Comments on EML 6.0 Schema Organization

November 28, 2009

Richard Cardone, Ph.D.

1 Introduction

EML specifies a wide range of voting system data formats and protocols, and this comprehensive approach makes its 36 primary schemas a formidable body of work to understand and use. As EML 6.0 proceeds through the approval process, automated tools can help us investigate, double-check and, possibly, improve upon, the organization of the specification.

The **EML60Analysis** program analyzes EML 6.0 schema files with an eye towards usability. The goal is to uncover complexity and inconsistencies that might hinder the ultimate acceptance of EML. In particular, the program records and analyzes how names are associated with types across all EML schemas. EML60Analysis examines EML's 798 element and 182 attribute definitions and writes its results to files. The **EML60Analysis.zip** archive contains these result files:

- *AnalysisSummary.txt*
- *NameAnalysisSummary.txt*
- *nameAnalysis/*.txt*

The first two files contain summary information about the structure of EML schemas and about the element and attribute names that the schemas define. The last item is a directory of 47 text files, each file documenting a name that is used differently in different contexts. This variability in name usage may reveal a source of unnecessary complexity or variability. The *nameAnalysis* files contain names that exhibit one or more of the following characteristics when they appear in different contexts:

1. The name uses different typographical cases.
2. The name defines both elements and attributes.
3. The name is defined with different types.

These characteristics do not necessarily indicate a problem, but they do point to areas of the specification that might warrant another look.

1.1 How to Use the Analysis Data

EML60Analysis generates statistics and discovers associations between schema definitions that *may* be useful during the EML 6.0 evaluation period. It is well understood that schema changes are essentially interface changes, so any gain in usability or comprehensibility has to be weighed against the cost of disrupting existing applications and the risk of breaking something that works.

Action	Description	Costs	Benefits
Do nothing	Don't change a definition	Current usability may not be optimal	No effort, little risk
Consolidate closely related types	Refactoring types to take advantage of commonalities and redundancies	Requires coordinated changes to type definitions and references to those types	Can expose commonality, improve reuse, and simplify definitions
Create an explicit type	Create a named type where currently anonymous types are used	Requires coordinated changes to type definitions and references to those types	Can expose commonality, improve reuse, and simplify definitions
Change an attribute or element name	Change a name in some or all of the contexts in which it appears	Requires changes to document instances that conform to the changed schemas	Can better reflect schema semantics, reduce overloading of terms, allow for precise descriptions of related but not identical concepts.

Each of the 47 files in the *nameAnalysis* directory describes how a single name is used in different ways in different contexts. The above table lists in increasing disruptive potential the possible actions that could be taken for each such name.

In some cases, the best thing might be to do nothing because the possible disruption (candidate: *EML*). In other cases, consolidating a type would remove redundancy and make explicit the relationship between different types (candidate: *Proposer*). Sometimes replacing an anonymous type with an named type is enough of a signal to users that the same name is defined differently in different places (candidate: *PreferredChannel*).

None of the examples just mentioned would change an element or an attribute name, so they can be thought of as schema housekeeping or reorganization. There are cases, however, where we may want to signal a semantic difference by introducing new names for different types. *Contest* is an example of the same element name being defined with numerous, disparate, anonymous types. Even though each definition of *Contest* is clearly different, the documentation in EML-v6-dictionary.xls is the same for all uses. In cases such as this, we may want to introduce new names to distinguish between uses and to allow for more precise documentation.

The next section lists the 47 names that are used differently in different contexts.

2 Names in Multiple Contexts

Each of the 47 subsections below corresponds to a file in the *nameAnalysis* directory. For example, the file *address.txt* contains detailed information about how the name *Address* (section 2.1) is used.

Each subsection contains a brief summary of how the name is used in its various contexts. The summary is followed by a *preliminary triage statement* about how we might want to deal with the name. This triage statement is just my initial take on how we might proceed.

When I reviewed the files referenced in this section, I found utility in keeping the following questions in mind:

1. Should an explicit type be defined in places where multiple, closely related, anonymous types appear?
2. Should elements and/or attributes be renamed if they have the same name, but do not have closely related types?
3. Should elements and/or attributes that have the same name, but do not have closely related types, use different explicitly defined types?

2.1 Address

Address is defined using two different types in different paths in emlcore-v6-0.xsd.

[Consider doing nothing since both types are based on the same actual type (*`xal:AddressType`*).]

2.2 Affiliation

Affiliation is defined using one use of `AffiliationStructure`, 3 anonymous types derived from that type, and `xs:token`.

[Consider changing names to distinguish different types.]

2.3 AuditInformation

AuditInformation is defined using two uses of `AuditInformationStructure`, and one anonymous type derived from that type.

[Consider changing the name of the occurrence that uses the derived type.]

2.4 Candidate

Candidate is defined using one use of `CandidateStructure` and another use of an anonymous type that does not reference `CandidateStructure`, but does contain a `CandidateIdentifierStructure` subelement.

[Consider changing names to distinguish different types.]

2.5 CastVote

CastVote is defined using two anonymous types that are derived from `CastVoteStructure`.

[Consider changing names to distinguish different types.]

2.6 Category

Category is defined four times using `xs:token` and once using a 3 member enumeration derived from `xs:token`.

[Consider changing names to distinguish different types.]

2.7 Channel

Channel is used both as an element name and an attribute name.

Two identical anonymous type definitions of *Channel* appear in 110-electionevent. *Channel* is also defined four times using `VotingChannelType` and once using of `ChannelStructure`. `VotingChannelType` is a simple type (an enumeration) used to define an attribute named *Channel*. `ChannelStructure` is a complex type; it is used to define an element named *Channel*, which unfortunately specifies a child element that is derived from `VotingChannelType`.

[Consider reorganizing channel related attributes and elements. This is non-trivial.]

2.8 Contact

Contact is defined eight times using `ContactDetailsStructure` as its type and once using a derivation of that same type.

[Consider doing nothing or changing the name in the derived case.]

2.9 Contest

Contest is defined using eight different anonymous types.

[Consider factoring out commonalities into one or more base types and/or changing names. This one is non-trivial.]

2.10 Contests

Contests is defined twice using anonymous types that are very closely related.

[Consider using a common base complex type for both definitions.]

2.11 CountMetric

CountMetric is defined once with the `CountMetricStructure` type and three times with the `CountMetricsStructure` type.

[Consider changing the name of the elements defined with type `CountMetricsStructure` to CountMetrics.]

2.12 Date

Date is defined once using `ComplexDateRangeStructure` and once using `xs:date`.

[Consider changing names to distinguish different types.]

2.13 Description

Description is defined eleven times using type `MessagesStructure` and once using `xs:token`.

[Consider changing the one `xs:token` instance of Description to use `xs:string`.]

2.14 DisplayOrder

DisplayOrder is used as the name in unique definitions and in attribute definitions. When used in attributes, its type is `xs:positiveInteger`, otherwise, it has no defined type. This is a proper use of unique.

[Consider doing nothing.]

2.15 Election

Election is defined in ten places using ten different anonymous complex types.

[Consider factoring out commonalities into one or more base types and/or changing names. This one is non-trivial.]

2.16 EML

EML is defined in 29 places using 29 different anonymous complex types.

[Consider doing nothing since this is the top-level element of every EML document and the question has come up before. See the "Language Binding Generators" thread in the January 2009 TC member discussion archive.]

2.17 End

End is defined in 8 places using `BallotIdentifierStructure` once and date/time oriented types in the remaining occurrences (`xs:date`, `xs:dateTime`, and `DateType`).

[Consider using different names in different cases. Also see *Start* below.]

2.18 Format

Format is defined as an attribute three times in emlcore-v6-0.xsd, twice with type `xs:NMTOKEN` and once with a restricted type based on `xs:NMTOKEN`.

2.19 Gender

Gender is defined twice, once with type `GenderType` and once with an inline type equivalent to `GenderType`.

[Consider replacing the inline type definition with a reference to `GenderType`.]

2.20 Id

Id is defined 37 times, 8 times with type `xs:token` and 29 times with no type specified. It appears that when no type is specified, *Id* always has a fixed value.

[Consider doing nothing.]

2.21 IdNumber

IdNumber is defined eleven times, ten time with type `xs:NMTOKEN` and once with type `xs:token`.

[Consider making using the `xs:NMTOKEN` type in all *IdNumbers*.]

2.22 MaxWriteIn

MaxWriteIn is defined twice with type `xs:nonNegativeInteger` (zero allowed) and once with type `xs:positiveInteger` (zero prohibited).

[Consider harmonizing types, changing names, or doing nothing.]

2.23 Message

Message is defined twice with type `xs:string` and once as an anonymous complex type.

[Consider doing nothing or changing names.]

2.24 Name

Name is defined five times with type `PersonNameStructure`, once with type `xs:token`, and once an anonymous complex type.

[Consider doing nothing: *Name* refers to different things in different contexts, so different types are appropriate.]

2.25 Nominate

Nominate is defined twice using two slightly different complex types.

[Consider consolidating types.]

2.26 Notes

Notes is define once as an element and once as an attribute, both times using type `xs:string`.

[Consider doing nothing.]

2.27 Position

Position is defined twice using type `xs:token` and once using type `PositionStructure`, which is based on `xs:token`.

[Consider changing names, creating a named type, or doing nothing.]

2.28 PreferredChannel

PreferredChannel is defined once with type `VotingChannelType` and once as a type based on `VotingChannelType`.

[Consider creating a named derived type, changing names, or doing nothing.]

2.29 ProposalIdentifier

ProposalIdentifier is defined as an element with type `ProposalIdentifierStructure` and as an attribute with type `xs:token`.

[Consider changing names or doing nothing.]

2.30 Proposer

Proposer is defined twice with identical anonymous complex types and once with type `ProposerStructure`.

[Consider consolidating the two anonymous complex types into a named type and changing names.]

2.31 Qualifier

Qualifier is defined once with type `xs:token` and once with an anonymous complex type.

[Consider changing names or doing nothing.]

2.32 Reason

Reason is defined nine times as an attribute with type `xs:token` and once as an element with an anonymous complex type that derives from `xs:token`.

[Consider changing names, creating an explicitly named type, or doing nothing.]

2.33 ReferendumOptionIdentifier

ReferendumOptionIdentifier is defined once with type `xs:token` and once with type `ReferendumOptionIdentifierStructure`, which derives from `xs:token`.

[Consider changing names, creating an explicitly named type, or doing nothing.]

2.34 ReportingUnitVotes

ReportingUnitVotes is defined twice with closely related anonymous complex types.

[Consider consolidating the two anonymous complex types.]

2.35 ReportType

The attribute *reportType* is defined once and the attribute *ReportType* is defined twice. *reportType* has no type specified; *ReportType* has type `xs:token`.

[Consider changing the name and type of *reportType*.]

2.36 Role

Role is defined twice with the type `xs:token` and once with an anonymous complex type that derives from `xs:token`.

[Consider changing names, creating an explicitly named type, or doing nothing.]

2.37 Selection

Selection is defined four times using similar, but not identical, anonymous complex types.

[Consider distinguishing the name used in each case.]

2.38 SequenceNumber

SequenceNumber is used once as an attribute and once as an element, both with type `xs:positiveInteger`.

[Consider doing nothing.]

2.39 ShortCode

ShortCode appears five times as an attibute and once as an element, always with type `ShortCodeType`.

[Consider doing nothing.]

2.40 Start

Start is defined in 7 places using `BallotIdentifierStructure` once and date/time oriented types in the remaining occurrences (`xs:date`, `xs:dateTime`, and `DateType`).

[Consider using different names in different cases. See *End* above.]

2.41 Status

Status appears twice as an attribute with no type specified and once as an element with an anonymous complex type.

[Consider explicitly specifying the attribute types.]

2.42 TotalVotes

TotalVotes is defined twice with closely related anonymous complex types.

[Consider consolidating the two anonymous complex types.]

2.43 Type

Type appears 14 times, as both an element and an attribute, using four different types: `xs:anyURI`, `xs:string`, `xs:NMTOKEN`, and `xs:token`.

[Consider reducing the number of types used to define *Type*.]

2.44 Voter

Voter is defined once using type `VoterIdentificationStructure` and 5 times using anonymous complex types. Two of these types are identical.

[Consider consolidating types and differentiating names.]

2.45 Votes

Votes is defined once with type `xs:positiveInteger` and once with an anonymous complex type.

[Consider changing the name to distinguish the two cases.]

2.46 VToken

VToken is defined once with type `VTokenStructure` and once with an anonymous complex type that derives from `VTokenStructure`.

[Consider changing names, creating an explicitly named type, or doing nothing.]

2.47 VTokenQualified

VTokenQualified is defined once with type `VTokenQualifiedStructure` and once with an anonymous complex type that derives from `VTokenQualifiedStructure`.

[Consider changing names, creating an explicitly named type, or doing nothing.]