XL language: Introduction

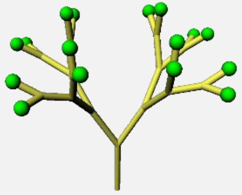
Katarína Smoleňová

FSPM & GroIMP Workshop

PMBDA 2025, Nanchang, China

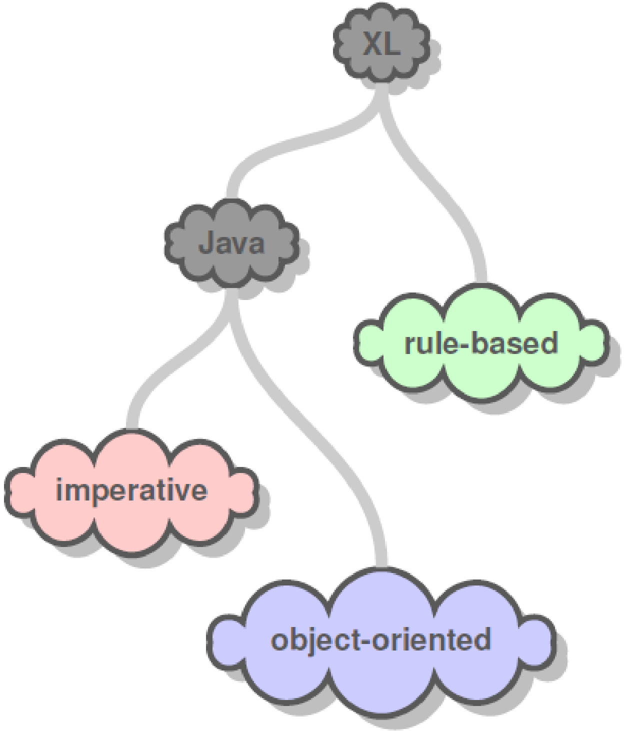
November 1, 2025

```
module A(float len) extends Sphere(0.1);

int time;

protected void init()
[
    { time = 0; }

    Axiom ==> A(1);
]

public void run()
[
    {
        time++;
        println("time: " + time);
    }

    A(x) ==> F(x) [RU(30) RH(90) A(x*0.8)] [RU(-30) RH(90) A(x*0.8)];
]
```



XL modelling language

- **XL**: e**X**tended **L**-system language
- Combines multiple programming paradigms (ways of structuring code)
- Extension of Java
  - Imperative & object-oriented
- Implements L-systems and graph grammars
  - Rule-based



Imperative programming in XL

- Explicit statements
- Code controls execution flow
- Tasks performed step-by-step

- E.g. photosynthesis calculation

```
// Non-rectangular hyperbola as described by Thornley
// ppfd - photosynthetic photon flux density (umol photons m-2 s-1)
// (absorbed radiation per unit area)
// amax - max photosynthesis rate (umol CO2 m-2 s-1)
// alpha - leaf photosynthetic efficiency (umol CO2 umol-1 photons)
// A - rate of photosynthesis  (umol CO2 m-2 s-1)
static double calculatePS_Thornley(double ppfd, double amax,
    double alpha, double theta) {

    double a = theta;
    double b = alpha*ppfd + amax;
    double c = alpha*ppfd * amax;

    double fn = b - Math.sqrt(Math.pow(b,2) - 4*a*c);
    double A = fn / (2*a);

    return A;
}
```

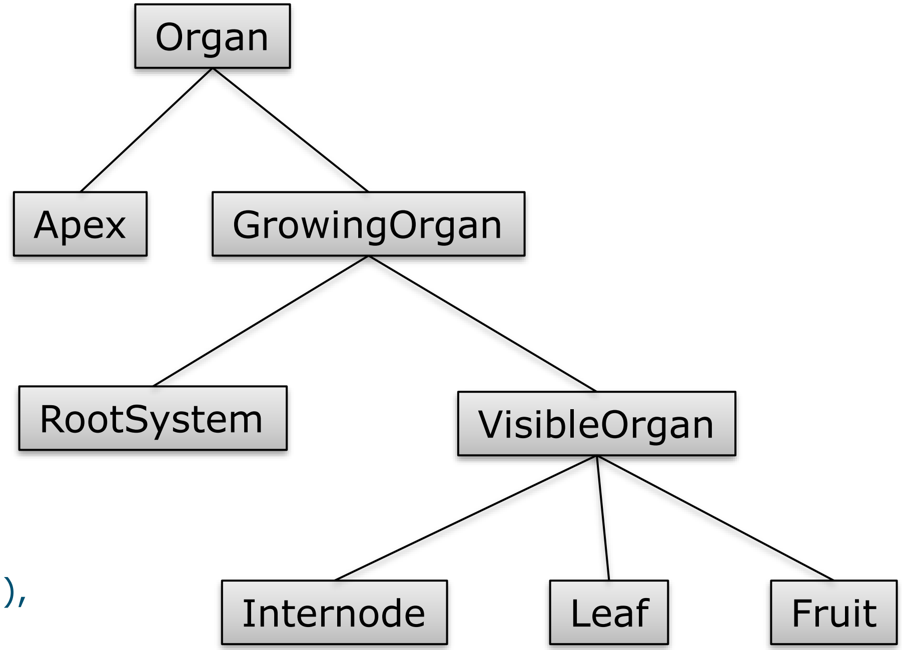
Object-oriented programming in XL

- Code organised around objects (with data and behavious)
- Supports inheritance

- E.g. organ modules

VisibleOrgan

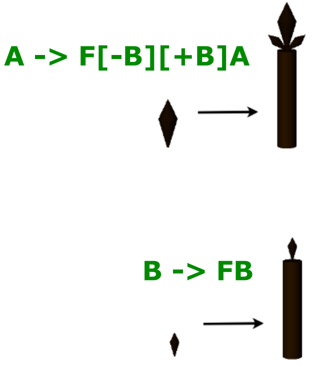
- attributes: absRadiation, assimilates
- methods: calcLight(), calcPhotosynthesis(), calcDimensions()

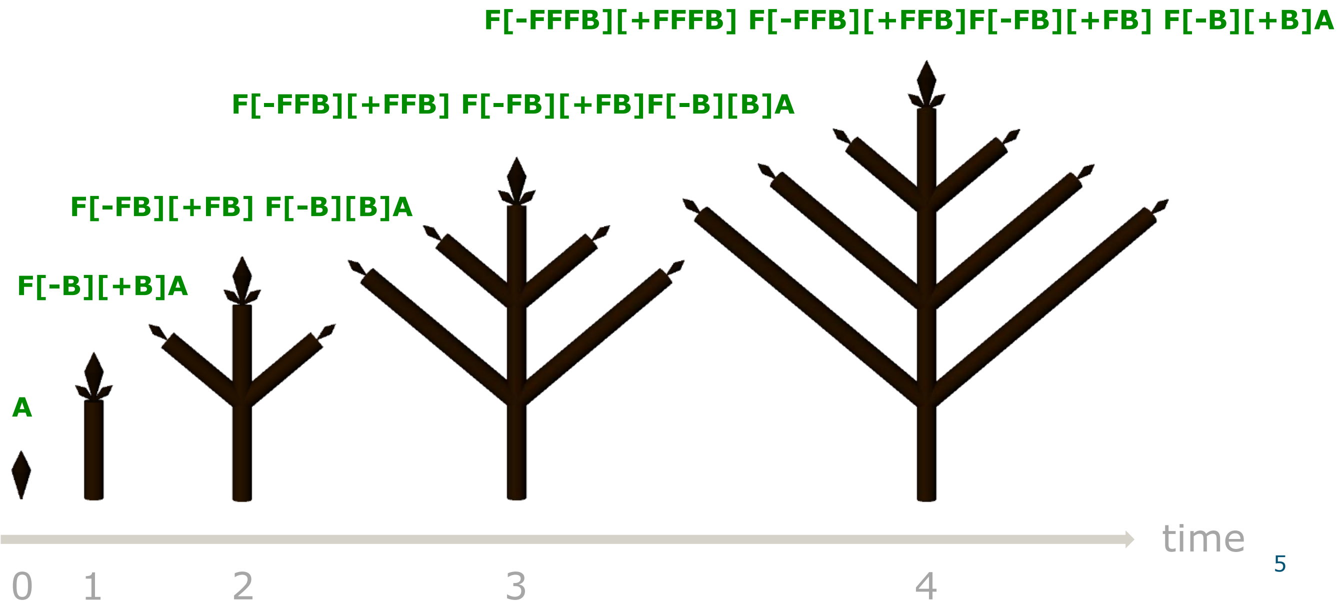


WAGENINGEN UNIVERSITY & RESEARCH

Rule-based programming in XL

- Knowledge represented in form of rules; rules define what to do
- Rules applied as long as possible: *matching* & *rewriting* process





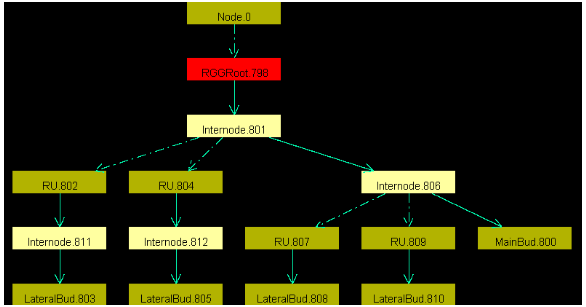
WAGENINGEN
UNIVERSITY & RESEARCH

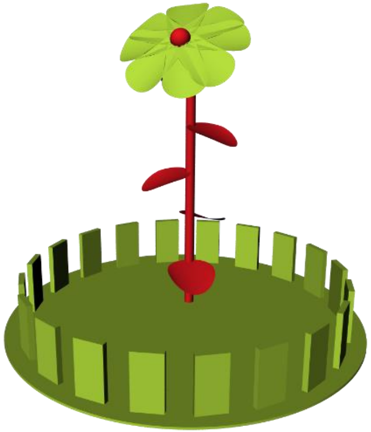
XL implements relational graph grammars

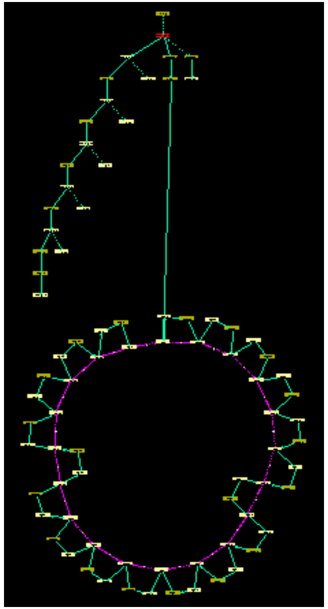
- **RGG - relational growth grammars**

  allow for the definition of L-system style rewriting rules on a *graph* structure and new type of *relations*

F[-FB][FB] F[-B][B]A







Use of imperative code in XL

Try this out

- XL rules and Java code can be mixed and nested

rule-based blocks: **[…]**
imperative blocks: **{ … }**

```
module A(float len) extends Sphere(0.1);

int time;

protected void init()
[
    { time = 0; }

    Axiom ==> A(1);
]

public void run()
[
    {
        time++;
        println("time: " + time);
    }

    A(x) ==> F(x) [RU(30) RH(90) A(x*0.8)] [RU(-30) RH(90) A(x*0.8)];
]
```

```
// Alternatively:
{
    time = 0;
    [
        Axiom ==> A(1);
    ]
}
```

WAGENINGEN
UNIVERSITY & RESEARCH

Some useful operators and methods

Comparison operators:

```
==      // equal to
!=      // not equal to
>       // greater than
<       // less than
>=      // greater than or equal to
<=      // less than or equal to
```

Logical operators:

```
&&      // logical AND
||      // logical OR
!       // logical NOT
```

Mathematical constants:

```
Math.PI     // the value of pi
Math.E      // Euler's number
```

Mathemathical functions:

```
Math.sin(x)                 // the sine of x (in radians)
Math.cos(x)                 // the cosine of x (in radians)
Math.tan(x)                 // the tangent of x  (in radians)
Math.toDegrees(radians)     // converts an angle from radians to degrees
Math.toRadians(deg)         // converts an angle from degrees to radians
Math.sqrt(x)                // the square root of x
Math.exp(x)                 // e raised to the power of x e^x
Math.pow(x, y)              // x raised to the power of y x^y
Math.log(x)                 // the natural logarithm (base e) of x
Math.abs(x)                 // the absolute value of x
Math.min(x, y)              // the smaller of two values
Math.max(x, y)              // the larger of two values
Math.round(x)               // rounds x to the nearest integer
```

XL functions for pseudorandom numbers

probability(x) - returns 1 with probability x, and 0 with probability 1-x

random(a, b) - generates a uniformly distributed pseudorandom number between *a* and *b*

irandom(a, b) - generates a uniformly distributed integral pseudorandom number between *a* and *b*

normal(mu, sigma) - generates a normally distributed pseudorandom number with mean *mu* and standard deviation *sigma*

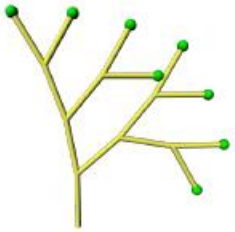
setSeed(s) - sets a starting value for the pseudorandom number generator (to produce identical sequences of pseudorandom numbers)

Math.random() - generates a pseudorandom double number between 0.0 and 1.0

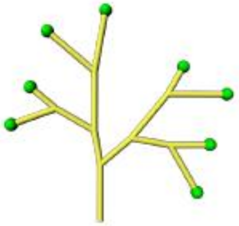
Adding randomness to XL models

Try this out

```
module A extends Sphere(0.1)
{
    { setShader(GREEN); }
}

protected void init()
[
    Axiom ==> A;
]

public void run()
[
    A ==>
        F(1, 0.1)
        [RU(50) A] [RU(-10) A]
    ;
]
```



```
module A extends Sphere(0.1)
{
    { setShader(GREEN); }
}

protected void init()
[
    Axiom ==> A;

    { setSeed(10); }
]

public void run()
[
    A ==>
        F(random(0.5, 1), 0.1)

        if (probability(0.5)) (
            [RU(50) A] [RU(-10) A]
        ) else (
            [RU(-50) A] [RU(10) A]
        )
    ;
]
```



RGG rule

- RGG – relational growth grammars - define rules for graph transformations

Essential RGG rule:

left hand side ==> right hand side

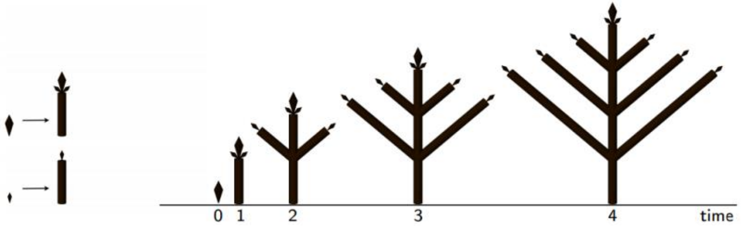
Complete RGG rule (5 parts):

(* context *), left hand side, (condition)

==>

right hand side { imperative code };

Types of rules



- 3 different types of rules

==> L-system rule

```
BudT ==> Internode(1) [RU(-30) BudL] [RU(30) BudL] BudT;
BudL ==> Internode(1) BudL;
```

::> Execution rule (no changes on the graph structure)

```
ModuleName ::> { imperative_code; }
i:Internode ::> { i[length] += 0.1; }
i:Internode ::> { i[age]++; }
```

==>> General graph-rewriting rule

```
b1:BudL, b2:BudL, ((b1 != b2) && (distance(b1, b2) < 0.5 )) ==>> b1 -c-> b2;
```

WAGENINGEN UNIVERSITY & RESEARCH

wiki.grogra.de

fspm

fspm.discourse.group