“ASLX”: Semantic Analyzer for programming languages based on XML

Gabriela Yanel.Buffa, Franco Gaston.Pellegrini
Licenciatura en Ciencias de la Computación
Universidad Nacional de Río Cuarto

Abstract. XML (Extensible Markup Language) is a language used to structure information in a document or any file containing text. An XML document is "well formed" if respects their *basic syntactic structure*, that is composed of elements, attributes, and specified as XML comments. Some ways to verify that an XML file is either formed is by: *Document Type Definition (DTD)* and *XML Schemas*. But these types of verification not include relevant aspects, such as, the semantic relationship between content attributes and tags, declarations of fields and check the scope of variables, among others. This raises the need for a tool for verifying programming languages based on XML in the semantic aspect. In addition to analyzing the scope and usability of the tool was used as a case study the NCL declarative language.

Acknowledgements

Mainly, we want to thank our families, who gave us the opportunity to study, to be where we are, who supported us from day one, whether in our achievements, failures and mainly because without their help it would have been impossible to reach these instances.
Also, thank you to those who helped us during this hard work, our Director Mg. Marcelo Arroyo, for proposing to address the issue, and our Director Dr. Francisco Bavera, being both present in each stage, mainly in every mistake, and without whom this work could not have been carrying out.
For my part (Yanel) I want to thank the angel that I have in heaven, "My Grandmother" or rather my second Mom, which helped me at the time to be here today and then from somewhere nursed and gave me strength each day to fulfill this achievement.

1 Introduction

This work aims to develop a solution to validate semantically languages programming or the like based on XML.

In **XML** (Extensible Markup Language), XML files are text files where symbols "greater than" and "less than" are used to delimit the brands that give structure to document. Each brand has a name, let's see an example:

The brand <figure>, may have one or more attributes:

```
file="foto1.jpg" <figure tipo="jpeg">
```

In this case you have two attributes, "file" and "type".

The attributes have values that have to be in quotes or apostrophes, the marks cannot be left open, i.e., for each brand, for each brand, <*figure*> for example, there should be a mark </ *figure*> indicating where it ends for the content of the brand. That a document is "well formed" only refers to its *basic syntactic structure* is say, which is composed of elements, attributes and XML comments as specified to be written. Now, each XML application, i.e., each language defined with this technology, need to specify what exactly the relationship which must hold between items present in the document.
Some ways to verify that an XML file is well formed, is by:

1. *Document Type Definition or DTD:* Defines the types of elements, attributes and entities permitted, and can express some limitations to combine. XML documents that conform to your DTD are called valid.
2. *XML Schemas:* A Schema is similar to DTD defines which may contain elements XML document, how they are organized, what attributes and what type may have their elements.

Between the two types of verification, there are several fields that are not covered, and they are very useful for verify programming languages based on XML:

1. Semantic relationship between content attributes and tags.
2. Importing variables (and type) of another file.
3. Semantic verification of file formats to variables.
4. Statements Scopes and variable scope verification.
5. Conditional checks the contents of an attribute.

Some of these can be done in a very basic way using regular expressions in XML Schemas, but the problem is limited to a particular attribute and *not* the relationship between several.

This raises, the need for a tool for verifying programming languages based in XML or similar in the semantic aspect. In short, to develop an application that can extend verification of validity that provides XML with the above points and more.

To apply our tool to a real problem, it was decided to use it to validate semantically source code of declarative language NCL.[1]. Create NCL applications is not very difficult for people in the field of programming, requires prior learning which is difficult because such language has only one basic data type ("String") which is used to reference to any basic entity or media item (video, image, etc.), and interpretation tools Ginga-NCL do not provide basic facilities of a modern compiler for relevant reports to the user from their errors.

[1] Gomes Soares, Luis Fernando: "Programando em NCL": Desenvolvimento de aplicações para middleware Ginga, TV digital e Web. (2009).

In this way and as previously mentioned, according to our knowledge, "ASLX" is the first semantics tool for the NCL language and one of the few semantic tools for XML-based languages or similar.

1.1 Development Objectives:

Considering that:

- It is very complex to find errors in a program if the compiler does not help.
- Find and correct errors confusing error outputs watching is very productive.
- That does not even cover all aspects of verification to verify if a file XML is well formed.

Main objectives are proposed, creating a semantic validator detected in the lower long as possible the different semantic errors and / or syntactic language can have a programming based on XML.

The tool can display a detailed report of the errors, thus this tool can be of great help to many developers at the time to correct their applications, since providing a detailed analysis may go directly to the point of failure and fix it. Another feature that should have the program, is to be used as a library (so that for example an IDE can leverage the speed and advantages of this solution).

Moreover, considering that Ginga-NCL does not verify semantics, such language was used as a case study and test, to thereby verify the semantic concepts (like well as in the syntax) that verifies Ginga-NCL alone.

1.2 Using a Scripting Language

A scripting language is a type of programming language that is generally interpreted (as opposed to compile). A script can be seen as a program that can accompany a document HTML or contained inside. The scripts remain in its original form (your source code Text) and are interpreted command by command whenever running.

Scripting language features:

1. Scripts are usually written more easily, but at a cost of its execution.
2. Usually implemented with interpreters rather than compilers.
3. They have strong communication with components written in other languages.
4. The scripts are usually stored as plain text.
5. The codes are usually smaller than the equivalent in the language of compiled programming.

In developing this tool, the Scripting language to create semantic rules are designed with the aim of providing comprehensive solutions to different problems that only verify semantics of a single language (eg Ginga-NCL). Through this mechanism, it can be adapted our program to verify semantics of programming languages based on XML.[2]

A Script begins with a tag called *"semanticParser"* which has reference to the schema script. Must also be specified using the attribute "fileFormat" a list (separated by commas) of extensions this script supports to validate.

Then, within the tag *"rules"* specifies all the rules. For this example we want to set a set of rules that should be applied only to tags labeled with the name "media". With this objective stated in the "name" attribute on the tag "tag" value "medium".

[2] For more information on scripting languages and their use in this tool, refer to Chapter II of the full report of the developed tool "ASLX".

2 Design

2.1. General Design

The development process is centered on a defined architectural with rules developed in a script based in XML. First developed an XML schema *"ScriptParserSchema.xsd"*, in the defined a language for creating scripts with the semantic rules to check.

Given the complexity, it was decided to design incrementally. Our program takes the script and a code based on XML as a parameter, follow the rules defined in the script and try to validate the XML code reporting errors or warnings.

Subsequently, we studied the specific failures of Ginga-NCL for rules and / or tools needed to incorporate.[3] In the case of Ginga-NCL designed a script, *"semanticCheck.xml"* with all the rules semantics of the same so that by giving one or more files with NCL code, the program can verify and create a report. This is defined by regular expressions and rules (applied to "Tags") the conditions to be met by any program NCL to be a valid application free the semantic errors.

The program was divided two sections: "*Parser"* and "*Validator"*. If no syntax errors, the Parser interprets the rules defined in the script and generates a Validator results. This Validator is used to check the rules on one or more files with the format specified in the script.

The basic set of rules that recognizes Parser is:

1. Existence of an attribute with a given name.
2. Right contents of an attribute or content right relationship between various attributes.
3. Check existence of local or external files.
4. Creating objects / variables referenceable.
5. Type checking (for references to objects / variables created).
6. Include code from other files.
7. Check text content of an XML node.
8. Conditional validation rules.

2.2 Structure and Course of an XML file

The design for parsing XML files based on the use of API *"SAX"* and *"DOM"*. These APIs are two tools used to analyze XML and define the structure of a document, although there are many others.

2.2.1 Using DOM and SAX in the design

Because SAX is more complete for information storage, and DOM is simpler to use since there is no need to implement a tree like structure data, we decided to combine both.

By SAX modify the XML file parsed nodes and we add location (for which line of code is the node). Using DOM created the tree data structure to tour files easily without having to implement anything. Also used SAX to validate XML files as one XML Schema, useful to verify that the script and files to be verified with a base stable.

[3] Gomes Soares, Luis Fernando. "Programando em NCL": Desenvolvimento de aplicações para middleware Ginga, TV digital e Web. 2009.
Associacao Brasileira de Normas Técnicas. "ABNT NBR 15606-2": Televisão digitalterrestre – Codificação de dados e especificações de transmissão para radiodifusão digital.

3 Implementation

3.1 Introduction

As programming language Java 7 was chosen only by two important features:

1. Write software on one platform and run it on virtually any other platform.
2. Simple to produce code quantity and quality (productivity).
3. *SAX* and *DOM* are included in the standard library.

For technical implementation details, refer to the technical documentation of the code (Javadoc or code). In this section we discuss only general details of the source code.

3.2 Implementation Details

The entire project was created under the IDE NetBeans[4], which facilitates file *"build.xml"* to compile your code without the need to install this IDE.

Using Apache ant[5] can run the file "build.xml" and automatically generate Javadoc documentation and executable JAR extension.

Since the design is very simple because it uses DOM and SAX to parse the scripts and validate files, only discussed in this section general details of the most important classes of program. For details, refer to the technical report or the source code.

3.2.1 Parser (ScriptParser.java)

You cover all valid nodes recursively with the defined rules (script) using an algorithm.[6] Whenever the node name matches the name of a reserved word of the script, it acts according to the meaning of it by creating rules, ER-Ex (patterns) or containers of rules associated with a tag (TagValidator.java). To facilitate the implementation and script writing, all white characters of ER-Ex will be ignored. If needed them, using suitable escapes characters.

To solve the macros of the ER-Ex, the parser tries to do the translation of the macro only one time, and stores the results. Since a macro always comes down to the same ER, is efficient only calculate this reduction only once. This allows more efficient management of memory and speed parsing. Also added is a limit to the calculation of the macros, as solving recursively ill-defined macro can stop the program in a state of infinite calculation. After obtaining the ER and rules, this returns a Validator

3.2.2 Container Rules (TagValidator.java) and Validator (Validator.java)

Represents a set of rules only applicable to a particular tag. They are stored rules created by the script ready, for easy retrieval. It all nodes recursively valid XML file being checked by the algorithm, and if the name of some of the nodes are in the container list of rules (TagValidator) apply all the rules contained therein to validate that node / tag.nodes in the container list of rules (TagValidator) apply all the rules that thiscontains to validate that node / tag.

[4] http://netbeans.org
[5] http://ant.apache.org
[6] Algorithm described in the section "General Program Design", Chapter II of the full report of the developed tool "ASLX".

4. Testing

4.1 Introduction

The tests performed to verify the correct operation of the program, were of “Black Box”. *JUnit* was used to design a set of test cases and found as it implemented would program the outputs are exactly as expected. The most complete test case is the demonstration of the script with validation rules semantics for Ginga-NCL language which uses most of the functionality of the program in a real case. All tests are described next to the source code and technical reports.

4.2 Optimizations

The first tests performed were of optimization. The evaluation was specifically related to the response time of the main data structures and management of ER-Ex. We implemented five versions of the program, which were evaluated with 4 different scenarios.

The 5 versions had the following characteristics:

Version	*Characteristics*
1	No optimizations
2	The ER equal solved only 1 time
3	The ER-EX Macros are solved only 1 time
4	Macros and ER are stored in HashMap instead of SortedMap (TreeMap)
5	HashMap across data structure that requires searchesin Validator and Parser

Each version has included optimizations of the previous version.

All performance measures were obtained on a Notebook with an Intel ® Core ™ i7-2630QM CPU@2.00GHz × 8 and 4Gb DDR3 RAM under Ubuntu 10.04 64bit (Linux kernel 3.2.0-25-generic).

The results were:

Version 1

Stage	The average time (ms)	ER created	ER reused	Macros analyzed	Macros reused
1	13	4	0	7	0
2	938	5000	0	0	0
3	2477	5000	0	0	0
4	17202	20	0	2334634	0

Three versions were made with the same number of stages, which will be omitted in this report.[7]

[7] For more details refer to Chapter V of the full report of the developed tool "ASLX".

The following shows in detail the latest version and optimizations with respect to the first:

Version 5

Stage	The average time (ms)	ER created	ER reused	Macros analyzed	Macros reused
1	6	4	0	4	3
2	700	5000	0	0	0
3	166	100	4900	3	5049
4	2717	10	10	104	2

This version besides being faster (in some cases) use less RAM than other versions, reason why it was chosen as the implementation.

4.3 JUnit and Test Scenarios of Script Language

For proper implementation incrementally, we chose to go to implement the program validating it through the JUnit[8] framework. JUnit return to class method successfully passed the test, if the expected value is different from that returned during the execution method, JUnit failure to return a corresponding method. JUnit is also a means of controlling regression testing, necessary when part of the code has been modified and you want to see the new code meets the above requirements and has not altered its functionality after the new amendment.

4.4 "Testing Case

4.4.1 Syntax Errors in XML Schema of Ginga-NCL.

For the testing phase, extracted oficial book[9], a total of 33 Ginga XML schemas-NCL. A Below is a detail of the scheme that had errors, the page which can be found in oficial book, a detail of the errors and possible solutions that could be implemented, taking into tool has developed "ASLX" to thus be valid applications.

"CausalConnector.xsd": The scheme is located on page number 48 of the official book.

Code where the error (line 40):

```
<complexType name="nclType">
     <complexContent>
          <restriction base="structure:nclPrototype">
               <sequence>
                   <element ref="structure:head" minOccurs="0"
maxOccurs="1"/>
                   <element ref="structure:body" minOccurs="0"
maxOccurs="0"/>
               </sequence>
          </restriction>
     </complexContent>
</complexType>
```

[8] http://www.junit.org//

[9] Gomes Soares, Luis Fernando. “Programando em NCL”: Desenvolvimento de aplicações para middleware Ginga, TV digital e Web. 2009.

Error type:

```
cos-particle-restrict.2: Forbidden particle restriction:
'choice:all,sequence,elt'.
```

Solution
Defining the maximum of occurrences (maxOccurs) body structure is 1 instead of 0.

4.4.4 Syntactic errors in examples official Ginga-NCL.

We extracted all the code examples from the book mentioned above for official to use them as evidence. In this way we obtained a total of 20 examples which first syntactic errors were corrected.

Beyond that the tool developed to detect syntax errors to achieve fully valid applications, as the main objective is the semantic, are omitted in this report[10].

4.4.5 Semantic errors in examples official Ginga-NCL.

Were detected errors in the following examples Ginga-NCL. These errors were found by the validator semantic created. As noted, various examples were no syntax errors, if they have them in the semantic aspect. In the absence of the tool created, these examples would be considered valid when in fact they are not.

"Example 1": This example is the initial release of the NCL document "O Primeiro João". This example can be seen on page number 51 of the official book.

Code where the error (line 36):

```
file:Ejemplo1.ncl: line:36: The symbol "conEx#onBeginStartDelay" is
not
declared in this scope.
file:Ejemplo1.ncl: line:45: The symbol "conEx#onBeginStart" is not
declared in this scope.
file:Ejemplo1.ncl: line:49: The symbol "conEx#onBeginStart" is not
declared in this scope.
file:Ejemplo1.ncl: line:53: The symbol "conEx#onEndStop" is not
declared
in this scope.
```

Solution
CausalConnBase.ncl Define a new file, which contains each of the connectors for compiling the examples present in Ginga-NCL. In this case specifically defined in the file, the connectors *"onBeginStartDelay"*, *"onBeginStart"* and *"onEndStop"*.

[10] For more details refer to Chapter V of the full report of the developed tool "ASLX".

Below is a table show examples present in the official book of GINGA_NCL possessing semantic errors:

Example	Description	Error line	Error type	Solution
Example 3.15 (Page 56)	Application to sync by interaction	41-50-54-58-63-67-77	Elements are not declared In this scope.	Define the file causalConnBase.ncl the connectors.
Example 3.19 (Page 63)	Application to Reuse	43-47-57-64-73-77-81	Elements are not declared In this scope.	Define the file causalConnBase.ncl the connectors.
Example 3.22 (Page 67)	Application to channel interactive	47-51-62-69- 78	Elements are not declared in this scope.	Define the file causalConnBase.ncl the connectors.
Example 3.27 (Page 73)	Application with content adaptation	60-64-75-82-91- 95-99	Elements are not declared in this scope.	Define the file causalConnBase.ncl the connectors.
Example 3.32 (Page 79)	Application-controlled interactive ads.	53-60-66-70-98-100-103-114-121-130- 134-138	Elements are not declared in this scope. Invalid value in attribute "role".	Define the file causalConnBase.ncl the connectors. Give a valid value to the attribute "interface" and to attribute "role".
Example 3.37 (Page 85)	Application purposes animations and transitions	60-67-77-80-85 90-120- 127-136-140	Elements are not declared in this scope. Invalid value in attribute "interface".	Define the file causalConnBase.ncl the connectors. Give a valid value to the attribute "interface"
Example 3.45 (Page 95)	Application menu with navigation keys	72-79-89-117-122-133-139-167-174-185-194-198-207-213	Elements are not declared in this scope.	Define the file causalConnBase.ncl the connectors.
Example 3.52 (Page 105)	Application in order NCLua	60-74-81-91-120-122-125-136-142-156-175-185-199-208-212-221-229	Invalid value in attribute “end” y “role”. Elements are not declared in this scope.	Give a valid value to the attribute "interface" and attribute “role” Define the file causalConnBase.ncl the connectors
Example 7.1 (Page 144)	Example showing the interactivity of a "button" for a fixed time	18	The symbol "dTVtelaInteira" is not declared in this scope <even forward>.	Writing the shape descriptor correct.

Developed tool also found errors in the examples: "10.12" (page 199), "10.14" (page 202), "12.4" (page 234), "14.4" (page 259), "14.6" (page 268), "15.3" (page 272) "15.6" (page 274).
In case you want to see in detail this type of error, refer to Chapter V of the full report tool developed "ASLX".

References

1. Gomes Soares, Luis Fernando: “Programando em NCL”: Desenvolvimento de aplicações para middleware Ginga, TV digital e Web, (2009).
2. Associacao Brasileira de Normas Técnicas: “ABNT NBR 15606-2”: Televisão digitalterrestre – Codificação de dados e especificações de transmissão para radiodifusão digital.
   Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações (2007).
3. Gomes Soares, Luis Fernando and Ferreira Rodrigues, Rogerio: “Nested Context Language 3.0 , NCL Digital TV Profiles” (2006).
4. Soares, Luiz Fernando Fernando Gomes: “Construindo Promgramas Audiovisuais Interativos Utilzando a NCL 3.0 e a la Ferramentea Composer” (2007).
5. Zambrano, Arturo: “Introducción a la TV Digital Interactiva y Ginga.ar”- Universidad Nacional de La Plata. La Plata – Argentina.
6. Balaguer Federico & Isasmendi, Leonardo: “Desarrollo de Aplicaciones para Televisión Digital”. Universidad Nacional de Río Cuarto. Rio Cuarto - Argentina (2011).