PhUSE 2012

Paper TS06

Deep Dive into ODM Validation

Ronald Steinhau, Entimo AG, Berlin, Germany

ABSTRACT

Checking an ODM file for correctness seems to be easy at first: an XML schema is defined and XML parsers that check XML against a given schema are widely available. But by a second look challenges not easy to tackle arise:

- Checking of valid cross-references (e.g. item ID's) in a given scope (e.g. specific version of the metadata)
- Forms and their items may form local sub-groups with repeated ID's
- The extension mechanism of ODM allows extension namespaces to be used, e.g. for define.xml. Because of name differences a namespace mapping is needed.
- Different ODM versions need different checks

For very large files some checks can be time intensive – here Scala (a modern language compatible with the Java Virtual Machine) can help to use parallel processing in an elegant way.

An extendable solution to ODM validation (often called ODM-Checker) with interesting examples and code snippets in Scala will be presented.

INTRODUCTION

ODM validation has several dimensions:

- ODM itself is moving forward in versions (e.g. 1.2, 1.2.1 and 1.3) with different schema's and therefore different checks.
- Very different types of checks are needed, mainly structural checks, value checks and referential integrity checks
- ODM extensions, like define.xml and various custom extensions, need to be handled properly (also in different versions)
- A useful error report with correct references to the illegal position in the document is needed.

This leads to an extendable solution, where different control files are used for different ODM versions and even users of the checker can extend those basic control files for their own extensions or purposes.

ODM has also some difficult to check corner cases, like the way forms and items are handled or the fact, that multiple metadata versions can be put into one ODM document.

Finally the validation needs to be fast, because ODM files can be very large. By using Scala - a type-safe, compiled language that conforms to the JVM – parallel execution of checks can be implemented very elegant and lead to a well performing solution.

EXTENDABLE VALIDATION

The ODM-Checker needs to have a control file for the different versions of ODM in order to be flexible enough. We have chosen an Excel format for such a control file to give customers a good and easy chance to extend or change the validation process. The control file also resembles the structural constraints of the XML-Schema – making a separate schema-check superfluous.

The control file consists of rows with various columns, where each row represents multiple checks of different purposes:

- Structural cardinality checks in the given context
- Value checks
- Referential integrity checks

Each row has a unique check number, which is used in the error report for referencing the failed check.

The rows carry also a category column, which allows users to include or exclude a complete category (group) of checks. It is also possible to exclude an individual check, but this will exclude all checks of that row and may impose side effects.

Finally another column depicts a special ODM-Type, e.g. "defxml" for define.xml, which can be selected for inclusion in checks. This type can be mapped to a concrete namespace prefix of the ODM file and an underlying schema. This way all shared and customer specific extensions can be included in the validation.

VALIDATION CHECK DETAILS

As mentioned there are three main types of checks, which are now explained in more detail.

STRUCTURAL CARDINATION CHECKS

These checks mimic the structural checks usually found in an XML schema. The advantage of having these checks being part of the control file is the ability to include or exclude them (e.g. by categories) or add details.
Examples for cardinality checks are a <CodeList> element, that can have multiple <CodeListItem> elements, but other elements may or not be allowed under a certain parent element.
Under some parent only one out of 2 or more alternatives is allowed, e.g. a <CodeList> element can either contain one or many <CodeListItem> elements or alternatively one <ExternalCodeList> element. Bu it must contain one of them. This is covered by so-called exclusive Or-Groups in the control file.

VALUE CHECKS

These checks take either the attribute value or the body value (between the tags) and check it against a specific type. Each type (e.g. “integer” or “sasName”) has a specific pattern or set of values to be checked. A special case are so-called list values, e.g. the “UserType” attribute of the <User> element can only have a value out of “Sponsor”, “Investigator”, “Lab” or “Other”.
Some pattern are not fix, they may be changed by the user when running the checker, e.g. the max. length of a SAS name can be 8 or 32.
Even in the value check domain the schema validation check would not be enough, as special values from SAS (e.g. missing values) cannot be expressed properly in an XML-Schema. This alone enforces an ODM checker going beyond simple schema validation.

REFERENTIAL INTEGRITY CHECKS

In ODM files many elements refer to other elements by a so-called OID. There are different (typed) OID’s, e.g. a Form-OID is referenced by a form reference and an Item-OID is referenced by an item reference, e.g. the “MeasurementUnitOID” attribute of the element <MeasurementUnit> refers to the “OID” attribute of the element <MeasurementUnit>. This relationship is configured in the control sheet by having a type “oidref” and a type “OID”, where the rows with a type of “oidref” are pointing to the defining element.
An OID is not the same as an XML ID. The latter are defined to be unique across the whole XML document, but an OID can have duplicates in its scope. E.g. the <MetadataVersion> element may occur several times in an ODM document and each may contain the same OID’s for sub-elements, because the OID of the metadata version element is different and serves as a scope.
The checker knows about the possible scopes in ODM (currently this is not configurable) and checks, whether an OID reference has a defined counterpart (OID) in the given scope.
The referential integrity checks are relaxed, e.g. a reference can be followed by a definition and vice versa. Therefore all unresolved references are kept until the end of a scope, when they will produce errors or get resolved.

ERROR REPORT

The error report is created as a CSV-File or Excel-File. It contains the error number defined in the control file, the row and column where the error occurred and a 40 character sequence of the XML row, where the error occurred.

SCALA – HELP FROM MODERN FUNCTIONAL PROGRAMMING

Scala (see http://www.scala-lang.org/) is a modern hybrid object oriented and functional language, which compiles to the Java Virtual Machine (e.g. is executable on any platform that has Java available). It is also compliant with the Java type system, which translates to the ability to run Java code from within Scala code and also to use Scala code from within Java code.
Scala comes with a built-in library for actors (see http://en.wikipedia.org/wiki/Actor_model) which greatly simplify the design of concurrent applications. The checker was designed to have the different validation check types as separate actors running in parallel. This way a multi-core machine can execute the ODM-Check up to three times as fast as a single-core machine. Beyond that the referential integrity check becomes the bottle-neck of concurrency.
An actor is like a very lightweight messaging system that can accept new messages at all time and executes the messages one at a time.
The powerful syntax of Scala allowed a functional instantiation of the control sheet, e.g. all checks and information of the control sheet are available on the API level in a very natural way.
Scala also comes with pattern matching and regular expressions (together much more powerful than what Java has to offer). Scala also has XML as a built-in data type, which in specific situations eases XML-Handling. In the validation checker we have used the Woodstox XML streaming library (integrated into Scala) for maximum performance. With streaming the XML is not read into memory and then processed, it is continuously read and the checks are executed on the fly. Only because of integrity checks some element information must be kept, until a resolution is possible (or errors have to be reported at the end of a scope).

CONCLUSION

Trying to reach the goal of a flexible (aka configurable) ODM-Checker, that is still very fast and does cover even the most complex checks required a fresh approach beyond XML schema validation by using control files to drive the validation process. This approach combined with the expressive power of the Scala programming language, the use of concurrent checks and the fast execution times reachable by compiled code in the latest Java Virtual machine (close to C performance) formed a deep diving ODM validation solution.

RECOMMENDED READING

Scala: Programming In Scala – By Martin Odersky, Lex Spoon and Bill Venners (Artima Verlag)
Actors: Actors in Scala - By Philipp Haller and Frank Sommers (Artima Verlag)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Ronald Steinhau
Entimo AG, Germany, Berlin
e-mail: st@entimo.de