# Execute OCL Constraints

## Execution

**Evaluating:** self.consistsOf.transition

**Results:**

Manufacturing System FurnitureSystem::Composite Manufacturing Step CuttingComposite

Manufacturing System FurnitureSystem::Transport Step StartStorageTransport

Manufacturing System FurnitureSystem::Composite Manufacturing Step CuttingComposite

Manufacturing System FurnitureSystem::Composite Manufacturing Step CuttingComposite::Manufacturing System ComponentPreparationSystem1::Manufacturing Step Cutting

Manufacturing System FurnitureSystem::Quality Assurance Step productQA

Manufacturing System FurnitureSystem::Transport Step CuttingCompositeTransport

Manufacturing System FurnitureSystem::Storage Point ProductStorage

Manufacturing System FurnitureSystem::Storage Point RawMaterialStorageExplanation

## Explanation

In this OCL-Expression we first iterate through all the **ManufacturingSystemElement** instances, which are referenced by our **ManufacturingSystem**’s **consistsOf** reference. Then for each of these we iterate through the **ManufacturingSystemElement** instances referenced by their **transition** reference. In the end we don’t check anything resulting in a Boolean, so the execution just returns all the **ManufacturingSystemElement** instances we iterated over. The final output lists all the elements which are directly **transition**ed into by elements directly referenced through **consistsOf** by our system.

## Execution

**Evaluating:** self.consistsOf->forAll(m:ManufacturingSystemElement | m.name <> null)

**Results:** true

## Explanation

Like in the previous expression we iterate through all the elements referenced by **consistsOf** from our system. Then we check for all of them, if their **name** property is not null. This will only return **true**, if every element referenced by **consistsOf** has a non-null **name** property. Unfortunately, an empty string is also not null, so this will return true even if elements are not properly named.

## Execution

**Evaluating:** self.consistsOf->select(oclIsKindOf(Step)).oclAsType(Step).speed->sum()

**Results:** 70

## Explanation

We first iterate through all the elements our system references with **consistsOf**. Then we limit the collection to only those which are based on the **Step** class. Then we cast these elements to the **Step** class, which has a **speed** property, for which we return the sum of. So basically, the sum of the speeds of all steps directly referenced by **consistsOf** from our system.

## Execution

**Evaluating:** self.output.input->forAll(i:InputCondition | self.input->includes(i))

**Results:** true

## Explanation

[…]

## Execution

**Evaluating:** self.transition->exists(m:ManufacturingSystemElement | m.oclIsTypeOf(QualityAssuranceStep)) implies self.oclIsKindOf(ManufacturingStep)

**Results:** false

## Explanation

First, we iterate through all **transition** references of element. Then we check if at least one of the elements we **transition** into is an instance of the **QualityAssuranceStep** class. If that is true, we also need to check if we are an instance of **ManufacturingStep** or any of its subclasses. So in our example, when the expression is executed for an instance of **CompositeManufacturingStep**, which has a **transition** reference to a **QualityAssuranceStep** instance, it will return false, because 1. It transitions into a **QualityAssuranceStep** and 2. It’s not an instance of **ManufacturingStep** *(or any of it’s subclasses, even though we don’t have subclasses of it).*

## Execution

**Evaluating:** ManufacturingSystem.allInstances()

->forAll(ms:ManufacturingSystem|ms.transforms.hasType

->includes(self) implies ms.uses->includes(self))

**Results:** true

## Explanation

We iterate through all instances of **ManufacturingSystem** and check for all of them, if at least one of their **WorkPiece** instances referenced by **transform** references the **WorkPieceType** instance, which we are running this expression on, via its **hasType** reference. If that is the case, we also need to check if this system’s **uses** reference collection includes our **WorkPieceType**. So this would return true for any system, which does not **transform** any workpiece of our type and for those which do but also includes our workpiece in their **uses** reference.