Tanner wrote:
>> Ich denke, man sollte trennen, ob man
>> a) Informationen aus der dispositiven Welt, oder
>> b) Ã„nderungen eines Planes
>
> Andreas' Vorschlag, Informationen aus der dispositiven Welt und
> PlanÃ¤nderungen konsequent zu trennen, erachte ich als sinnvoll.
> GrundsÃ¤tzlich muss ich jedoch Dirk beipflichten.
>
>> Bisher war die Philosophie in RailML so, dass eine RailML-Datei nur einen
>> aktuellen Datenstand absolut und vollstÃ¤ndig abbildet: Sie bezieht sich
>> nicht auf einen frÃ¼heren Datenstand.
>
> Meines Erachtens sollte an dieser Philosophie festgehalten werden.
>
>> Ich mÃ¶chte nur darauf hinweisen, dass wir damit ein kleines FÃ¤sschen
>> aufmachen, bei dem sich vielleicht herausstellt, dass es so schnell keinen
>> Boden hat: Wenn wir einmal mit â€žÃ„nderungsinformationenâ€œ anfangen,
> werden
>> wir sie vmtl. so schnell nicht wieder los.
>
> Ich denke dass wir mit der von mir vorgeschlagenen Ã„nderung nicht nur ein
> kleines FÃ¤sschen, sondern ein ziemlich grosses Fass aufmachen wÃ¼rden!
> Dirk ist ja bereits auf mÃ¶gliche Konsequenzen eingegangen.
>
>> Es kÃ¶nnte daher sinnvoll sein â€žwenn schon,
>> dann richtigâ€œ, also quasi bei _jedem_ Element in RailML (per Vererbung)
>> solche Informationen wie â€žDatensatz/Element gÃ¼ltig von ... bis ...â€œ
>> einzufÃ¼hren.
>
> Von einer halbfertigen, nicht zu Ende gedachten LÃ¶sung profitiert
> schlussendlich niemand. Andererseits fÃ¼hrt der Einsatz zeitabhÃ¤ngiger
> Attribute zu einer stark erhÃ¶hten KomplexitÃ¤t und blÃ¤ht die Daten
> weiter auf. Dies bringt insbesondere dann keinen Mehrwert, wenn dieser
> KomplexitÃ¤tsgrad gar nicht gefordert ist (z.B. weil das Zielsystem gar
> keine Historie bzw. zeitabhÃ¤ngigen Attribute kennt oder benÃ¶tigt).
>
>> Letztendlich kann man es auch anders aufrollen: Bei DatenÃ¼bertragung in
>> einem komplexen Prozess muss Ã¼blicher Weise irgendwo ein Ã„nderungsabgleich
>
>> stattfinden. Dies kann beim Export _vor_ RailML stattfinden - RailML
>> Ã¼bertrÃ¤gt dann Ã„nderungsinformationen und bezieht sich auf einen
> frÃ¼heren

>> Export - oder es kann beim Import _nach_ RailML stattfinden, d. h. beim
>> â€žmergenâ€œ (=soll ein Anglizismus sein - bitte englisch
â€žauslesenâ€œ)
> der
>> RailML-Daten mit den aktuell im System vorhandenen Daten.
>>
>> Es scheint, dass beide Wege gleichwertig sind. Die bisherige
>> RailML-Philosophie war â€žkeine Ã„nderungsinformationenâ€œ, d. h.
> â€žmergenâ€œ beim
>> Einlesen.
>
> Wie bereits erwÃ¤hnt, erachte ich es als sinvoll weiterhin an der
> aktuellen RailML-Philosophie festzuhalten und Ã„nderungen beim Import bzw.
> â€žmergenâ€œ zu erfassen.
>
> Wir handeln dies nun auch so. Beim Import von FahrplÃ¤nen wird der bereits
> im System vorhandene Datenbestand mit den zu importierenden RailML-Daten
> abgeglichen. FÃ¼r den Datenabgleich bieten wir folgende Import-Optionen an:
> 1. neue ZÃ¼ge importieren: ja/nein
> 2. bestehende ZÃ¼ge aktualisieren: ja/nein
> 3. nicht mehr vorhanden ZÃ¼ge lÃ¶schen: ja/nein
>
> Damit sind fÃ¼r uns zumindest die wichtigsten Use-Cases im Zusammenhang
> mit PlanÃ¤nderungen abgedeckt.
>
> Viele GrÃ¼sse,
> Christian
>
>
>
> Dirk BrÃ¤uer wrote:
>>
>> Hallo Christian, Andreas und alle Mitlesenden,
>>
>> mit den â€žÃ„nderungsinformationenâ€œ oder â€žHistorieâ€œ im
Allgemeinen ist
> das
>> eine schon mehrfach andiskutierte Sache in RailML.
>>
>> Bisher war die Philosophie in RailML so, dass eine RailML-Datei nur einen
>> aktuellen Datenstand absolut und vollstÃ¤ndig abbildet: Sie bezieht sich
>> nicht auf einen frÃ¼heren Datenstand. Wenn sich Daten geÃ¤ndert haben,
>> musste man alles mit RailML neu Ã¼bertragen, auch das, was sich nicht
>> geÃ¤ndert hat.
>>
>> Zum Ausfall gekennzeichnete ZÃ¼ge sind nun eine besondere Form von
>> â€žÃ„nderungsinformationenâ€œ: Die Datei bezieht sich damit zumindest
> formell

>> auf einen frÃ¼heren Stand, nÃ¤mlich den, zu dem der Zug noch nicht ausfallen
>
>> sollte. Das ganze wird deutlich, wenn man bedenkt, dass ein Zug mehrfach
>> ein- und ausgelegt werden kÃ¶nnte, d. h. erst soll er fahren, dann wieder
>> nicht, dann doch usw. Und das in unterschiedlicher Abfolge fÃ¼r jeden
>> mÃ¶glichen Verkehrstag des Zuges.
>>
>> Es ist mir klar, dass man konkret den Ausfall eines Zuges nicht so
>> â€žbÃ¼rokratischâ€œ sehen muss: Einfach wÃ¤re es vielleicht, eine
>> operatingPeriodRef bzw. Verkehrstage-Bitmaske zu ergÃ¤nzen, die aussagt,
>> wann der Zug abweichend von der eigentlichen (bisher einzigen)
>> operatingPeriodRef _nicht_ verkehrt - also an denen er irgendwann in der
>> Vergangenheit mal verkehren sollte und nach aktuellem Erkenntnisstand mal
>> jemand entschieden hat, dass er nicht verkehren soll.
>>
>> Ich mÃ¶chte nur darauf hinweisen, dass wir damit ein kleines FÃ¤sschen
>> aufmachen, bei dem sich vielleicht herausstellt, dass es so schnell keinen
>> Boden hat: Wenn wir einmal mit â€žÃ„nderungsinformationenâ€œ anfangen,
> werden
>> wir sie vmtl. so schnell nicht wieder los. Der Anwender kennt ja den
>> Werdegang von RailML nicht und akzeptiert auch nicht unbedingt einen
>> willkÃ¼rlichen Zwischenstand. Es kÃ¶nnte daher sinnvoll sein â€žwenn schon,
>> dann richtigâ€œ, also quasi bei _jedem_ Element in RailML (per Vererbung)
>> solche Informationen wie â€žDatensatz/Element gÃ¼ltig von ... bis ...â€œ
>> einzufÃ¼hren. Dann kann man â€žgenau genommen genauâ€œ erkennen, ob
ein
> Element
>> - auch Zug, Zugteil usw. - noch â€žgÃ¼ltigâ€œ ist und von wann bis wann das
>> jemand mal so geplant hatte.
>>
>> Auch diese vermeintliche â€žIdeallÃ¶sungâ€œ wÃ¤re u. U. noch nicht der
Boden
> des
>> Fasses, wenn man bedenkt, dass im Falle von ZÃ¼gen eventuell noch wichtig
>> sein kÃ¶nnte, auf welcher Planungsebene er ein- und ausgelegt wurde. War er
>> schon beim Infrastrukturbetreiber bestellt und muss daher abbestellt
>> werden? Oder war es â€žnur so eine Ideeâ€œ, die zu keinem Zeitpunkt
>> offiziellen Status erlangte? Und das nach Verkehrstagen unterschieden: Zug
>> ist im Juli bis September bestellt worden, im Juli soll er immer noch
>> fahren, im August ist er bereits abbestellt, im September ist er noch
>> abzubestellen...
>>
>> - - -
>> Letztendlich kann man es auch anders aufrollen: Bei DatenÃ¼bertragung in
>> einem komplexen Prozess muss Ã¼blicher Weise irgendwo ein Ã„nderungsabgleich
>
>> stattfinden. Dies kann beim Export _vor_ RailML stattfinden - RailML
>> Ã¼bertrÃ¤gt dann Ã„nderungsinformationen und bezieht sich auf einen

> frÃ¼heren
>> Export - oder es kann beim Import _nach_ RailML stattfinden, d. h. beim
>> â€žmergenâ€œ (=soll ein Anglizismus sein - bitte englisch
â€žauslesenâ€œ)
> der
>> RailML-Daten mit den aktuell im System vorhandenen Daten.
>>
>> Es scheint, dass beide Wege gleichwertig sind. Die bisherige
>> RailML-Philosophie war â€žkeine Ã„nderungsinformationenâ€œ, d. h.
> â€žmergenâ€œ beim
>> Einlesen. Das Einlesen ist ohnehin ein komplexerer Prozess als das
>> Rausschreiben, wegen der zusÃ¤tzlich notwendigen DatenintegritÃ¤tsprÃ¼fungen.
>>
>> Zu bedenken ist immer auch, dass u. U. verschiedene Datenquellen in
>> Betracht kommen: Ein â€žmergenâ€œ beim Einlesen kann auch Daten
>> zusammenfÃ¼hren, die aus verschiedenen Quellen kommen. Ein Ã„nderungsexport
>> hingegen wÃ¼rde sich immer auf die zuvor aus dem eigenen System
>> exportierten Daten beziehen, d. h. hier ist im Gesamtprozess kein
>> Informationsgewinn mÃ¶glich.
>>
>> Wir (iRFP) haben uns dieser Lesart angepasst, d. h. wir kÃ¶nnen beim
>> Einlesen â€žmergenâ€œ, kÃ¶nnen aber derzeit keinen Ã„nderungsexport
> anbieten.
>> Wir wÃ¼rden aus AufwandsgrÃ¼nden in absehbarer Zeit nicht beides anbieten,
>> zumal wie gesagt derzeit kein Mehrwert erkennbar ist. Insofern wÃ¼rde ich
>> das von unserer Seite erst einmal zurÃ¼ckhaltend betrachten wollen.
>>
>> Ich verstehe, dass die Situation vielleicht anders aussieht, wenn zwei
>> nicht gleichwertige Systeme Daten austauschen - d. h. wenn beim Einlesen
>> aus irgendwelchen GrÃ¼nden weniger ProzesskapazitÃ¤t vorliegt als beim
>> Rausschreiben. Das wÃ¤re dann aber vielleicht ein Fall, der nicht mehr im
>> allgemeingÃ¼ltigen RailML auftauchen muss - eher ein individueller Aufsatz
>> oder eine bilaterale LÃ¶sung.
>>
>> Viele GrÃ¼ÃŸe,
>> Dirk.
>>
>>
>
>
>

Tickets #188 und #247
Posted by on Fri, 18 Sep 2015 08:14:03 GMT
View Forum Message <> Reply to Message

Hallo allerseits,

Ticket #247 wurde als für railML 2.3 geschlossen und das damit zusammenhängende Problem damit als gelöst deklariert.

Wir haben in den letzten Jahren schon oft über die Problematik "Änderungsübertragung vs. Gesamtübertragung" und damit im Zusammenhang stehende Aspekte gesprochen. Bisher war die dabei unter'm Strich herauskommende Lesart immer "erfordert weitgehendere Lösungen, die erst mit 3.x und Refactoring möglich werden", u. a. nachzulesen beim Diskussions-Thread 26.02.2013..24.09.2013 [1].

Insofern sollte Einigkeit darüber bestehen, dass die Schließung von Ticket #247 bei Weitem keine Lösung ist, sondern eher ein Provisorium, über das man geteilter Meinung sein kann. Möglicherweise wäre hier eine Verwendung von any-Attributen die bessere Wahl gewesen, um auszudrücken, dass ein allgemeingültiger railML-Konsens noch nicht erreicht ist.

Ich möchte anregen, beim Ticket #247 einen Querverweis auf Ticket #188 (gleiches Thema bei railML 3.0) anzubringen und zu vermerken, dass es sich um eine provisorische, nicht allgemeingültige Lösung handelt.

Viele Grüße,
Dirk Bräuer.

[1] news://news.railml.org:119/l176vgtbv1@sifa.ivi.fhg.de

P.S.:
Gleichzeitig sollte für das provisorische "Löschkennzeichen" eines <trainPart>s von railML 2.3 im Wiki durch den Einführer oder Koordinator festgehalten werden,
  a) ob ein <trainPart> mit "Löschkennzeichen" immer an allen seinen Verkehrstagen auf seinem gesamten Laufweg ausfällt und er demzufolge keine /operatingPerodRef/ und keine <ocpTT>s haben darf bzw. dass diese zu ignorieren sind oder
  b) ob ein <trainPart> mit "Löschkennzeichen" nur an dem konkret genannten Laufweg und Verkehrstagen ausfällt, d. h. eventuell im Zielsystem existierende weitere Verkehrstage / Laufwegteile unverändert erhalten bleiben.

Da im Ticket #247 schon erwähnt, dass die Bezugsbasis in railML derzeit nicht definiert ist, sollte es auf (a) hinauslaufen (was hiermit als mein konkreter Vorschlag gelten kann).

Wenn wir das undefiniert belassen, werden wir ein großes Problem auf die Praxis abwälzen mit der Folge, dass die meisten Anwendungen das "Löschkennzeichen" ablehnen oder ignorieren. Dieses Dilemma verdeutlicht schon etwas, wo das Unbefriedigende an der 2.3er Lösung liegt.

Tickets #188 und #247
Posted by Philip Wobst on Tue, 29 Sep 2015 11:34:05 GMT
View Forum Message <> Reply to Message

Hallo Dirk,

danke für die Rückmeldung - dein Einwand ist durchaus gerechtfertigt.
Das Ticket #247 entspricht nicht der tatsächlich durchgeführten Änderung
in 2.3.
Ich möchte vorschlagen, dass ich dies entsprechend aktualisiere und dann
hier im Forum um Rückmeldung bitte.

EN: There has been a discussion regarding the TRAC ticket #247 changes
for railML 2.3. The ticket will be updated to better show the actual
changes (also in English) so that feedback can be given in the forum.

Best regards /Gruß aus Hannover,

Philip Wobst

Am 18.09.2015 um 10:14 schrieb Dirk Bräuer:
> Hallo allerseits,
>
> Ticket #247 wurde als für railML 2.3 geschlossen und das damit
> zusammenhängende Problem damit als gelöst deklariert.
>
> Wir haben in den letzten Jahren schon oft über die Problematik
> "Änderungsübertragung vs. Gesamtübertragung" und damit im Zusammenhang
> stehende Aspekte gesprochen. Bisher war die dabei unter'm Strich
> herauskommende Lesart immer "erfordert weitgehendere Lösungen, die erst
> mit 3.x und Refactoring möglich werden", u. a. nachzulesen beim
> Diskussions-Thread 26.02.2013..24.09.2013 [1].
>
> Insofern sollte Einigkeit darüber bestehen, dass die Schließung von
> Ticket #247 bei Weitem keine Lösung ist, sondern eher ein Provisorium,
> über das man geteilter Meinung sein kann. Möglicherweise wäre hier eine
> Verwendung von any-Attributen die bessere Wahl gewesen, um auszudrücken,
> dass ein allgemeingültiger railML-Konsens noch nicht erreicht ist.
>
> Ich möchte anregen, beim Ticket #247 einen Querverweis auf Ticket #188
> (gleiches Thema bei railML 3.0) anzubringen und zu vermerken, dass es
> sich um eine provisorische, nicht allgemeingültige Lösung handelt.
>
> Viele Grüße,
> Dirk Bräuer.
>
> [1] news://news.railml.org:119/l176vgtbv1@sifa.ivi.fhg.de
>

> P.S.:
> Gleichzeitig sollte für das provisorische "Löschkennzeichen" eines
> <trainPart>s von railML 2.3 im Wiki durch den Einführer oder Koordinator
> festgehalten werden,
> a) ob ein <trainPart> mit "Löschkennzeichen" immer an allen seinen
> Verkehrstagen auf seinem gesamten Laufweg ausfällt und er demzufolge
> keine /operatingPerodRef/ und keine <ocpTT>s haben darf bzw. dass diese
> zu ignorieren sind oder
> b) ob ein <trainPart> mit "Löschkennzeichen" nur an dem konkret
> genannten Laufweg und Verkehrstagen ausfällt, d. h. eventuell im
> Zielsystem existierende weitere Verkehrstage / Laufwegteile unverändert
> erhalten bleiben.
>
> Da im Ticket #247 schon erwähnt, dass die Bezugsbasis in railML derzeit
> nicht definiert ist, sollte es auf (a) hinauslaufen (was hiermit als
> mein konkreter Vorschlag gelten kann).
>
> Wenn wir das undefiniert belassen, werden wir ein großes Problem auf die
> Praxis abwälzen mit der Folge, dass die meisten Anwendungen das
> "Löschkennzeichen" ablehnen oder ignorieren. Dieses Dilemma verdeutlicht
> schon etwas, wo das Unbefriedigende an der 2.3er Lösung liegt.