DAT240 / DIT596
Assignment 2
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

There are several different ways to specify OCL constraints in Ecore. Here, we will use both the (interactive) OCL console, where you can directly execute expressions on single model elements in a console-like fashion, and the OCLinEcore editor, which allows you to place constraints in the meta-model in a Java-like fashion. In both cases, the context is implicitly given. For the OCL console, the context is dependent on which model element you currently have selected. For the OCLinEcore editor, the context depends on where you place the constraint in the code.

2. Class Diagram

Manufacturing System meta-model from assignment 1.

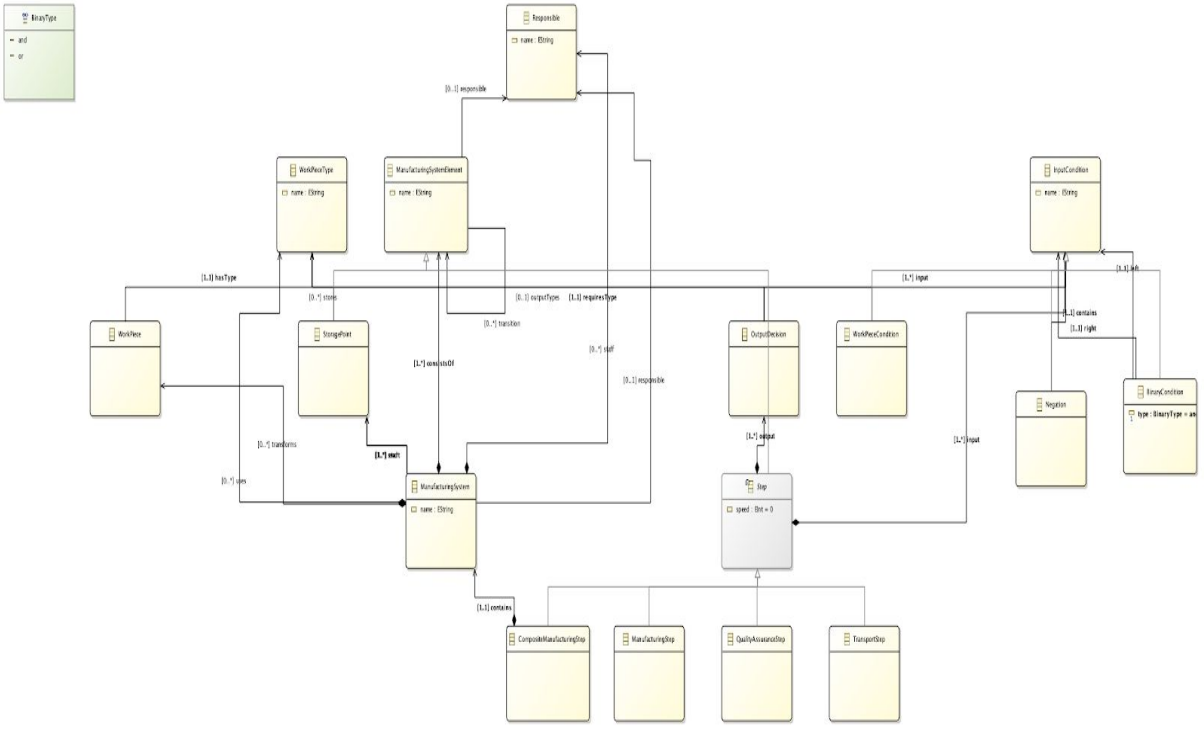

Figure 1: Class Diagram

3. Section A - Execute OCL constraints

3.1. **Expression:** `consistsOf.transition`
Context: `ManufacturingSystem FurnitureSystem`
This line returns a collection of transitions (the steps) of the Manufacturing System.

3.2. **Expression:** `self.consistsOf ->forAll(m:ManufacturingSystemElement| m.name <> null)`
Context: `ManufacturingSystem FurnitureSystem`
For all the items in the returned collection of ManufacturingSystemElement, enforces that the name cannot be null and returns true if none of the names are null.

3.3. **Expression:** `consistsOf ->select(oclIsKindOf(Step)).oclAsType(Step).speed->sum()`
Context: `ManufacturingSystem FurnitureSystem`
Create a subcollection of all the steps that are direct to the Manufacturing System, get access to the variable speed of all items in the collection and return the sum.

3.4. **Expression:**
`self.output.input->forAll(i:InputCondition|self.input->includes(i))`
Context: `TransportStep EndTransport`

Enforces all items of the collection to include the InputCondition and returns true if InputCondition is included in the collection (Binary Condition input_left and Binary Condition input_right in this instance)

[Or, To check if all the input workpieces to the ending transportstep contain the condition on them or not by returning a Boolean solution]

3.5. **Expression:**
`self.transition->exists(m:ManufacturingSystemElement| oclIsTypeOf(QualityAssuranceStep)) implies self.oclIsKindOf(ManufactoringStep)`
Context: `CompositeStep CuttingComposite`
Checks if QualityAssuranceStep exists in the collection of transitions and which implies the transition is either the direct type or one of the supertypes of the ManufactoringStep.

3.6. **Expression:** `ManufacturingSystem.allInstances()->forAll(ms:ManufacturingSystem|ms.transforms.hasType->includes(self) implies ms.uses->includes(self))`
Context: `WorkPieceType CutMetal`
Returns a Set containing all of the existing instances of all ManufacturingSystem having WorkPieceType CutMetal and then iterates over all ManufacturingSystems transforming a WorkPiece of a WorkPieceType and checks if all the WorkPieces are contained within the same ManufacturingSystem and implies that the same ManufacturingSystem is using the same WorkPieceType..
Return true if both the conditions are false *(false implies false is true)*

4. Section B - Extend the Meta-model

4.1. Each Step shall have exactly one Responsible.

```
invariant oneResponsible: self.consistsOf
->forAll(m:ManufacturingSystemElement| m.responsible
-> size() = 1);
```

This result is validated by checking the responsible in the properties of the ManufacturingSystem class.

4.2. Each Responsible shall be responsible for between one and three (inclusive) Steps.

Expression:

```
invariant checkResponsbile:

self.consistsOf

->forAll(m:ManufacturingSystemElement| m.responsible
-> size() > 1 and m.responsible -> size() <= 3)
```

Context: `ManufacturingSystemElement`

This result is validated at each step in ManufacturingSystem has one responsible assigned to it and none has been assigned to more than to more than 3.

4.3. There shall not exist a single Step in a ManufacturingSystem which has the same Responsible as the ManufacturingSystem itself.

Expression:

Invariant limitResponsible:

```
self.consistsOf ->
forAll(m:ManufacturingSystemElement|
m.responsible.name <> self.responsible.name);
```

Context: ManufacturingSystemElement

The constraint checks if the responsible name within the ManufacturingSystem is not the same in the ManufacturingSystemElement This constraint is violated as there exists the same responsible for the step and the system.

4.4. No start StoragePoint shall have incoming Transitions.

Expression:

```
invariant startLimit: self.consistsOf ->
forAll(m:ManufacturingSystemElement| m.transition <>
self.start);
```

Context: ManufacturingSystemElement

4.5. No end StoragePoint shall have outgoing Transitions.

Checks whether the ending storagepoint of a composite step has any transitions leaving from it. The result is validated

Expression:

```
invariant endLimit:

self.contains.end.transition -> isEmpty()
```

Context: StoragePoint CompositeManufacturingStep

4.6. Transitions shall only connect ManufacturingSystemElements within the same ManufacturingSystem. E.g. in the example in Figure 1, it would not be allowed to connect RawMaterialStorage directly to the Step Cutting, as Cutting is inside another ManufacturingSystem (CuttingComposite).

Expression:

```
invariantlinkElem:
self.consistsOf ->
forAll(m:ManufacturingSystemElement| m.transition ->
includes(self.consistsOf));
```

Context: ManufacturingSystem

4.7. Outputs and inputs of connected Steps shall be compatible. That is, the input condition of a Step has to include at least one condition on a WorkPieceType that is produced by at least one Step connected to it via an incoming transition. E.g. in the example in Figure 1, the output of Step Cutting is compatible with Step Drilling, as at least one of the possible outputs of Cutting (CutWood and CutMetal) occur in the input of Drilling.

Expression:

```
self.input -> intersection(self.output.input)
->notEmpty ();
```

Context: Manufacturing Step CuttingComposite

4.8. For composite steps, the input needs to be compatible (see 7.) with the types stored in one of the start points and the output needs to be compatible with the types stored in one of the end points.

Expression:

```
invariant compatible_stores:

self.input ->
select(oclIsKindOf(WorkPieceCondition)).oclAsType(Wo
rkPieceCondition)
-> intersection(self.contains.start.stores)
->union(self.output.outputTypes
-> intersection(self.contains.end.stores))
->notEmpty ();
```

Context: Manufacturing Step CuttingComposite

4.9. For each Step, the previous ManufacturingSystemElement (if existent) shall either be a StoragePoint or the speed of that Step shall be lesser or equal than the speed of the current Step. That is, the speed of the

manufacturing system cannot suddenly become lower without a storage (bottleneck).

Expression:

invariant Storagepoint_bottleneck:

```
self.consistsOf.transition

-> exists(m:ManufacturingSystemElement|
oclIsTypeOf(StoragePoint)) implies
self.consistsOf.transition
-> select(oclIsKindOf(Step)).oclAsType(Step)
-> forAll(p1, p2 | p1 <> p2
implies p2.speed <= p1.speed)
```

Context: ManufacturingSystem FurnitureSystem

Acronyms

1. OCl Object Constraint Language
2. UML Unified Modeling Language

References

1. Course “OCL By Example“
   http://st.inf.tu-dresden.de/files/general/OCLByExampleLecture.pdf

2. Eclipse Org EMF
   https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.emf.doc%2Freferences%2Foverview%2FEMF.html