Subject: Re: Some problems with/questions about the infrastructure schema...
Posted by Volker Knollmann on Fri, 18 Jun 2004 09:08:47 GMT
View Forum Message <> Reply to Message

Wolfgang Keller wrote:
> Am Wed, 16 Jun 2004 17:49:26 +0200 schrieb Volker Knollmann:
>> Hmmm, I would prefer the opposite syntax. For example, I would
>> appreciate a common attribute "elemID" for all elements. Right now, we
>> have "ID", "elemID", "ocpID", "connectionID", etc. "elemID" is used in
>> many children of <trackData>, why not everywhere?
>
> Within bulkloads of code, it's easy to get lost, especially
> but not only for others than the individual who wrote the code. Instantly
> figuring out what all that code is about can be pretty difficult when
> everything is identified by an elemID. However, if you read trackID, you
> instantly know that the element in question is a track element. The same
> applies for all other identifiers. Having unique identifiers everywhere
> avoids lots of mistakes.

Sorry again, but your arguments don't convince me ;)

If you are reading a XML-file, you can immediately figure out with which
kind of element an "elemID" is associated - just take a look at the tag
the "elemID" belongs to:

<track elemID="42" ....>,
<ocp elemID="kjdf" ....>,
<switch elemID="666" ...> and so on.

On the other hand if you use different names (trackID, switchID,
trackRefID, lineID, branchID, whateverID) for the same thing (a ID), you
get puzzled while writing XML or a parser. You always have to take look
at the reference what the exact name of the ID-attribute for a certain
element is. If you only have to remember "elemID", things are a lot
easier and less vulnerable to mistakes.

> Just like XML, which was unknown ten years ago, other information
> representation technologies may come up in the future. If you take care of
> technology-independence now, you can take advantage of these later.
> Otherwise, most if not all the work invested may be lost.

How could a technology-indepent representation of the data be realized?
I can only think of diagrams like UML, ER or other formal methods (and
UML itself was unknown some years ago as well as XML).
And don't forget: as soon as you what to actually _USE_ the language /
structure / schema / system you invented, you have to make a decision
for a certain technology. AFAIK, railML had always the intention to be
applicable and usable from the beginning. And XML is a good compromise

between instantly usable technology on the one hand and a structured describtion on the other hand.

>> I agree, that due to a certain redundancy within the file (opening / >> closing tags, ...) the file size is not optimal.
>> But for the transmission over bandwidth- or volume-critical connections, >> you can apply $FAVOURITE_COMPRESSION_UTILITY to your data and you should >> get satisfying results...
>
> Nope, sorry. The experts who are working on such topics know why they are
> doing things they way they do them. And they don't use zipped XML for their
> datagrams.

"Datagrams"? That sounds like dynamic data (e. g. track occupation, current train speed, etc) which is send periodically between network nodes. Indeed, compressed XML is not the first choice for such purposes (although I would like to hear the arguments of "the experts who are working on such topics").
railML focuses on _STATIC_ data like infrastructure and timetables. And this static information is not as time- and volume-critical as dynamic process data which you need to update periodically or in "realtime" (whatever "realtime" means in your context).

> Under different conditions, different information representation
> technologies have different advantages. If you want to allow for automatic
> processing of data without extensive handwork, then you will have to take
> this into account.

See above: railML's primary intention is the platform- and vendor-independent representaion of static railway data. And for this purpose, I still think XML is a good choice. I definitely prefer parsing a structured text file over fiddling around with multiple binary tables of a relational database for instance.
For XML, various libraries for various languages and platforms are available. They simply work. And the file itself can be reduced to use 7-bit ASCII only. It simply works. Everywhere. Even with slow telnet-connections, VT100-Terminals or a Z80-machine.
Have you in contrast ever tried to access a MySQL-Server on Linux from a Windoze-Workstation and MS-Access via ODBC and SQL? I was close to biting my keyboard and sending a letter bomb to Redmond, USA. To cite Murphy: some systems are more compatible than others. You _NEVER_ have this kind trouble with XML. This is why I really like it.

Best regards and a nice weekend,
Volker Knollmann