# A Programmatic View and Implementation of XML

**Ghassan ElNemr, Ph.D.** *ghassan.nemr@lcu.edu.lb*
*Head of CCE department*
*Faculty of Engineering*
*Lebanese Canadian University*
*Aintoura, Lebanon*

**Pierre Gedeon, Ph.D.** *pierre.gedeon@lcu.edu.lb*
*Campus Director*
*Faculty of Engineering*
*Lebanese Canadian University*
*Aintoura, Lebanon*

**Abstract**

XML as a markup language defines rules to encode data in a free format comprehensive by both human and machines. Usage of XML as a support for data integration, file configuration and interface definition is widely adopted and implemented by the software industry community.

The purpose of this paper is to examine an implementation of XML as a programming language, extending the capabilities offered by frameworks and simplifying the coding tasks. The code becomes a set of functions sharing the same pattern all written as XML parts. The defined language takes advantage from the predefined common libraries and provides a mean to invoke handlers from user interface components. Programmers take benefits from the simplicity of this language to apprehend quickly the logic implemented by a function, which result in an increase in maintenance quality and rapid development stability.

**Keywords:** Software Quality, Developer Productivity, XML Programming, Framework Extension, Coding XML, Easy Maintenance.

## 1. INTRODUCTION

Billions of dollars are spent every year for building and maintaining software. External quality refers to users' perception and managers' evaluation of developers work. The number of defects and the maintenance efforts to adjust code are two main indicators considered to measure software quality [1]. Software reliability is clearly important and mandatory, however it is costly. For many applications, the increase of reliability go through an incremental increase in programmer effort mastering a programming language and coding with it [2].

The complexity of a programming language has direct impact on the developers' productivity. The recommended principles of language design include: readability, reliability, simpler syntax, portability, no name hiding, eliminate machine dependencies, graceful degradation, and support for reuse.

But what is a programming language? [3] Defines it as "A set of commands, instructions, and other syntax use to create a software program". Harper in [4], Programming languages express computations in a form comprehensible to both people and machines. The syntax of a language specifies how various sorts of phrases (expressions, commands, declarations, and so forth) may be combined to form programs.

This last principle will guide this work. We will present a reliable programming language, based on XML, simple to understand and learn, reliable and extensible.

## 2. WHY GENXML?

One of the most important indicators to measure a programming language complexity is the number of keywords. A keyword is a word that is reserved by a program because it has a special meaning. C++ has 92 keywords, while Java has only 50 [5], programmers have less symbols to learn and less to manipulate.

The control flow of a program defines its procedural structure [6]. Hence the structure of a program is built from block of code or units having a single entry and single exit in the control flow. The purpose is here to offer to the programmer a simple programming syntax allowing to write code fast. It is not to write a full programming language, having its own syntax and compiler such as o-xml where all of polymorphism, function overloading, exception handling, threads are offered [7], but to offer a mean for developers to standardize and simplify the code structure while taking benefit of a well-known framework such as Microsoft.Net.

This generic syntax must offer all the components of a program: declarations, operators, arrays, branching and loops, scope of variable and input/output primitives as well as function/procedure calls and system calls.

A generic XML based instruction form is

$$<\text{tag } attrib_1=value_1\ attrib_2=value_2 \ldots attrib_n=value_n > \ldots </\text{tag}>$$

The attributes names and values provide the support for all the program components. Hence the program becomes a sequence of tagged text specifications, read by an interpreter in run time.

Similarly to programming languages C++, C#, java and php, we have chosen to enrich genXML by a graphical library. The generic instruction form is used to make calls to the library.

### 2.1 Flow of Execution

Instruction respecting the generic format are written in text files. Then the text file is loaded upon an event and parsed. The type of the instruction along with a reflective process that determines if the attributes are well formed and linkable to the underlying framework. In this first implementation we have opted to use .NET framework since it offers the reflection namespace necessary to link our instructions to system calls and offers the necessary memory management tools and structures such as hash maps, lists, and array support.

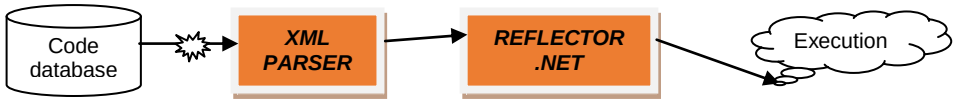


**FIGURE 1:** Execution of GENXML Instruction.

As shown in Figure1, the code is hosted in a relational database rather than in files. This allows an easier maintenance of the code and allow a continuous run of the application without restarting it. Each script is assigned a name and an XML text. The XML text is first loaded from the database when an event occurs (button click, load, grid selection and others) then parsed and verified. The next step is to map the tag to a method call over object created in the .NET environment [8].

Events that load the code from the database are: time triggers, user events such as click or select line in a grid, and application event. The Parser verifies the well formatting of the xml program and calls the reflector which has two principal roles:

- Loader: the tags determine the type to instantiate so the corresponding libraries are loaded.
- Linker: the declared objects are added to the application context. In a later time, the application would use the newly declared objects referring to their name preceded with the char %

Once the reflector evaluates the different parts of the function tag, it executes the instruction.

### 2.2 Generic Instruction Syntax

Learn Curve decrease necessitates two factors: Reduce the number of keywords, and simplify the syntax. Tags are representative of an operation, while attributes of a tag specify the parameters of the function. As an example a declaration of a variable dbac of type database connection is done through the line:

```
<Line Id="1" Operation="FunctionCall">
   <Function id="1.2" fname="Declare" name="dbac" type="GenDataLayer.DbAccess" />
</Line>
```

**FIGURE 2:** GENXML Instruction: Declare A Variable.

A line may be a set of Functions tags with an id. The attribute fname is equal to Declare, which means a declaration of variable, and the type of the new variable named dbac is DbAccess from the library GenDataLayer.

The same logic applies when the operation is a method call. The following line:

```
<Line Id="1" Operation="FunctionCall">
            ……..
   <Function id="1.3" fname="dbac.changeDataConnection" param1="%dbName" />
</Line>
```

**FIGURE 3:** GENXML Instruction: Call Object Method.

Expresses a call to the method changeDataConnection on the object passing the content of the variable dbName.

From the preceding, we note that a declaration, or a function call are coded with the same pattern. Subsequent programming instructions follow this pattern. We retain here that one tag <function> is sufficient for both declarations and method call.

The reserved keywords/tags here are tag function, and the attributes of the tag: fname, name, param1, param2, out. The values assigned to these values will serve to determine which objects to call, which methods to evaluate and which controls in a user interface are affected.

### 2.3 Language Components

First implementation of GenXML is done using .NET framework. Hence .NET classes are available through the reflection namespace and classes are used in XML instructions to be instantiated. Data structures such as Hash maps, Dictionaries, and Arrays are used as instances so their methods are also callable through xml instructions.

#### 2.3.1 Declarations and Objects

In Figure 2, the attribute fname=”Declare” specifies a declaration operation. All classes variables may be declared using the generic instruction format

<Function id="1.2" fname="Declare" name="*varname*" type="*vartype*" />

A new object of the type vartype shall be instantiated, then added to the application context as referred by the variable varname.

### 2.3.2 Operators

Operators are implemented through method call over an object instance of a .NET class, or a class defined in an extension library as in the sample script:

```
<Function id="1.1" fname="Declare" name="p1" type="System.Int16" param1=”2”/>
<Function id="1.2" fname="p1.Parse" param1=”3” out=”p2”/>
<Function id=”1.3" fname="Operators.add" param1=”%p1” param1=”%p2” out=”p3”/>
```

**FIGURE 4:** GENXML Add Operator.

Where Operators is an extended library offering the methods the functions add, substract, multiply, divide, modulo, remainder, and, or, xor. This library may be extended with new operators.

### 2.3.3 Arrays

Operators are implemented through method call over an object instance of a .NET class, or a class defined in an extension library. In the sample statement:

```
<Function id="2.3" fname="dbac.getArrayList" out=”arr”/>
```

getArrayList takes no parameters returns an object of type ArrayList, referenced in the application context by the name arr. All other functions may use the arraylist object through a call

```
<Function id="1.3" fname="arr.Add" param1="%par1" />
```

**FIGURE 5:** GENXML Arrays.

### 2.3.4 Conditionals and Loops

The execution of the script is done using a parsing of the XML document tree, depth-first navigation. GenXML allows nested blocks in if function tags like.

```
<Function Id="3.5" fname="If" Condition="(%dv_UserRolePasswordValid.Count &gt; 0)">
<Then>
…….
</Then>
<Else>
…….
</Else>
</Function>
```

**FIGURE 6:** GENXML If Block.

The condition is here expressed as the count property of a data view variable. The then block is a starting of a new sequence of tags that may be including another if block. Loops are expressed by the tag <Loop> as in the example:

```
<Function id="5.14" fname="While" Condition="(%nbColumns &gt; 1)">
  <Loop>
  <Function id="1.4" fname="par4.setValue"
param1="%grdStudentCourseGrade.Columns.[%nbColumns].Binding.Path.Path"/>
  <Function id="1.5" fname="dbac.getDViewFromProc" param1="SR_getHeader" param2="%arrEx"
out="dv_lstColHeader" />
  <Function Id="1.6" fname="If" Condition="(%dv_lstColHeader.[0].[defined] &gt; 0)">
  <Then>
    <Function id="1.7" fname="grdG.Columns.[%nbColumns].setProperty" param1="Header"
param2="%dv_lstColHeader.[0].[0]" />
```

```
    <Function id="1.8" fname="grdG.Columns.[%nbColumns].setProperty" param1="Visibility"
param2="Visible" />
   </Then>
   <Else>
    <Function id="1.9" fname="grdG.Columns.[%nbColumns].setProperty" param1="Visibility"
param2="Hidden" />
    </Else>
   </Function>
   <Function id="2.0" fname="this.setCalculated" param1="nbColumns" param2="%nbColumns - 1" />
  </Loop>
</Function>
```

**FIGURE 7:** GENXML Loops.

While instructions blocks begin with a generic function tag format, and include a loop tag block that delimits the boundaries of the while block. In the sample, the while condition is executed as long the variable nbColumns is greater than 0. This variable is decreased at each iteration by the function id = 2.0.

### 2.3.5 InputOutput

GenXML as XML specification language doesn't recognize directly any output. It relies on variables properties to read and write values. Those variables being defined as application context objects, such as components in a form, it becomes easy to set their properties across method calls.

A direct consequence of this principle is the current implementation of GenXML on top of .NET framework generally and WPF [9] more specifically. The good news here is that WPF is xml based and used a syntax called xaml, which specifies the visual tree of any form as xaml script. GenXML applications start then by considering all the visual tree components as objects in the global context, which allows the scripts to act on the components appearance as well on their contents. The function 1.9 in the figure 7 shows a hide of a grid column depending on a value read from a database. Function 1.7 shows the update of a column header to a value specified in the database, read through a function call.

## 3. IMPLEMENTATION

In a first real-time implementation, we have developed a student management system (SIS) based on GenXML. The overall effort for development has shown the following:

- Two developers were needed to implement the code in a period of six month (development effort)
- The learning curve of the coding practices and style was reduced to few hours (learning curve)
- Since all the code is similar, developing a new function using GenXML was reduced to a code duplicate followed by adjustment. The percentage of the adjusted code is less than 20 % leading to a cost minimization (cost reducing)
- The maintenance of the code is simple since the application is organized in group of GenXML scripts (modularity).

A quick comparison with the modern methods to evaluate the cost of development shows that the impact of the chosen language is important. Function Points (FP) [10] introduced by Albrecht define an empirical estimated of the needed effort, where the Lines of code (LOC) measure the quantity of lines of code needed to implement a given functionality.

The language complexity has the final word on the determination of a program complexity. The relation LOC/FP [11] depends greatly on the language complexity where 320 lines of code are necessary for each function point if the assembly language is used. This number shuts to 30 if an

object oriented programming language is used. In the case of GenXML, we noticed an average estimate of 15 Lines of code per function point, which is dividing the effort of programming by 2.

## 4. CONCLUSION

In this paper we have introduced GenXML, a programming language using XML syntax and inheriting the benefits of .NET framework. The simplicity of the language has been shown through examples citing the components of a simple syntax that reduces the flow of execution of a program to the interpretation of a uniformly expressed sequence of xml tags.

Commercial software take benefits of this extensibility and simplicity. Code is stored in a relational database and loaded on demand. This feature makes the update of scripts easy and run independent where developers may update an application code while code is run.

In the current implementation, WPF and .NET are used to build applications. However the extension of supported framework will result in a portability of the same code to different environments such as spring framework [12][13].

## 5. FUTURE WORK

In the first implementation of GenXML, we have opted for an integration with .NET framework which is tied closely to windows operating system. Future implementation of this platform will be based on Java, which will provide Operating system portability. Reflection in Java will be extensively used.

The storage of the XML scripts actually located in the databases shall be improved to implement a caching mechanism allowing to load the graphical interface expressed in xml faster. In this area we will examine the modern caching mechanisms to increase the performance of the system.

Finally we will improve the dynamic loading of external libraries to extend the capabilities of GenXML.

## 6. REFERENCES

[1] Assessing programming language impact on development and maintenance: a study on C and C++. Pamela Bhattacharya && Iulian Neamtiu, Proceeding ICSE '11 Proceedings of the 33rd International Conference on Software Engineering Pages 171-180.

[2] Designing Programming Languages for Reliability. Harry H. Porter. CS Department, Portland State University, 2001.

[3] WWW: Programming Language. https://techterms.com/definition/programming_language. Accessed Dec 2018.

[4] Robert Harper. Practical Foundations for Programming Languages. Carnegie Mellon University. 2016 pp 334.

[5] WWW: Reserved Words of Programming languages. https://halyph.com/blog/2016/11/28/prog-lang-reserved-words.html. Accessed Dec 2018.

[6] Evaluating and Mitigating the Impact of Complexity in Software Models. Delange, Hudak, Nichols, McHale and Nam. Technical report CMU/SEI-2015-TR-013. Carnegie Mellon University, Software Engineering Institute. Dec 2015.

[7] WWW: o-xml the object oriented programming language. https://www.o-xml.org. Accessed Dec 2018.

[8] Meta Programming in .NET. Kevin Hazzard and Jason Bock. Manning Shelter Island, 2013, 365 p.

[9] WPF4.5 Unleashed. Adam Nathan. Pearson Education. 2014. 847 p.

[10] Measuring Application Development Productivity. Albrecht, A., Proceedings of IBM Application Development Symposium, October 1979, 83-92.

[11] Software Function, source Lines of Code and Development Effort Prediction: A Software Science Validation. Albrecht A., Gaffney J., IEEE Transactions on Software Engineering 9(11), November 1983, 639-648.

[12] Spring Framework Reference Documentation. Rod Johnson & authors. Spring framework official documentation, 2015, 904 p.

[13] https://docs.spring.io/autorepo/docs/spring/4.3.0.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf. Accessed Dec 2018.