Domain Specific (Modelling) Languages

Lecture: Model Transformations with ETL

by Prof. Vasco Amaral

2022/2023

ETL

Within the Epsilon Framework can be used to:

Transform an arbitrary number of input models into an arbitrary number of output models.

Abstract Syntax

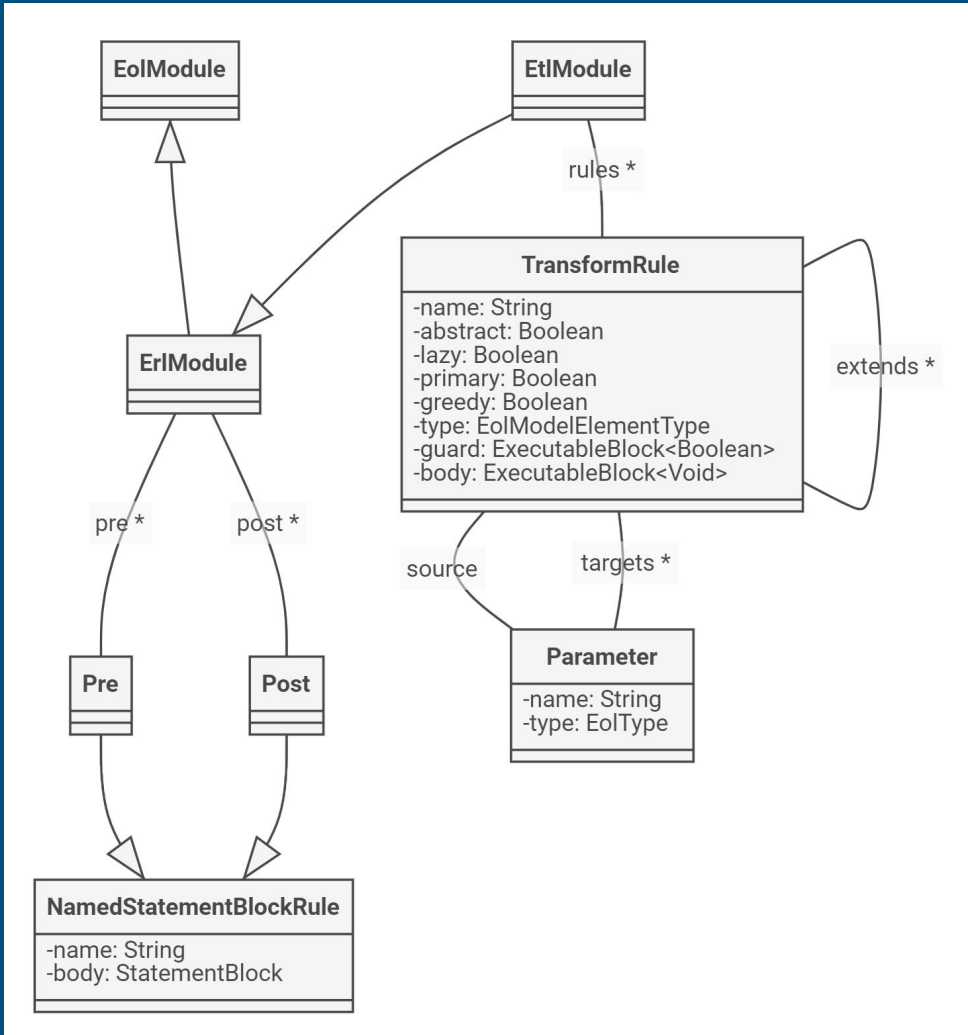
Transformation Rules can be:

- abstract
- primary
- or lazy

Guards (EOL statements) to limit its applicability

Body has EOL statements for populating the property values

There can be pre and post blocks



Concrete Syntax and Transformation Rule

```
1  (@abstract)?
2  (@lazy)?
3  (@primary)?
4  rule <name>
5    transform <sourceParameterName>:<sourceParameterType>
6    to <rightParameterName>:<rightParameterType>
7       (, <rightParameterName>:<rightParameterType>)*
8    (extends <ruleName>(, <ruleName>)*)? {
9
10   (guard (:expression)|({statementBlock}))?
11
12   statement+
13 }
```

Concrete Syntax Pre and Post blocks

```
1  (pre|post) <name> {
2    statement+
3  }
```

ETL example of a tree to graph

```
1  rule Tree2Node
2   transform t : Tree!Tree
3   to n : Graph!Node {
4
5   n.label = t.label;
6
7   if (t.parent.isDefined()) {
8    var edge = new Graph!Edge;
9    edge.source = n;
10   edge.target = t.parent.equivalent();
11  }
12 }
```

equivalent() and equivalents() - automatically resolve source elements to their transformed counterparts in the target models.
It inspect the established transformation trace and invokes the applicable rules(if necessary) to calculate the counterparts in the target model

Execution Semantics - Rule and Block overriding

ETL can import other ETL modules.

The importing ETL module inherits all the rules and pre/post blocks specified in the modules it imports (recursively).

If the module specifies a rule or a pre/post block with the same name, the local rule/block overrides the imported one

Rule Execution Scheduling

Per order of execution:

1. the first things to be executed are the pre blocks
2. following that, each non-abstract and non-lazy rule for all elements which it is applicable
   a. the element must have a **type-of** relationship with the defined **sourceParameter** type in the rule
   b. an element can have the **kind-of relationship** with the defined **sourceParameter** type in the rule, if it is annotated as @greedy
   c. the **guard** must be satisfied

Rule Execution Scheduling

3. when the rule is executed on an applicable element

    a. the target elements are created by instantiating the **targetParameter** of the rules
    b. the contents are populated using the EOL statements of the body of the rule

4. Finally, when all the rules have been executed, the **post** blocks of the module are executed in the order in which they have been declared

Source Elements Resolution

The body of the rule has frequently to solve target elements

that have been or can be transformed from source elements of other rules

EOL provides the following operations to help this task:

equivalents() -

- when applied to a single source object, inspects the transformation trace and invokes the applicable rules to calculate the counterparts of the element in the target model
- when applied to a collection, it returns a Bag containing Bags that contain counterparts of the source elements contained in the collection

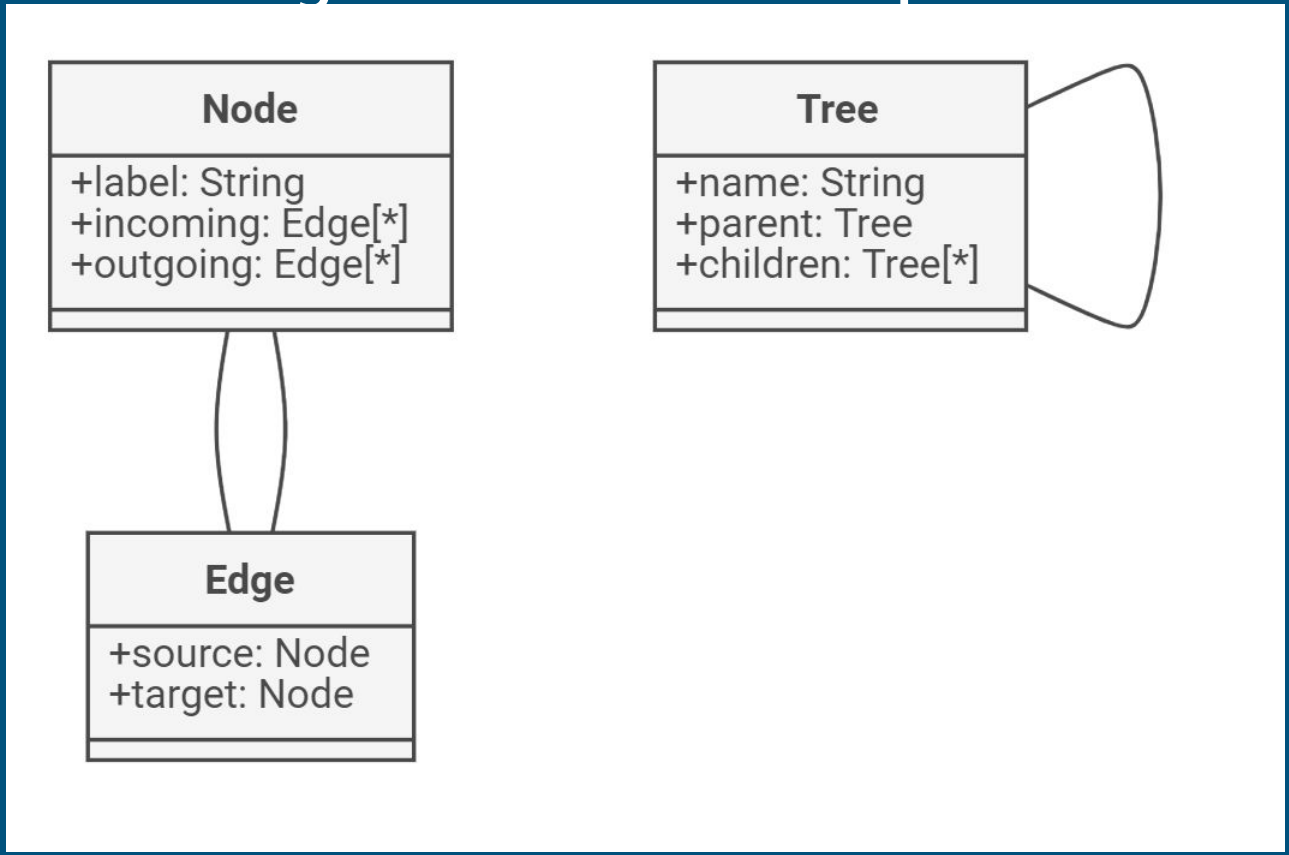
Source Elements Resolution

ETL also provides:

equivalent() -

- when applied to a single source object, returns only the first element of the respective result that would have been returned by the equivalents()
- when applied to a collection, it returns a flattened collection

Transforming a Tree to a Graph



Transforming a Graph to Tree

```
rule Tree2Node
    transform t : Tree!Tree
    to n : Graph!Node {

    n.label = t.label;

    if (t.parent.isDefined()) {
        var edge = new Graph!Edge;
        edge.source = n;
        edge.target = t.parent.equivalent();
    }
}
```

Transforming a Graph to Tree (more simply)

```
rule Tree2Node
    transform t : Tree!Tree
    to n : Graph!Node {

    n.label = t.label;

    if (t.parent.isDefined()) {
        var edge = new Graph!Edge;
        edge.source = n;
        edge.target ::= t.parent;
    }
}
```

where the **::=** does the same as equivalent()

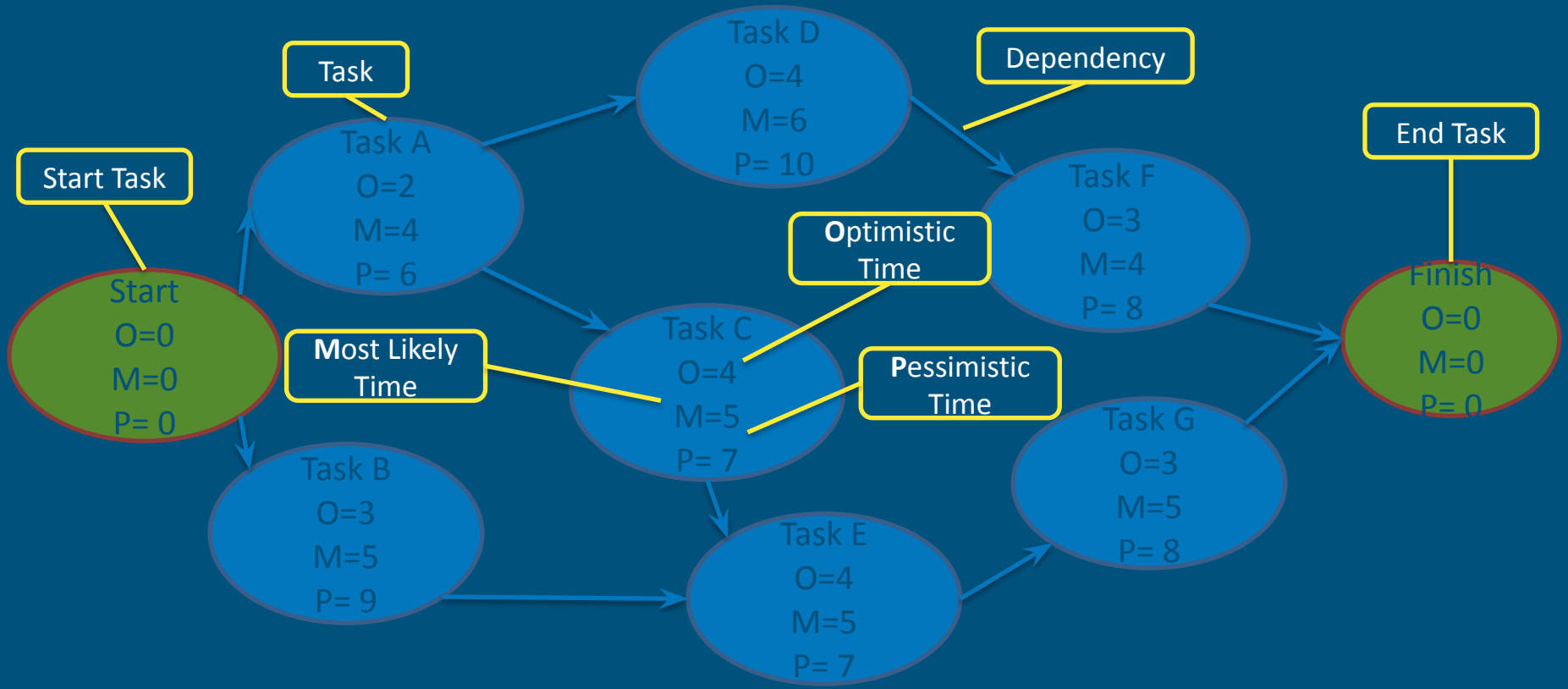
ETL - Interactive Transformations

```
rule Tree2Node
    transform t : Tree!Tree
    to n : Graph!Node {

    guard : UserInput.confirm
        ("Transform tree " + t.label + "?", true)

    n.label = t.label;
    var target : Graph!Node ::= t.parent;
    if (target.isDefined()) {
        var edge = new Graph!Edge;
        edge.source = n;
        edge.target = target;
    }
}
```

Colours Study

Pert Case Study

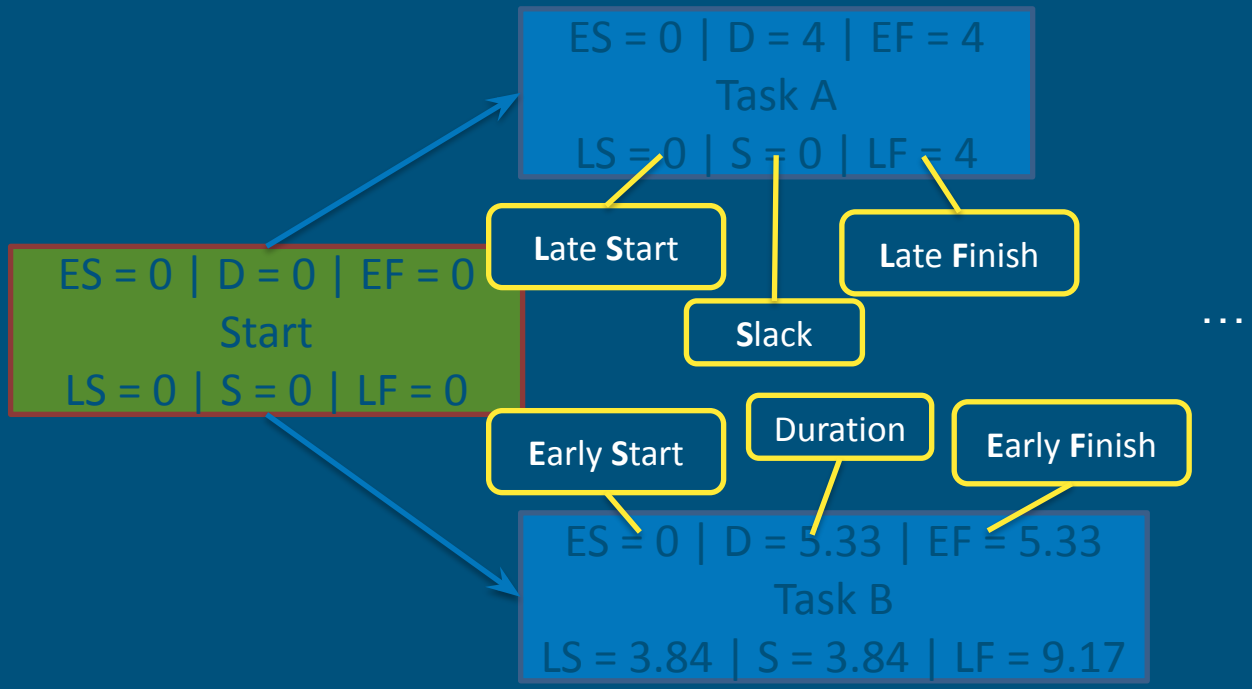
Domain Analysis (PERT)



Model Validation

- No cyclic dependencies.
- No loose tasks.
- One and only Start Task.
- One and only End Task.
- Pessimist is pessimist.
- Optimism is optimist.
- End is End.
- Start is Start.
- Etc…

Domain Analysis (Activity on Node)

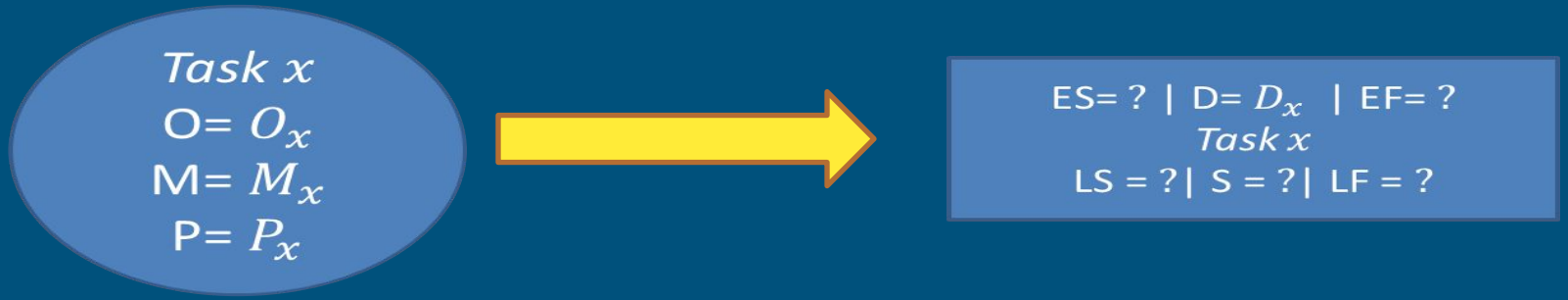


…

To be completed…

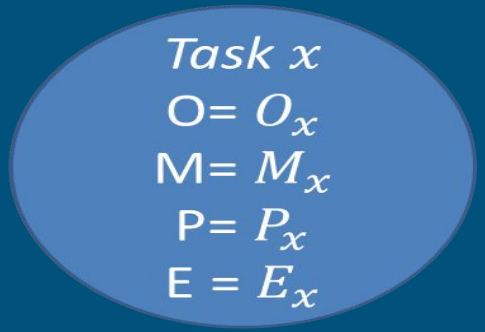
ES = 0 | D = 0 | EF = 0
End
LS = 0 | S = 0 | LF = 0

M2M (out-place)



$$D_x = \frac{O_x + 4M_x + P_x}{6}$$

M2M (in-place)

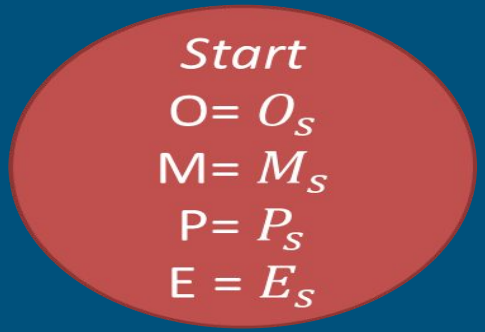


ES= ES_x | D= D_x | EF= EF_x
Task x
LS = LS_x | S = S_x | LF = LF_x

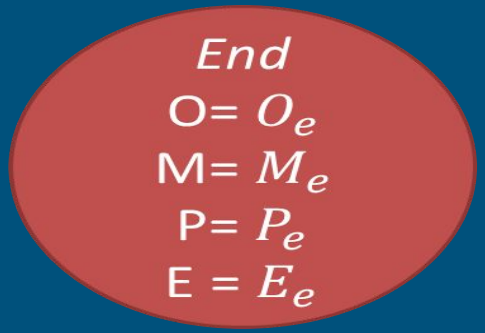
$$ES_x = \max(EF_{y \to x})$$
$$EF_x = ES_x + D_x$$
$$LF_x = \min(LS_{x \to y})$$
$$LS_x = LF_x - D_x$$
$$S_x = LF_x - EF_x$$

Legend:
$EF_{y \to x}$ Set of EF of all predecessors of x
$LS_{x \to y}$ Set of LS of all successors of x

M2M (in-place)

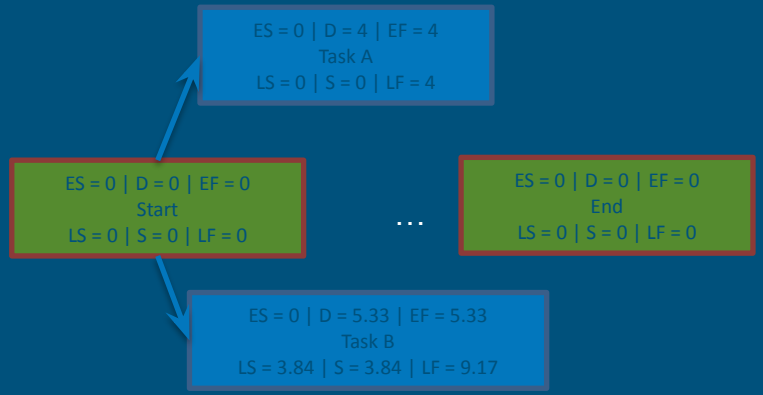


ES = 0 | D = D_s | EF = EF_s
Start
LS = LS_s | S = S_s | LF = LF_s

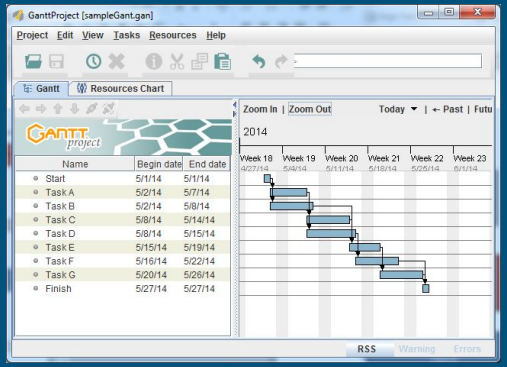


ES = ES_e | D = D_e | EF = EF_e
End
LS = LS_e | S = S_e | **LF = EF_e**

M2T




```
<task id="0" name="Start" … >
 <depend id="2" type="2"
      difference="0" hardness="Strong“ />
 …
</task>
…
```



Bibliography

- http://web2.concordia.ca/Quality/tools/20pertchart.pdf
- http://en.wikipedia.org/wiki/Program_Evaluation_and_Review_Technique