ISSN: 2350-0743

www.ijramr.com

International Journal of Recent Advances in Multidisciplinary Research

Vol. 06, Issue 04, pp.4847-4851, April, 2019

RESEARCH ARTICLE

A UNIVERSAL PROGRAMMING PLATFORM BY EXTENDING XML

***1Ghassan El Nemr and 2Yara AlAli**

1Head of CCE Department, Faculty of Engineering, Lebanese Canadian University, Aintoura, Lebanon
2Technical Director, Key In Hands Corp. Damascus, Syria

ARTICLE INFO

Article History:
Received 07th January, 2019
Received in revised form
09th February, 2019
Accepted 11th March, 2019
Published online 30th April, 2019

Keywords:

Programming Platform,
Development Cost, XML Programming,
Framework Extension, XML Coding,
Maintenance.

ABSTRACT

XML is everywhere: Data Integration, File configuration and Interface definition are areas where XML is intensively present. We may define rules, constraints and configuration settings, or specify graphical interfaces, export data and define rules. This markup language allows the extensibility of any data specification, graphical interface design, or settings storage in an easy and flexible way, which allowed a wide usage of XML in the industry community. Programmers write programs using a learned syntax that depends of a specific language. Languages may share the same paradigm but are different in their syntax and reserved words. Productivity depends of the chosen language, and the developer knowledge of it. Programmers switching from a language to another may build on the paradigm knowledge, however they must learn a new syntax since no flexible universal programming syntax is offered. The purpose of this paper is to offer to the development community a uniform programming platform based on XML, and running on top of different baseline languages and frameworks. By joining a simple language specification to a powerful framework, we reduce the set of required reserved words and syntax mechanisms to learn in a language. We propose a programming platform that increases the productivity while reducing the learning time and error rate. This results in an increase in maintenance quality and rapid development, and code readability.

INTRODUCTION

Learning new programming languages is highly expansive: Billions of dollars are spent every year in learning new programming languages syntax, extend software written in specific languages, and maintaining legacy code software. The quality of a software is determined by the number of defects in one hand [1], and the maintenance efforts that are invested to maintain the software in production. Hence Software reliability is mandatory and related closely to the used programming platform, it remains very costly given many facts: One is technical resources turnover, which introduces a hidden cost in the overall maintenance global effort, another one is the complexity of adding or modifying a module in a running environment. For mostly all applications, the increase of reliability goes through an incremental increase in the overall programmer effort, who is supposed to maintain a high level of knowledge in a specific programming language with a specific programming paradigm, and coding with it [2]. The productivity of developers is affected in principle by the complexity of the tasks they have to implement and the complexity of the used programming language. Modules that developers are expected to develop include the design and development of a Graphical User Interface, Core components, Data persistency scripts, and business tasks. The recommended principles of language design include: readability, reliability, simpler syntax, portability, no name hiding, eliminate machine dependencies, graceful degradation, and support for reuse. A programming language [3] is defined as "A set of commands, instructions, and other syntax use to create a software program". Harper in [4] stipulates that "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. These considerations will guide the scope of this work. We will introduce a reliable programming language, based on XML, capable of defining user interfaces, link the user events to the core modules, simple to understand and learn, reliable and extensible.

***Corresponding author: Ghassan El Nemr,**
Head of CCE Department, Faculty of Engineering, Lebanese Canadian University, Aintoura, Lebanon

Genxml and Genplatform

We have chosen the name of GenXml (Generic XML) to express the nature of our presented language: generic programming language based on XML. The generic aspect of the language consists in defining a common normalized way of handling user interface design and link the core modules to it, so a developer sees all the code as a sequence of instruction lines that are interpreted in a uniform manner. We have chosen to reduce the number of keywords in GenXML since reserved words are a key factor in measuring a language complexity [5]. C++ has more than 90 keywords, while Java has only 50. The developers' preference goes naturally to Java for all non-embedded type application since they have less symbols to learn, and less structures to manipulate. The procedural structure of a programming language [6] is determined by its control flow of instructions. Languages as Scala [6a] adopt a sequential procedural paradigm, while Java is based on object oriented paradigm. Many blocks with different structure define a program or a module, program will possess a single entry and

single exit in the control flow, while modules define a set of exported functions and structures. The goal here is to offer to the developer the tools to implement and manipulate different kind of entities, objects, and execute sequence of instructions.
Our purpose is to simplify this tasks by offering to the programmer a simple linear programming syntax allowing to write code fast. The approach is different from other languages who start from scratch and are implemented using a rule based grammar [6b], identify tokens and process both syntax and semantic analysis prior to generate an abstract tree and a code machine [6c]. 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. Hence we choose to define a language specification based on stable frameworks such as Microsoft.Net, implement the specification and realize programs that demonstrate the proof of concept in one hand and allow write a complete application in another hand. This platform will be called GenPlatform since it used the GenXML language to define and interpret XML scripts for both user interface definition and code handling. This generic syntax must offer all the user interface design by using visual trees expressed in XML, and offering all the components of a program such declarations, operators, arrays, branching and loops, scope of variable and input/output primitives as well as function/procedure calls and system calls. The combination of visual trees xml based scripts with XML code scripts will define a program written under GenPlatform.

A generic XML based instruction form is

<tag $attrib_1$=$value_1$ $attrib_2$=$value_2$ ... $attrib_n$=$value_n$ > … </tag>

The tag will be the same for all code script definitions, and expresses the graphical component in the case of the visual tree support. The attributes names and values provide the support for all the program components. Hence the program becomes a set of xml streams where each one is a combination of xml sequence of tagged text specifications, read by an interpreter in run time. Similarly to programming languages C++, C#, java and php[7a], we have chosen to enrich GenPlatform with a graphical library offering all the richness of desktop environment. We have based our first specification on XAML[7b] for the graphical part.

Flow of Execution

Code in GenPlatform can be classified in two types:

- Xml files containing graphical specification of user interface
- Xml files containing code instructions.

Many patforms use an xml approach to design and implement the interface. Using Android's XML vocabulary, one can quickly design UI layouts and the screens, and attach code handlers functions to components.

Microsoft. Net does the same with its aspx pages where a graphical namespace is loaded and defines a set of tags, which will be used to express a graphical tree of components. This approach is known to offer a RAD (rapid application design) method to construct interfaces and prototypes quickly through tools and wizards such as android studio or Microsoft Visual Studio. Our approach to define the interface takes benefit from the existing .NET wizard that allows create XAML files from user actions. Hence the assistant uses WYSIWYG approach to visualize user interface design in real time, and modify them consequently. A 3 step process is performed to allow the execution of a GenPlatform application: First the code is scanned from a repository of xml files and parsed to check the well forming of the xml syntax according to the supported file type. If the file is a visual tree implementation then it is loaded in the controller, else it goes through the reflector to be translated to a binary form of library entries calls. The third step is the coordination of code calls and graphical user interfaces so a context will be created to start the execution.
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 [7c].

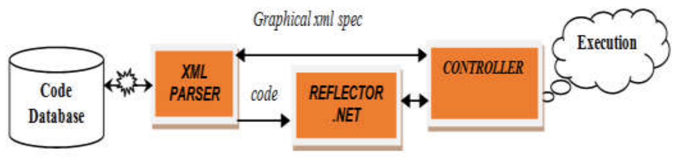

Figure 1. Program Execution in GenPlatform

As shown in Figure1, the code may be hosted in a relational database rather than in files. This kind of data support manages the code as data content and allows an easier maintenance and indexing as well as a continuous run of the application without the need to restart the client applications in case of script changes. Scripts are stored in tables, depending on their type. A script is assigned a name and an XML text, as well as an event handler and a graphical component name to which it is attached. 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].

The code may be loaded from the database when a specific event occurs. Such events are: time triggers, user interaction 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 the responsibility of dynamically load the libraries and provides the xml-equivalent binary handlers to the controller.

The controller has two principal roles:

- **Loader:** the tags determine the type to instantiate so the corresponding libraries are loaded. So the controller instructs the reflector to pull the code from the database according to the flow of execution
- **Runner:** 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 %, so a dynamic context is maintained to insure the well execution.

The controller acts as a code flow regulator where all logging of execution is performed for ulterior trace analysis.

Generic Instruction Syntax

The context maintained by the controller is populated by objects created while the application runs. A common line syntax specifying the generic function form may be defined as:

```
<Line Id="1" Operation="FunctionCall">
<Function id="id" fname="operation" attrib1="val1" attrib2="%val2" attrib3="%val2.prop" out="%ob" />
</Line>
```

As the application runs, the context is enriched with new named objects that will be called in the subsequent ran scripts using the XML syntax. When an *operation* function is called, passing the parameters val1, the content of the object named val2 is obtained through the operator %, and the value of the property *prop* of the object named *val3*. The result will be stored in the context under the name *ob* passed to the function call using the attribute out. The context will be enriched by a new object named ob after the execution of the function. This object may be used later in all functions referencing it as an attribute value preceeded by % sign. The result of such notation is the restraint number of reserved words, and the syntax simplification, which contributes to decrease the learning curve of our language. The total number of symbols is then reduced and normalized to a simple form, so developers get to learn one common syntax and one mechanism that applies to all of declarations, arrays, loops, conditionals and method calls. As an example, a declaration of a variable dbc of type database connection is done through the line:

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

Figure 2. Genplatform 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="dbc.changeDataConnection"
param1="%dbName" />
</Line>
```

Figure 3. Genplatform Instruction: Call Object Method

Expresses a call to the method changeDataConnection on the object dbc passing the content of the variable dbName which is evaluated in the context. Subsequently the object named dbName is called to manage the connection to the database.
The memory manager will keep the references to the live referenced objects as long they are named in the context. There will be a need for an operation to free the object by resetting the name to another object, which will decrease the number of references to the real object and signals its state as disposable to the garbage collector.

We note that all of declarations and function calls share the same function call pattern. Subsequent programming instructions follow this pattern too: all function calls will contain the same function-{att=val}* format, which simplifies the mechanism and familiarized the developer quickly with the code. 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.

Language Components: First implementation of GenXML[REF IJCSS] 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.

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.

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.

Arrays: Operators are implemented through method call over an object instance of a .NET class, or a class defined in an extension library. An array is an object of a predefined dynamic library that may be either loaded as an extension or provided by the hosting environment (which is in this case .NET environment).

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

We note here that the method getArrayList is a defined method, local to our platform. However, the returned object is a .NET framework object, which activates all the calls to its .NET defined methods such as the method Add.

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. The syntax of the If block requires an in-context validation of the block by the Loader module that will insure the well-formed structure of the if-Block prior to its submission to the reflector.

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. Loops and Conditionals take advantage of the hierarchical nature of XML segments to recursively define loops in loops, and conditionals in loops, or loops in conditionals.

Input Output: In general, a process has three streams for data input, output and errors. GenXML as XML specification language doesn't recognize directly any of such. It relies on variables properties to read and write values. These 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.

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.

Conclusion

In this paper we have introduced GenXML and GenPlatform, a programming language using XML syntax and implementation that takes the benefits of underlying .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, and the utility of the platform has been exposed through a first implementation. 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].

Future Work

In the first implementation of GenPlatform, 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.

REFERENCES

A Programmatic View and Implementation of XML, G.EL NEMR, P.Gedeon. International Journal of Computer Science and Security (IJCSS), Volume-13 : Issue-1 : 2019

A Rule-Based Style and Grammar Checker, Daniel Naber, Universität Bielefeld. Thesis. 2003.

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.

Compilers: Principles, Techniques, and Tools, Aho, Lam, Sethi. Addison Wesley 2006, ISBN-13: 978-0321486813

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

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.

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

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

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

Modern PHP: New Features and Good Practices, Lockhart, Oreilly, 2015. ISBN-13: 978-1491905012

Programming in Scala: A Comprehensive Step-by-Step Guide, Odersky, Spoon and Venners, 2nd Edition,2011. ISBN-13: 978-0981531649

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

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.

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

WPF4.5 Unleashed. Adam Nathan. Pearson Education. 2014. 847 p.

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

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

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

XAML Syntax In Detail, Microsoft documentation. https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-syntax-in-detail, 2017.

XAML Syntax In Detail, Microsoft documentation. https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-syntax-in-detail, 2017.

****