**9/2** (2011), 209–224
tmcs@math.klte.hu
http://tmcs.math.klte.hu
Teaching Mathematics and Computer Science

# Can a language be before "the first programming language"?

László Menyhárt

*Abstract.* I would like to present a potential new language which can be before "the first programming language". We can use this to write down the algorithms and source code can be generated from this. The keyword is XML. This can be used for describing algorithms, easy to check the syntax and the semantic. 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?

*Key words and phrases:* programming, XML, programming languages.

*ZDM Subject Classification:* P40, P50, U70.

## 1. Introduction

I have never entered into the discussion which is the best first programming language. I would like to stay far 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/articles/TMCS_2011_AML/`

## 2. Today's programming languages

In the 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 is good for children to create games. Or there are some environments with simple text editor, but there are with graphical interface. At last but not at least there are the professional languages for using in the “business”. There are a lot of new products for helping the children’s learning of programming. For example scratch is a new developing.

## 3. Algorithm Markup Language (AML)

I would like to present a new approach to define algorithms. In my opinion it won’t 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 profession languages. It can help to understand the structural of the programming languages. This method is not so spread, but its usage is going up. A lot of meta-programming language uses it. For example similar XML file format is in background of BPEL in SUN implementation, o:XML or MetaL.

### 3.1. Properties of AML

The XML – eXtensible Markup Language – provides a lot of possibility. We can check easy the well-formed files. XSD – XML Schema Definition – can define the syntax and the data types of values. There are applications which can validate an XML file based on an XSD. 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’s see the algorithm of linear seeking.

**linearSeeking.xml:**

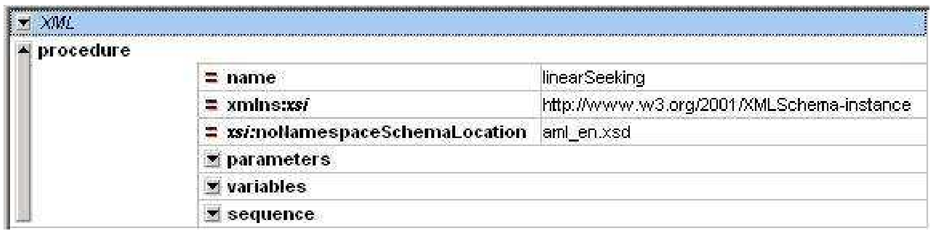
```
<?xml version="1.0" encoding="ISO-8859-2"?>
<procedure name=“linearSeeking”
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=“aml_en.xsd”>
 <parameters>
  <parameter type=“TIndex” mode=“constant”>N</parameter>
  <parameter type="TArray" mode="constant">arr</parameter>
  <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&lt;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&lt;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>
```

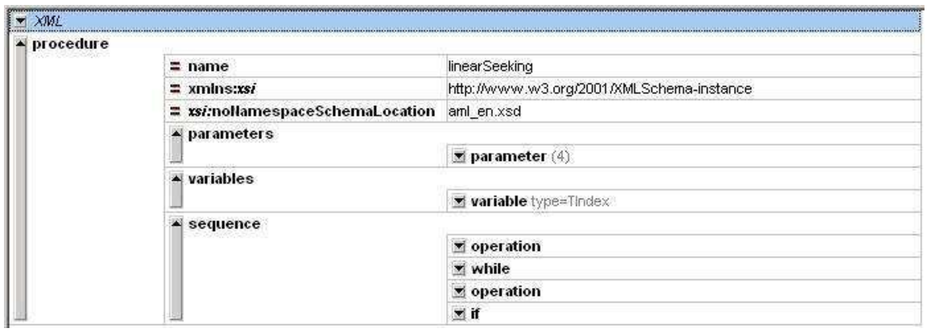
There is a procedure named "linearSeeking". This procedure is called with more parameters. For example there is a parameter named "N" with TIndex type and constant mode. There are more variables. For example the type of "ind" variable is TIndex. There are a lot of commands in sequence. The first is an 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 simpler with the next figures.



```
XML
procedure
                  = name                               linearSeeking
                  = xmlns:xsi                          http://www.w3.org/2001/XMLSchema-instance
                  = xsi:noNamespaceSchemaLocation      aml_en.xsd
                  parameters
                  variables
                  sequence
```

*Figure 1.* Procedure contains parameters, variables and sequence



```
XML
procedure
                  = name                               linearSeeking
                  = xmlns:xsi                          http://www.w3.org/2001/XMLSchema-instance
                  = xsi:noNamespaceSchemaLocation      aml_en.xsd
                  parameters
                                                       parameter (4)
                  variables
                                                       variable type=Tindex
                  sequence
                                                       operation
                                                       while
                                                       operation
                                                       if
```

*Figure 2.* Each node contains other nodes

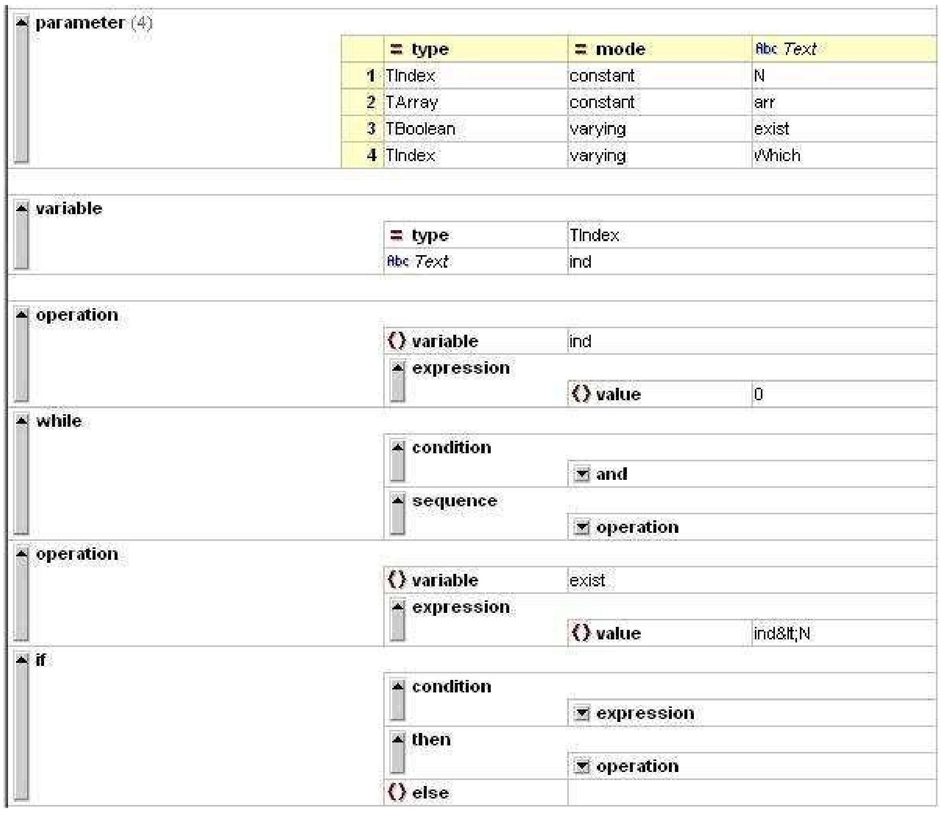
“Parameters” node contains more “parameter” nodes. “Variables” node contains more “variable” nodes. “Sequence” node contains more nodes.

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

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

There are four command 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 stands 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 stands three parts. The first is the “condition” again; the second is the “then” node, which is a sequence, too. Because it is that part of this node which runs when the result of the condition is true.



parameter (4)

| | type | mode | Text |
|---|---|---|---|
| 1 | TIndex | constant | N |
| 2 | TArray | constant | arr |
| 3 | TBoolean | varying | exist |
| 4 | TIndex | varying | Which |

variable

| | |
|---|---|
| type | TIndex |
| Text | ind |

operation

| | | |
|---|---|---|
| variable | ind | |
| expression | value | 0 |

while

| | |
|---|---|
| condition | and |
| sequence | operation |

operation

| | | |
|---|---|---|
| variable | exist | |
| expression | value | ind&lt;N |

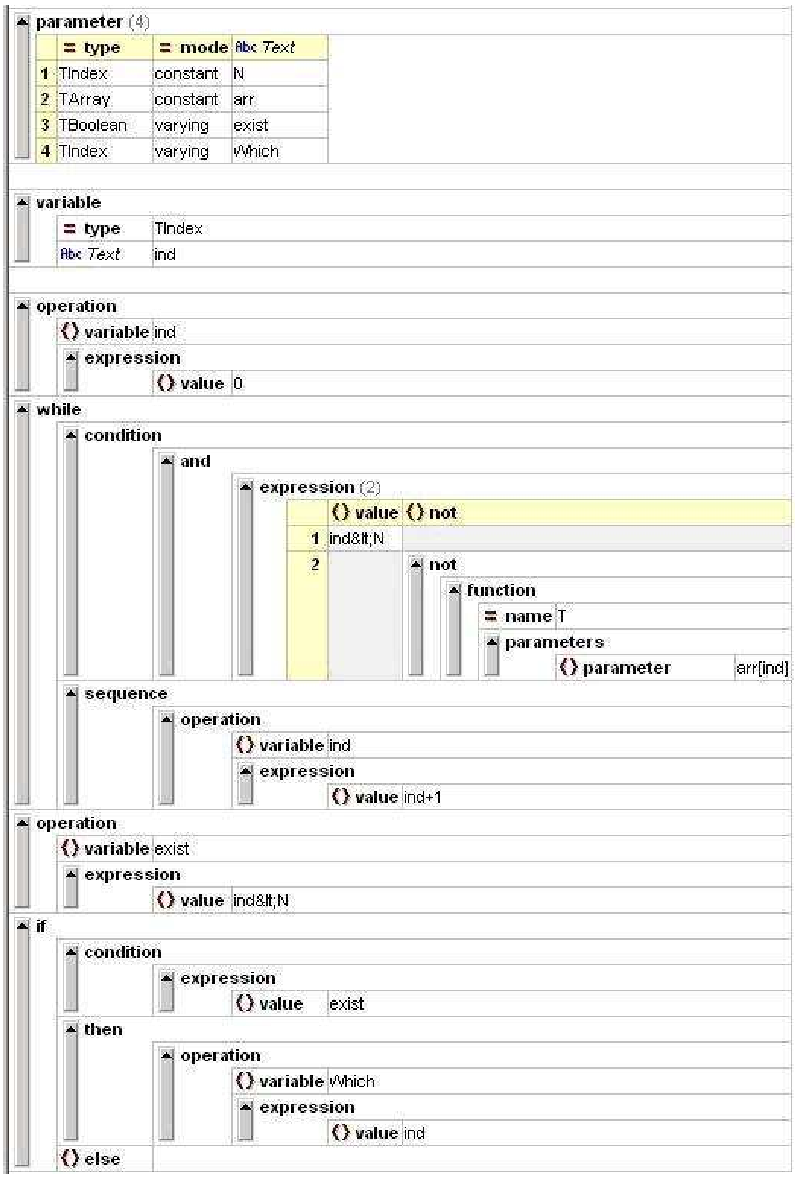
if

| | |
|---|---|
| condition | expression |
| then | operation |
| else | |

*Figure 3.* Expanded values

The third part is the "else" node, which runs at false result of the condition and it is a sequence too.

The whole algorithm can be expanded and understood here.



```
parameter (4)
      type       mode      Text
  1   TIndex     constant  N
  2   TArray     constant  arr
  3   TBoolean   varying   exist
  4   TIndex     varying   Which

variable
  type   TIndex
  Text   ind

operation
  variable ind
  expression
      value 0
while
  condition
      and
          expression (2)
                 value     not
              1  ind&lt;N
              2            not
                              function
                                 name T
                                 parameters
                                     parameter      arr[ind]
  sequence
      operation
          variable ind
          expression
              value ind+1
operation
  variable exist
  expression
      value  ind&lt;N
if
  condition
      expression
          value     exist
  then
      operation
          variable Which
          expression
              value ind
  else
```

*Figure 4.* The whole algorithm

## 3.2. Schema definition

A part of the syntax can be check on an Internet Explorer or any other browser application. These can check the well-formedness 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 -->
  <!--Complext 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 >
```

An XSD schema is also an XML document. So it can be handled easily. It can be understood simpler with the next figures:

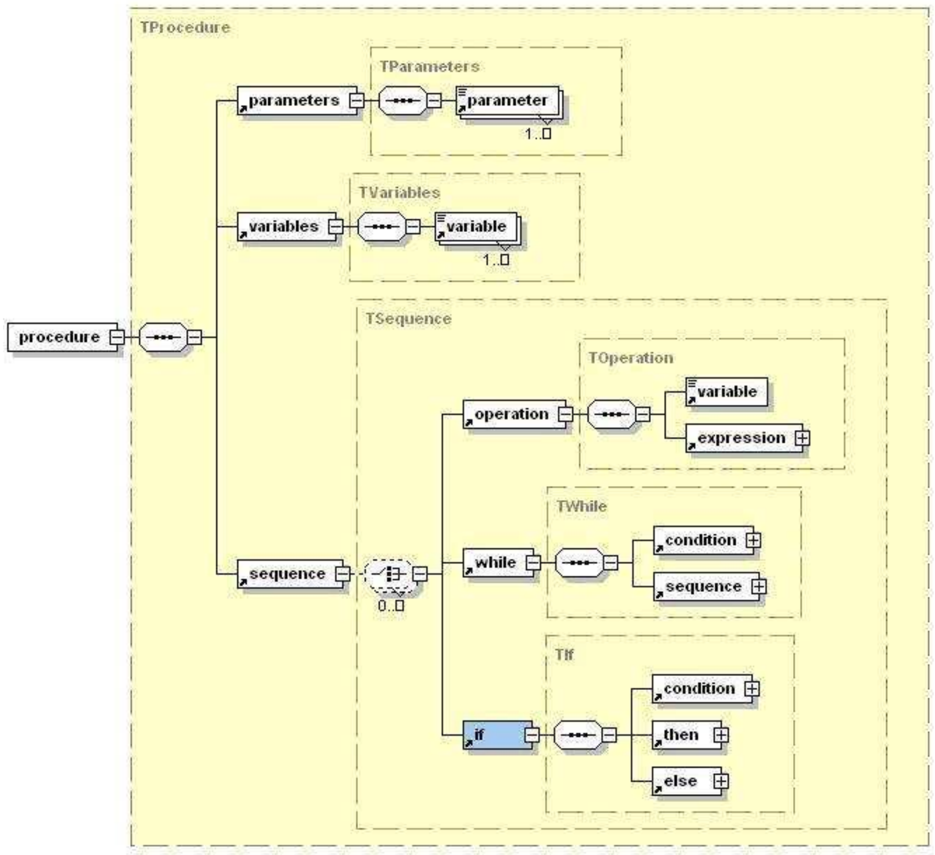
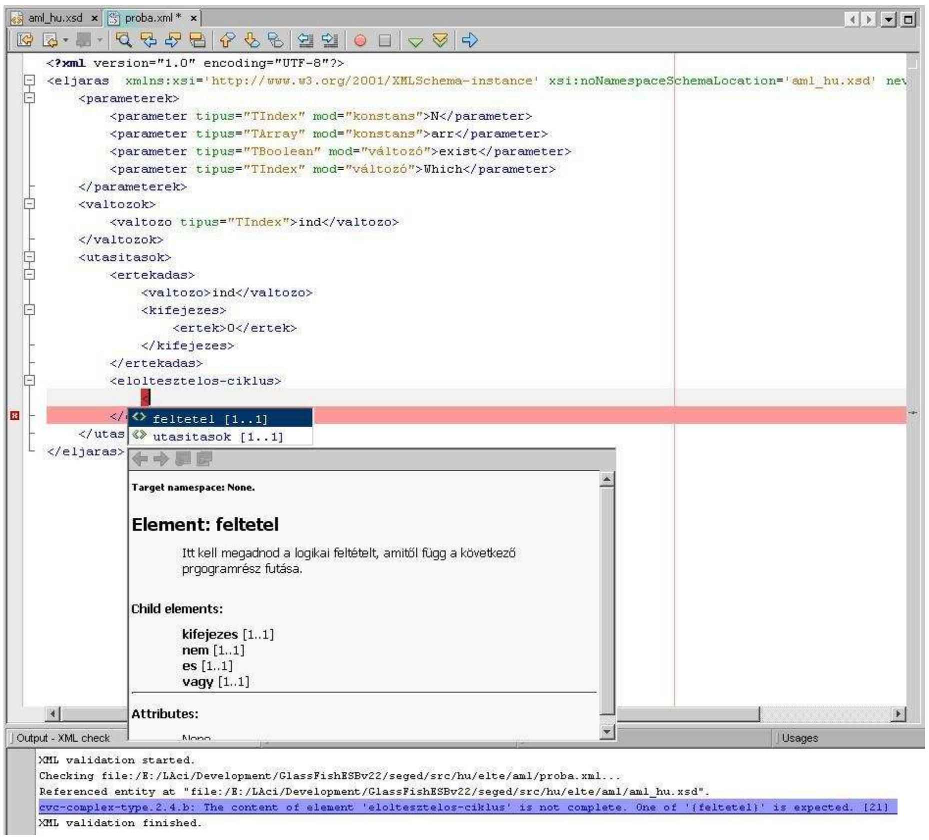


*Figure 5.* Types in the XSD

Figure 6 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 missed more commands for example the "for". But it was enough for the POC to show the functioning.

This schema can help the editor. There are XML editor applications which offer the possible identifiers in the given part of the file. For example in NetBeans IDE the following picture can be seen in the native (now in Hungarian) language:



```
<?xml version="1.0" encoding="UTF-8"?>
<eljaras  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='aml_hu.xsd' nev
    <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>
            <
        </
    </utas
</eljaras>
```

```
feltetel [1..1]
utasitasok [1..1]
```

Target namespace: None.

**Element: feltetel**

Itt kell megadnod a logikai feltételt, amitől függ a következő prgogramrész futása.

**Child elements:**

**kifejezes** [1..1]
**nem** [1..1]
**es** [1..1]
**vagy** [1..1]

**Attributes:**

None

```
XML validation started.
Checking file:/E:/LAci/Development/GlassFishESBv22/seged/src/hu/elte/aml/proba.xml...
Referenced entity at "file:/E:/LAci/Development/GlassFishESBv22/seged/src/hu/elte/aml/aml_hu.xsd".
cvc-complex-type.2.4.b: The content of element 'eloltesztelos-ciklus' is not complete. One of '{feltetel}' is expected. [21]
XML validation finished.
```

At first I run a validation which result was that there is a missing field. When I started to add the new field and I hit the “<” character the editor offered the possible fields and presented a native language description.

## 3.3. Transformation to other formats

Any transformation can be defined with an XSL file. So if we have a good XSL file we can use it on more AML files. But each XSL stylesheet can produce a specific format. If we would like to transform to more format we need more XSL file. I prepared more transformation.

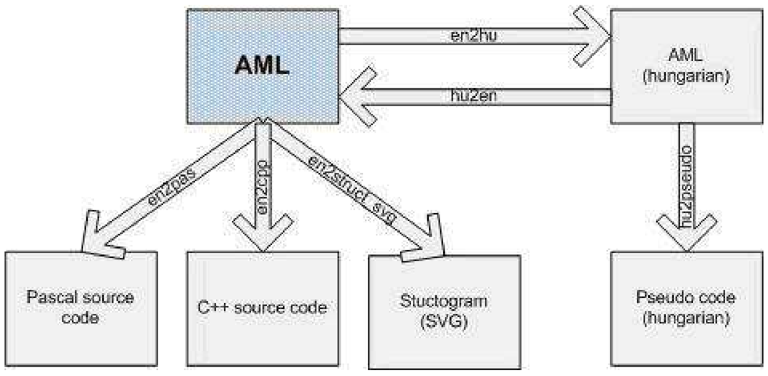


*Figure 6.* Direction of transformations

If students cannot speak English they can use their native language in AML. Transform between the AML and the own national AML is possible. I created the transform to Hungarian and back to 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 word should be translate 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&lt;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&lt;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. Pseudo code can be generated from our native language AML with transform_hu2pseudo.xslt.

Generated pseudo code is the next:

```
    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 same method 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 generates 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;
}
```

At last I would like to present that stuctogram 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.

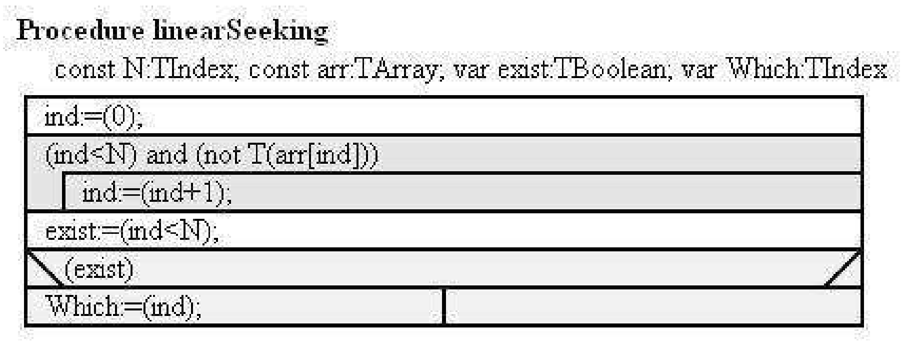

Procedure linearSeeking
const N:TIndex; const arr:TArray; var exist:TBoolean; var Which:TIndex
ind:=(0);
(ind<N) and (not T(arr[ind]))
ind:=(ind+1);
exist:=(ind<N);
(exist)
Which:=(ind);

*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 files implies much other information apart from the important data. So we should write a lot. There are applications, for example XMLSPY, that have 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.

## References

[1] P. Szlávi and L. Zsakó, $\mu$lógia 18, *Módszeres programozás: Programozási bevezető*, 2002.

[2] P. Szlávi and L. Zsakó, $\mu$lógia 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/`.

LÁSZLÓ MENYHÁRT
ELTE IK
BUDAPEST
HUNGARY

*E-mail:* `menyhart@elte.hu`

*(Received November, 2010)*