

Can a language be before “the first programming language”?

Menyhárt, László

menyhart@elte.hu
ELTE IK

Abstract. I would like to present a potential new language which can be used before „the first programming language”. We can use this to write down the algorithms and source codes can be generated from this. The keyword is XML. This can be used for describing algorithms and for checking the syntax and the semantics easily. Source code can be transformed with XSLT. So the usage of this new language can help us to answer the question, which is the best first programming language?

1. Introduction

I have never entered into the discussion about which the best first programming language is. I would like to stay away from it now, too. But I would like to present a new approach to construe this problem.

There is a potential language which is good for writing down algorithms.

This article can be found on the following URL:

<http://xml.inf.elte.hu/conference/2009/INFODIDACT/>

2. Today’s programming languages

At present we have a lot of programming languages and environments. These can be categorized based on the language, users, target and characteristics. For example there are languages with graphical appearance which are good for children to create games. Or there are some environments with a simple character editor and others with graphical interface. At last but not at least there are the professional languages for using in “business”. There are a lot of new products for helping the children’s learning of programming. For example scratch is a new development.

3. Algorithm Markup Language (AML)

I would like to present a new approach to define algorithms. In my opinion it will not be good for the beginners and for the children but it might be good for those students who can create games and who would like to teach professional languages. It can help them to understand the structure of the programming languages.

3.1. Properties of AML

The XML – eXtensible Markup Language – provides a lot of possibilities. We can check easily the well-formed files. XSD – XML Schema Definition – can check the syntax and the semantics. XSL – eXtensible Stylesheet Language – can help us to transform the source to a lot of format. So we can transform to other source codes and any other text files.

Now I would like to show the proof of this concept (POC) with the following example. Let us see the algorithm of linear seeking.

linearSeeking.xml:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<procedure name="linearSeeking"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="aml_en.xsd">
  <parameters>
    <parameter type="TIndex" mode="constant">N</parameter>
    <parameter type="TArray" mode="constant">arr</parameter>
    <parameter type="TBoolean" mode="variable">exist</parameter>
    <parameter type="TIndex" mode="variable">Which</parameter>
  </parameters>
  <variables>
    <variable type="TIndex">ind</variable>
  </variables>
  <sequence>
    <operation>
      <variable>ind</variable>
      <expression>
        <value>0</value>
      </expression>
    </operation>
    <while>
      <condition>
        <and>
          <expression>
            <value>ind<N</value>
          </expression>
          <expression>
            <not>
              <function name="T">
                <parameters>
                  <parameter>arr[ind]</parameter>
                </parameters>
              </function>
            </not>
          </expression>
        </and>
      </condition>
      <sequence>
        <operation>
          <variable>ind</variable>
          <expression>
```

```

        <value>ind+1</value>
      </expression>
    </operation>
  </sequence>
</while>
<operation>
  <variable>exist</variable>
  <expression>
    <value>ind<N</value>
  </expression>
</operation>
<if>
  <condition>
    <expression>
      <value>exist</value>
    </expression>
  </condition>
  <then>
    <operation>
      <variable>Which</variable>
      <expression>
        <value>ind</value>
      </expression>
    </operation>
  </then>
  <else/>
</if>
</sequence>
</procedure>

```

There is a procedure named “linearSeeking”. This procedure is called with more parameters. For example there is a parameter named “N” with TIndex type and constant mode. There are more variables. For example the type of “ind” variable is TIndex. There are a lot of commands in sequence. The first is an operation; the value of “ind” will be 0. The next command is a “while” with a condition and a sequence of commands. And so on.

This code can be understood more simply with the following figures:

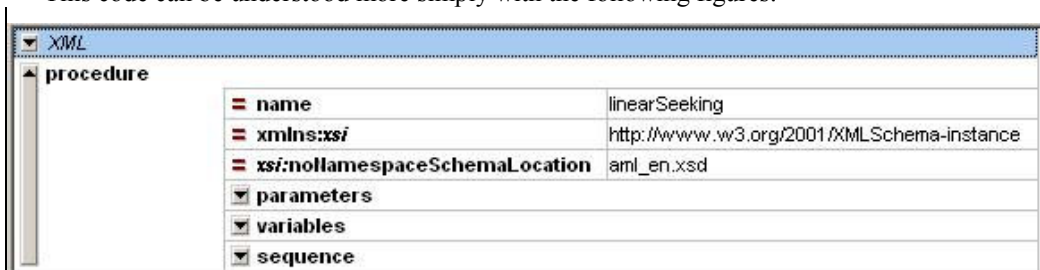


Figure 1: Procedure contains parameters, variables and sequence

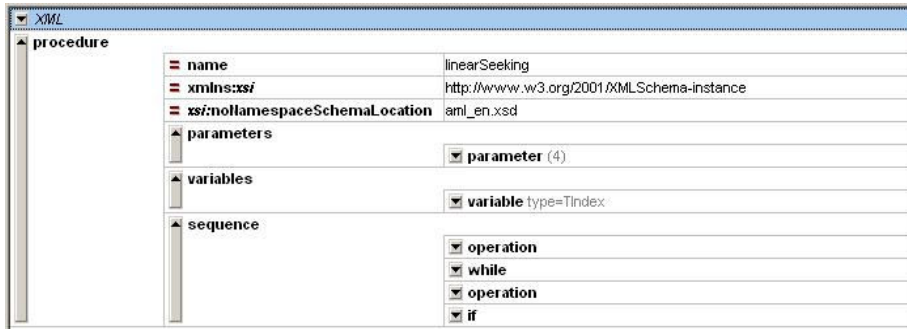


Figure 2: Each node contains other nodes

“Parameters” node contains more “parameter” nodes. “Variables” node contains more “variable” nodes. “Sequence” node contains more nodes.

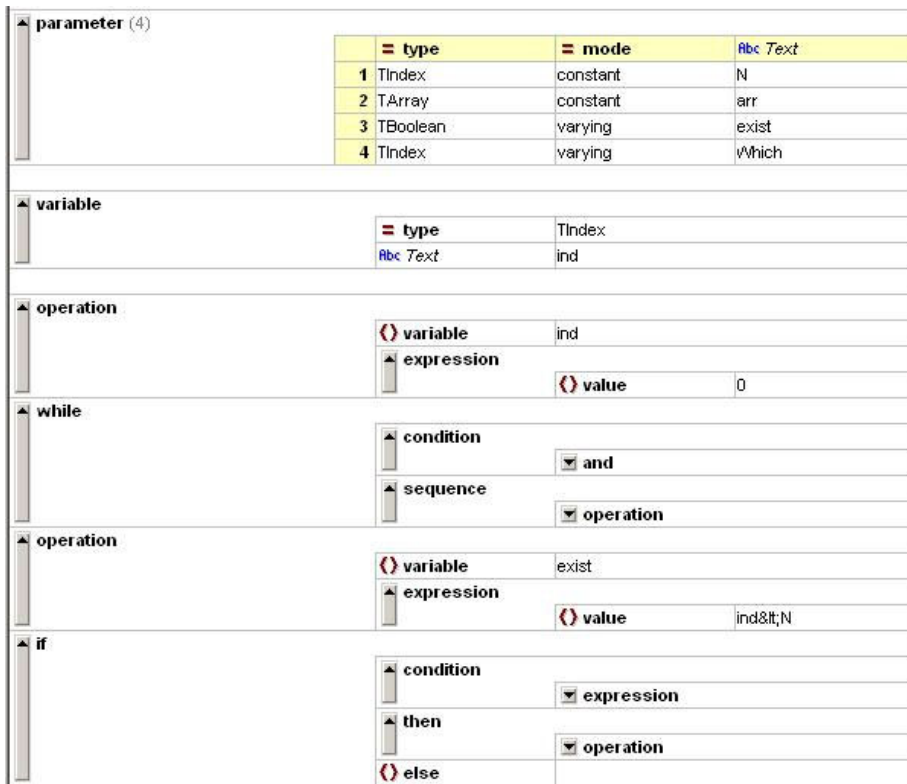


Figure 3: Expanded values

A parameter has a name and two properties. The “type” attribute defines the type of the variable. “Mode” attribute defines how the given parameter is handled in the procedure. The value can or can not be modified.

A variable has a name and “type” attribute like the parameter.

Can a language be before „the first programming language”? (AML)

There are four commands in the sequence.

- The operation has a variable (left side) and an expression (right side) which contains only a “0” value now.
- The “while” node has two parts. The first is the “condition”, the second is another sequence which is run more times.
- There is an operation again.
- The “if” is the last one. It consists of three parts. The first is the “condition” again. The second is the “then” node, which is a sequence, too. It runs when the result of the condition is true. The third part is the “else” node, which runs at a false result of the condition and it is a sequence too.

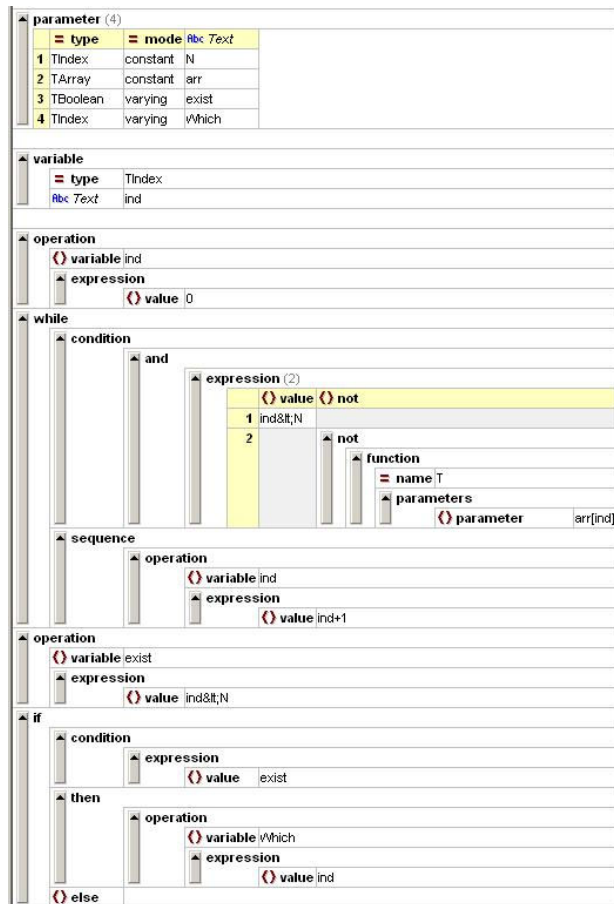


Figure 4: The whole algorithm

The whole algorithm can be expanded and understood here.

3.2. Schema definition

A part of the syntax can be checked with an Internet Explorer or any other browser applications. These can check the well-formed property.

XSD can define structure of the commands.

For example: “if” contains three parts: a “condition”, a “then ” and a “else”.

I began to write the definition XSD. Now it is the following:

A part of aml_en.xsd:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <!-- Simple Types -->
  <!-- Complex Types -->
  <xs:complexType name="TAnd">
    <xs:sequence>
      <xs:element ref="expression" minOccurs="2" maxOccurs="2"/>
    </xs:sequence>
  </xs:complexType>
  ...
  <xs:complexType name="TProcedure">
    <xs:sequence>
      <xs:element ref="parameters"/>
      <xs:element ref="variables"/>
      <xs:element ref="sequence"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="TSequence">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="operation"/>
      <xs:element ref="while"/>
      <xs:element ref="if"/>
    </xs:choice>
  </xs:complexType>
  ...
  <!-- Elements -->
  <xs:element name="procedure" type="TProcedure"/>
  ...
</xs:schema>
```

XSD is an XML, too. So it can be handled easily. It can be understood more simply with the next figure:

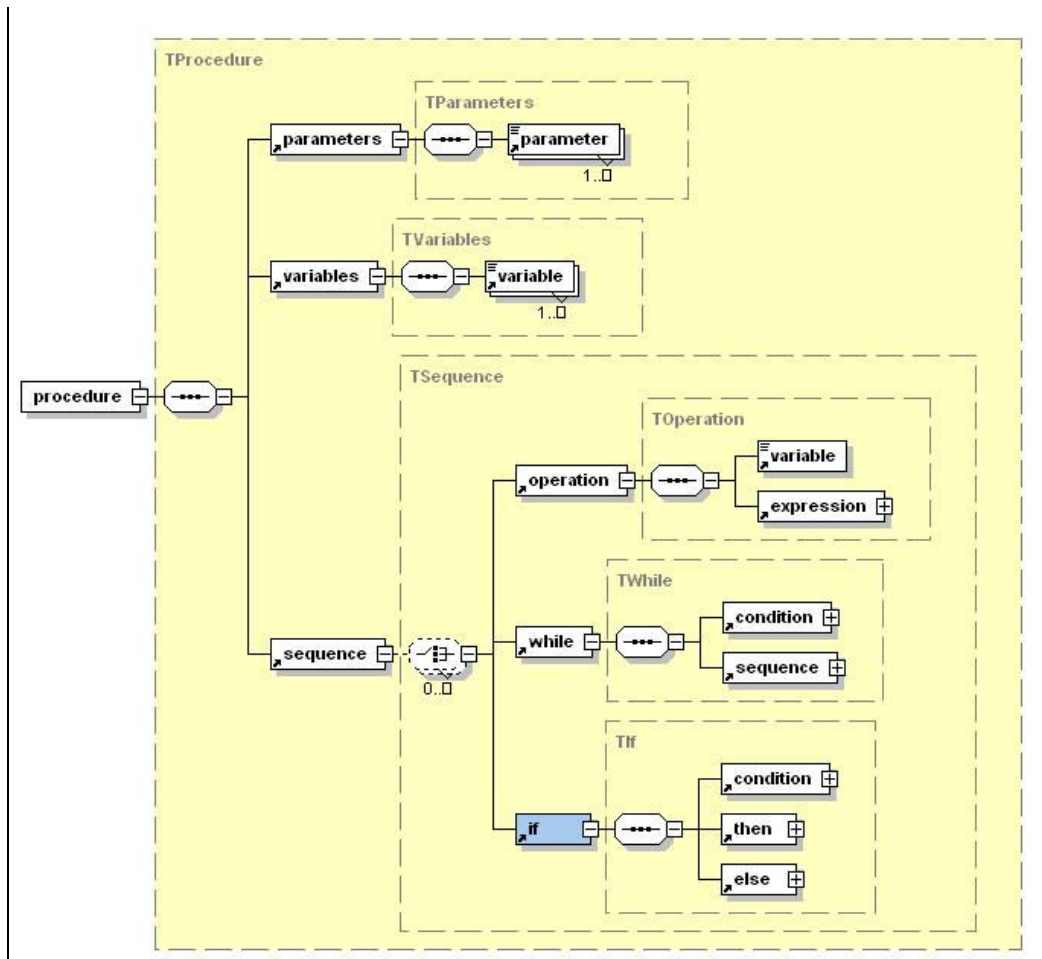


Figure 5: Types in the XSD

Figure 5 shows the possible structure and types of nodes. In the sequence each node can be chosen in any order and any times. There can be more commands for example the “for”, but I omitted these. These commands were enough for the POC to show the functioning.

3.3. Transformation to other formats

Any transformation can be defined by an XSL file. Since this article is aiming at introducing AML, XSL files are not going to be dealt with here. So if we have a good XSL file we can use it on more AML files. Each XSL can define a new format. If we would like to transform an AML file to more formats we need more XSL files. I prepared more transformations.

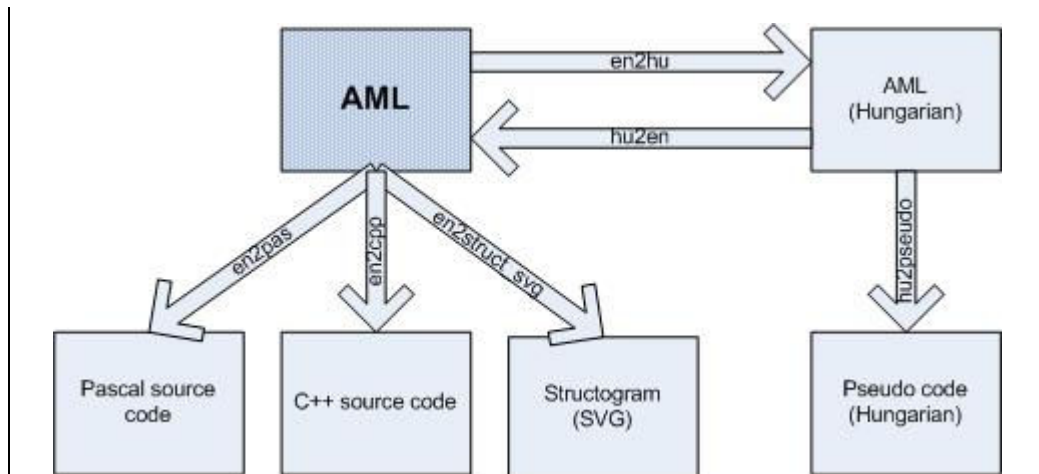


Figure 6: Direction of transformations

If students can not speak English they can use their native language in AML. Transformation between the AML and their own national AML is possible. I created the transformation to Hungarian and back to the English language. I have to note that now only the limited commands can be transformed. Transform_en2hu.xlst file can be used for transforming AML to the Hungarian AML. Transform_hu2en.xlst file can be used for transforming Hungarian AML to the English AML. Some words should be translated in the sources. That is why a dictionary is needed. dictionary.xml is available for these few words in the POC.

Generated Hungarian AML is this:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<eljaras nev="linearSeeking">
  <parameterek>
    <parameter tipus="TIndex" mod="konstans">N</parameter>
    <parameter tipus="TArray" mod="konstans">arr</parameter>
    <parameter tipus="TBoolean" mod="változó">exist</parameter>
    <parameter tipus="TIndex" mod="változó">Which</parameter>
  </parameterek>
  <valtozok>
    <valtozo tipus="TIndex">ind</valtozo>
  </valtozok>
  <utasitasok>
    <ertekadas>
      <valtozo>ind</valtozo>
      <kifejezes>
        <ertek>0</ertek>
      </kifejezes>
    </ertekadas>
    <eloltesztelos-ciklus>
      <feltetel>
        <es>
          <kifejezes>
            <ertek>ind<N</ertek>
```



```
</kifejezes>
<kifejezes>
  <nem>
    <fuggveny nev="T">
      <parameterek>
        <parameter>arr[ind]</parameter>
      </parameterek>
    </fuggveny>
  </nem>
</kifejezes>
</es>
</feltetel>
<utasitasok>
  <ertekadas>
    <valtozo>ind</valtozo>
    <kifejezes>
      <ertek>ind+1</ertek>
    </kifejezes>
  </ertekadas>
</utasitasok>
</eloltesztelos-ciklus>
<ertekadas>
  <valtozo>exist</valtozo>
  <kifejezes>
    <ertek>ind<N</ertek>
  </kifejezes>
</ertekadas>
<ha>
  <feltetel>
    <kifejezes>
      <ertek>exist</ertek>
    </kifejezes>
  </feltetel>
  <akkor>
    <ertekadas>
      <valtozo>Which</valtozo>
      <kifejezes>
        <ertek>ind</ertek>
      </kifejezes>
    </ertekadas>
  </akkor>
  <kulonben/>
</ha>
</utasitasok>
</eljaras>
```

In our native language we use the pseudo codes. A pseudo code can be generated from our native language AML with transform_hu2pseudo.xslt.

Generated pseudo code is the following:

```
eljárás linearSeeking(konstans N:TIndex; konstans arr:TArray;
változó exist:TBoolean; változó Which:TIndex);
```

```

változók
    ind:TIndex;

ind:=(0);
ciklus amíg ((ind<N) és (nem T(arr[ind])))
    ind:=(ind+1);
ciklus vége;
exist:=(ind<N);
ha ((exist)) akkor
    Which:=(ind);
különben

elágazás vége;

```

eljárás vége;

With the same method a source code of other programming languages can be generated by XSLT.

Transform_en2pas.xslt generate the source code of Pascal:

```

Procedure linearSeeking(const N:TIndex; const arr:TArray; var
exist:TBoolean; var Which:TIndex);

```

```

    var
        ind:TIndex;

begin
    ind:=(0);
    while ((ind<N) and (not T(arr[ind]))) do begin
        ind:=(ind+1);
    end;
    exist:=(ind<N);
    if ((exist)) then begin
        Which:=(ind);
    end
    else begin

    end;

end;

```

Transform_en2cpp.xslt generate the source code of C++:

```

void linearSeeking(TIndex N, TArray arr, TBoolean & exist,
TIndex & Which){

```

```
TIndex ind;

ind=(0);
while ((ind<N) && (!T(arr[ind]))) {
    ind=(ind+1);

}
exist=(ind<N);
if ((exist)) {
    Which=(ind);

} else {

}

}
```

These source codes can be copied to a part of the source code of a real program. The following frame is good for testing the C++ source (main.cpp; project: frame):

```
#include <iostream>

using namespace std;

const int MaxN=100;
typedef int TIndex;
typedef bool TBoolean;
typedef int TArray[MaxN] ;

bool T(int a){
    return (a<5);
}

// GENERATED - START
//     COPY HERE THE SOURCE!
// GENERATED CODE - END

int main()
{

    TIndex N=3;
    TArray tomb={6,3,9};
    TBoolean b;
    TIndex ind;

//CALL GENERATED CODE - START
    linearSeeking(N,tomb,b,ind);
//CALL GENERATED CODE - END

    if (b) {
        cout<<"OK"<<endl;
    }
}
```

```

        cout<<"index :"<<ind;
    } else {
        cout<<"NO";
    }
    return 0;
}

```

Finally I would like to present that structogram can be generated, too. This graphical appearance of algorithms can be drawn in browsers by SVG. SVG is an XML file, too. So it can be generated by XSLT.

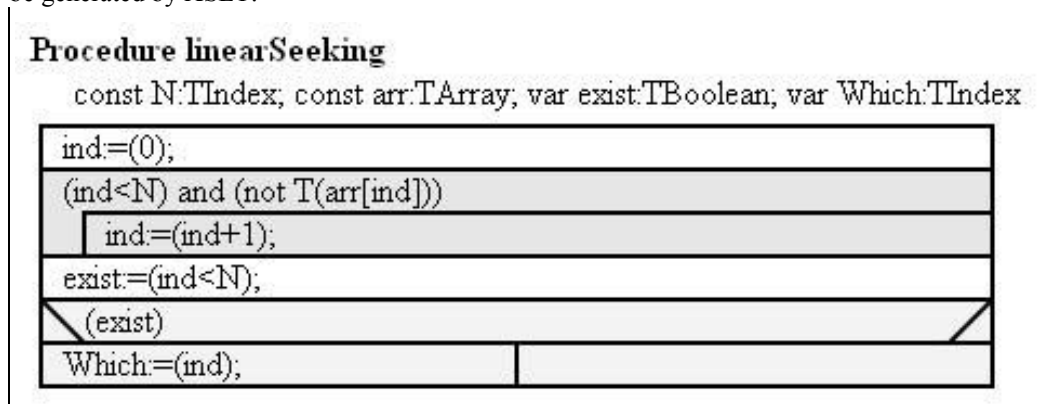


Figure 7: transform_en2struct_svg.xslt generates the source of this

4. Conclusion

This new language can be used for learning the structural programming languages. It helps to understand the parts of the command. It can help to check its syntax. It can generate other source codes and additional files. But this XML file implies a lot of other information apart from the important data. So everybody applying XML should consider that it takes a lot of typing. There are applications, for example XMLSPY that have a code supplement and can check the syntax at the time of editing. It might be the best if there might be an application in which we can *drag and drop* our procedures on graphical interface.

Literature

1. P. Szlávi, L. Zsakó: *µlógia 18. Módszeres programozás: Programozási bevezető* (2002)
2. P. Szlávi, L. Zsakó: *µlógia 19. Módszeres programozás: Programozási tételek* (2002)
3. World Wide Web Consortium, (1994-2009) <http://www.w3.org/>
4. W3Schools Online Web Tutorials, (1999-2009) <http://www.w3schools.com/>
5. Scratch Magyarország Portál, Budapest (2007-2009) <http://scratch.inf.elte.hu/>