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.