Subject: XSD design patterns (was: Re: railML 3.x: Data Modelling Patterns)
Posted by Vasco Paul Kolmorgen on Thu, 10 Jan 2019 10:39:50 GMT
View Forum Message <> Reply to Message

Dear all,

I'm moving the topic of XSD design patterns from the thread "railML 3.x:
Data Modelling Patterns"
(https://www.railML.org/forum/index.php?t=msg&th=573& goto=2057&#msg_2057)
into this new thread to make discussion better visible and understandable.

Am 28.12.2018 um 21:39 schrieb Thomas Nygreen:
> The xsd design pattern was not mentioned in the
> presentation, but I will include a comment here anyway.
> Mostly, it currently looks like a Garden of Eden pattern,
> but the local and global element names do not match. Mostly,
> it is just a matter of capitalization, but there are also a
> lot of cases where the difference is more substantial
> (mostly in IL, but also in IS). And, as XML is
> case-sensitive even differences in capitalization matter.
> Also, there is no point in generating global elements for
> types that are never used in any local elements such as
> abstract types. The pattern should be to define elements
> also globally, not to generate elements for all types.
> Currently there are some global elements that cannot be used
> in valid xml (because they implement abstract types), some
> that should not be used (because their names are not similar
> to any local elements), and no local elements are really
> defined globally (case-sensitivity + other differences).

A compressed description of the individual modelling variants can be
found on an Oracle page:
https://www.oracle.com/technetwork/java/design-patterns-1421 38.html

In today's conference call of the Timetable developers the topic was
discussed. After a - still short - discussion of the topic, the
following level of discussion is emerging:
- The railML 3 schema should be aligned with one of the patterns, but
this should then be consistently adhered to.
- There is a preference for the variants "Garden of Eden" or "Venetian
Blind", a preference will be communicated by the individual developers
until January 16, 2019.

The consequence of the implementation of the proposals would be that the
railML 3.1 RC 2 (expected to be published on January 29, 2019; see also
https://www.railml.org/en/public-relations/news/reader/delay
ed-3.1-release-and-updated-license.html)
will look significantly different, but the example file will remain

approximately the same.

We are very interested in further opinions and hints in the railML 3 context and railway area, but ask for quick feedback.

Thank you and kind regards,
--
Vasco Paul Kolmorgen - Governance Coordinator
railML.org (Registry of Associations: VR 5750)
Phone railML.org: +49 351 47582911
Altplauen 19h; 01187 Dresden; Germany www.railML.org

Subject: Re: XSD design patterns (was: Re: railML 3.x: Data Modelling Patterns)
Posted by Thomas Nygreen JBD on Thu, 10 Jan 2019 18:04:33 GMT
View Forum Message <> Reply to Message

I find it useful to be able to create files that do not start at the railML root, and will still validate against the XSD and be read by a schema-aware parser. This is a benefit of using a pattern with global elements (such as Garden of Eden). Without global element definitions, one has to include all ancestor elements (which in turn can force inclusion of required subelements of these ancestors). However, even with global element definitions, such partial files require the use of UUIDs if referencing elements that are not included in the file.

A couple of use cases for such partial files:
Splitting data into separate files (e.g. separate IS, IL, TT, RS files). With global elements these files can start at their domain element, with a separate file starting at <railML> referencing them all by UUID. Without global elements, each file must start at <railML>, but nothing else changes.Sending an update for one specific object as a snippet. With global elements + UUIDs it is possible to create a valid file containing just the desired object(s). Without global elements this must be handled via methods outside of railML (e.g. diff files in version control systems).

Apart from the ability to start at any place in the tree, I do not really se an additional benefit in our case from global element definitions compared to just having global type definitions (Venetian Blind). Element definitions can easily be reused through reusing types.

Subject: Re: XSD design patterns (was: Re: railML 3.x: Data Modelling Patterns)
Posted by on Fri, 11 Jan 2019 16:01:40 GMT
View Forum Message <> Reply to Message

Hello, Thomas,

thank you for clarifying which use cases can be implemented with global elements. In particular, I see potential for transferring updates for individual elements instead of the entire railML document. The support of this use case is also within the railML-TT developers a goal for a future railML3-TT.

Christian Rahmig's design principle of keeping the XSD structure as flat as possible in order to transfer subsets of elements without unnecessary ballast goes in the same direction. I see global elements as design principle as a more consistent solution to this problem.

From my point of view, the greater flexibility in reusing the schema and the fact that railML3 apparently already follows the pattern "Garden of Eden" more than "Venetian Blind" (in contrast to railML2.x) speak in favor of a consequnt adaptation of railML3 to the design pattern "Garden of Eden".
The (subjectively) better comprehensibility of the "Venetian Blind" pattern speaks against this. In addition, as a developer of railML interface software, I do not (yet) see any need for the transfer of subsets of the railML schema, all our current applications are based on the transfer of complete railML files.

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

Subject: Re: XSD design patterns (was: Re: railML 3.x: Data Modelling Patterns)
Posted by Thomas Nygreen JBD on Fri, 11 Jan 2019 17:08:54 GMT
View Forum Message <> Reply to Message

Hi Christian,

Christian Rößiger wrote on Fri, 11 January 2019 17:01
From my point of view, the greater flexibility in reusing the schema and the fact that railML3 apparently already follows the pattern "Garden of Eden" more than "Venetian Blind" (in contrast to railML2.x) speak in favor of a consequnt adaptation of railML3 to the design pattern "Garden of Eden".

I have no experience using Enterprise Architect, but from what I read in the manual, the change away from global elements requires only deselecting one checkbox when exporting. So changing to Venetian Blind is quite effortless. On the other hand, Garden of Eden would require renaming of all existing complexTypes to work according to the pattern's purpose. For many types this is just a matter of different capitalisation, which could maybe be automated, but hundreds of types have other differences as well. Normally, Garden of Eden schemas use @ref instead of @type on elements within types. This ensures equal naming of local and global definitions. When using the option in Enterprise Architect to export global elements for complexTypes to achieve a similar pattern, care must be taken to align type names and element names, and each unique element name must have its own global type of the same name.

Regarding the reuse of elements: I know that this is always listed as a strength of Garden of

Eden, but what does it really add compared to extending types from a Venetian Blind schema? Maybe I am missing something. I hope someone can enlighten me. (For Salami Slice this is different, since it does not have global types.)

Quote:
The (subjectively) better comprehensibility of the "Venetian Blind" pattern speaks against this. In addition, as a developer of railML interface software, I do not (yet) see any need for the transfer of subsets of the railML schema, all our current applications are based on the transfer of complete railML files.

The issue that is normally raised against Garden of Eden is that it can be difficult to find the (intended) root element. (But that is a consequence of allowing all elements to be the root.) Apart from that, the two patterns are so similar that I do not find one of them to be more readable or comprehensible than the other. Garden of Eden is just slightly more verbose when viewing the schema as plain text. Although, when listing global definitions in an XML tool, the list will be twice as long for GoE as for VB.

As mentioned, the two patterns are very similar, makimg it quite easy to mix them. Even in a generally Venetian Blind schema, key elements can be defined globally to allow using them as the root. And in a generally Garden of Eden, it is posible to skip global definitions for some elements. For instance, in the railML case, it does not make much sense to use some elements such as a spotLocation as root. Such a mixing of patterns will probably rquire more manual care than completely adopting one pattern.