# Some experiences in adding a new language to SSQSA architecture – a part of SSQSA front-end

Jozef Kolek, Gordana Rakić, Zoran Budimac

# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

# Given Tasks

- Tasks
  - To add new language to SMIILE - Delphi
  - To add new metric to added language - Halstead

# Motivation

- Everyone who wants to use SMIILE for some new language and for some new metrics, can do this by following our guide.

# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

# Adding a Language

- Steps

  1) Find the specification of desired language (for example generic EBNF grammar, which is also most suitable to rewrite with ANTLR)

  2) Rewrite given grammar with ANTLR

  3) Add generation of CST to ANTLR grammar (*rule rewriting*)

  4) Add the SMIILE universal nodes (CST → eCST)

# Example of mentioned Steps

***1) EBNF production***

```
UsesClause = "USES" UsedUnit { "," UsedUnit } ";" .
```

***2) Corresponding ANTLR rule***

```
usesClause : USES usedUnit ( ',' usedUnit )* ';'
```

***3) Same ANTLR rule with CST output (without any of universal nodes)***

```
usesClause
    :   USES usedUnit ( ',' usedUnit )* ';'
        -> ^( USES usedUnit ( ',' usedUnit )* ';' )
```

***4) ANTLR rule with universal node (IMPORT_DECL)***

```
usesClause
    :   USES usedUnit ( ',' usedUnit )* ';'
        -> ^( USES ^( IMPORT_DECL usedUnit )
                ( ',' ^( IMPORT_DECL usedUnit ) )* ';' )
```

# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

# Some of the SMIILE Universal Nodes

- There are eight SMIILE nodes incorporated:
  - **COMPILATION_UNIT**
  - **MAIN_BLOCK** (main blocks)
  - **CONDITION** (conditions of *if*, *while*, ...)
  - **BRANCH_STATEMENT** (*if*, *case*, *try*)
  - **BRANCH** (branches of *if*, *case* and *try*)
  - **JUMP_STATEMENT**
  - **LOOP_STATEMENT** (*while*, *repeat*, *for*)
  - **LOGICAL_OPERATOR** (*and*, *or*, *xor*)

# Jump Statements

- Few standard procedures of Delphi are added to act as keywords (**exit**, **continue**, **abort**, **runerror**, **break** and **halt**)
- This is because they can also change the flow of the program and we have to mark them as **jump statements** (JUMP_STATEMENT).
- And this must be done at **syntax level**.

# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

# Adding a Metric

- Steps
  1) Analize what new nodes are needed
  2) Add new nodes to existing ANTLR grammar as described before
  3) Generate the lexer and parser
  4) Traverse the tree of the parser, use incorporated universal nodes and calculate the metrics

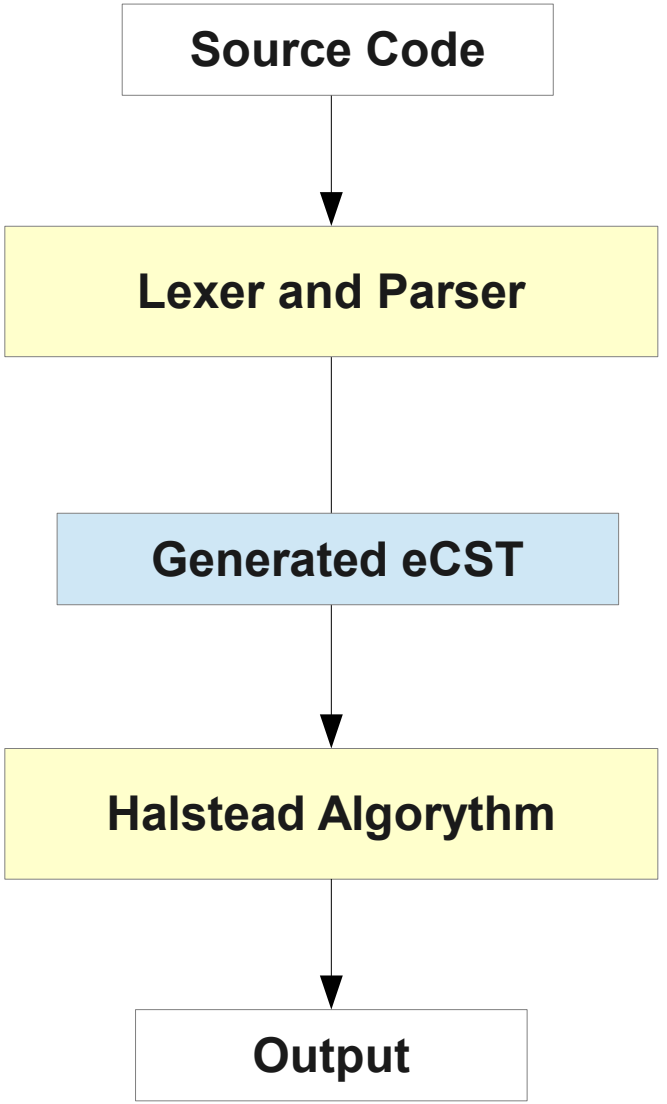
# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

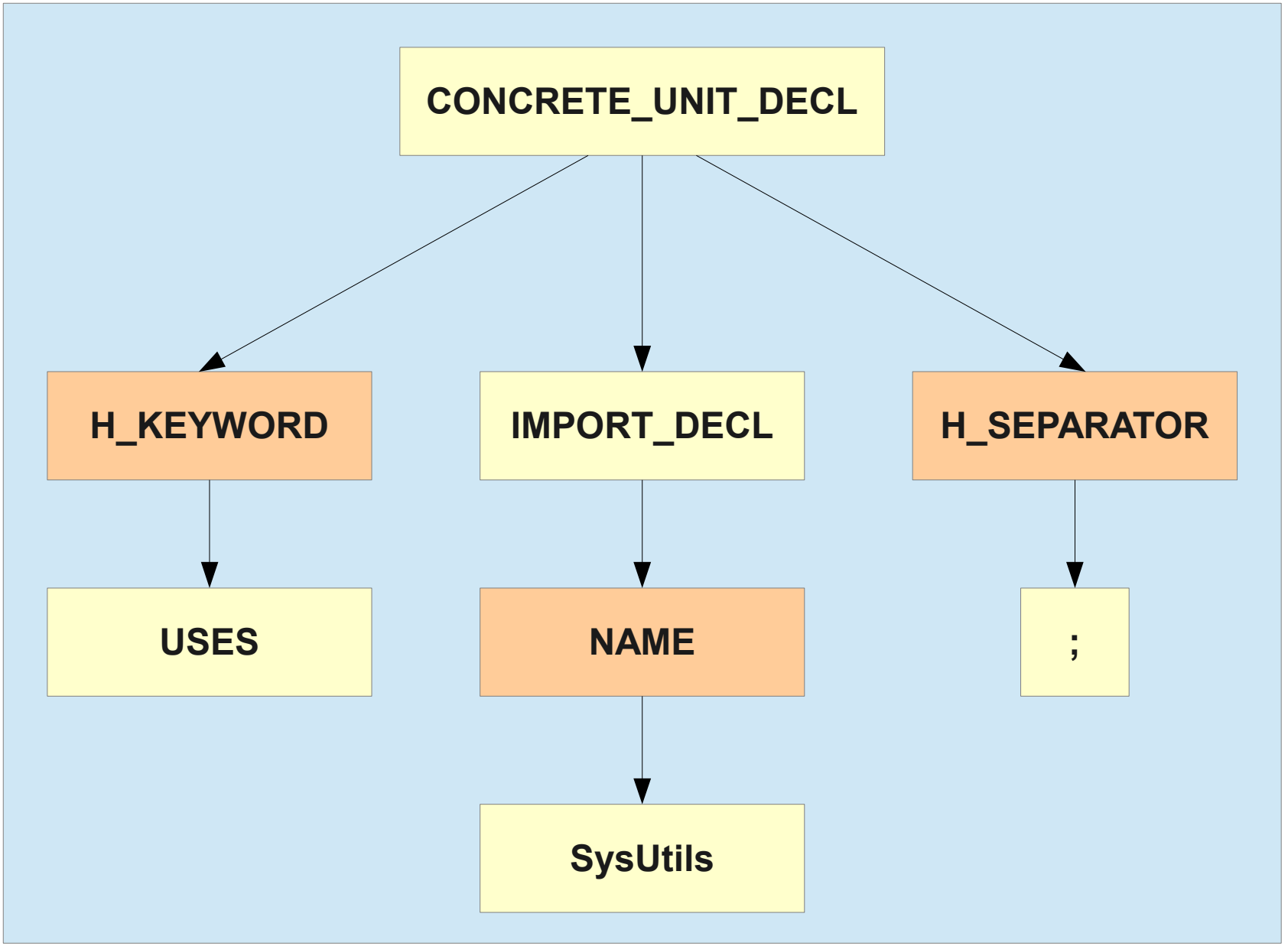
# Halstead Metrics

- Measures program's complexity directly from source code.

# Metric Calculation Phases



# Generated eCST



# Halstead Metrics Evaluation

- While traversing the tree Halstead algorythm must count *total* and *distinct* number of occurrences of operators and operands.
- Operators:
  - keywords,
  - operators such as “+” and “-”,
  - separators
- Operands:
  - names, constants, types and directives.

# Halstead Metrics Evaluation

- While traversing the tree Halstead algorythm must count *total* and *distinct* number of occurrences of operators and operands.
- Operators:
  - keywords,
  - operators such as “+” and “-”,
  - separators
- Operands:
  - names, constants, types and directives.

# Halstead Metrics Algorythm

1) Compute the following:
- *n1* - the number of distinct operators
- *n2* - the number of distinct operands
- *N1* - the total number of operators
- *N2* - the total number of operands

2) Calculate the measures:
- *Program vocabulary*: n = n1 + n2
- *Program length*: N = N1 + N2
- *Program volume*: V = N * log2(n1 + n2)
- *Program level*: L = (2/n1)*(n2/N2)
- *Program difficulty*: D = (n1/2) * (N2/n2) = 1/L
- *Programming effort*: E = D * V
- *Programming time*: T = E/18 seconds
- *Intelligent content*: I = V / D

# Output

| Description | Value |
|---|---|
| Distinct Operators (n1) | 63 |
| Distinct Operands (n2) | 101 |
| Total Operators (N1) | 743 |
| Total Operands (N2) | 338 |
| Program Vocabulary (n) | 164 |
| Program Length (N) | 1081 |
| Program Volume (V) | 7953.5137 |
| Program Level (L) | 0.00948624 |
| Program Difficulty (D) | 105.41584 |
| Programming Effort (E) | 838426.3 |
| Programming Time (T) | 46579.24 |
| Intelligent Content (I) | 75.448944 |

# Halstead Metrics

- For purpose of **Halstead Metrics** we have introduced few new universal nodes:
  - Operators: **H_KEYWORD**, **H_OPERATOR**, **H_SEPARATOR**;
  - Operands: **H_TYPE**, **H_DIRECTIVE**, **H_CONST**.
- To count identifiers as operands the **NAME** universal node is reused.
- Every one of these Halstead universal nodes has exactly one child.

# Example of modified usesClause rule

```
usesClause
    :    USES usedUnit ( ',' usedUnit )* ';'

         → ^( H_KEYWORD USES )
           ^( IMPORT_DECL usedUnit )
           ^(
               ^( H_SEPARATOR ';' )
               ^( IMPORT_DECL usedUnit )
            )*
           ^( H_SEPARATOR ';' )
    ;
```

# Example of the Tree traversing

- To count distinct names we are using corresponding universal nodes (such as *VAR_DECL* and *FIELD_DECL*) in generated syntax tree, so every place in the tree where new declarations of names can be found is investigated and this names are added in the **list of names**.
- At the end we just count how many elements in this list we got.

# Content

1. Introduction
2. Adding a Language
3. Some of the SMIILE Universal Nodes
4. Adding a Metric
5. Halstead Metrics
6. Conclusion

# Conclusion

- This was typical example that eCST is very flexible and extensible.
- That means eCST can be used in some other project with minor or none modifications.
- Or it can be extended with totally new nodes to satisfy various needs.