Run-Time Validation of Inbound XML Documents – Yea or Nay?

Issue
An application receives an XML document. Should the application validate the XML document prior to processing it? That is, should applications perform run-time validation of inbound XML documents?

Discussion
There is no right or wrong answer to this issue. There are only engineering tradeoffs. So, before making a decision for your particular application, it is important to understand the approaches, their advantages, and their disadvantages.

Approach #1: Nay to Run-Time Validation
The inspiration for this approach is captured by Postel's Law [1]:

Be liberal in what you accept, conservative in what you generate.

Here's the idea: when an application receives an XML document, it extracts what it needs and ignores the parts it doesn't need. If the XML document has errors, the application does its best to determine what the intent is, and proceed accordingly.

The most widely known applications that follow this approach are web browsers. Web browsers are very forgiving in the HTML they receive.

Advantages
1. It is easy for clients to work with the application since it is forgiving. For example, a forgiving purchasing application doesn't turn away a client just because his purchase order contained a spelling mistake.

Disadvantages
1. Recall what was stated in the above text:

   If the XML document has errors, the application does its best to determine what the intent is …

   What if the application guesses wrong? A wrong guess may result in the application doing the something terribly wrong. Web developers can give many examples of web browsers displaying their web page in a completely unexpected and undesired way (or not displaying it at all) because of a mistake in their HTML.

2. Even if an application is liberal with the input it receives, it will still likely need some error-checking code. In the worst case it will need a lot of error-checking code. The net result is that validation is reinvented in hand-crafted validation code that is intertwined with application

processing code. That is not cost effective. It is unlikely that hand-crafted validation code will be as robust as a COTS tool that is dedicated to validation. Furthermore, there is no division of labor; this means there is less efficiency.

Recommendation

1. Do not use this approach for mission-critical applications where a misinterpretation may have catastrophic effects.

Approach #2: Yea to Run-Time Validation

With this approach, the application validates all inbound data and only permits valid data into the application. There are two inspirations for this approach:

1. **Division of labor**: by separating the validation task and the data processing task into independent parts, a tremendous enhancement of productivity may be achieved. See Adam Smith's *Wealth of Nations* [2].

2. **Trust**: invalid data may result in corrupting the application or have other undesirable consequences. The data may be invalid as a result of accidental data entry or it may be due to malicious design. In either case, inbound data cannot be trusted.

Advantages

1. There is no need for applications to guess about the intent of users.

2. There is a higher quality of information.

3. The application code is simpler since the error checking code is offloaded to a COTS tool.

4. Validation is performed using a COTS tool. In comparison with hand-crafted validation code, the use of a COTS tool for validation results in reduced cost and less risk of bugs.

Disadvantages

1. This approach is not forgiving to users. It requires fairly sophisticated users, which are able to validate their XML documents.

Recommendation

1. Do not use this design approach where there is a diverse group of unsophisticated users (provided that the users can be trusted and misinterpretation of their data is not catastrophic).

Cost of Validation

A common argument that is given for why run-time validation should not be performed is that the cost in time and memory resources for validating inbound XML documents in too high. Let's analyze that argument. Consider two cases:

Case 1: an engineering assessment is made and it is determined that validation of inbound data is *not* needed. In this case the argument is irrelevant.

Case 2: an engineering assessment is made and it is determined that validation of inbound data *is* needed. There are two ways to accomplish the validation:

a. Create hand-crafted validation code and embed it in the application code.

b. Use a validation language (XML Schema, RELAX NG, Schematron) along with a COTS validator.

In light of this, we must dissect the argument. Presumably the argument is:

> Run-time validation *using a validation language along with a COTS validator* should not be performed because the cost in time and memory resources for validating inbound XML documents is too high.

Clearly that is a fallacious argument. It is obviously better to use a validation language along with a COTS validator for run-time validation than it is to use hand-crafted validation code that is embedded in application code.

Noah Mendelsohn points out [3] that validation may have little or no overhead compared to non-validated processing:

> Starting in 2001 my group at IBM built a prototype system that demonstrated that, with care, Schema Validation can often be done with little or no overhead compared to non-validated processing of the same data.

JAXB is a widely deployed commercial system that provides runtime validation. This is from Michael Glavassevich [4]:

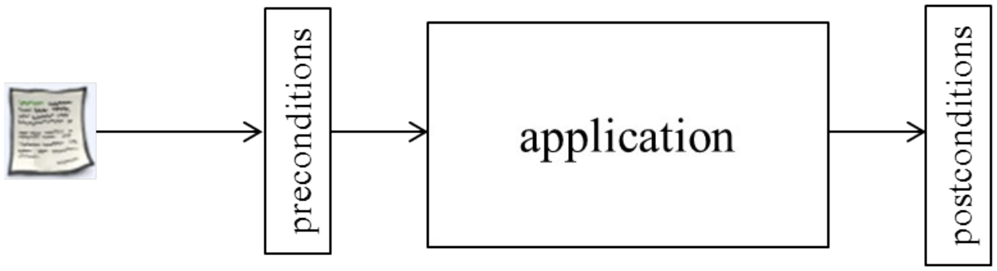
> If you don't want invalid documents to be accepted quietly you can register a schema (not necessarily XML Schema; could also be Relax NG, Schematron or something else) and if you want you can force JAXB to fail if an error is reported by the validator or respond in some other way with the error handler you've registered.

Feedback

Here is feedback since the publication of this article:

1. The schema languages (XML Schema, RELAX NG, Schematron) cannot express some things, so it may be necessary to embed within the application some code to check the data. Be sure that the code is isolated. Only write code to perform things that cannot be expressed by the schema languages. That should be very little.

2. Validation is used to ensure that the input is consistent with the preconditions of the application. Validation can also be used to ensure that the output is consistent with the postconditions of the application.



Acknowledgements
Thanks to the following people for their contributions to this paper:

- Roger Costello
- Bruce Cox
- James Garriss
- Michael Glavassevich
- Stephen Green
- Marc Hadley
- Ken Holman
- Casey Jordan
- Michael Kay
- Noah Mendelsohn
- Scott Renner

References
[1] http://en.wikipedia.org/wiki/Robustness_principle

[2] http://www.amazon.com/Inquiry-Nature-Causes-Wealth-Nations/dp/193604188X/ref=sr_1_1?ie=UTF8&qid=1305985547&sr=8-1

[3] http://lists.w3.org/Archives/Public/xmlschema-dev/2011May/0027.html

[4] http://lists.w3.org/Archives/Public/xmlschema-dev/2011May/0028.html