Tight or Loose Data Constraints?

Roger L. Costello
January 2011

Issue

Should XML Schemas implement tight or loose data constraints?

In this paper I argue that XML Schemas should implement tight data constraints.

Example

Suppose you are building a system in which you will exchange English family names. You do research and find that all your users have these data constraints: their names are not longer than 100 characters and their names consist exclusively of the characters a-z, A-Z, space, hyphen, apostrophe, and period.

Figure 1: Sample Family Names

Set of Characters	**Example Family Name**
The letters a-z, A-Z	Costello
Hyphen	Parsons-Kerns
Apostrophe	O'Donnell
Space	de La Cruz
Period	St. Ives

The following simpleType exactly (tightly) matches the data constraints. It uses the minLength and maxLength facets to constrain the length. The regular expression in the pattern facet constrains the set of characters.

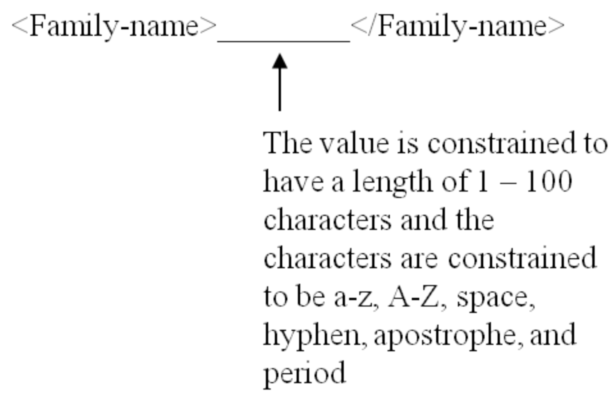
Figure 2: Tight simpleType

```
<simpleType name="English-language-family-name">
     <restriction base="string">
            <minLength value="1" />
            <maxLength value="100" />
            <pattern value="[a-zA-Z' \.-]+" />
     </restriction>
</simpleType>
```

The XML Schema also declares a Family-name element and constrains it using the above simpleType:

```
<element name="Family-name" type="English-language-family-name" />
```

Here is a sample XML instance document:



<Family-name>________</Family-name>

The value is constrained to have a length of 1 – 100 characters and the characters are constrained to be a-z, A-Z, space, hyphen, apostrophe, and period

Consider validating the XML instance document against the XML Schema. If the value of <Family-name> exceeds the length constraint or uses characters other than those identified by the regular expression then a message will be generated: "The value is invalid."

Contrast with Loose Data Constraints

For comparison, let's see what it means to implement loose data constraints. The following simpleType implements loose data constraints. Family name values are allowed to be any string. This means there is no restriction on the length or on the set of characters.

Figure 3: Loose simpleType

```
<simpleType name="English-language-family-name">
     <restriction base="string" />
</simpleType>
```

Of course, there is a broad spectrum of restrictiveness. This simpleType restricts the length to 10000 characters and it has no restriction on the set of characters:

Figure 4: Another Loose simpleType

```
<simpleType name="English-language-family-name">
     <restriction base="string">
            <minLength value="1" />
            <maxLength value="10000" />
     </restriction>
</simpleType>
```

Benefits of Tightly Implemented Data Constraints

To see the benefits of tight data constraints, let's expand our example. Suppose that you creating a system for users to apply for a credit card. You've created a web form that allows users to enter their personal information (including family name). When a user presses "Submit" his data is first validated and then is sent to a server which routes it to a credit card printer machine that stamps his name on a credit card. A few weeks later the user gets his new credit card in the mail.

Data Entry: the first step that a user takes to getting a credit card is to fill in the web form with his personal data. Suppose the user fills in the family name field with this value:

Lecœur

Notice the œ ligature. That is an invalid character. When the name is validated against the tight simpleType then the invalid character is detected and the user is notified: "Sorry, the œ character is not a valid character." Also, this message is logged and later analyzed.

> Note: An XML Schema simpleType is used to validate the data entered at the web form. Alternatively (in fact, more typically) the data could be validated using JavaScript (or some other language). The issue is not the particular language used to do the validation. The issue is tightness versus looseness. A JavaScript program could exactly (tightly) implement the data constraints and detect the invalid œ ligature. For this article I assume that XML Schema is the validation language. Just remember that other languages may be used.

We immediately start seeing the benefits of tight constraints:

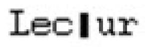
1. Early detection of errors
2. Immediate feedback to users
3. Analysis of user inputs

The earlier errors are detected, the cheaper they are to fix. The user was alerted that the name he entered will cause problems for the card printer machine, so he was able to instantly adapt.

The Lecœur name was flagged as invalid and a report was logged. The report was later analyzed by business and IT managers to determine: "Is the understanding of our users incomplete – are there users whose family name contains the œ ligature? Or, is this someone attempting to attack our system?" If it is determined that there are users for which the simpleType does not account for, the simpleType can be adjusted. Thus, this leads to another advantage:

4. Evolutionary change to systems is possible due to early identification of changes in the user base

Data Processing: Suppose that a loose simpleType was used to validate the user's input, and the invalid family name value was not detected at data entry. The data is then sent to a server, which gives it to the credit card printer machine. Suppose the printer only understands ASCII characters. The œ ligature is not an ASCII character, so when the printer stamps the card here is the result:


Lec▮ur

A few weeks later the user receives the credit card in the mail, only to discover that it is bad and he has to start the process all over again. That's wasted time. That's a financial loss for the credit card maker. Everyone loses.

In this case the invalid œ ligature resulted in a black rectangular box being stamped on the credit card. However, there are more catastrophic scenarios possible. For example, suppose the invalid character damages the credit card printer. This could result in a huge loss of time and revenue.

We see another advantage of tight constraints:

5. Reduced vulnerability. Reduced risk of malicious attacks.

Impact of Change

Clearly things will change over time. Your user base will change. As we've seen, with tight constraints you will be instantly alerted to changes. You need to design the rest of your system to be adaptable.

Your XML Schemas will need periodic updating. For example, you may get new users with long family names (up to 150 characters in length) and you may get users that have family names containing the œ ligature. The simpleType is easily updated to reflect changes in the ecosystem:

```
<simpleType name="English-language-family-name">
     <restriction base="string">
            <minLength value="1" />
            <maxLength value="150" />
            <pattern value="[a-zA-Zœ' \.-]+" />
     </restriction>
</simpleType>
```

Data Binding Tools: Data binding tools are used to automatically generate code that corresponds to structures in an XML Schema. A data binding tool maps an XML Schema to imperative code such as Java or C++ code. For instance, JAXP is a data binding tool that maps XML Schemas to Java code.

What will be the impact of changes to the XML Schema on the Java or C++ code? For example, will the change in the tight simpleType result in breaking the Java or C++ code? Will the Java or C++ code need to be changed each time the simpleType changes? Answer: No. The binding Java or C++ objects will not change at all. Validation is applied externally to the binding objects.

Know Your Users

It is vital to know your users. Turning away users due to negligence in understanding the user base is bad business. For example, if there are legitimate users with family names longer than 100 characters then it is vital that your system quickly (instantly) recognize that the simpleType's constraints are weeding out legitimate users.

Suppose a user enters a family name that is determined to be invalid. There are three possibilities:

1. It is a legitimate family name that we have not accounted for.
2. The user accidentally entered a bad name.
3. The user is trying to attack the system.

You must decide, in real-time, which is the case. That's not easy.

Discovering User Data Constraints

Above, when first introducing the example I said, " You do research and find that all your users have these data constraints …" How does one discover the user's data constraints?

I discovered the set of characters used in English language family names by going to the U.S. Census Bureau. They have a list of all the family names in the U.S. I mined that data and extracted the set of characters. I looked at the length of each family name. They were all well under 100 characters. I checked the Guinness World Book of Records for the longest name (590 characters). I made a judgment call that 100 characters is probably a reasonable length.

I spent several hours researching the data constraints for English language family names. That's a lot of work for one data item. But it's necessary. We need to create a library of well-researched simpleTypes.

Don't Let IT Limitations Drive the Business

In the above example I stated, "Suppose the credit card printer only accepts ASCII characters." Please do not interpret that statement as, "The system is designed to accept only certain users because of the limitations with the IT technology (the printer)." No! That is bad business. IT supports the business; IT must not drive the business.

Suppose that initially your user base consists exclusively of users with ASCII family names. You purchase a credit card printer that supports only ASCII characters. Later, your user base changes and some users have family names containing non-ASCII characters. A credit card printer that supports non-ASCII characters is much more expensive. You must make a business decision on whether the number of new users with non-ASCII characters is sufficient to justify the large capital expenditure. For example, if there are only one or two users with non-ASCII family names then it may not be worth the expenditure. But if you want to expand your business from USA-only to a global market, then it will likely be worth the expenditure.

Summary

Understand your users and then implement your XML Schemas to exactly match your user's constraints. Do this and your users will be happy because they get immediate feedback when they do something wrong. And you will be happy because you will be instantly alerted to changes to your user base and you will be positioned to adapt to changes. Also, your security people will be happy because you've reduced the vulnerability of your system and reduced the risk of malicious attacks.