Meta-modeling and instance generation with Refinery Report

Jason Liu

5. October 2024

1 Constraints + answers

1.1 A Factory contains at least one Machine.

```
contains Machine[1..*] hasMachine
```

This is done by adding the brackets and specifying a lower bound. This lower bound tells us that their needs to be at least one machine in the factory.

1.2 A Factory has at least one Worker working for it.

```
contains Worker[1..*] hasWorker
```

This can be done in the same manner as for 1.1.

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

```
% There exists a 'morning', 'afternoon' and 'night' Shift
Shift(morning).
Shift(afternoon).
Shift(night).

% There exist no extra shifts
!Shift(Shift::new).
```

By manually creating these three shifts and specifying that their can't be new shifts created, we can achieve this constraint. The last line tells us that we generate new Shift, but adding a „!“ before the instance, tells us that we don't want any new Shifts generated.

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

```
Shift[1..2] worksShift opposite assignedWorkers
```

This can be done in the same manner as 1.1 but this time we also at an upperbound. This forces the worker that their can not be more than two shifts assigned to one single worker.

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

```
contains Source[1] hasSource
contains Sink[1] hasSink
```

This can be done by forcing a number on the member of the class. This forces the constraint that their can only be exactly one of this member in this class.

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

1. Every Machine has one or two inputs, that connect to (a) another Machine's output, or (b) the Factory's Source.
2. Every Machine has one or two outputs, that connect to (a) another Machine's input, or (b) the Factory's Sink.
3. Cycles among Machines, such as the following, are allowed: Machine feedsTo feedsTo Machine Machine feedsTo Cycles between machines are common in industry.
4. The Factory's Source and Sink both must be connected to at least one Machine, and never more than two Machines.
5. A Source cannot be connected directly to a Sink.

1.6.1 Answers

1.
```
% At least one connects input
error pred hasAInput(Machine m) <-> !Receives(m, _).
```

   This line forces that the machine has to receive some sort of input. If no input is received, the constraint will be invalidated.

2.
```
% At least one connects output
error pred hasAOutput(Machine m) <-> !Provides(m, _).
```

   This line forces that the machine outputs to some other connectable.

3. This constraint is not a constraint that is coded. Here is an example instead that it does work:

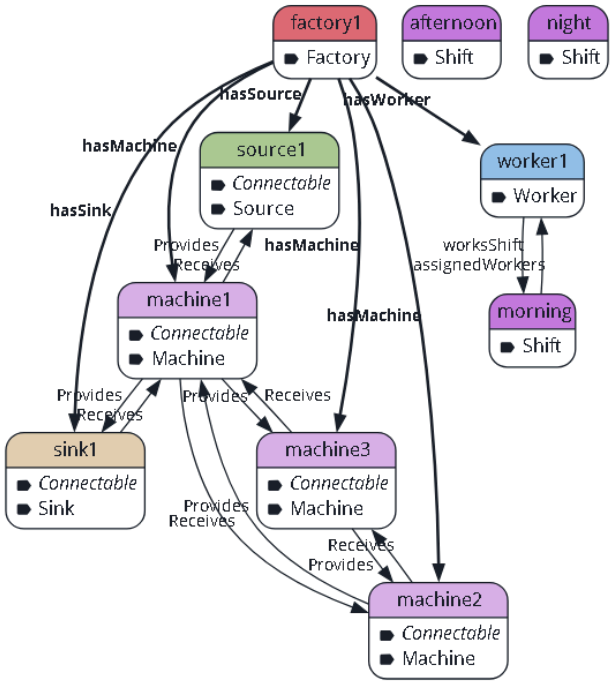

Abb. 1: Example circular production

   Here you can clearly see that machine 1 provides for machine 3, machine 3 provides for machine 2 and machine 2 back to 1.

4.
```
Connectable[0..2] Provides opposite Receives
Connectable[0..2] Receives opposite Provides
```

   This part forces an upperbound and tells us that it has max 2 connections. Sink and source both inherents the attributes of Connectable.

```
error pred hasAOutputSource(Source s) <-> !Provides(s, _).
error pred hasAInputSink(Sink t) <-> !Receives(t, _).
```

These two lines forces that the Sink and Source are connected to at least one other Connectable. The next constraint (in constraint 5) forces the Sink and Source to be disconnected from eachother, so only machines are connected.

5.
```
% Factory Source and Sink have no direct connection
error pred disconnectSourceSink(s, t) <-> Source(s), Sink(t), Provides(s, t).
```

   This forces the Sink and Source to be disconnected from each other. Stating that the source can not provide to the sink.

1.7 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).

```
Worker[0..*] worker opposite operates
```

This part is just specifying a lower bound again.

1.8 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.

```
% Machine has no workers
pred noWorkers(Machine m) <->
    Machine::worker(m, _).

pred shiftExists(Machine m, Shift s) <->
    Machine::worker(m,w),
    Worker::worksShift(w, s).

% Constraint has a worker => all three shift or has no workers
error autoMachine(Machine m, Shift s) <->
    noWorkers(m),
    !shiftExists(m, s).
```

Here we make a predicate of noWorkers that states if there is a worker, it returns true. We also make a predicate telling us if a shift exists work a worker. If there is a worker and no shiftsExist then an error occures because we want to force that every shifts is taken when there is a worker.

1.9 Extra constraint: There exist no self-loops for a machine. So a machine cannot provide resources to himself.

```
% No self loops for a machine
pred selfLoopMachine(Machine m) <->
    Connectable::Provides(m, m).

error noSelfLoop(Machine m) <->
    selfLoopMachine(m).
```

We do not want any self loops in our factory, so we state that a machine can not provide for himself. This can be done by just saying that machine that it connects to, is not himself, if it is, then it is an error.

1.10 Extra constraint: Source can not have an input and Sink cannot have an output to a machine.

```
% Source has no input
error hasAInputSource(Source s) <->
    Receives(s, _).

% Sink has no output
```

```
error hasAOutputSink(Sink t) <->
    Provides(t, _).
```

Because we only have one machine, we only want the source and sink to work within the machine. The source can not have an input within the machine and the sink can not output something within the machine.

So we set a constraint that it can not provide or receive things depending on which.

2 Example images

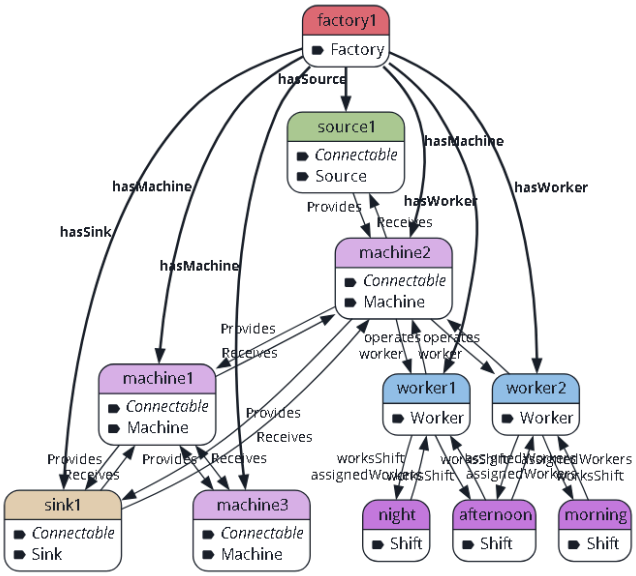

Abb. 2: Autonomous machine with no workers and not autonomous machine with a worker for every shift.

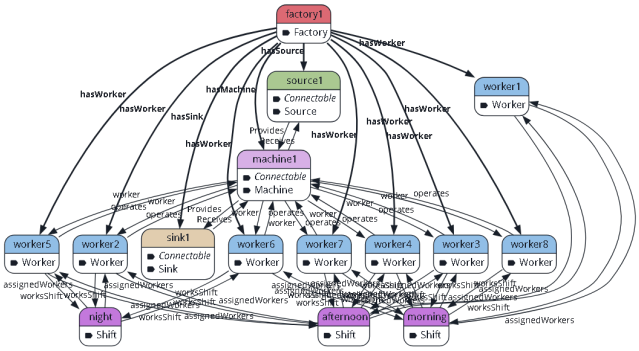

Abb. 3: Big example.

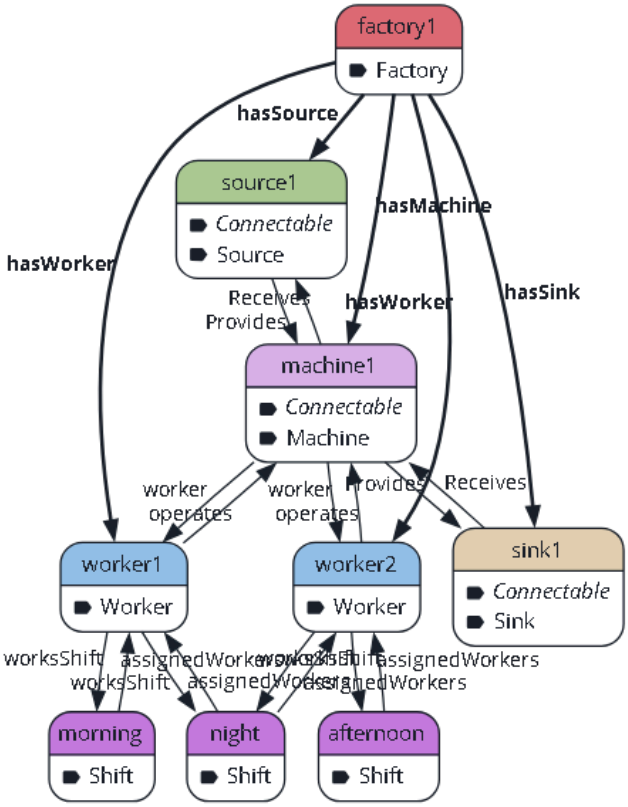

Abb. 4: One worker for every shift.

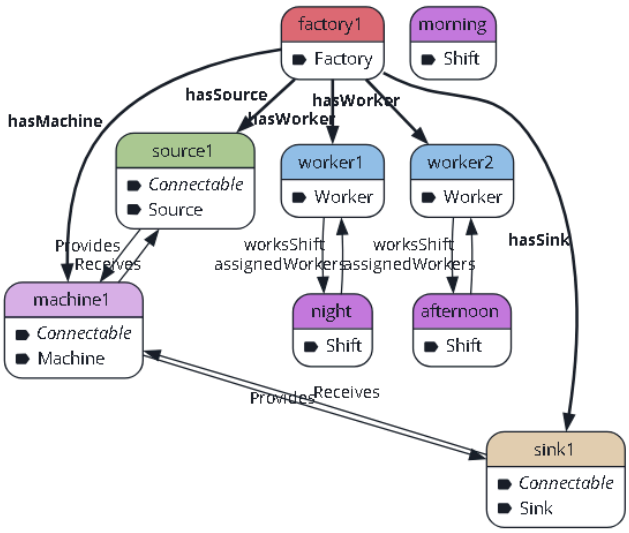

Abb. 5: No workers for a machine but there are workers in the factory.