Jason Liu

Assignment 2 results

Woods.py

Does the example model conform to the example meta-model or are there errors?

Can you explain each constraint violation?

Target cardinality of type afraidOf (0) out of bounds (1..inf) in billy.

- Needs to be afraid of at least one person

Local constraint of "Man_weight" in "george_weight" not satisfied.

- Gearge is too skinny; he needs to be 21 kg at least.

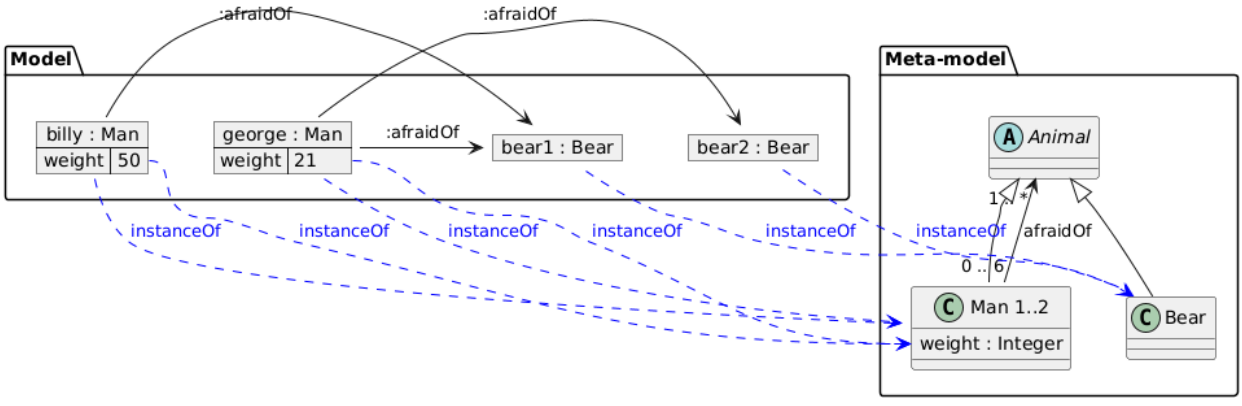
Local constraint of "Man" in "george" not satisfied.

- Same as above

Global constraint "total_weight_small_enough" not satisfied.

- Sum of weight exceeded 85

Generate PlantUML for the updated model



Jason Liu

Factory.py

Specification + Constraints

A Factory has at least one Worker working for it.

```
hasWorker:Association (Factory -> Worker) {
    target_lower_cardinality = 1;
}
```

This constraint tells us that the lower bound is 1, so a factory needs to be connected to at least one worker.

A Factory contains at least one Machine.

```
hasMachine:Association (Factory -> Machine) {
    target_lower_cardinality = 1;
}
```

This constraint is the same as the one above.

There are exactly three Shifts: morning, afternoon, and night.

```
threeShifts:GlobalConstraint {
    constraint = ```
        len(get_all_instances("Shift")) == 3
    ```;
}

oneMorningShift:GlobalConstraint {
    constraint = ```
        morning_shift = False
        for shift_name, shift_id in get_all_instances("Shift"):
            if shift_name == "morning":
                morning_shift = True
        morning_shift
    ```;
}

oneNightShift:GlobalConstraint {
    constraint = ```
        night_shift = False
        for shift_name, shift_id in get_all_instances("Shift"):
            if shift_name == "night":
                night_shift = True
        night_shift
    ```;
}

oneAfternoonShift:GlobalConstraint {
    constraint = ```
        afternoon_shift = False
        for shift_name, shift_id in get_all_instances("Shift"):
            if shift_name == "afternoon":
                afternoon_shift = True
        afternoon_shift
    ```;
}
```

The first constraint specifies that there needs to be exactly three shifts. The other three specify which shifts need to exist. The combination of these constraint form that these three shifts has to exist and only these three.

Workers work one or two Shifts. No Workers work three Shifts (this would be tough for them) and no Workers work zero Shifts.

```
workShift:Association (Worker -> Shift) {
    target_lower_cardinality = 1;
    target_upper_cardinality = 2;
}
```

This is the same as the first constraint but here we also specify an upperbound.

A Factory has exactly one Source (to receive parts), and one Sink (to send out assembled products).

```
hasSink:Association (Factory -> Sink) {
    target_lower_cardinality = 1;
    target_upper_cardinality = 1;
}

hasSource:Association (Factory -> Source) {
    target_lower_cardinality = 1;
    target_upper_cardinality = 1;
}
```

By specifying that the upper and lower bound is the same, we bound it to exactly that number. In our case this is one.

Connections exist between Machines and the Factory's Source/Sink:

Every Machine has one or two inputs, that connect to (a) another Machine's output, or (b) the Factory's Source.

```
inputsFromAConnectable:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for machine_name, machine_id in get_all_instances("Machine"):
            if len(get_outgoing(machine_id, "receives")) < 1:
                constraintValid = False
        constraintValid
    ```;
}
```

This constraint let us know that every machine has an input connected to a connectable.
```

```
sinkNoProvider:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for sink_name, sink_id in get_all_instances("Sink"):
            outgoing = get_outgoing(sink_id, "provides")
            if len(outgoing) > 0:
                constraintValid = False
        constraintValid
    ```;
}
```

Adding this constraint tells us that the sink can't provide, so the only other one is the source or a machine.

Every Machine has one or two outputs, that connect to (a) another Machine's input, or (b) the Factory's Sink.

We apply the same methodology here but in reverse.

```
outputsToAConnectable:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for machine_name, machine_id in get_all_instances("Machine"):
            if len(get_outgoing(machine_id, "provides")) < 1:
                constraintValid = False
        constraintValid
    ```;
}
```

```
sourceNoReceipiant:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for source_name, source_id in get_all_instances("Source"):
            outgoing = get_outgoing(source_id, "receives")
            if len(outgoing) > 0:
                constraintValid = False
        constraintValid
    ```;
}
```

Cycles among Machines, such as the following, are allowed: Machine feedsTo feedsTo Machine Machine feedsTo Cycles between machines are common in industry. For instance, think of a cooling circuit.

There is no constraint that allows it, it is by default allowed. Our Plantuml image will showcase this.
```

Jason Liu

The Factory's Source and Sink both must be connected to at least one Machine, and never more than two Machines.

```
sourceToMachine:GlobalConstraint {
    constraint = ```
        firstCheck = True
        for source_name, source_id in get_all_instances("Source"):
            outgoing = get_outgoing(source_id, "provides")
            if len(outgoing) < 1:
                firstCheck = False

        secondCheck = False
        for machine_name, machine_id in get_all_instances("Machine"):
            outgoing = get_outgoing(machine_id, "receives")
            if len(outgoing) > 0:
                for link in outgoing:
                    target_id = get_target(link)
                    if get_type_name(target_id) == "Source":
                        secondCheck = True

        firstCheck and secondCheck
    ```;
}
```

This constraint checks if the source provides some connection. Then we also check if the machine actually receives one.

Because it is a connectable, it is also bounded by that it can only have at most 2 outputs.

```
provides:Association (Connectable -> Connectable) {
    target_upper_cardinality = 2;
}
```

The same methodology applied for Sink.
```

```
machineToSink:GlobalConstraint {
    constraint = ```
        firstCheck = False
        for machine_name, machine_id in get_all_instances("Machine"):
            outgoing = get_outgoing(machine_id, "provides")
            if len(outgoing) > 0:
                for link in outgoing:
                    target_id = get_target(link)
                    if get_type_name(target_id) == "Sink":
                        firstCheck = True

        secondCheck = False
        for sink_name, sink_id in get_all_instances("Sink"):
            outgoing = get_outgoing(sink_id, "receives")
            if len(outgoing) > 0:
                for link in outgoing:
                    target_id = get_target(link)
                    if get_type_name(target_id) == "Machine":
                        secondCheck = True

        firstCheck and secondCheck
    ```;
}
```

```
receives:Association (Connectable -> Connectable) {
    target_upper_cardinality = 2;
}
```
```

A Source cannot be connected directly to a Sink.

```
disconnectSinkSource:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for source_name, source_id in get_all_instances("Source"):
            outgoing = get_outgoing(source_id, "provides")
            if len(outgoing) > 0:
                for link_id in outgoing:
                    target_id = get_target(link_id)
                    if get_type_name(target_id) == "Sink":
                        constraintValid = False

        for sink_name, sink_id in get_all_instances("Sink"):
            outgoing = get_outgoing(sink_id, "receives")
            if len(outgoing) > 0:
                for link_id in outgoing:
                    target_id = get_target(link_id)
                    if get_type_name(target_id) == "Source":
                        constraintValid = False

        constraintValid
    ```;
}
```

We just check all outgoing links from source and sink, if these two don't have any that connect to each other the constraint is valid.

Every Machine is operated by zero or more Workers. A Machine that is operated by zero Workers, is considered autonomous (it can function without an operator). For every Shift, every non-autonomous Machine (i.e., one that needs an operator), must have at least one Worker operating it during that Shift.

By default a machine can have zero workers.
```

```
autonomousMachineCheck:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for machine_name, machine_id in get_all_instances("Machine"):
            incoming = get_incoming(machine_id, "operates")
            if len(incoming) >= 1:
                # check if it has a worker for each shift
                morning = False
                afternoon = False
                night = False
                for link_id in incoming:
                    worker_id = get_source(link_id)
                    for link_id2 in get_outgoing(worker_id, "workShift"):
                        shift_id = get_target(link_id2)
                        shift_name = get_name(shift_id)
                        if shift_name == "morning":
                            morning = True
                        elif shift_name == "afternoon":
                            afternoon = True
                        elif shift_name == "night":
                            night = True
                if not (morning and afternoon and night):
                    constraintValid = False
        constraintValid
    ```;
}
```

We just add if there is a worker, we check if every shift is occupied by at least one worker. If this is the case then it is all good. If not, the constraint will be invalidated.

This is simply done by checking every incoming link that operates this machine and checking what shifts that worker is working for.

## Every Worker has a monthly salary, which must be at least 1000 (could be Euros, Dollars, Belgian Francs, ...) to comply with minimum wage legislation.

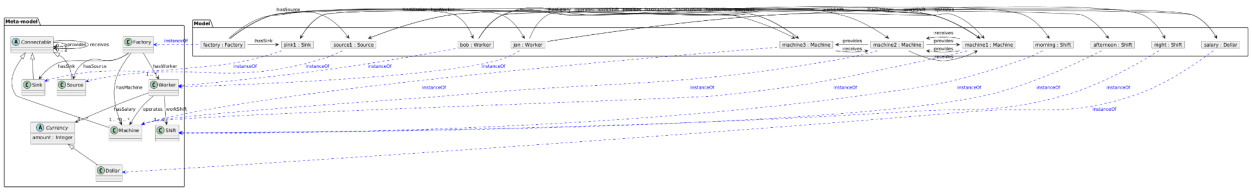
We do this in a more complicated way than needed. We could have just added an attribute that is bound to a worker but instead I added a new class called currency. I chose this method over adding one salary attribute, because currencies have different values and this ensures that the model is extendable with little change to the original model.
```

Jason Liu

First I made a class called currency, that will be an abstract class and inherit any other class that is a currency from this class. I only made use of Dollar in my example but adding euros wouldn't be much work.

```
Currency:Class {
    abstract = True;
}
```

```
Dollar:Class
:Inheritance (Dollar -> Currency)
```

The constraint will be set on the Dollar, because the value of how much you get paid is different for every currency.

```
Dollar_amount:AttributeLink (Currency -> Integer) {
    name = "amount";
    optional = False;

    constraint = ```
        tgt = get_target(this)
        amount = get_value(tgt)
        amount >= 1000
    ```;
}
```

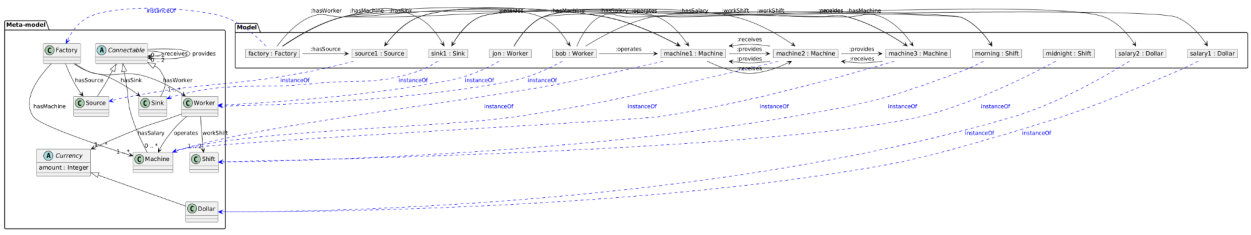
Here we set the target to integer and the value needs to exceed 1000.
```

Jason Liu

The sum of all monthly salaries of all Workers of the same Factory must not exceed 5000. The shareholders are asking for this.

```
totalSalary:GlobalConstraint {
    constraint = ```
        constraintValid = True
        for factory_name, factory_id in get_all_instances("Factory"):
            total_salary = 0
            worker_links = get_outgoing(factory_id, "hasWorker")
            for link_id in worker_links:
                worker_id = get_target(link_id)
                salary_links = get_outgoing(worker_id, "hasSalary")
                for link_id2 in salary_links:
                    salary_id = get_target(link_id2)
                    total_salary += get_value(get_slot(salary_id, "amount"))
            if total_salary > 5000:
                constraintValid = False
        constraintValid
    ```;
}
```

This is simply done by looking at every employee that works in that specific factory and adding up the amount of salary.

## Conforming model



## Non-conforming model



### Errors

Local constraint of "Dollar_amount" in "salary1_amount" not satisfied.
```

Jason Liu

Global constraint "threeShifts" not satisfied.
Global constraint "everyWorkerBelongsToFactory" not satisfied.
Global constraint "oneAfternoonShift" not satisfied.
Global constraint "oneNightShift" not satisfied.
Global constraint "autonomousMachineCheck" not satisfied.

These constraints are invalidated in this example.