Structured Authoring with XML Process Documentation

Table of Contents

Matthew Murphy
WRIT 4662W Writing with Digital Technology
2018-10-18

Background

Previously, I met with my client to discuss our assignment and talk about how XML might be useful in accomplishing the goals we established during the analysis phase of our website project. The take away from our meeting was a description of the data which we were to define in our tag set.

The XML file

Sample data

My first task in creating our tag set was to create sample data. Creating the sample data would show me if my client and I had missed anything in our definitions. It would also form the basis for my XML file. Using the table of possible elements and attributes from my client meeting, (See Appendix A: Client meeting.) I created one sample event and one sample news announcement.

Root element

My first challenge in creating my XML file was to combine `news` and `events` into a single root element. Because "news" and "current events" are so often associated in the media, it was natural for my client to make the same association. However, the `news` and `event` elements my client and I discussed have different sub-elements. I considered combining them into a single element and allowing NULL values for their sub-elements. However, because they differ in their purpose, I felt they should be treated as separate elements.

My solution was to create a new root element called `post`. This root element would contain `news` and `event` as child elements, and would allow us to add additional `post` types in the future.

Child elements

After deciding on my root element, I turned to creating my child elements, starting with `event`. I made several modifications to the `event` fields my client identified:

- Event title was added.
- Event time became event begin time and end time.
- Event description became event short description and event long description.

I followed the same process with the `news` element. However, since `news` already contained a `title` field, I found I only needed to alter the description field to make it similar to my `event` element.

Metadata

As I explained to my client, there would be several fields in our data which were used to describe the data itself. For each type of `post`, I wanted to capture the date the post was created, the post author, and the date it was last updated. I also wanted the ability to include descriptive tags in each post.

The DTD file

In creating my DTD, my first task was to define my root `post` element. A `post` could be either an `event` or a `news` type, but it could not be both. Also, it must contain one of those values and must not contain any other value.

My root element is defined as:

```
<!ELEMENT post (postEvent | postNews)> I
```

My child elements were defined as containing each of their respective sub-elements. In each child, only the `tags` element was allowed to be NULL. Each of the sub-elements contained character data as in the example below:

```
<!ELEMENT eventType (#PCDATA)>
```

Attributes

To creating my attributes, I found it easiest to work in both my XML file and DTD file, adding attributes and their definitions together. In several cases, I found it difficult to decide if a data item should be an element or an attribute. In a quick Google search, I found I was not alone. I did not find a set of standards or rules for determining if something should be an element of its own, however. The most common advice seemed to be to use one's best judgement.

For my client's tag set, I decided to limit my use of attributes to data that would be likely to be used by an application or database (rather than data meant to be displayed to a user).

I'm sure that many would caution be against designing my structure to fit an imagined future scenario. On the other hand, I think that the question of element vs. attribute agrees with Self's assessment that "form, structure, and style can never be fully separated". cite

In my XML file, I first assigned "id" attributes:

```
<post postID="">
  <postEvent eventID=""></postEvent>
  <postNews newsID=""></postNews>
</post>
```

Next, I assigned an attribute for the date an event or news announcement was created, and the last date that each was updated.

Because every post (and post type) will have a unique ID, I used a default attribute value of #REQUIRED. Because every event or news announcement will have a date on which it was created, I used a default attribute value of #REQUIRED. However, since not all posts will be updated once they are posted, my default value for this attribute is #IMPLIED.

Among the other items which I considered metadata were the author and tags data items. After consideration, I created elements for these items, thinking that they are more likely to be displayed for the user than be required by an application or database.

Naming conventions

While the Lynda.com Essential XML tutorial cite talked briefly about the naming of elements, my naming convention is more related to working with database tables than it is to web-centric languages like JavaScript. Each element is named in relation to it's parent. For example:

```
<post>
  <postEvent>
    <eventType></eventType>
    <eventLocation></eventLocation>
    <eventCost></eventCost>
  </postEvent>
</post>
```

I've found several benefits to using this naming convention:

- Avoids name collisions
- Assists grouping and sorting
- Identifies the hierarchy of data elements

Because of this, I am able to include sub-elements in each `post` child that serve the same purpose. For example, `eventAuthor` and `newsAuthor`. Although I could accomplish this by specifying a namespace for each, I would have to do so every time my client requested a new `post` type; addressing this in the element names seemed to be a more future-friendly method.

Concerns

My primary concern is that my client's tag set is over-designed. It may be that there are elements which contain data that will never be used. Yet, these elements and their data source must be maintained. The time and cost of maintaining unused data elements may outweigh the value of using XML, at least in my current design. My design experience comes from working with large databases containing many tables. It may be that these conventions don't scale down to a data set as small as my client's.

My other concern is that I'm unclear on how our tag data will be applied, and this impacted my design. It's true that we can use XML to separate form from content. However, in creating the DTD, I was left to imagine future scenarios in order to define my elements. For example, in the DTD, I can require a value for my ID attributes. However, I still must tell my data source that the

ID must be in the correct form (e.g. sequentially numbered positive values, etc.). Without knowing how the data in my XML file will be used, it's difficult to define, and I find myself making assumptions about my data source.

Course material and other references

Much of the course material during the last two weeks has been helpful conceptually. I appreciated how Samuels demonstrated how structured writing, topic-based writing, and single-source publishing supported each other. cite The Usability.gov article included a helpful visualization of the "interdependent nature of users, content, and context. cite I also found the XML Essentials course from Lynda.com cite valuable, in particular in showing how CSS can interact with XML.

Mai's workshop cite very nicely condensed the practice of XML. However, I still had several questions about creating an XML file and the associated DTD. I often turn to the W3C's website for examples, but I did not find their DTD tutorial cite as thorough as I'd hoped. Searching the Internet, I located a tutorial on the website Quakit.com cite which included the information I was looking for. Most helpful were the tables which listed and defined the default values for both elements and attributes.

I also spent significant time searching the Internet regarding when a data item should be an element vs. an attribute. As I mentioned, several of my data items could be either. The best advice I found came from IBM, the developers of DITA. The author, Uche Ogbuji, calls this "the oldest question in XML design". cite Ogbuji is very thorough and does a great job at considering other common approaches before giving his recommendations.

Conclusion

In the end, I found that the course resources provided me with concept and context, but left me with questions about applying those concepts. The XML Essentials course demonstrated practical application, but it was cumbersome to consult when I had a specific coding question. For our assignment, I found myself coming back to websites like Stack Overflow or previously mentioned Quackit.com more than our other resources.

I also found that my previous experience working with large databases was helpful in coming up with my design, but I'm not certain that my background knowledge is directly transferrable to XML design. As a result, I may have provided an over-engineered solution for my client.

Regardless, I found value in both the exercise and in the tag set I've provided my client. Defining the data we use leads us to thinking about how it might be used in the future. For my client, this meant imagining entirely new ways to bring their writing to an audience.

References

Appendix A: Client meeting

Table 1. Possible elements and attributes

Event	**News**	**metadata**
type of event	title	post date
date	description	author
time	date written	last updated
location		tags

Event	News	metadata
cost		
description		

Appendix B: Project Links

Diamond Dust Writers:
https://telecommatt.github.io/diamond-dust-writers/

Github profile: https://github.com/telecommatt/

Structured Authoring with XML Assignment: https://github.com/telecommatt/diamond-dust-writers/tree/master/structured-authoring-xml-assignment