

Fifth Workshop on Graph-Based Tools

Live Challenge Problem

Zurich, 1-2 July

Pieter Van Gorp
Tihamér Levendovszky
Arend Rensink

Table of Contents

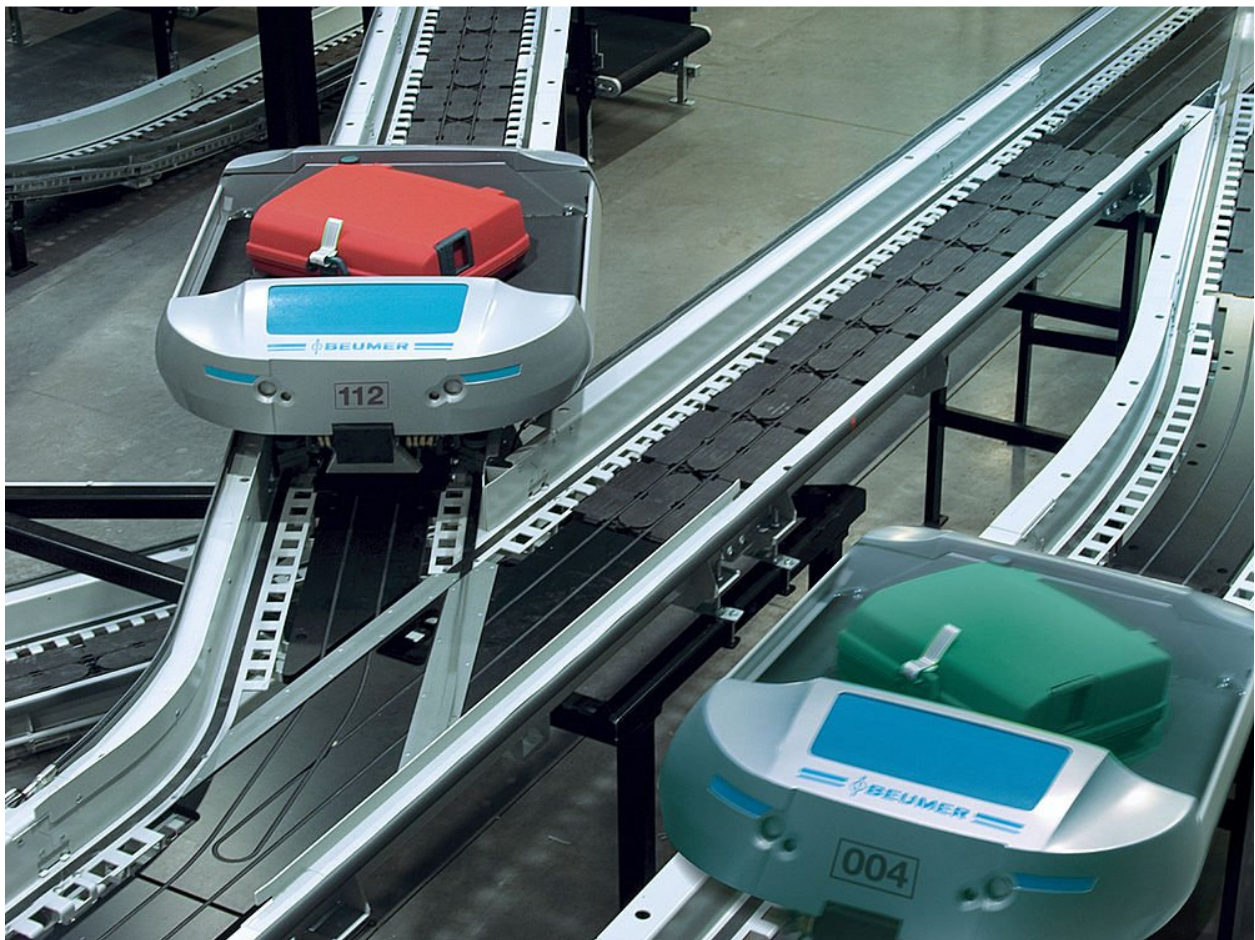
1	Domains	3
1.1	Source Domain.....	3
1.2	Target Domain	4
2	Transformation to Statechart	5
3	Simulation of Statechart Models	7
4	Challenge Problems	8

1 Domains

The domains of the transformations are (i) a conveyor belt system and (ii) a simplified statechart diagram.

1.1 Source Domain

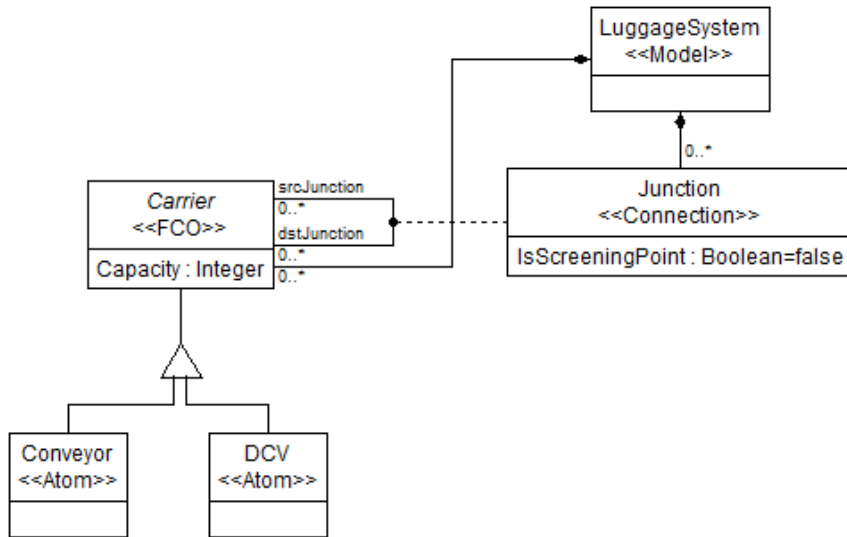
The **destination-coded vehicle** (DCV) is a metal cart with wheels on the bottom and a plastic tub on top. Its only electronic device is a passive radio-frequency circuit that broadcasts a unique number identifying that particular car. The job of the DCV is to move your bag quickly to an off-ramp at the gate. DCVs are used at airports because the distance from the main terminal to the passenger terminals is quite long, and passengers make the commute fairly quickly by train. The DCV can travel up to five times faster than a conveyor at 20 mph.



A conveyor belt is a device that conveys (carries) large quantities of material from place to place. It consists of an endless belt that is looped over two pulleys. One of the pulleys is called the drive pulley, and supplies the power that keeps the belt moving. Most conveyor belts are powered by an electric motor. Its speed is 4 mph.



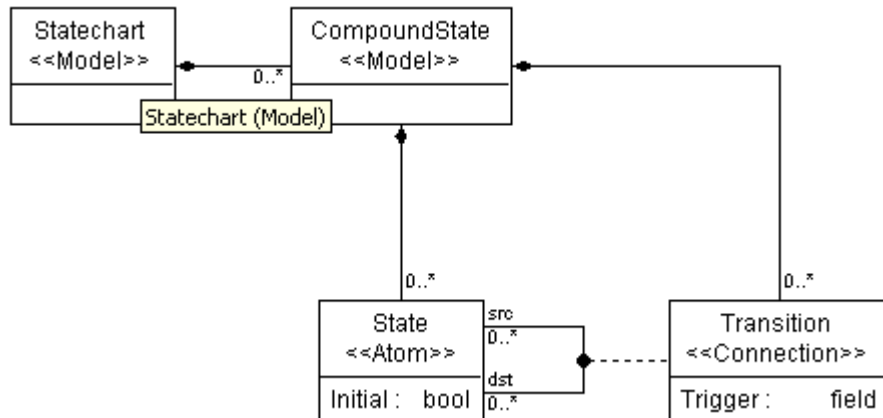
The metamodel we use for the domain is as follows (the terms of the Generic Modeling Environment):



The abstract ancestor class **Carrier** holds an integer attribute, which is the maximum capacity of the device. There is a many-to-many relationship called **Junction** between two carriers. A junction may be a screening point, which means a delay of 20 seconds. The information whether a junction is a screening point is stored as an edge attribute (denoted by an association class in the figure) **IsScreeningPoint**. A **Carrier** can be either a **Conveyor** or a **DCV**, as it is described in the figure.

1.2 Target Domain

The target domain is a simplified statechart diagram depicted in the following figure.



This is a two level statechart. The **CompoundState** is on the higher level. It may contain **States** (lower level), which can be connected to each other (many-to-many relationship) via an association called **Transition** having the string attribute **Trigger**. The trigger is an event that fires the transaction.

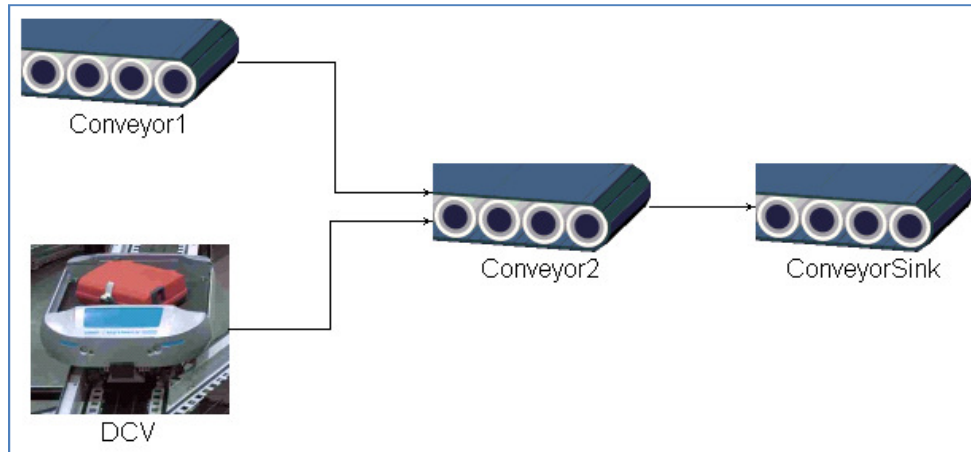
The compound states are executed in a parallel way; there is no parallel execution for the lower level.

2 Transformation to Statechart

The description of the transformation has been broken down into the following steps.

1. Create a **CompoundState** for each **Carrier** with the same name.
2. Within the compound state, create as many **States** as the capacity of the **Carrier** is. The name convention is *<carrier name>+_accept+_<number of bag the carrier can still accept>*, where + denotes concatenation. Each state corresponds to the number of bags a **Carrier** can still accept.
3. Create a transition between the **States** correspond to adjacent bag numbers: (i) one that points to the higher number, where the event is *<carrier name>+_bag_out*, one that points to the lower number, where the event is *<carrier name>+_bag_in*.
4. Add start states.

Below is an example Luggage input model and the corresponding Statechart output model.

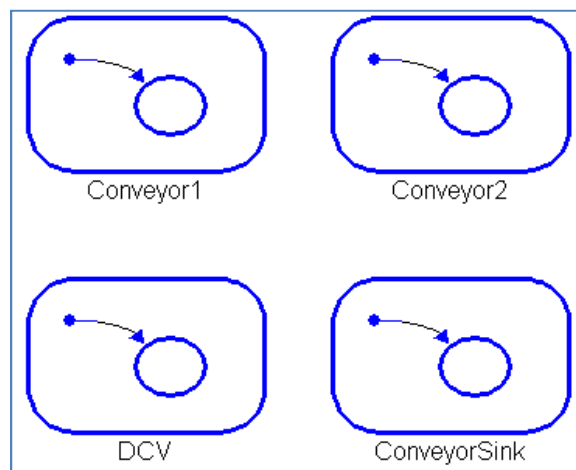


Example input model

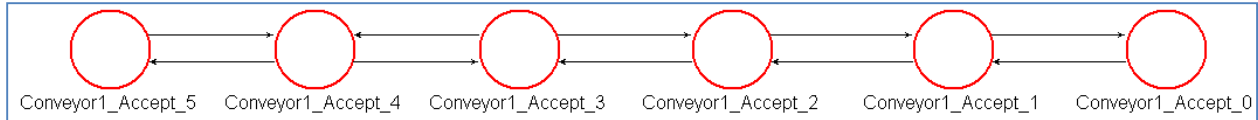
The capacities for the input model are:

- Conveyor1: Capacity = 5
- DCV: Capacity = 4
- Conveyor2: Capacity = 3
- ConveyorSink: Capacity = -1 (this is a sink)

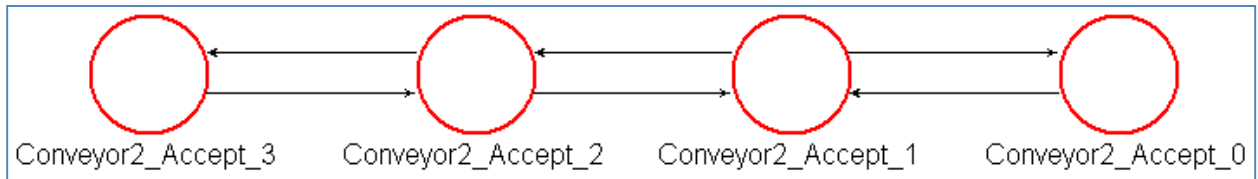
The corresponding output model is as follows.



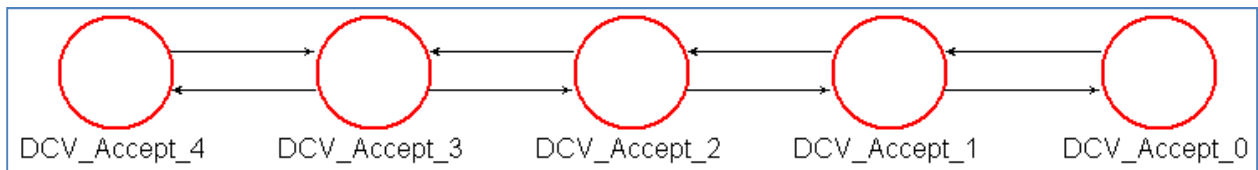
High level view of the output. 1 Compound state for each conveyor/DCV



Inside the compound state Conveyor1. 6 states corresponding to the number of bags the conveyor can accept at a given time. Conveyor1_Accept_5 is the initial state.



Inside the compound state Conveyor2. 4 states corresponding to the number of bags the conveyor can accept at a given time. Conveyor2_Accept_3 is the initial state.



Inside the compound state DCV. 5 states corresponding to the number of bags the DCV can accept at a given time. DCV_Accept_4 is the initial state.



Inside the compound state ConveyorSink. 1 states with an unlimited capacity because this is a sink. AcceptAll is the initial state.

3 Simulation of Statechart Models

The simulation of the generated statechart is performed by a simulation engine, which is responsible for the following:

- Providing the *xxx_bag_in* and *xxx_bag_out* messages.
- It must consider the topology of the *LuggageSystem* model.
- A bag can enter the system in any of the **Carrier** that has no incoming **Junctions**.

- A bag can leave the system via a **Carrier** having no outgoing junctions.
- In case of fork, the simulation engine decides the path of a bag randomly with uniform distribution.
- (Extension) It must consider the traveling time given the speed in the description and the distance to travel given as an attribute in the **Carrier**.
- (Extension) It must consider the delay caused by the screening points.

4 Challenge Problems

1. Modeling the domains

- Create a Domain-Specific Environment for the source and the target domain. Extend **Carriers** of the source domain with the distance to travel (use the metric system you are the most comfortable with). Extend the statechart with a start state (it can be either an attribute or a separate element, choose what fits the best for your tool).
- Create a concrete syntax for the domain. As a direction, statechart diagrams should follow the UML concrete syntax, try to be as close as possible to the view of the system above as possible.
- Create an example model which has at least 3 conveyor belts, 2 DCVs, 4 junctions, at least 1 with screening point, at least one one-to-many outgoing junction (“fork”), and many-to-one incoming junction (“join”).

2. Creating the transformation

- Create a transformation from a luggage system to the statechart model.
- Transform the example model.
- Assuming that the transformation has been executed, let your tool prove that
 - There is no circle in the input model
 - For each **Carrier** a **CompoundState** has been created.

3. Simulation

- Create a simulation engine that takes the luggage system model as an input, and is able to simulate the corresponding statechart diagrams possibly with visual effects on the models.
- Simulate the sample statechart model according to the given conditions.
- (Extension) Create a simulation directly for the luggage domain too.
- (Extension) Use traceability between the two domains to connect the two simulations: transitions in one domain should trigger transitions in the other domain automatically. This enables luggage domain experts to use the statechart simulation more easily.