Software Language Engineering Report

Final Report

Gurbir Singh

Lecturer: Adrian Johnstone

Department of Computer Science
Royal Holloway, University of London

Table of Contents:

Introduction

This project is a reflection of the work that I have done to create a domain-specific language. I decided to pick 3D modelling as I have a greater interest in this topic as compared to the others that were on offer to select from. One of the key points in choosing this project was to enhance my 3D design capabilities. Using the logic that I have learnt from 3D modelling, I can use these skills to make more complex languages. Through this report I will be going through the functionalities that I have implemented in my language which make it accessible to everyone to use at any skill level whilst going through some important topics and concepts that I learnt about creating a domain-specific language.

Chapter 1: informal specification and internal syntax

In this section of the report, I'll discuss the details I have put into my informal specification and internal syntax document. The 3D modelling language documentation outlines the framework I have used to manipulate/create my domain-specific language (DSL), particularly the 3D printing of complex shapes using a valueuserplugin. The document discusses the details regarding the language's operation and structure and highlights the language's ability to handle and generate complex 3D models/shapes such as a torus. The document also covers the 3D model's creation through a user-friendly and simple syntax.

Key sections that I would like to highlight in this particular part of the report are the arithmetic operations and comparison statements that allow for precise and complex calculations. One of the most essential features that I would like to point out is the creation of geometric shapes and other complex shapes such as a torus. Another key point about 3D modelling is that it is accessible to everyone at all levels of learning.

The specification also contains a backend function where specific operations can be called to perform functionalities or just for additional features. To create a shape, for example, a torus, I first pass an eSOS rule where all the requirements(in the case of a torus a major radius and minor radius are required. The Major radius is the overall radius and the minor radius is the tube radius) are stated, a try statement with the value is added and then simply linked to the plugin. The shape is then created on a separate window once the correct commands have been run.

Chapter 2: eSOS rule

In this section of the report, I will focus on the intricacies of eSOS rules as these rules serve as the foundation for the 3DModelling language. These rules are crafted in a way to manage and create 3D models giving more emphasis to 3D printing of complex geometric shapes. These rules are created by following a semantic and syntax structure. A syntax is defined first, followed by specifying the semantics, then the rule is implemented. To see if a rule is successful, it can be tested using !try statements. The most important and notable eSOS rules in my document are the shape-related creation rules such as the sphereCreation rule,

torusCreation rule, cuboidCreation rule etc. These are the most important rules as they help in implementing the main feature of the language which is to display shapes of different sizes and colours on the screen.
Key functionalities also include the control flow constructs such as looping(while) and conditional(if) statements. The control flow statements can then be used to create responsive and dynamic 3D models. Combining the dynamic controls with the arithmetic operations and logical operations enhances the versatility of the language which essentially allows to define and operate more complex behaviours for the 3D models. In the images below, some of the eSOS rules that have been written can be seen:

Image 1:

```
-ifTrue
---
if(True, _C1, _C2),_sig -> _C1, _sig

-ifFalse
---
if(False, _C1, _C2),_sig -> _C2,_sig

-ifResolve
_E, _sig ->_EP, _sigP
---
if(_E,_C1,_C2),_sig -> if(_EP, _C1, _C2), _sigP

-while
---
while(_E, _C),_sig -> if(_E, seq(_C, while(_E,_C)), __done), _sig
```

Image 2:

```
-cubeCreation
_width |> __real64(_) _height |> __real64(_) _depth |> __real64(_)
---
cubeCreation(_width, _height, _depth), _sig -> __user("cubeCreation", _width, _height, _depth), _sig

-sphereCreation
_radius |> __real64(_)
---
sphereCreation(_radius), _sig -> __user("sphereCreation", _radius), _sig

-cylinderCreation
_radius |> __real64(_) _height |> __real64(_)
---
cylinderCreation(_radius, _height), _sig -> __user("cylinderCreation", _radius, _height), _sig

-coneCreation
_radius |> __real64(_) _height |> __real64(_) _slantHeight |> __real64(_)
---
coneCreation(_radius, _height, _slantHeight), _sig -> __user("coneCreation", _radius, _height, _slantHeight), _sig

-cuboidCreation
_width |> __real64(_) _height |> __real64(_) _depth |> __real64(_)
---
cuboidCreation(_width, _height, _depth), _sig -> __user("cuboidCreation", _width, _height, _depth), _sig

-pyramidCreation
_baseWidth |> __real64(_) _height |> __real64(_)
---
pyramidCreation( baseWidth,  height),  sig ->  user("pyramidCreation",  baseWidth,  height),  sig
```

Image 3:

```
-backend
---
backend(_P1, _P2, _P3), _sig -> __user(_P1, _P2, _P3), _sig

!try backend(1,2,3), __map
!try add(15.0,25.0), __map
!try sub(6.0,4.0), __map
!try multiply(23.0,3.0), __map
!try divide(20.0,2.0), __map
!try modular(12.0,7.0), __map
!try cubeCreation(200.0,200.0,200.0), __map
!try sphereCreation(100.0), __map
!try cylinderCreation(80.0, 180.0), __map
!try coneCreation(50.0, 200.0, 800.0), __map
!try cuboidCreation(300.0, 150.0, 145.0), __map
!try pyramidCreation(120.0, -150.0), __map
!try dodecahedronCreation(2.5), __map
!try torusCreation(50.1, 70.5), __map
!try hexagonalPrismCreation(100.0, 200.0), __map
!try create_new_object("Sphere"), __map
!try cubeRotation(1.5, 90.0), __map
```

Image 4:

```
*** try backend(1, 2, 3), {} with relation  ->
Step 1
  Rewrite call 1 backend(1, 2, 3), {}  ->
  -backend  --- backend(_P1, _P2, _P3), _sig -> __user(_P1, _P2, _P3), _sig
Command 1 is not supported.
  -backend rewrites to "Command has been executed: 1", {}
Step 2
  Rewrite call 2 "Command has been executed: 1", {}  ->
    Terminal "Command has been executed: 1", {}
Normal termination on "Command has been executed: 1", {} after 2 steps and 2 rew
rites
*** try add(15.0, 25.0), {} with relation  ->
Step 1
  Rewrite call 1 add(15.0, 25.0), {}  ->
  -add _n1 |> _  _n2 |> _ --- add(_n1, _n2), _sig -> __add(_n1, _n2), _sig
  -add rewrites to 40.0, {}
Step 2
  Rewrite call 2 40.0, {}  ->
    Terminal 40.0, {}
```

In Image 1, the if and while related eSOS rules are defined and as we move to Image 2, the eSOS rules to represent shapes can be seen. In Image 3, the use of try statements can be seen where values can be passed into the rules given the variables. Upon execution, the result in Image 4 is produced if the rules are successful(Image 4 just includes a small section of the terminal). If not an error will be thrown.

Overall, This section of the report is about the technical sophistication and originality of the 3DModelling language with the use of eSOS rules.

Chapter 3: Plugin

In the development of the 3DModelling language, a crucial advancement was accomplished by implementing the ValueUserPlugin. This was a pivotal component that used the eSOS rules to generate 3D-modelled shapes on separate windows. I coded these so that almost all the shapes are dynamically displayed by the use of JavaFX i.e. rotate as they are being displayed until the popup window has been closed manually. Central to the functionality of the plugin is the processing of commands which consists of cubeCreation, cyclinderCreation, spehereCreation etc. These commands, are identified through eSOS rules within the language syntax and are then executed by passing parameters to define attributes like dimensions and colours. The dimensions are specified in the !try commands section in the eSOS rules.

The ValueUserPlugin follows a systematic and structured approach when executing the commands. It first identifies the name of the command from the passed arguments, and then locates the command to the corresponding shape-creation method. Each of the methods has been coded in such a way that they handle specific shape parameters and elements. Some of the elements include ‘Group’, ‘Scene’ and various shapes such as ‘Box’, and ‘Sphere’. To achieve the desired colour effect ‘PhongMaterial’ is used and for the transformations to move the shapes in a dynamic motion ‘Rotate’ is used. Below are given few snippets from the code and it’s working :

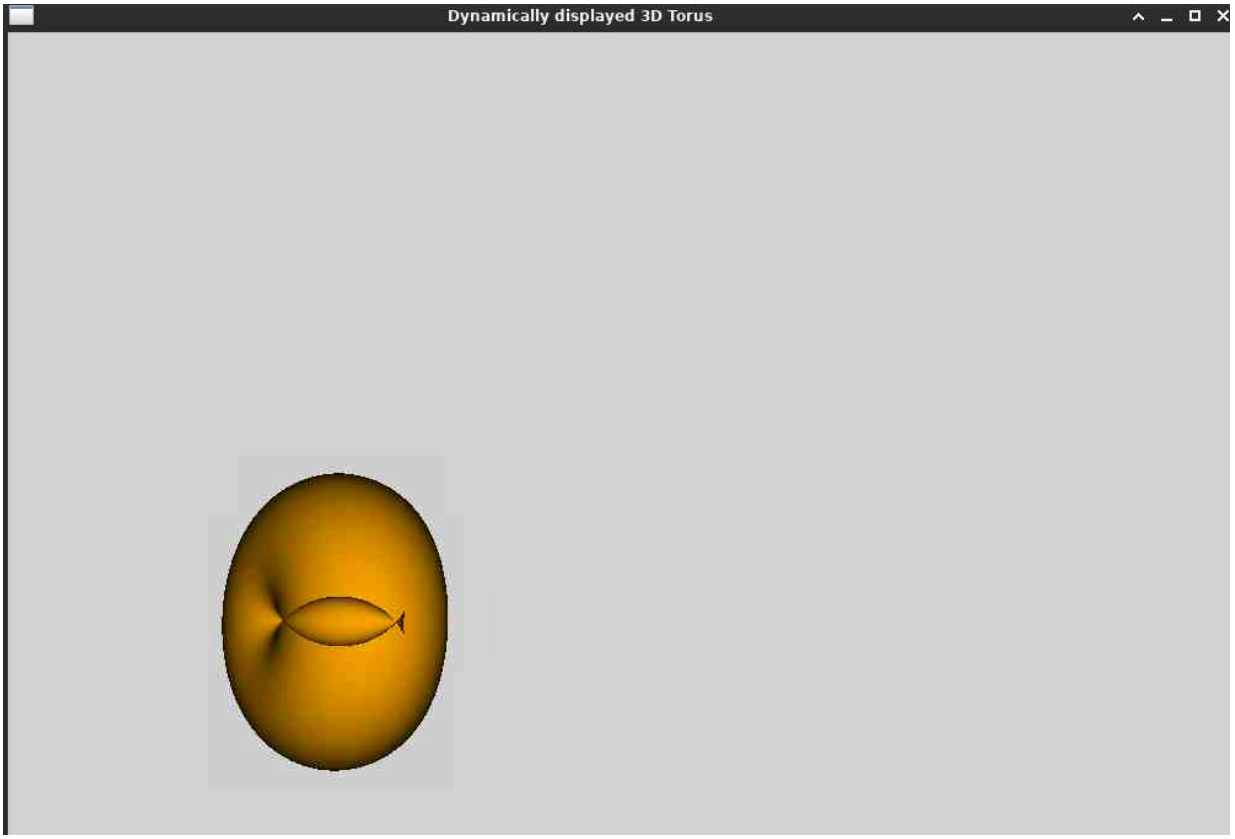
Image 1:

```
@Override
public Value user(Value... args) throws ARTException {
    String commandName = String.valueOf(args[0].value());
    switch (commandName) {
        case "coneCreation":
            if (args.length >= 4) {
                coneCreation(args);
            } else {
                System.out.println("Not enough arguments for cone creation.");
            }
            break;
        case "sphereCreation":
            sphereCreation(args);
            break;
        case "cylinderCreation":
            cylinderCreation(args);
            break;
        case "cubeCreation":
            cubeCreation(args);
            break;
        case "cuboidCreation":
            cuboidCreation(args);
            break;
        case "pyramidCreation":
            pyramidCreation(args);
            break;
         case "torusCreation":
            torusCreation(args);
            break;
        case "hexagonalPrismCreation":
            if (args.length >= 3) {
            hexagonalPrismCreation(args);
            } else {
            System.out.println("Not enough arguments for hexagonal prism creation.");
            }
            break;
        default:
            System.out.println("Command " + commandName + " is not supported.");
    }
```

Image 2 :

```
private void torusCreation(Value[] args) throws ARTException {
    double majorRadius = ((Number) args[1].value()).doubleValue();
    double minorRadius = ((Number) args[2].value()).doubleValue();

    Platform.runLater(() -> {
        Group root = new Group();
        MeshView torus = createTorus(majorRadius, minorRadius, 100, 100);
        PhongMaterial material = new PhongMaterial();
        material.setDiffuseColor(Color.ORANGE);
        torus.setMaterial(material);

        Rotate rotateY = new Rotate(0, Rotate.Y_AXIS);
        torus.getTransforms().add(rotateY);

        new Thread(() -> {
            while (true) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                Platform.runLater(() -> rotateY.setAngle(rotateY.getAngle() + 5));
            }
        }).start();

        setupScene(root, torus, "Dynamically displayed 3D Torus");
    });
}
```

Image 3:



In Image 1, the identification of the name of the command from the passed arguments can be seen. In image 2, the torusCreation method can be seen where different elements have been used to display the shape dynamically. In Image 3, the shape being displayed can be seen.

Chapter 4: External Syntax with GIFT rewrites

In this part of the report, I will delve into the context-free grammar that delineates the structure of statements within the 3DModelling language. The external to the internal translator is decorated with hat operators that produce terms which run with the eSOS rules, at the core of the translator lies a robust syntax design which organises the commands into sequences, assignments, conditional statements, loops and it also directs the calls to the backend function. The grammar has been further broken down into relational expressions(relExpr) which consist of logical operators such as greater than(gt), less than(lt) etc. and sub-expressions (subExpr), which consist of arithmetic operations such as addition(add), subtraction (sub) and more complex operations like modulus(mod). The defined ID and INTEGER tokens are essential as they essentially help with the textual representation of identifiers and numbers with their corresponding semantic values in the language runtime. Through this grammar, the 3D modelling language can achieve a high level of expressiveness which is accessible to any user of any skill level.

Image 1:

```
statement ::= seq^^ | assign^^ | if^^ | while^^ | backend^^

seq ::= statement statement
assign ::= ID ':='^ subExpr ';'^
if ::= 'if'^ relExpr statement 'else'^ statement
while ::= 'while'^ relExpr statement
backend ::= 'backend'^ '('^ subExpr ','^ subExpr ','^ subExpr ')'^

relExpr ::= subExpr^^ | gt^^ | ge^^ | e^^ | lt^^ | le^^ | ne^^
gt ::= relExpr '>'^ subExpr
ge ::= relExpr '>='^ subExpr
e  ::= relExpr '=='^ subExpr
lt ::= relExpr '<'^ subExpr
le ::= relExpr '<='^ subExpr
ne ::= relExpr '!='^ subExpr

subExpr ::= operand^^ | sub^^ | add^^ | mult^^ | div^^ | exp^^ | mod^^
sub ::= subExpr '-'^ operand
add ::= subExpr '+'^ operand
mult ::= subExpr '*'^ operand
div ::= subExpr '/'^ operand
exp ::= subExpr '**'^ operand
mod ::= subExpr '%'^ operand

operand ::= deref^^ | INTEGER^^ | '('^ subExpr^^ ')'^
deref ::= ID


ID <leftExtent:int rightExtent:int lexeme:String v:ARTValueString> ::=
  &ID^^ {ID.lexeme = artLexeme(ID.leftExtent, ID.rightExtent);
  ID.v = new ARTValueString(artLexemeAsID(ID.leftExtent, ID.rightExtent)); }

INTEGER <leftExtent:int rightExtent:int lexeme:String v:ARTValueInteger32> ::=
  &INTEGER^^ { INTEGER.lexeme = artLexeme(INTEGER.leftExtent, INTEGER.rightExtent);
  INTEGER.v = new ARTValueInteger32(artLexemeAsInteger(INTEGER.leftExtent, INTEGER.rightExtent)); }
```

Image 2:

```
a := 15; b := 9; c := 15; d := 20; e := 1; f := 2; g := 12; h := 3; i := 6; j := 23;

while a != b
  if a > b
    a := a - b;
  else
  b := b - a;

gcd := a;

while a <= b
  if a < b
    a := a + b;
  else
  b := b + a;

while a == c
  if a < b
    a := a - c;
  else
  c := c + a;

while a == c
  if a < b
    a := a - c;
  else
  c := c + a;

while d > c
  if d == c
    d := d % c;
  else
  c := c + d;

while a != b
  if a * b > a
    a := a * b;
  else
    b := b / a;
```

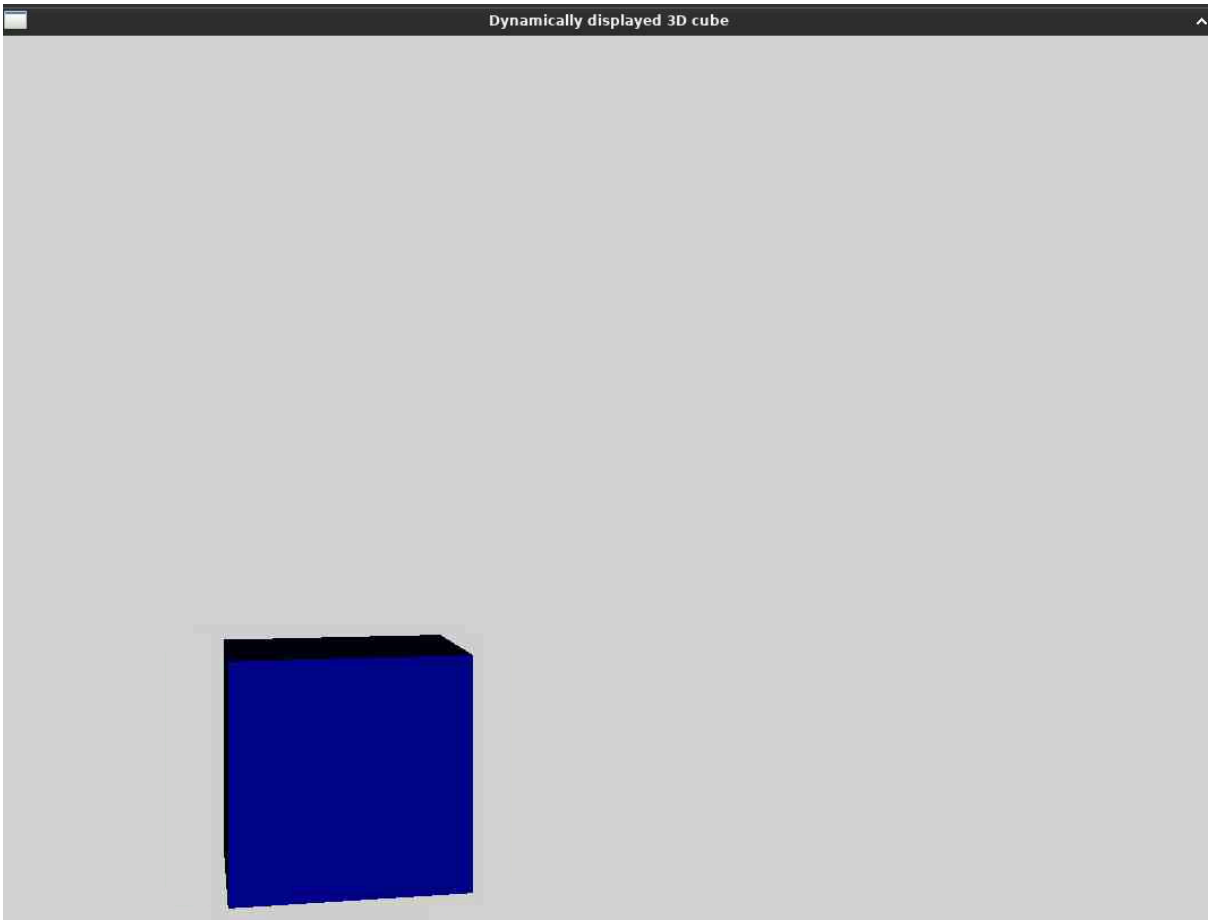
In image 1, the grammar can be seen and how it has been structured properly. Image 2 contains the tests used to verify that the grammar works as it's supposed to.

Chapter 5: External syntax with attributes and actions

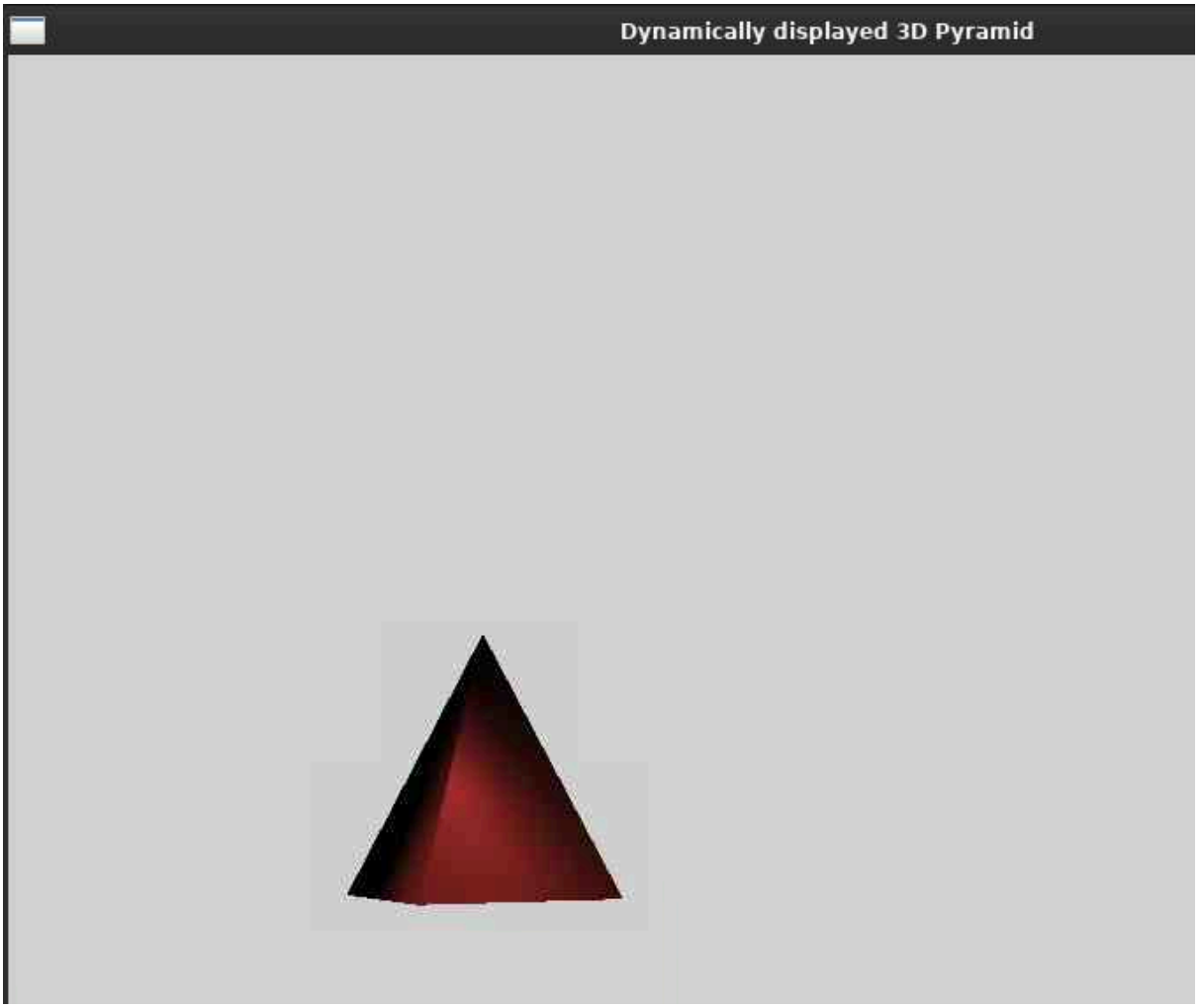
The grammar for the 3D modelling language uses Java to construct dynamic 3D models using statements for different tasks such as conditions, assignments, loops and a HashMap is used for symbol tracking. Complex operations can be easily supported with the given arithmetic and relational expressions which allow the proper detailing of a 3D model. This framework allows for the execution of advanced commands showcasing the language's precision and creative potential in 3D design.

Chapter 6: Example programs

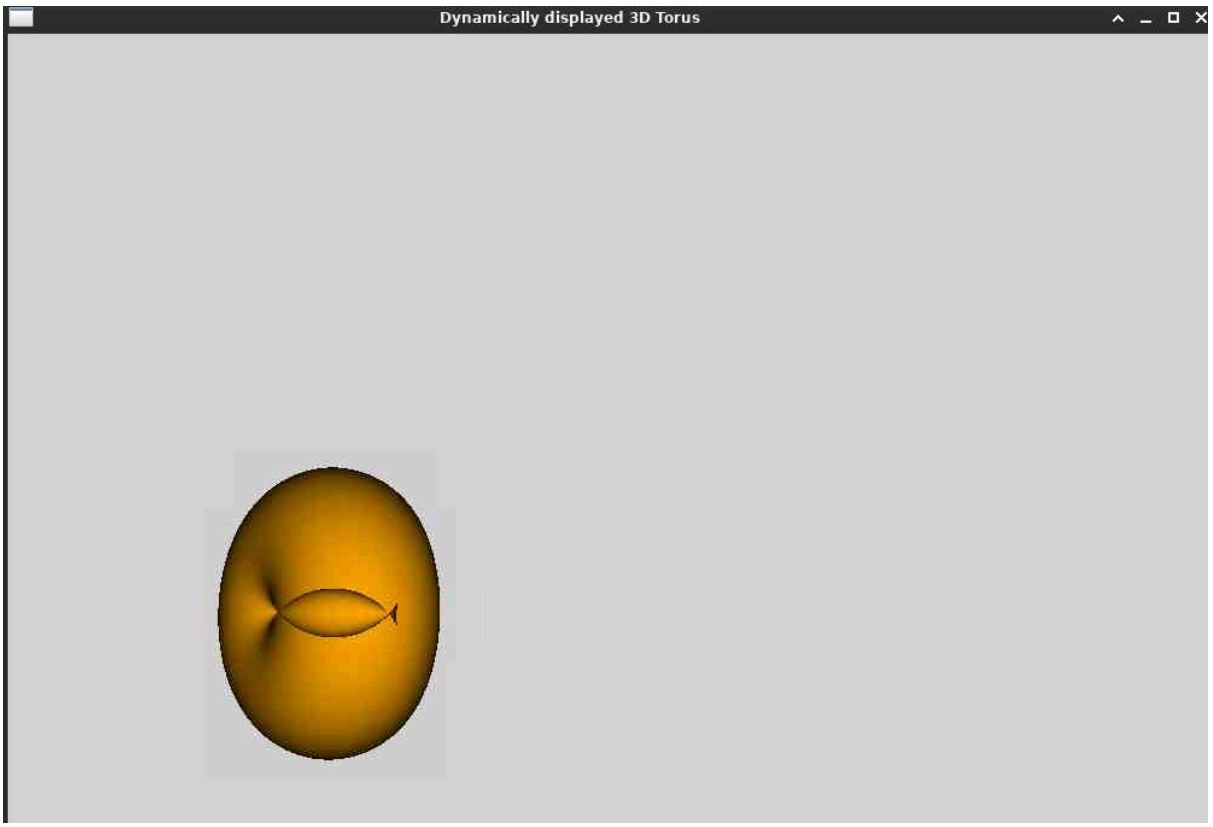
Below are the examples of the running the commands following the readme file:



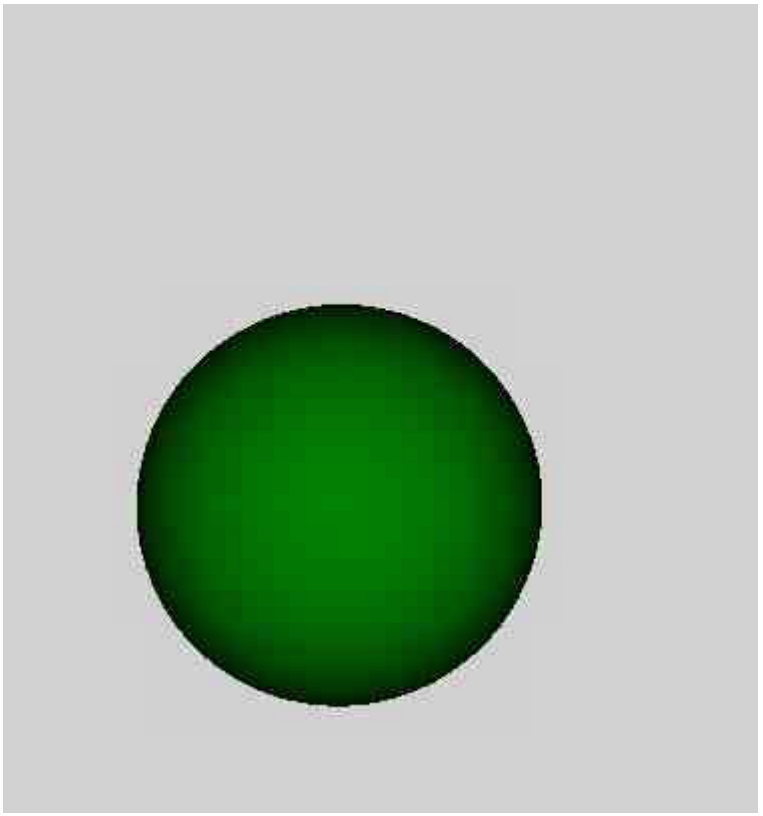
Cube Creation



Pyramid Creation



Torus Creation



Sphere Creation

Upon running eSOS rules:

```
*** try cubeCreation(200.0, 200.0, 200.0), {} with relation  ->
Step 1
  Rewrite call 1 cubeCreation(200.0, 200.0, 200.0), {}  ->
  -cubeCreation _width |> _  _height |> _  _depth |> _ --- cubeCreation(_width, _height, _depth), _sig -> __user("cubeCreation", _width, _height, _depth), _sig
  -cubeCreation rewrites to "Command has been executed: cubeCreation", {}
Step 2
  Rewrite call 2 "Command has been executed: cubeCreation", {}  ->
    Terminal "Command has been executed: cubeCreation", {}
Normal termination on "Command has been executed: cubeCreation", {} after 2 steps and 2 rewrites
*** try sphereCreation(100.0), {} with relation  ->
Step 1
  Rewrite call 1 sphereCreation(100.0), {}  ->
  -sphereCreation _radius |> _ --- sphereCreation(_radius), _sig -> __user("sphereCreation", _radius), _sig
  -sphereCreation rewrites to "Command has been executed: sphereCreation", {}
Step 2
  Rewrite call 2 "Command has been executed: sphereCreation", {}  ->
    Terminal "Command has been executed: sphereCreation", {}
Normal termination on "Command has been executed: sphereCreation", {} after 2 steps and 2 rewrites
*** try cylinderCreation(80.0, 180.0), {} with relation  ->
Step 1
  Rewrite call 1 cylinderCreation(80.0, 180.0), {}  ->
  -cylinderCreation _radius |> _  _height |> _ --- cylinderCreation(_radius, _height), _sig -> __user("cylinderCreation", _radius, _height), _sig
  -cylinderCreation rewrites to "Command has been executed: cylinderCreation", {}
Step 2
  Rewrite call 2 "Command has been executed: cylinderCreation", {}  ->
    Terminal "Command has been executed: cylinderCreation", {}
Normal termination on "Command has been executed: cylinderCreation", {} after 2 steps and 2 rewrites
*** try coneCreation(50.0, 200.0, 800.0), {} with relation  ->
Step 1
  Rewrite call 1 coneCreation(50.0, 200.0, 800.0), {}  ->
  -coneCreation _radius |> _  _height |> _  _slantHeight |> _ --- coneCreation(_radius, _height, _slantHeight), _sig -> __user("coneCreation", _radius, _height
slantHeight), _sig
  -coneCreation rewrites to "Command has been executed: coneCreation", {}
Step 2
```

Upon running ExtToInt_PiM

```
cim-ts-node-01$ ../artv3.sh ExtToInt_PiM +showAll
** Accept
1: seq
  2: seq
    3: seq
      4: seq
        5: seq
          6: seq
            7: seq
              8: seq
                9: seq
                  10: seq
                    11: seq
                      12: seq
                        13: seq
                          14: seq
                            15: seq
                              16: seq
                                17: seq
                                  18: seq
                                    19: seq
                                      20: seq
                                        21: seq
                                          22: seq
                                            23: seq
                                              24: seq
                                                25: seq
                                                  26: seq
                                                    27: seq
                                                      28: seq
                                                        29: seq
                                                          30: seq
                                                            31: assign
                                                              32: a
                                                              33: 15
                                                            34: assign
```

```
                                    31: assign
                                      32: a
                                      33: 15
                                    34: assign
                                      35: b
                                      36: 9
                                  37: assign
                                    38: c
                                    39: 15
                                40: assign
                                  41: d
                                  42: 20
                              43: assign
                                44: e
                                45: 1
                            46: assign
                              47: f
                              48: 2
                          49: assign
                            50: g
                            51: 12
                        52: assign
                          53: h
                          54: 3
                      55: assign
                        56: i
                        57: 6
                    58: assign
                      59: j
                      60: 23
                61: while
                  62: ne
                    63: deref
                      64: a
                    65: deref
                      66: b
                  67: if
                    68: gt
                      69: deref
                        70: a
                      71: deref
```

The file runs all the way upto:

```
              743: sub
                744: deref
                  745: e
                746: deref
                  747: f
              748: mod
                749: deref
                  750: g
                751: deref
                  752: h
        753: assign
          754: c
          755: add
            756: mult
              757: mod
                758: deref
                  759: c
                760: deref
                  761: d
              762: add
                763: deref
                  764: e
                765: deref
                  766: f
            767: div
              768: sub
                769: deref
                  770: g
                771: deref
                  772: h
              773: exp
                774: deref
                  775: i
                776: deref
                  777: j
  778: backend
    779: 1
    780: 2
    781: 3
```

Upon running Attribute_PiM :

```
cim-ts-node-01$ ../artV3.sh Attribute_PiM
** Accept
x is 5
x is 6
x is 7
x is 8
x is 9
x is 10
y is 6
y is 5
y is 4
y is 3
y is 2
y is 1
y is 0
The result of the addition expression 52+69+79 is : 200
The result of the Multiplication expression 30*45 is : 1350
The result of the Division expression 20/5 is : 4
The result of the modular expression 76%7 is : 6
The result of the exponential expression 2**3 is : 8
The result of the subtraction expression 7-3-1 is: 3
The result of the complex expression 7+3-1 is: 9
The result of the complex expression 10-11+1 is: 0
The result of the complex expression 5*10%3 is: 2
The result of the complex expression (7**3)+(10-5) is: 348
The result of the greater than expression 36>24 is : 1 therefore true
The result of the greater than expression 3>4 is : 0 therefore false
The result of the greater than equal to expression 20>=19 is : 1 therefore true
The result of the less than expression 24<36 is : 1 therefore true
The result of the less than expression 6<3 is : 0 therefore false
The result of the less than equal to expression 19<=20 is : 1 therefore true
cim-ts-node-01$
```

Adapt Rules for Compatibility — Mozilla Firefox

Achievements:

During this term, I managed to implement a Domain-specific Language that is accessible to everyone and can perform complex operations. The following are some of the achievements I would like to point out:

- Successfully implemented eSOS rules and tested the rules using various !try commands.
- Successfully implemented a Java plugin which dynamically displays various complex shapes of different colours and sizes ranging from a cube to a pyramid whilst using transformations.
- Successfully implemented Appropriate test strings.
- Accomplished an attribute/action interpreter.
- Successfully implemented a translator with hat operators that produce terms which run with my eSOS rules.

Overall, I thoroughly enjoyed this module throughut the term and with the helpp of the TA's and lecturer I learned new valuable key skills which I can implement in my professional career.