DAT240 / DIT596
Assignment 3- Concrete Syntax

Model-Driven Engineering

Group 1

Supriya Supriya (supe@student.chalmers.se)
Himanshu Chuphal (guschuhi@student.gu.se)
Kristiyan Dimitrov (gusdimkr@student.gu.se)

List of Figures

1. Introduction

This assignment address the generation of textual concrete syntax for the manufacturing system metamodel. We focus on mapping the abstract syntax into the concrete syntax using Xtext to make the meta model more easily understandable in terms of natural language.

2. Concrete Syntax

Images 1, 2 and 3 (in *Appendix*) show examples of the concrete syntax and its usage. First, a 'ManufacturingSystem is created and named, we decided to use the curly brackets {} for indentation, resembling java. This will allow whoever is using the language and has had java experience to be familiar with the indentation, nevertheless some differences do exist. After the ManufacturingSystem is created the user of the language must declare which its starting storage should be, followed by the end storage. At first, it will give errors as the storages need to be actually initiated but that is done further down. We decided to keep it up similar to how variable declarations are done on top, frankly, it's a design decision. Next, the responsible people need to be created, separated by a ','. Once all of that is done the lead responsible is set (the person in charge of the manufacturing system), workpiece types The first Method is clearly more readable and easier to understand but increases the amount of typing the programmer has to conduct, whereas the second method decreases the tedious typing but can have an impact on the understandability of the code. For simpler systems it can be good to nest inside one initiate, encapsulating everything, but for more complex systems initiating each one separately will make it easier to track, particularly so due to the hierarchical implementation of the system where a CompositeManufacturingStep contains an entirely new manufacturing system and it's a trivial task to get lost in the code nests. Similarly to the manufacturing system, each Step or Storage has its own encapsulation using {}, where its inner variables are declared (e.g. speed). Note that the responsible people declared in the manufacturing system are actually global variables that can be assigned inside the steps.

The snapshots of Concrete Syntax examples (Image 1-3) are shared in the ***Appendix***.

3. Discussion

Apart from the changes made in the original grammar generated from xtext through the modifications of terminals. The following main modifications were made:

- Modification to the binary condition:

A binary condition takes in two conditions with the type ‘or’ or ‘and’ in between. Instead of a prefix expression as in original grammar, we modified it to infix, with input conditions in the left and the right and type in between, to make it easily understandable.

```
BinaryCondition returns BinaryCondition:
    'BinaryCondition'
    name=EString
    '{'
        'left' left=[InputCondition|EString]  type=BinaryType 'right' right=[InputCondition|EString]

    '}';
```

This is illustrated in Image 2, where the binary condition ‘cond’ is imposed on the input workpieces, with either the satisfaction of the input condition in the left or the right of ‘cond’.

- Modifications to ManufacturingSystem

The main changes were done within the ManufacturingSystem as it is the class that encompasses all of the functionality that the language supports. Most of the changes done from the generated code were namely keyword additions, restructuring as to determine the ordering of the declarations (as mentioned in section 2, first the storages are set then responsible people are created in that explicit order, etc.). Additionally, more critical changes were made to the ‘consistsOf’ to make it more sensible and allow the 2 approaches of initiation which were not pre-defined by the generated code.

```
7  ManufacturingSystem returns ManufacturingSystem:
8      'ManufacturingSystem'
9      name=EString
10     '{'
11         'set' 'start' 'storage' start+=[StoragePoint|EString] ( "," start+=[StoragePoint|EString])*
12         'set' 'end' 'storage' end+=[StoragePoint|EString] ( "," end+=[StoragePoint|EString])*
13         'create' 'responsible' staff+=Responsible ( "," staff+=Responsible)*
14         'set' 'lead' 'responsible' responsible=[Responsible|EString]
15         ('workpiece' 'type' uses+=WorkPieceType ( "," uses+=WorkPieceType)*)?
16         ('processes' transforms+=WorkPiece ( "," transforms+=WorkPiece)*)?
17         ('initiate' ((consistsOf+=ManufacturingSystemElement)*   | ('{' consistsOf+=ManufacturingSystemElement*'}')))*
18
19     '}';
```

- Other modifications

Most of the other modifications done to the rest of the elements are namely removal of unnecessary keywords, repositioning of keywords to make more sense and be more user-friendly. These changes are simple, straightforward and clear enough to be understood by viewing the xtext file provided with this pdf.

Appendix

ManuSys.mydsl

```
ManufacturingSystem ManuSys300
{
    set start storage raw
    set end storage processed
    create responsible Mike, Charlie, Willies
    set lead responsible Mike
    workpiece type Wood, Metal, Stone
    processes WorkPiece {of type Stone}, WorkPiece {of type Metal}

/*Way 1 - initiate each seperately */
    initiate StoragePoint raw {
            set responsible Charlie
            stores Stone
    }
    initiate StoragePoint processed {
            set responsible Willies
            stores Metal
            transition manuStep
    }
    initiate CompositeManufacturingStep underling {
            speed = 30
            set responsible Willies
            input {InputCondition imp}
            output {input condition imp}
            contains ManufacturingSystem ManuSys500{
                set start storage rawMats
                set end storage processedMats
                create responsible Himanshu, Rick, Morty
                set lead responsible Rick
                workpiece type Glass, Plastic, Marble

                initiate StoragePoint rawMats{
                    set responsible Himanshu
                    stores Glass
                }
                initiate StoragePoint processedMats {
                    set responsible Morty
                    stores Marble
                }
```

Image 1: Concrete Syntax - 1

ManuSys.mydsl

```
                initiate ManufacturingStep melt {
                    speed = 10
                    set responsible Charlie
                    input{InputCondition imp,
                        Negation neg {contains {condition imp}},
                        BinaryCondition cond {left imp or right neg}
                    }
                    output{input condition imp}
                }

            }

    }

/*Way 2 - nest the manufacturing steps inside a single initiate */
/*Note - increases complexity but reduces typing */
    initiate {
        StoragePoint something {
            set responsible Charlie
            stores Wood
        }
        StoragePoint another {
            set responsible Willies
        }
        ManufacturingStep manuStep {
            speed = 10
            set responsible Charlie
            input{InputCondition imp,
                BinaryCondition cond {left imp or right imp}
            }
            output{input condition imp}
        }
        TransportStep wagon {
            speed = 10
            set responsible Charlie
            input {InputCondition imp}
            output {input condition imp}
        }
```

Image 2: Concrete Syntax - 2

```
        QualityAssuranceStep qas {
            speed = 10
            set responsible Willies
            input {InputCondition con}
            output {input condition con}

        }
        CompositeManufacturingStep manu600 {
            speed = 30
            set responsible Willies
            input {InputCondition imp}
            output {input condition imp}
            contains ManufacturingSystem ManuSys400 {
                set start storage something
                set end storage another
                create responsible Kris, Supria
                set lead responsible Kris
                initiate ManufacturingStep CutWood {
                    speed = 20
                    set responsible Supria
                    input {InputCondition imp}
                    output {input condition imp}
                }
                initiate StoragePoint bang{
                    set responsible Supria
                }
            }
        }
    }

}
```

Image 3: Concrete Syntax - 3

References

1. https://eclipse.org/Xtext/documentation/101_five_minutes.html (Links to an external site.)Links to an external site.
2. Eclipse Org EMF https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.emf.doc%2Freferences%2Foverview%2FEMF.html