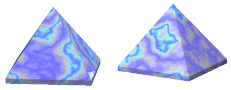
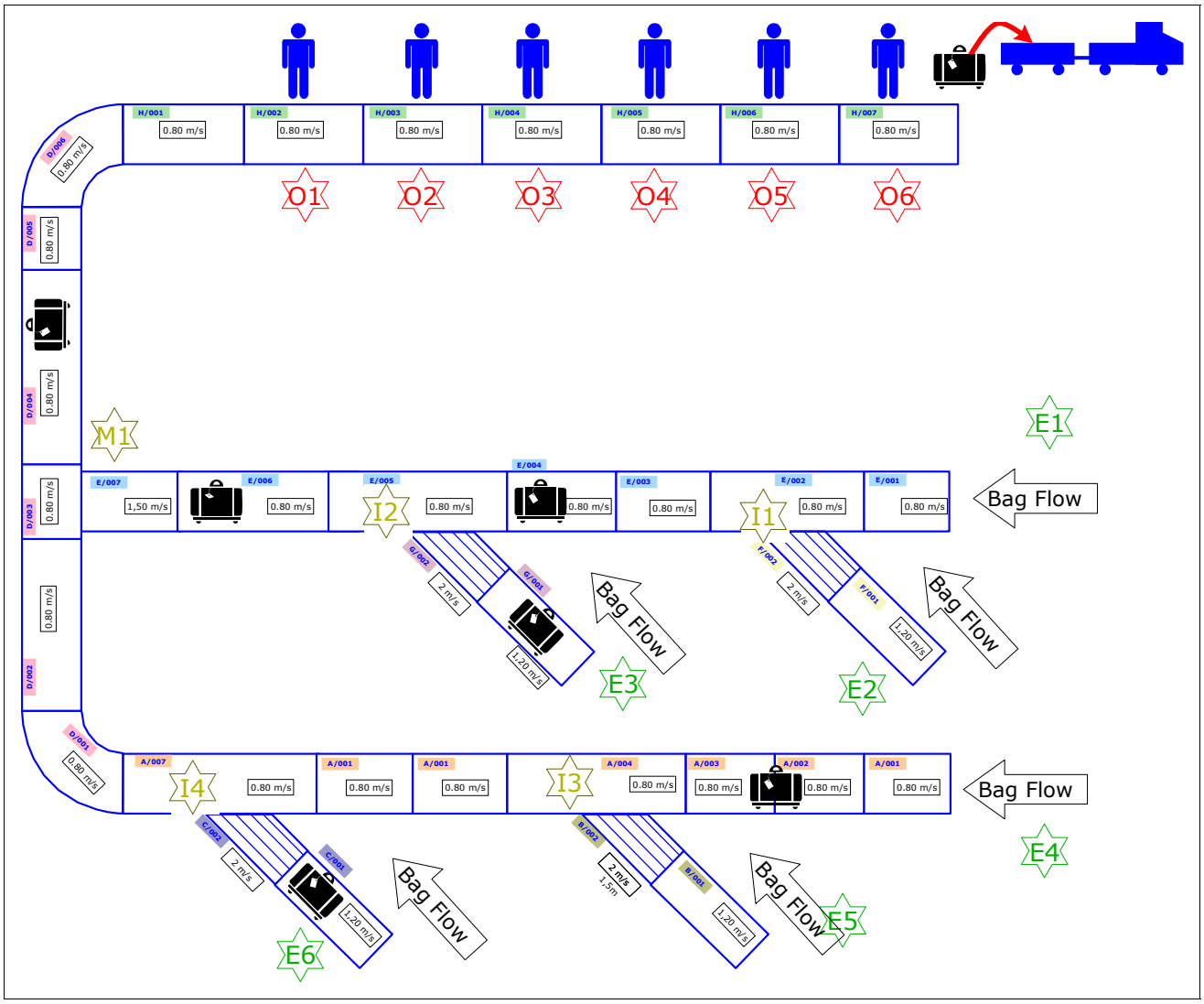


A baggage handling simulation software OO agile development with an OO Programming language and UML
Case Study

Table of Contents

1. Problem description

Considering the hypotheses given in #2. Hypotheses, and the baggage handling configuration described on the layout:



with:

- **E**x = entrance number x
- Ox = output (exit) number x
- Ix = Induction number x
- **M**x = Merge number x

The trainees (by groups of 2 or 3) have to realize a Baggage Handling simulation system.

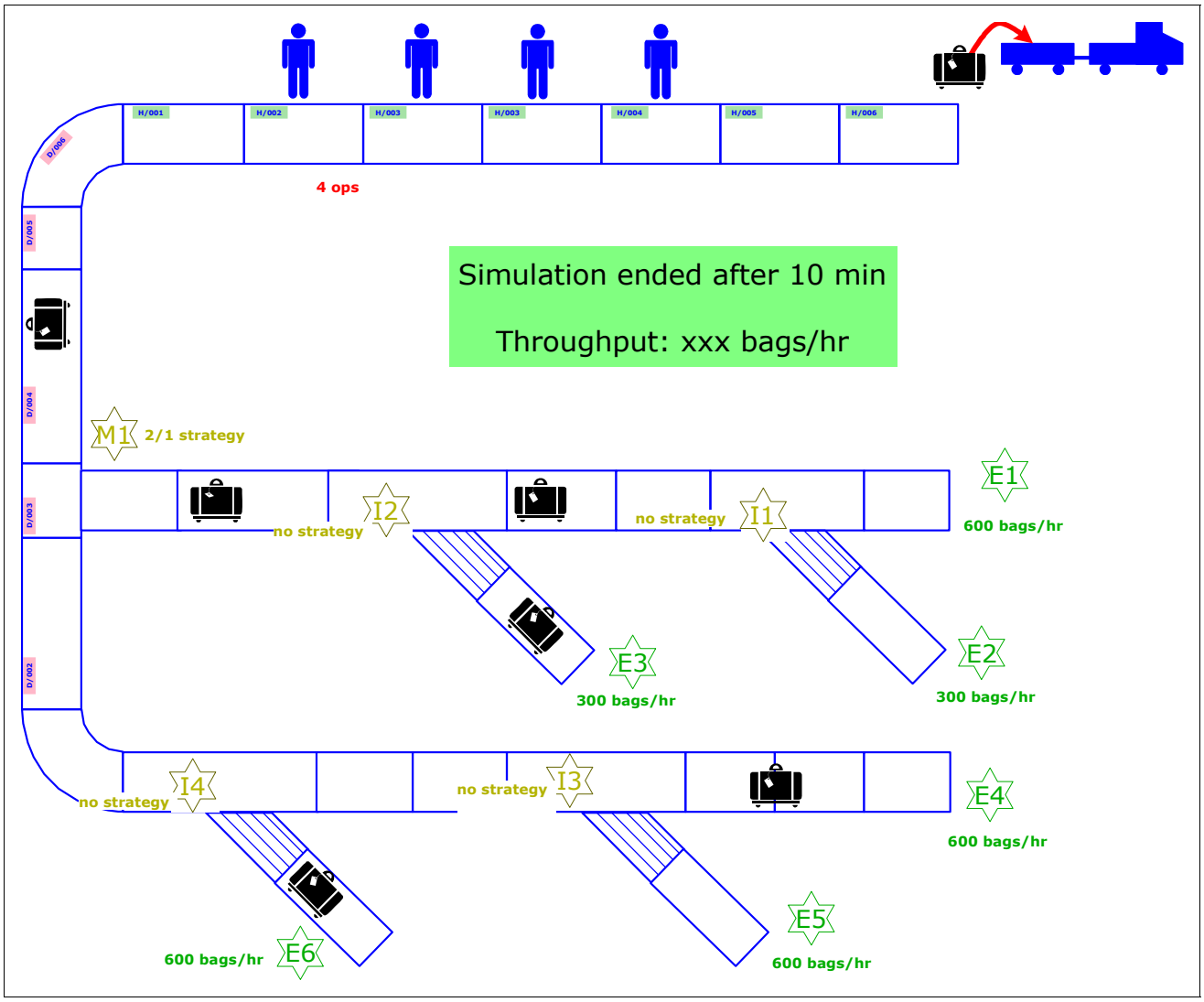
Such a system should be used to verify that the configuration allows for the required throughput of baggage.

The points indicated by stars represent points where parameters could be modified. The chosen parameters are indicated besides their position (see following result screens).

When done, the user may launch the simulation. Then, two kinds of results are possible:

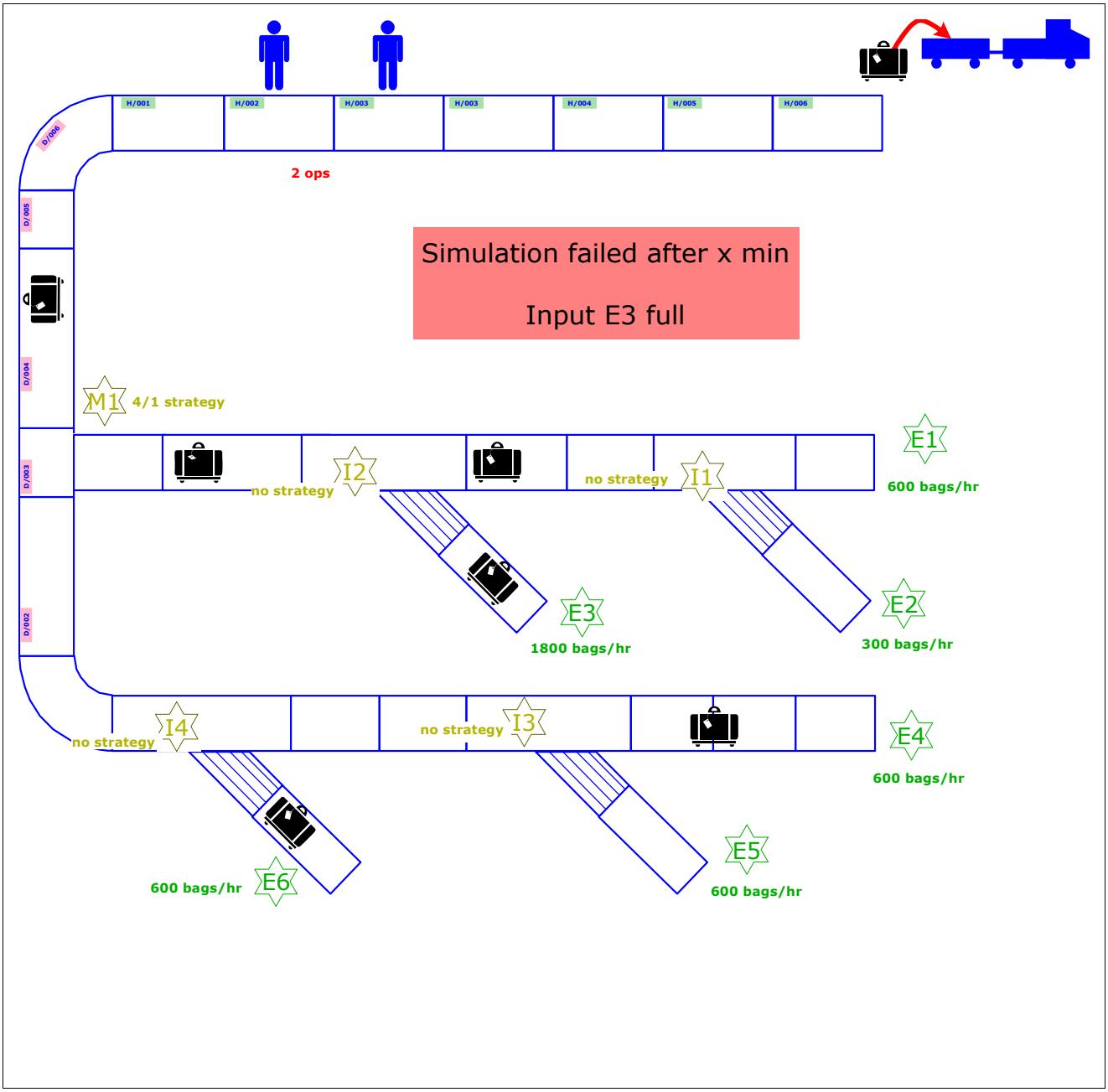
1.1. Simulation finished normally

In this case, the reached throughput should be given on a result screen that should appear like:



1.2. Simulation failed

Before the end of the simulation, one or more input(s) become blocked. In this case, the system has to give an error message specifying the blocked input, and asks the user for new parameters.



The results of the simulation, together with the date and time and the different parameters applied, could be saved in a text file on user's request.

1.3. Hypothesis

1.3.1. Depending on the OO programming language

- Java: IDE Eclipse, GUI Swing
- Python: IDE IDLE, GUI TKinter

1.3.2. Technological hypothesis

- The conveyors, with their length and speed, are stored in a relational DB (mySQL)
- Optionally, an administrator may modify the conveyor speed through a special menu of the application.
- The software has to be realized in the imposed programming language
- Unit tests have to be applied
- Simulation time has to be parameterizable (from 1 min to 60 min).
- Simulation data and image have to be updated every 100 ms.

1.3.3. Bag Handling Simulation Hypotheses

- Length of the bags = 1 m. The modifications of this length on intersections (inductions, merge) is neglected.
- The speed difference between two conveyors is neglected for the bag movement, except on Intersections (see hereunder). The considered speed is the one of the current Conveyor.
- Conveyor data (stored in the relational DB) are given in #1.4. Conveyors list.
- Conveyors are normally running at their maximum speed, unless they have to be stopped because of the bag flow (no standby,....). This speed transition and the inverse one are considered as instantaneous.
- A conveyor line is a straightforward line of conveyors, with one input and one output, either from/to another line, or not (real inputs, operator outputs). Other inputs/outputs correspond to intersections with other lines.
- There can be from 1 to 6 operators, one per output conveyor. They are placed at the middle of their conveyor. The first operator is always the upper left, and the following ones are added towards the right (no holes between operators).
- One operator may take 10 bags/min maximum. A taking off of a baggage item by an operator is considered as instantaneous.
- When specifying a bag flow, we mean a constant flow (e.g. 1800 bags/hr => one bag of 1 m every 2 sec), we do not consider statistical repartitions. Default bag flow = 400 bags/hr.
- Inputs are considered full if they are waiting for a time equivalent to 4 bags of their flow (this parameter could also be modified)
- The induction conveyor never stops, it is then the preceding one that should be stopped if necessary.
- Merge end inductions have:
  - one “Main stream”, corresponding to the receiving (“output”) conveyor
  - one “Input stream”, corresponding to the “input” conveyor
  - during the input of a bag from the input stream, the bag speed is the one of the Input conveyor until it is totally transferred on the main stream.

- A bag may arrive on the main stream (“output”) conveyor of a merge or of an induction only if it is empty
- Merge and inductions may have different strategies:
  - by default: the first bag arriving from one of the streams has priority. If two bags arrive exactly at the same time from the two streams, the main stream has priority.
  - the user may choose repartitions of baggage on the main stream relative to the input stream (e.g. **2/1**: there should be 2 bags passed on the main stream before allowing a bag from the input stream to pass).
- Induction angle = 50 °; a bag begins to enter on the reception CV at 100cm
- The length given for curve conveyors is the middle one.

1.4. Conveyors list

Nbr	Line	Number	Speed [m/s]	Length [m]	Entrance nbr	Output nbr	Output of induction nbr	Input of Induction nbr	Output of merge nbr	Input of Merge Nbr
1	A	001	0,8	2	E4					
2	A	002	0,8	2,08						
3	A	003	0,8	2,09			I3			
4	A	004	0,8	4,16						
5	A	005	0,8	2,2						
6	A	006	0,8	2,25						
7	A	007	0,8	4,62			I4			
8	B	001	1,2	2,72	E5					
9	B	002	2,0	2,3				I3		
10	C	001	1,2	2,72	E6					
11	C	002	2,0	2,3				I4		
12	D	001	0,8	2,11						
13	D	002	0,8	4,03						M1
14	D	003	0,8	1,72						
15	D	004	0,8	4,54						
16	D	005	0,8	1,48						
17	D	006	0,8	2,11						
18	E	001	0,8	1,97	E1					
19	E	002	0,8	3,58			I1			
20	E	003	0,8	2,2						
21	E	004	0,8	2,57						
22	E	005	0,8	4,14			I2			
23	E	006	0,8	3,5						
24	E	007	1,5	2,25					M1	
25	F	001	1,2	2,72	E2					
26	F	002	2,0	2,3				I1		
27	G	001	1,2	2,72	E3					
28	G	002	2,0	2,3				I2		
29	H	001	0,8	2,81						
30	H	002	0,8	2,81		O1				
31	H	003	0,8	2,81		O2				
32	H	004	0,8	2,81		O3				
33	H	005	0,8	2,81		O4				
34	H	006	0,8	2,81		O5				
35	H	007	0,8	2,81		O§				

The conveyors are supposed to be ordered in this table => first one = input of the corresponding line, last one = output.

2. Inception – iteration 1

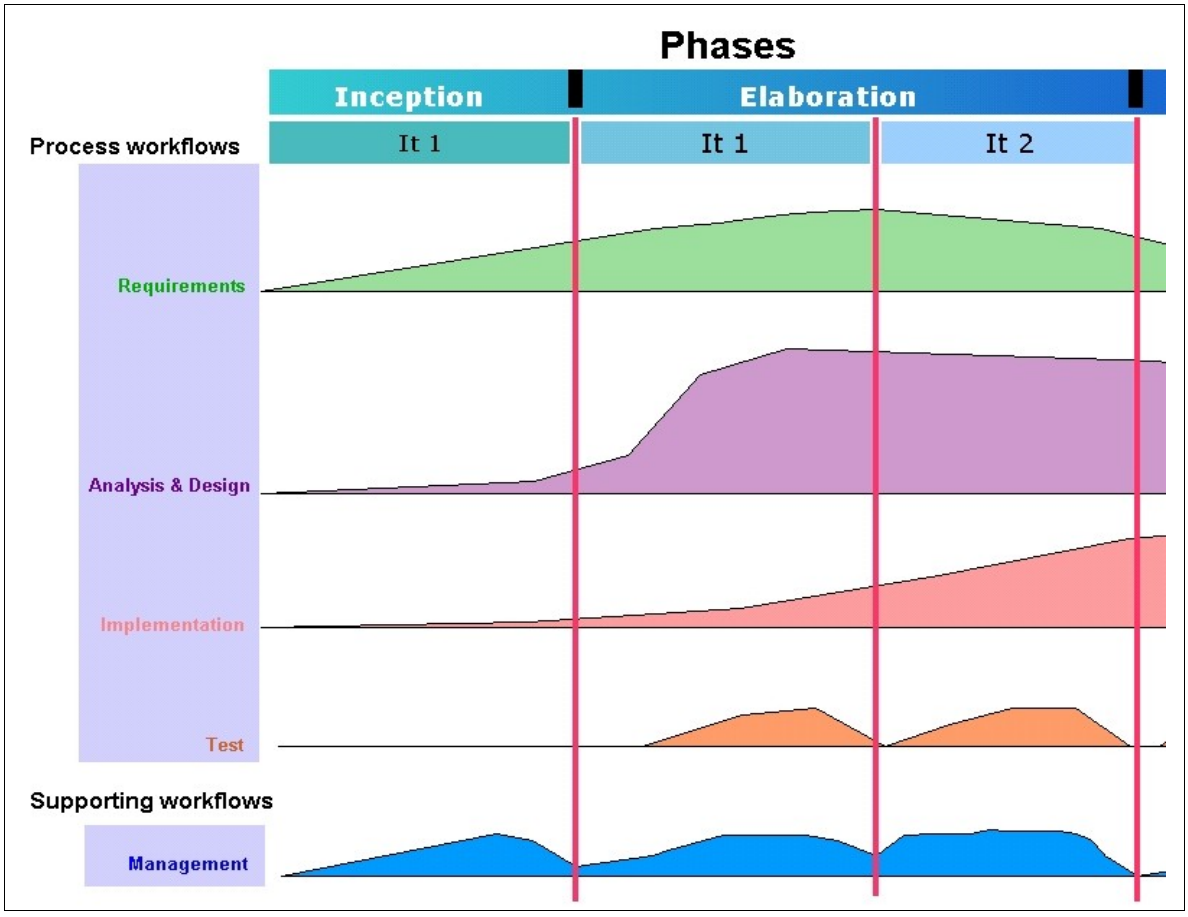
2.1. Vision – iterations

Small problem => **inception** 1 iteration, to determine the work

Main risks => Technological (language and related technologies unknown) and over user requirements (mainly about GUI) => 2 **elaboration** iterations proposed

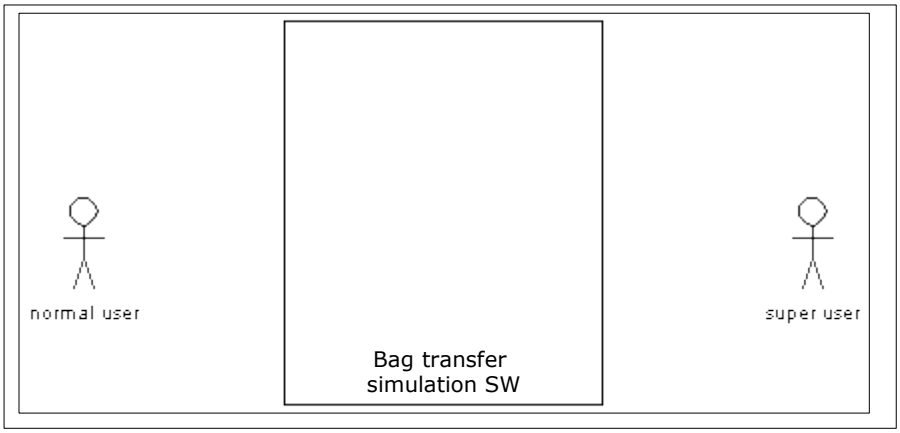
Afterwards, not much work left => probably one iteration in **construction**

Iterations of typically one week.

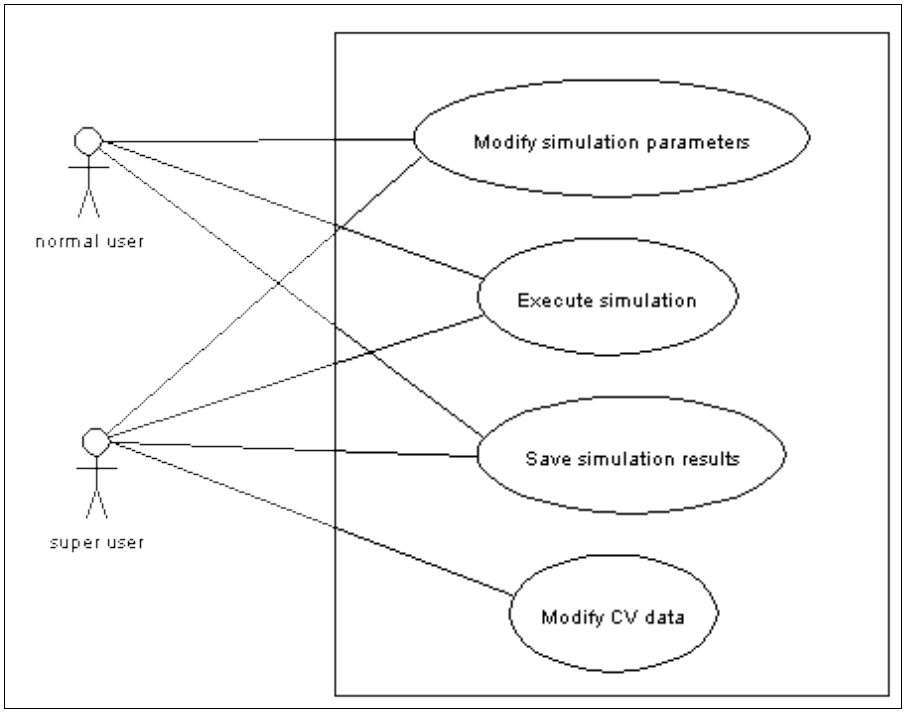


2.2. Use-Case analysis

2.2.1. System and actors



2.2.2. Use-Case model



2.2.3. High-level use-cases

2.2.3.1. Modify simulation parameters (UC1)

Before (and not during) a simulation, the user may modify some parameters of the circuit (at least from the menu, and possibly on the image itself, to be confirmed with the second elaboration iteration):

- simulation time (in min, default = 10 min, from 1 to 60)
- operators number (from 1 to 6, automatically set at points **O1→ O6**)
- bags flow at inputs of the circuit (bags/hour, default = 400 bags/hr, at points **E1→ E6**)
- number of bags to block an input (at points **E1→ E6**)
- merge and inductions strategies (equivalent priority, or x/y strategy, at points **I1→ I4** and **M1**)

2.2.3.2. Execute simulation (UC2)

From the main screen of the application, after possibly setting the parameters (see #2.2.3.1. Modify simulation parameters (UC1)), the user may launch the simulation. The program calculates and displays periodically (each 100 ms) the position of the different bags, until either:

- the simulation finishes normally. It then displays the resulting flow, and asks the user if he/she wants to save the results in a file (see #2.2.3.3. Save simulation results (UC3)).
- the simulation fails. It then displays an error message specifying which input became blocked, after which period).

Afterwards the program returns to the main screen.

2.2.3.3. Save simulation results (UC3)

When a simulation is normally finished, the user may ask for saving the results in a file. If yes, it the program asks for the file directory and name.

2.2.3.4. Modify CV data (UC4)

A “super” user (administrator) may modify the conveyors speed in the DB, through a particular menu.(not the length nor the number, that would imply we should change the physical layout).

2.2.4. Detailed use-cases

Detailing the first and the second use-case (#3.1.3.1. Modify simulation parameters and #3.1.3.2. Execute simulation), ***without detailing the GUI part***, we obtain:

2.2.4.1. **Use Case :** *Modify simulation parameters (UC1)*

Author : I. Leclercq/R. Florquin

Date updated : 11/03/03

Summary : from #2.2.3.1. Modify simulation parameters (UC1):

Before (and not during) a simulation, the user may modify some parameters of the circuit.

Use Context: this use-case may be called before launching a simulation (UC2).

Actors : the normal user of the simulation software.

Pre-conditions : the software has to be initialized

Description :

The system asks the user if he/she wants to modify the simulation parameters. If no, this use-case finishes.

If yes: the system proposes the different parameters that can be modified by the user:

- simulation time (in min, default = 10 min, from 1 to 60)
- *operators* number (from 1 to 6, automatically set at points **O1→ O6**, default = 2)
- bags flow at *inputs* of the *circuit* (bags/hour, at points **E1→ E6**, default = 400)
- number of bags to block an input (at points **E1→ E6**, default = 4)
- *merge* and *inductions strategies* (no strategy, or x/y strategy, at points **I1→ I4** and **M1**, default = no strategy).

The user may select one parameter at a time, and enter a new value for this parameter. The system verifies the range of the value, and accepts the parameter if correct.

The use-case finishes when the user indicates that he/she has finished to enter the parameters.

Exceptions :

If an entered parameter is incorrect, the program displays an error message, and then returns waiting for a new user input.

Post-conditions :

The values are then stored in the program for subsequent simulation, and displayed on the layout at the corresponding positions.

2.2.4.2. *Use Case : Execute simulation (UC2)*

Author : I. Leclercq/R. Florquin

Date updated : 11/03/03

Summary : from #2.2.3.2. Execute simulation (UC2), we have:

> From the main screen of the application, after possibly setting the parameters, the user may launch the simulation. The program calculates and displays periodically (each 100 ms) the position of the different bags, until either the simulation finishes normally or fails.

Use Context: this use-case is the reason of being of this application.

Actors : the normal user of the simulation software.

Pre-conditions : the software has to be initialized

Description :

this Use Case starts when the normal user asks for launching the simulation. First, the use-case UC1 is executed (see #2.2.4.1. Use Case : Modify simulation parameters (UC1)).

Then, the simulation begins:

- *Bags* enter at the inputs, following the specified *flows*
- are induced at *inductions* or m*erge :*
  - from *upstream* conveyor lines (“Input stream”), from “*input*” conveyors. Note that the real induction conveyor cannot be stopped, it is then the preceding one in the induction line which is stopped if necessary.
  - on *downstream conveyor lines* (“main stream”), on receiving (“*output*”) *conveyors*
  - following specified *strategies*
- and exit at the *output*, following the specified number of *operators* (operators are placed beginning from the upper-left).

The bags position is calculated periodically (every 100 ms), and is displayed, until the simulation time is elapsed, or until one of the inputs is blocked (exception).

Conveyors are normally running, unless the strategy at intersection results in stopping one of the streams.

The bags (length = 1m) enter on running conveyors at position “zero”, and their subsequent position results from the speed of the conveyor, and from its state (if stopped, the bag position doesn't change). When arriving at the end of the conveyor, a bag is being transmitted to the following conveyor, unless it is stopped. In this case, the current conveyor also stops.

The transfer of bags on intersections (inductions/merge) is managed through the specified strategies.

When finished, the resulting flow is displayed. The user is asked if he/she wants to save the results in a file (call of UC 3, not detailed yet), and then the use-case finishes.

Exceptions :

If the simulation fails, the system displays an error message, and when acknowledged by the user, the use-case finishes.

Post-conditions : None

2.3. Other requirements

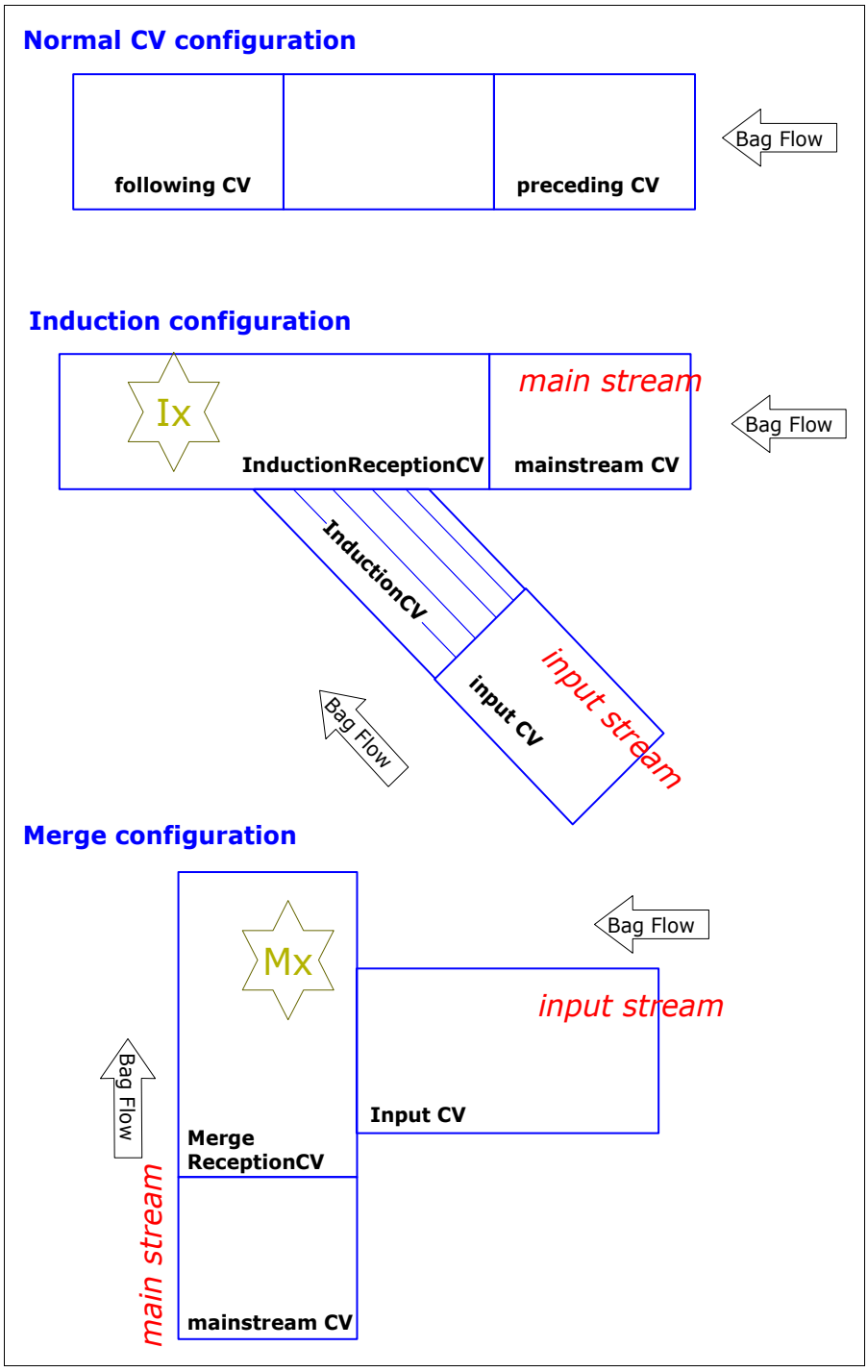
OO development and programming language (following chosen course) imposed.

Easy to use (see GUI prototypes, elaboration iteration 2)

No particular performance requirements, but should be reasonable.

2.4. Domain model

With the configurations:



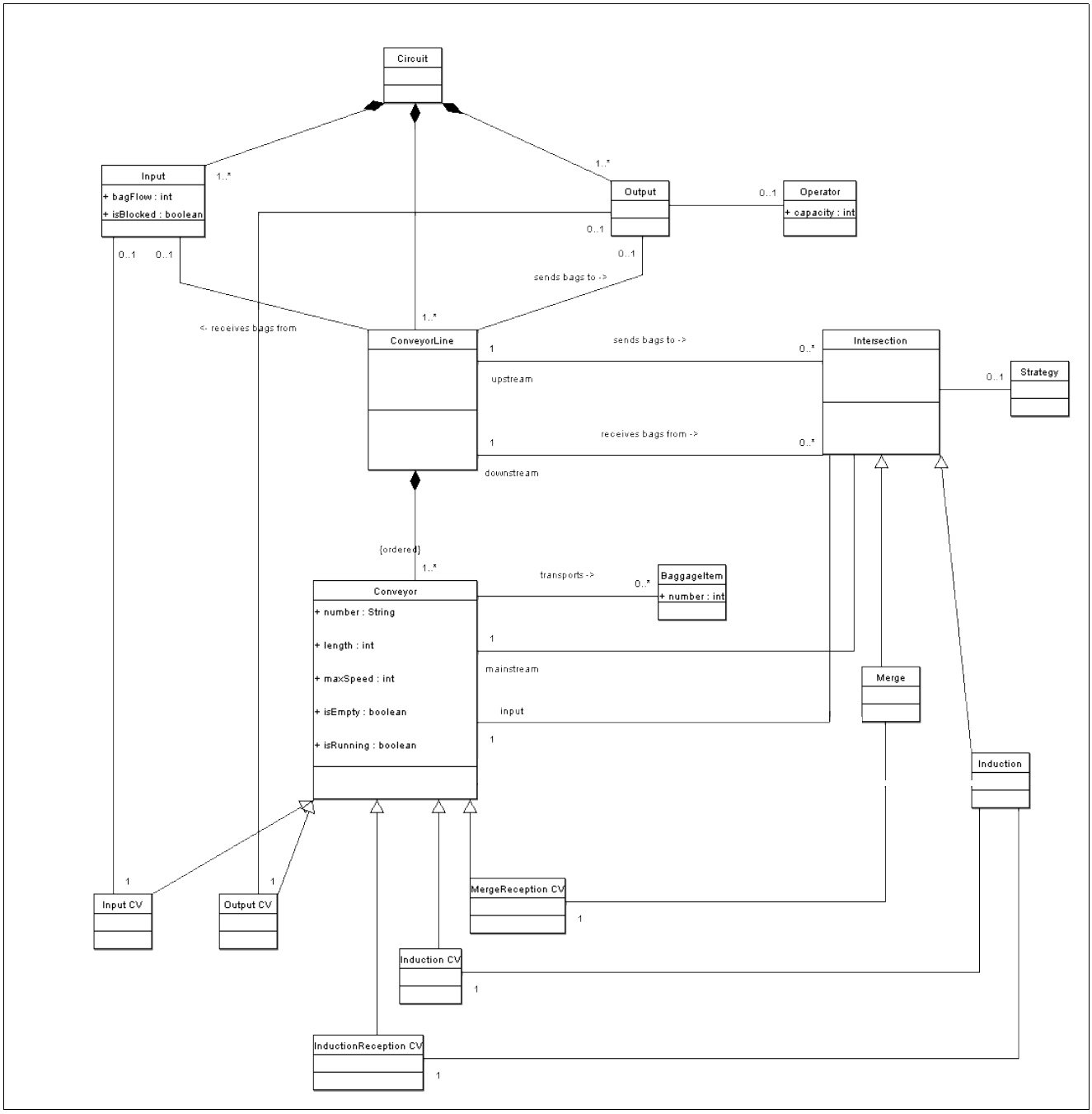
2.4.1. Candidate concepts

Conceptual class categories	
Physical or tangible objects	Circuit
	Conveyor
	Baggage item
Specifications, descriptions of things	Merge and Induction Strategies
	Bag Position
	Bag Flow
Places	Input Point
	Output Point
	Merge Point
	Induction Point
Transactions	Simulation
Roles	Baggage Operator
	System User
	Upstream Conveyor Line
	Downstream Conveyor Line
	“Output” Conveyor
	“Input” Conveyor
Containers	Conveyor Line
	Merge
	Induction

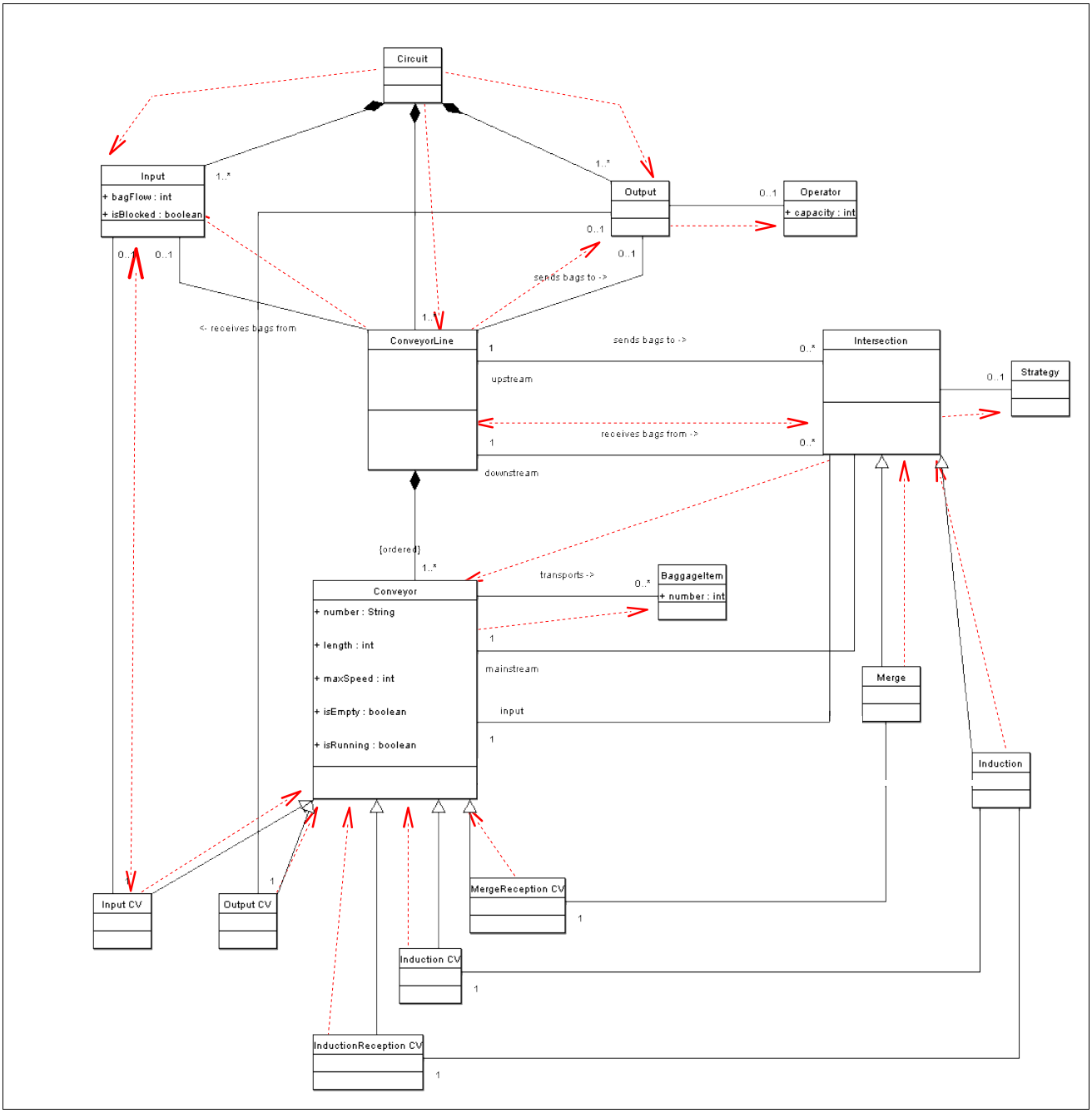
2.5. Candidate associations

Conceptual class categories	
A is a physical part of B	Conveyor Line, Input Points, Output Points – Circuit
	Conveyor – Conveyor Line
	“Input” and “Output” Conveyors – Merge
	“Input”, “Output”, Induction Conveyors – Induction
A is a logical part of B	Simulation Steps-Simulation
	Bag Position
	Bag Flow
A uses B	Merge – Merge Strategy
	Induction – Induction Strategy
A manages B	Circuit – Simulation
	Operator – Flow at Output Point
	Input – Flow at Input Point
	Merge – manages Flow at Merge Point
	Ditto for Induction
	Conveyor – Baggage Item Position

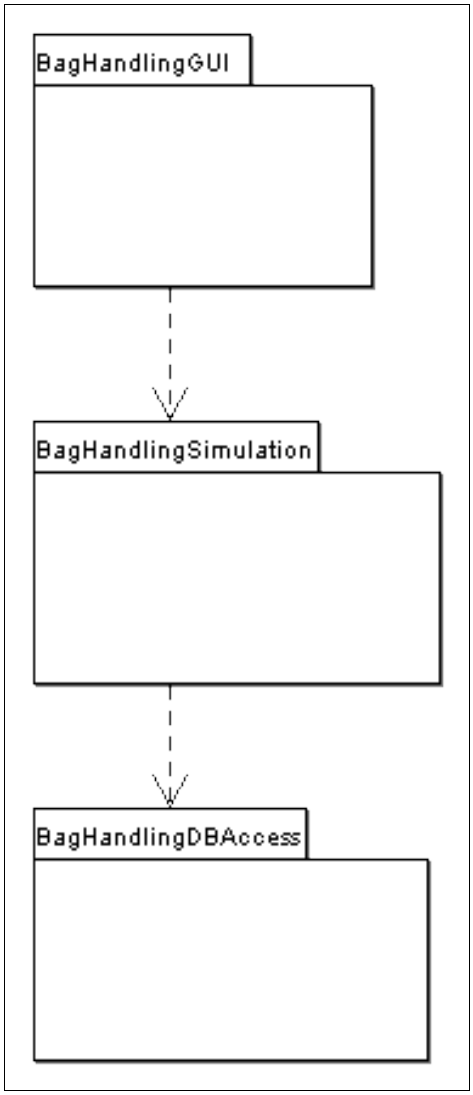
2.5.1. First domain model



2.5.2. Dependencies



2.5.3. Package name



2.6. UC selection

2.6.1. UC risks

	Architectural	Business critical	Risk estimation
Modify simulation parameters (UC1)	+	+++	+
Execute simulation (UC2)	++++ (DB read, GUI)	++++	+++
Save simulation results (UC3)	+++ (files)	+	+
Modify CV data (UC4)	++ (DB write)	+	+

Confirms the rough planning given in #2.1. Vision – iterations:

- elaboration it 1: UC1 + UC2 without GUI

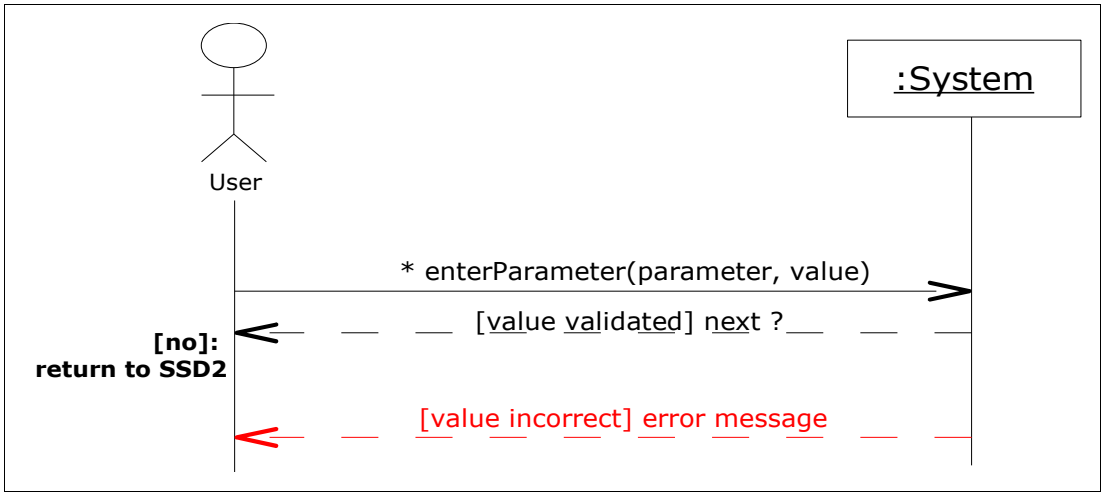
- elaboration it 2: UC1 + UC2: GUI particular
- construction: UC3 + UC4

3. Elaboration iteration 1

3.1. System sequence diagram

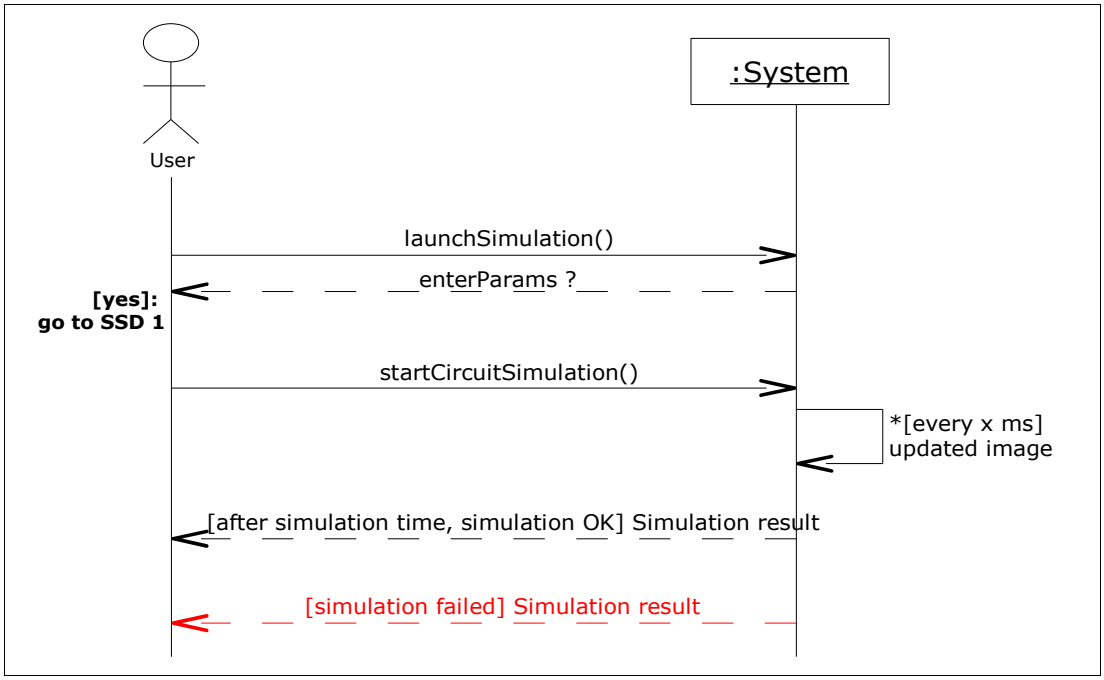
3.1.1. SSD1: Modify simulation parameters

From UC1:



3.1.2. SSD2: Execute Simulation

From UC2:



3.2. Operation contracts

3.2.1. OC1: launchSimulation()

From UC2, SSD2 and Domain model:

Preconditions:

The system is running.

Postconditions:

Static topology:

If not already done (first time since the system has began to run), create the static circuit instances (***instances creation***):

- **Circuit** itself
- the seven **ConveyorLine**s
- the **Merge** and the 4 **Induction**s (**Intersection**s)
- the corresponding **Conveyor**s:
  - **InputConveyor**s if there is an associated **Input**
  - **OutputConveyor**s if there is an associated **Output**
  - for the Merge:
    - the normal “mainstream” **Conveyor** (preceding in the mainstream **ConveyorLine** the **MergeReceptionConveyor**)
    - the **MergeReceptionConveyor**
    - the normal “input” **Conveyor**
  - for the Inductions:
    - the normal “mainstream” **Conveyor** (preceding in the mainstream **ConveyorLine** the **InductionReceptionConveyor**)
    - the **InductionReceptionConveyor**
    - the **InductionConveyor** (last one of Induction ConveyorLine)
    - the normal “input” **Conveyor** (preceding the InductionConveyor)
  - and all the other **Conveyor**s
- the six **Input**s
- the six **Output**s
- all the associations between these objects (***association creation***) have been created.

Default parameters:

Then, all the default parameters have been created:

- Simulation time: the default simulation time (10 min) has been set in the **SimulationController** (***attribute modification***).
- Number of operators (2): the corresponding objects have been created (***instance creation***), and associated with the **Output**s, beginning by the upper-left one in the **ConveyorLine** H (***association creation***).
- Default bag flow on the **Input**s (400 bags/hr): modification of the attribute (***attribute***

modification)

- Default blocking number of baggage items waiting on **Input**s: 4 (***attribute modification***).

3.2.2. OC2: enterParameter()

From UC1, SSD1 and Domain model:

Preconditions:

The circuit topology and default parameters have been created (see OC1).

Postconditions:

Depending on the input parameter:

- Simulation time: the simulation time has been modified in the **SimulationController** (***attribute modification***).
- Number of operators: if more than 2 (default number): the corresponding objects have been created (***instance creation***), and associated with the **Output**s, beginning by the upper-left one in the **ConveyorLine** H (***association creation***). If only one operator: the second one has been deleted (***instance deletion***).
- Bag flow on one **Input**: modification of the attribute (***attribute modification***).
- Blocking number of baggage items waiting on **Input**s (***attribute modification***).
- Strategy on one Intersection: if one already existing, delete of the current Strategy object (***instance deletion***). Then, creation of the corresponding Strategy object (***instance creation***) and association with the Intersection object (***association creation***).

3.2.3. OC3: startCircuitSimulation()

From UC1, SSD1 and Domain model:

Preconditions:

The circuit topology and default parameters have been created (see OC1), and optionally the parameters have been modified (see OC2).

Postconditions:

Periodically (every 100 ms):

- **Input**s have verified that a **BaggageItem** should have been created at the beginning of their **Conveyor**, following their bag flow parameter. If yes, they verified that the **Conveyor** was running.
  - If yes, they have created a **BaggageItem** instance, and associated it with the **Conveyor** (***instance creation*** and ***association creation***).
  - If no, they have incremented their counter of **BaggageItem**s waiting (***attribute modification***).
- Normal **Conveyor**s (not involved in **Intersection**s):
  - Updated their associated **BaggageItem**s position, if they were running, using their maximal speed (***attribute modification***).
  - Using their length, they verified that the **BaggageItem** reached their end, and if yes, they signaled the arrival of this **BaggageItem** to their following **Conveyor** (***association creation***). The **BaggageItem** is then considered as “Exiting”

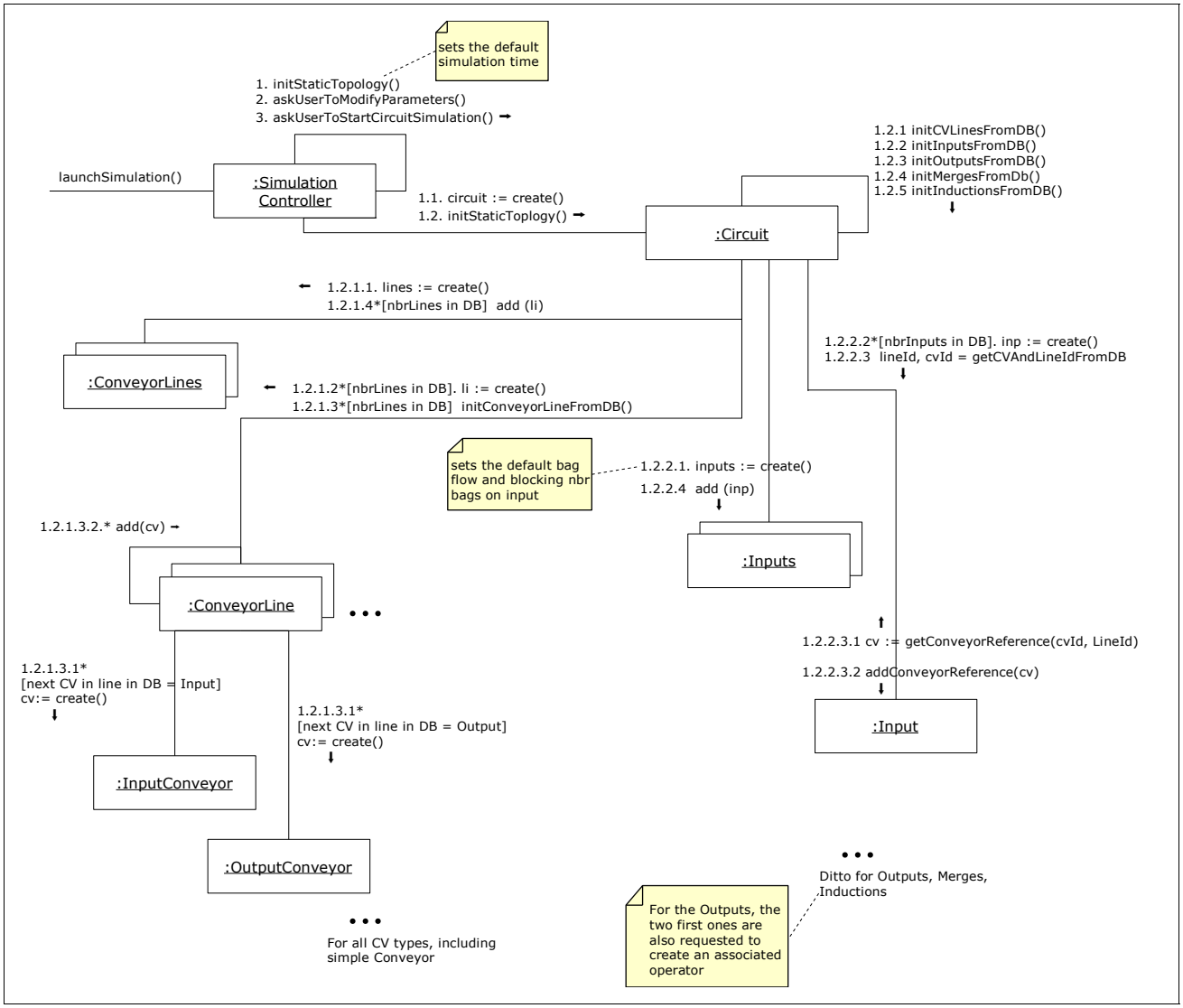
(***attribute modification***).

- They also verified that the exiting **BagaggeItem** have quit them. If yes, the association of this **Conveyor** with this **BaggageItem** has been deleted (***association deletion***).

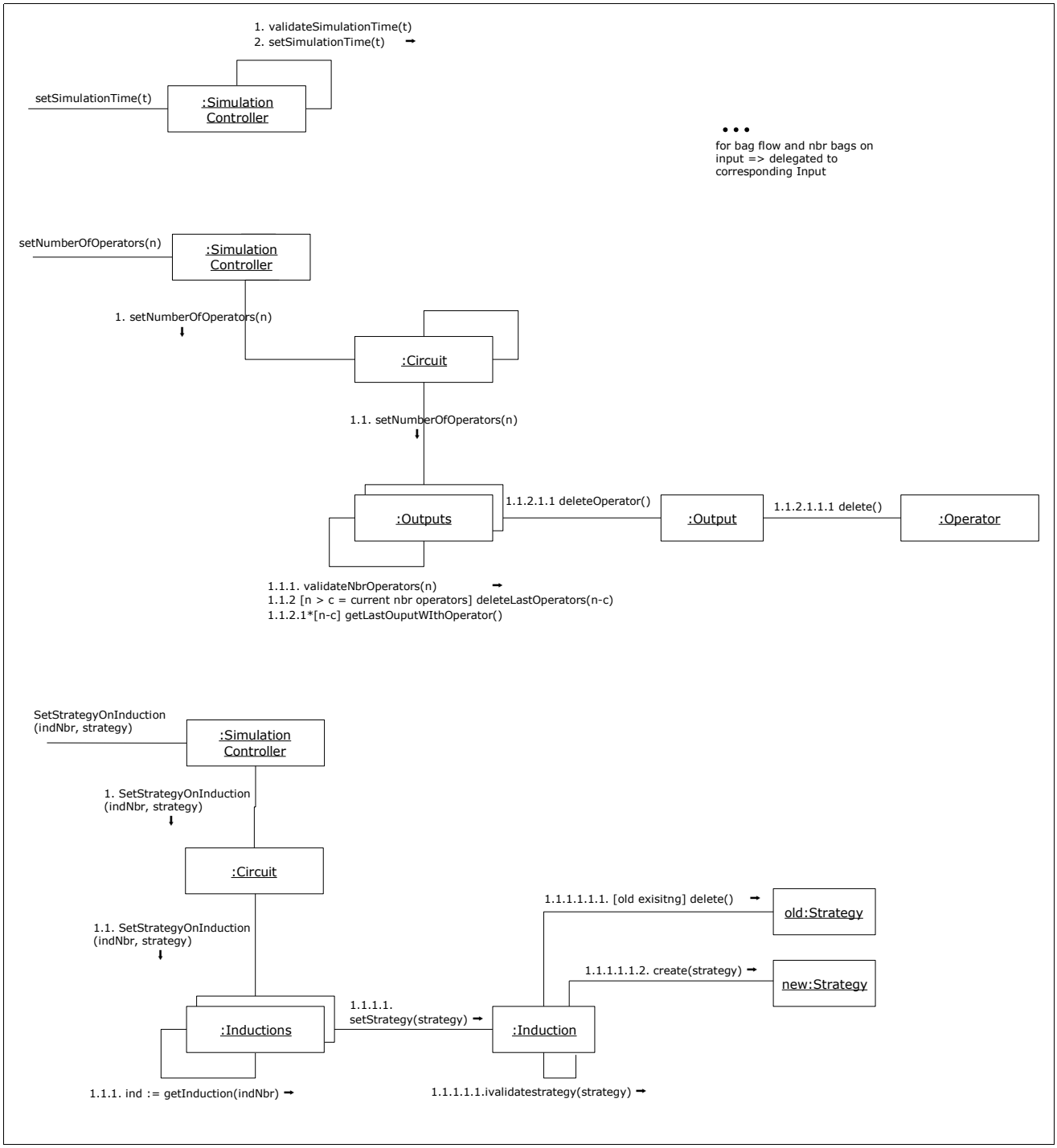
- **Intersection**s have managed the baggage flow following their **Strategy** if existing:
  - if **BaggageItem**s arrived at the end of the input **Conveyor** and/or the main stream **Conveyor**:
    - if the **InductionReceptionConveyor** or the **MergeReceptionConveyor** was not empty, both input and main stream **Conveyor** have been stopped (***attribute modification***).
    - Otherwise, if there was no **Strategy**, the mainstream **Conveyor** had priority, and thus the input **Conveyor** has been stopped (***attribute modification***)
    - If there was a **Strategy**, the corresponding **Conveyor** (input or main stream) has been stopped (***attribute modification***)
  - If the **InductionReceptionConveyor** or the **MergeReceptionConveyor** became empty back, and if **BagageItem**(s) was(were) waiting (input and/or main stream **Conveyor**(s) was(were) stopped):
    - if there was no **Strategy**, the mainstream **Conveyor** had priority, and thus it has been de-blocked if needed (***attribute modification***)
    - If there was a **Strategy**, the corresponding **Conveyor** (input or main stream) has been de-blocked (***attribute modification***)
- **Operator**s on **Output**s have verified that they should have taken a **BaggageItem** following their specified capacity, and that their **Conveyor** had a BaggageItem on their position. If yes, the corresponding **BaggageItem** has been deleted (***instance deletion***), together with its association with the **Conveyor** (***association deletion***).
- The **SimulationController** has verified that none of the **Input**'s baggage item waiting counters reached the limit blocking value. If this was the case, the simulation has been declared as failed (***attribute modification***).

3.3. Collaboration diagrams (Interaction diagrams)

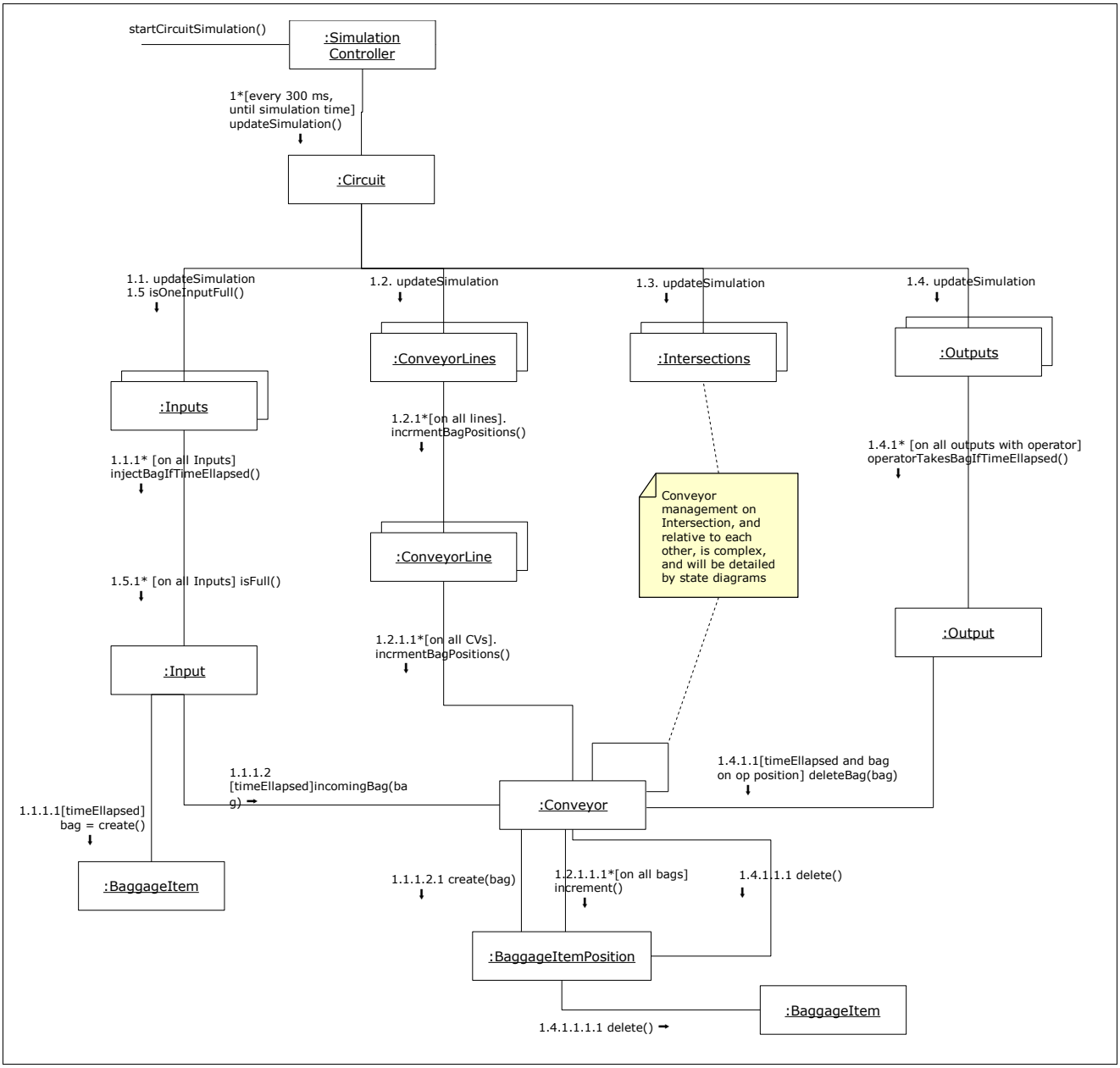
3.3.1. ID1: launchSimulation()



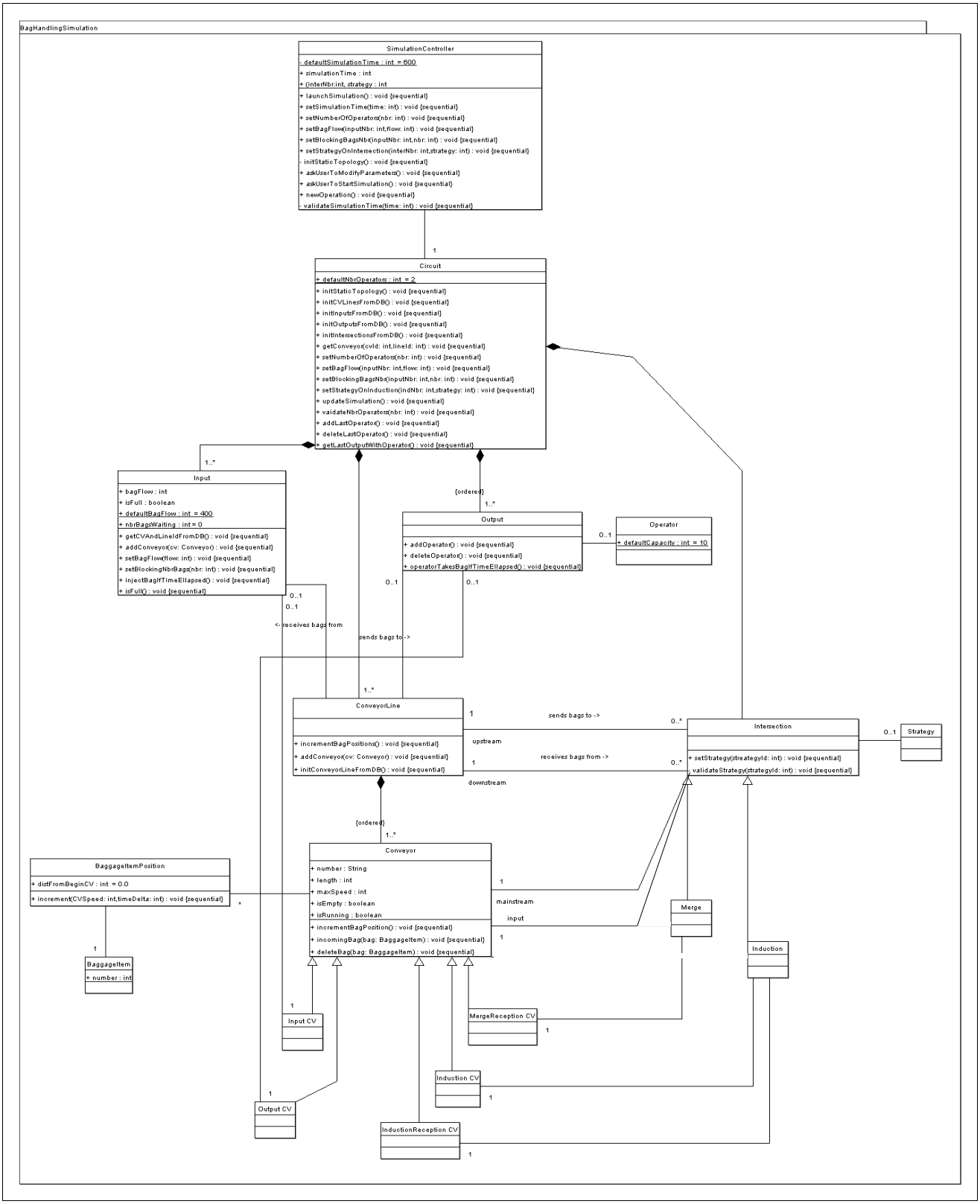
3.3.2. ID2: enterParameter()



3.3.3. ID3 startCircuitSimulation()



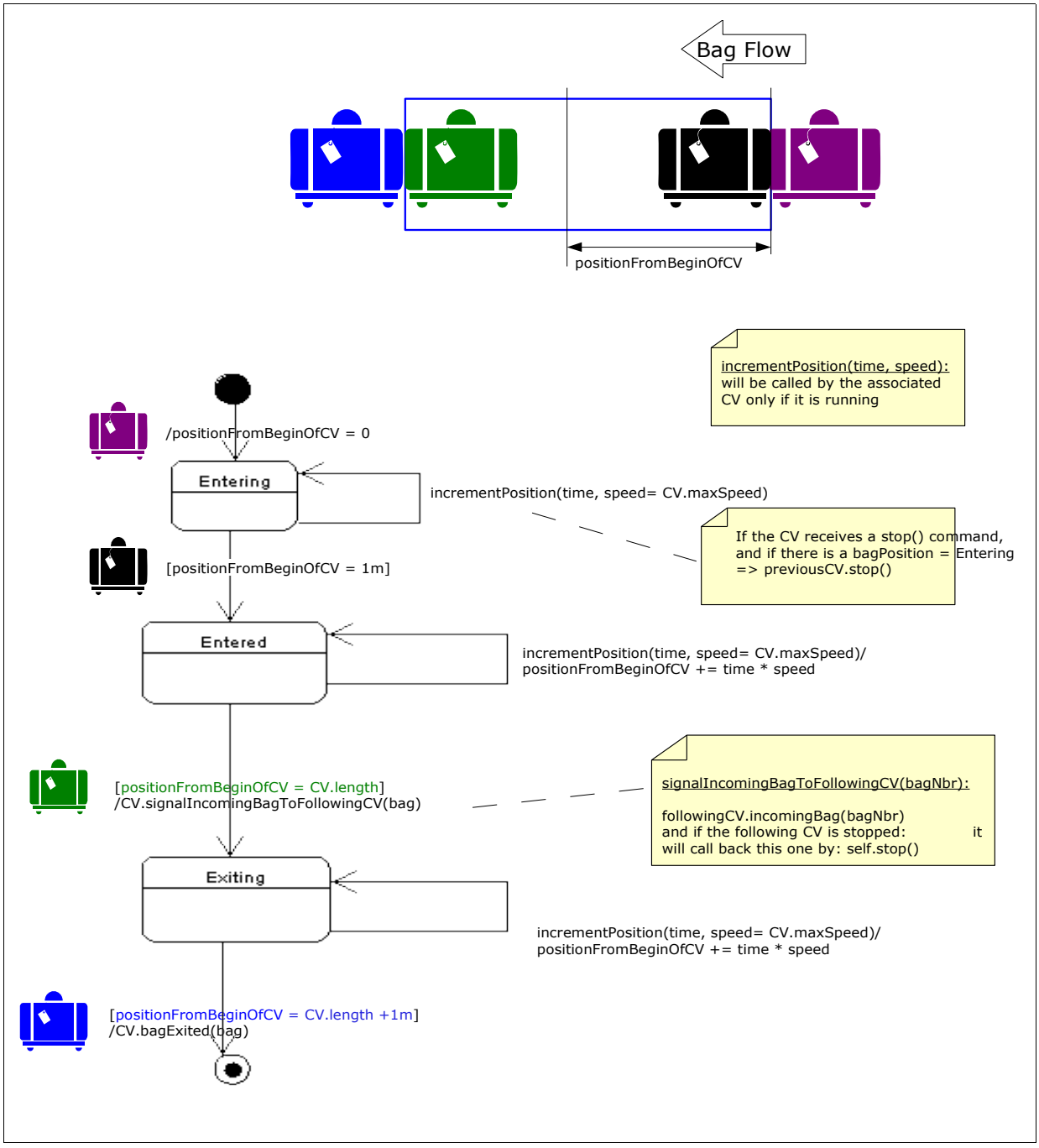
3.3.4. Updated class diagram



3.4. State diagrams

3.4.1. Normal Conveyor trajectory

3.4.1.1. FSM 1: BaggageItemPosition

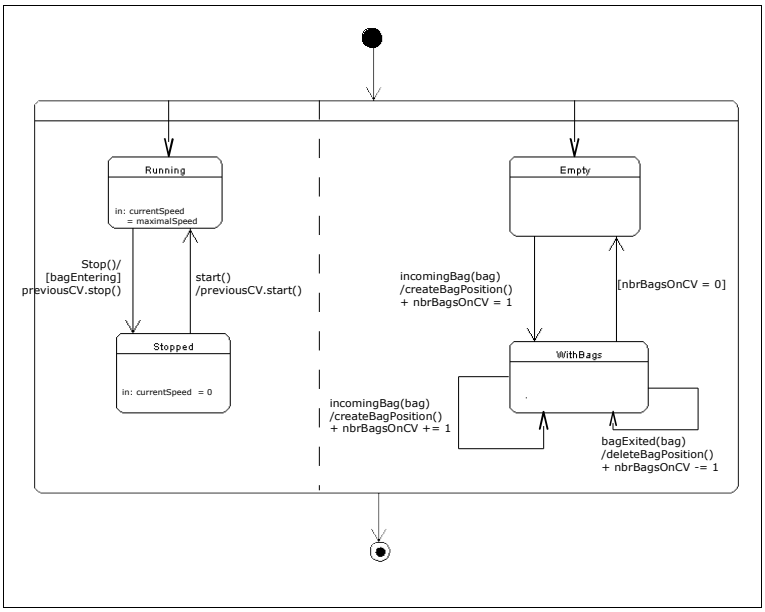


OK for horizontal movement, but some conveyors are curve, and some move the bag "vertically" => we should manage the x and y position of the bag on a normal conveyor,

- only x incremented
- only y incremented
- x and y incremented following a "9-12" curve (analogy with a clock)

– x and y incremented following a “6-9” curve (analogy with a clock)

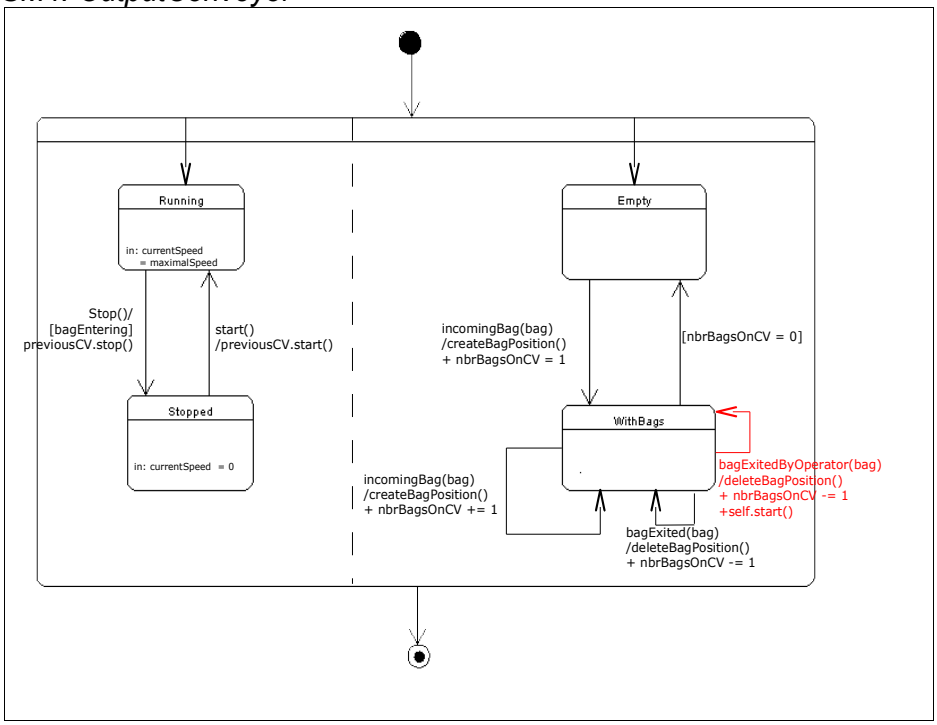
3.4.1.2. FSM2: Conveyor



3.4.1.3. FSM3: InputConveyor

Same as normal Conveyor, except that **previousCV** does not exist => not a particular type.

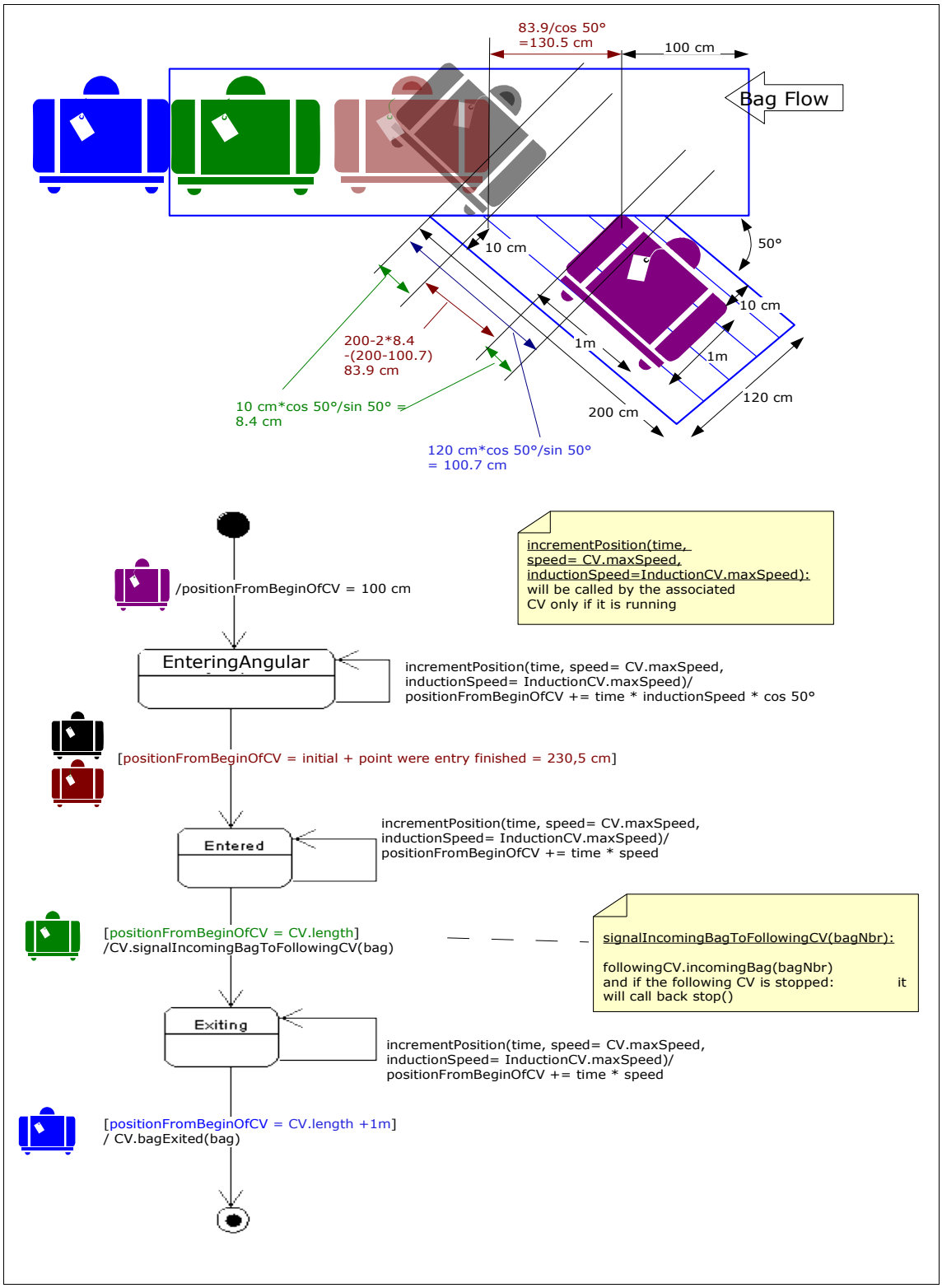
3.4.1.4. FSM4: OutputConveyor



Note also that for the last Conveyor, **followingCV** does not exist, and thus followingCV.isRunning() is always equal to false

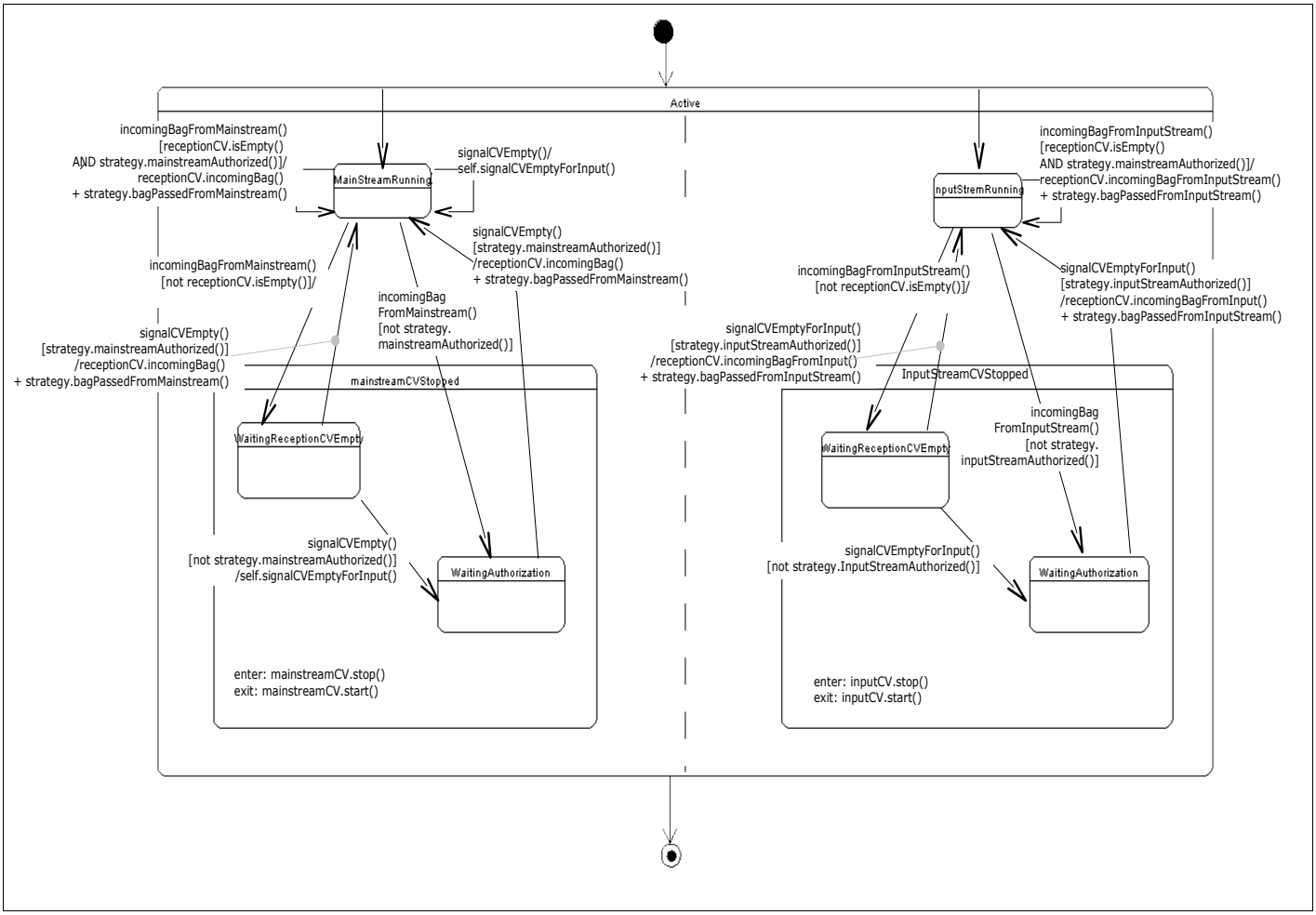
3.4.2. Induction

3.4.2.1. FSM5: BaggageItemAngularFirstPosition



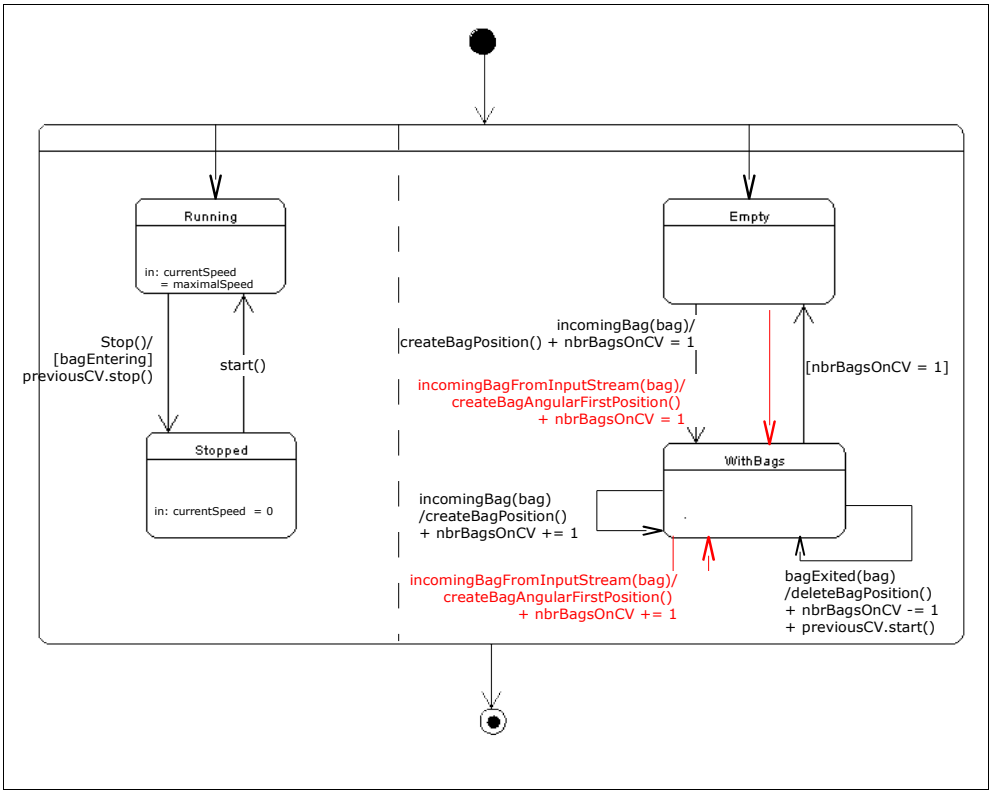
3.4.2.2. FSM6: Induction

The generic FSM for an intersection is:

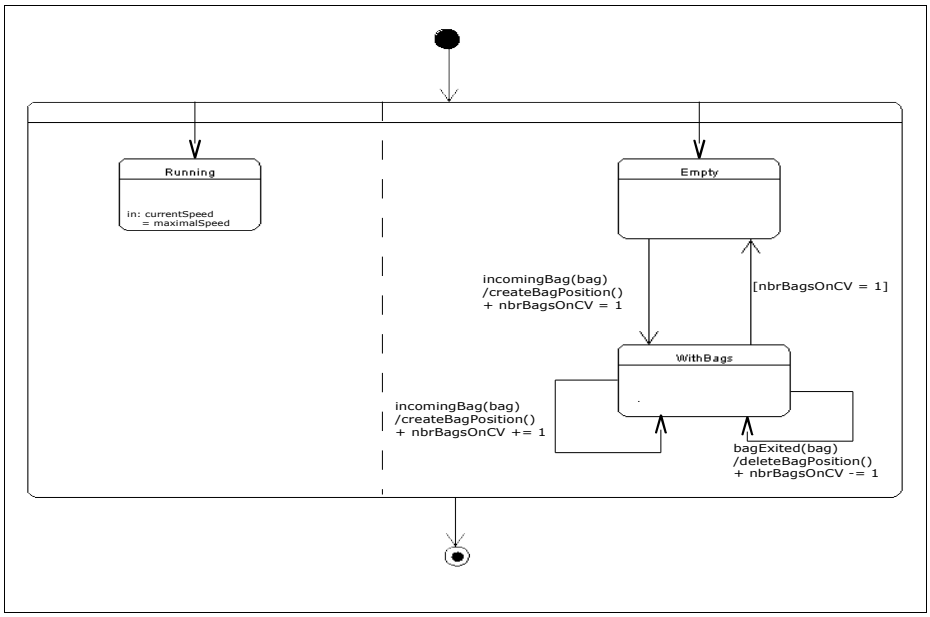


This diagram underlines the priority of the main stream on the input stream. The messages are passed to the InputStream part only if not usable by the MainStream (either because no bag is coming from the main stream, or because the strategy doesn't authorize a bag coming from the mainstream for the moment).

3.4.2.3. FSM7: InductionReceptionConveyor

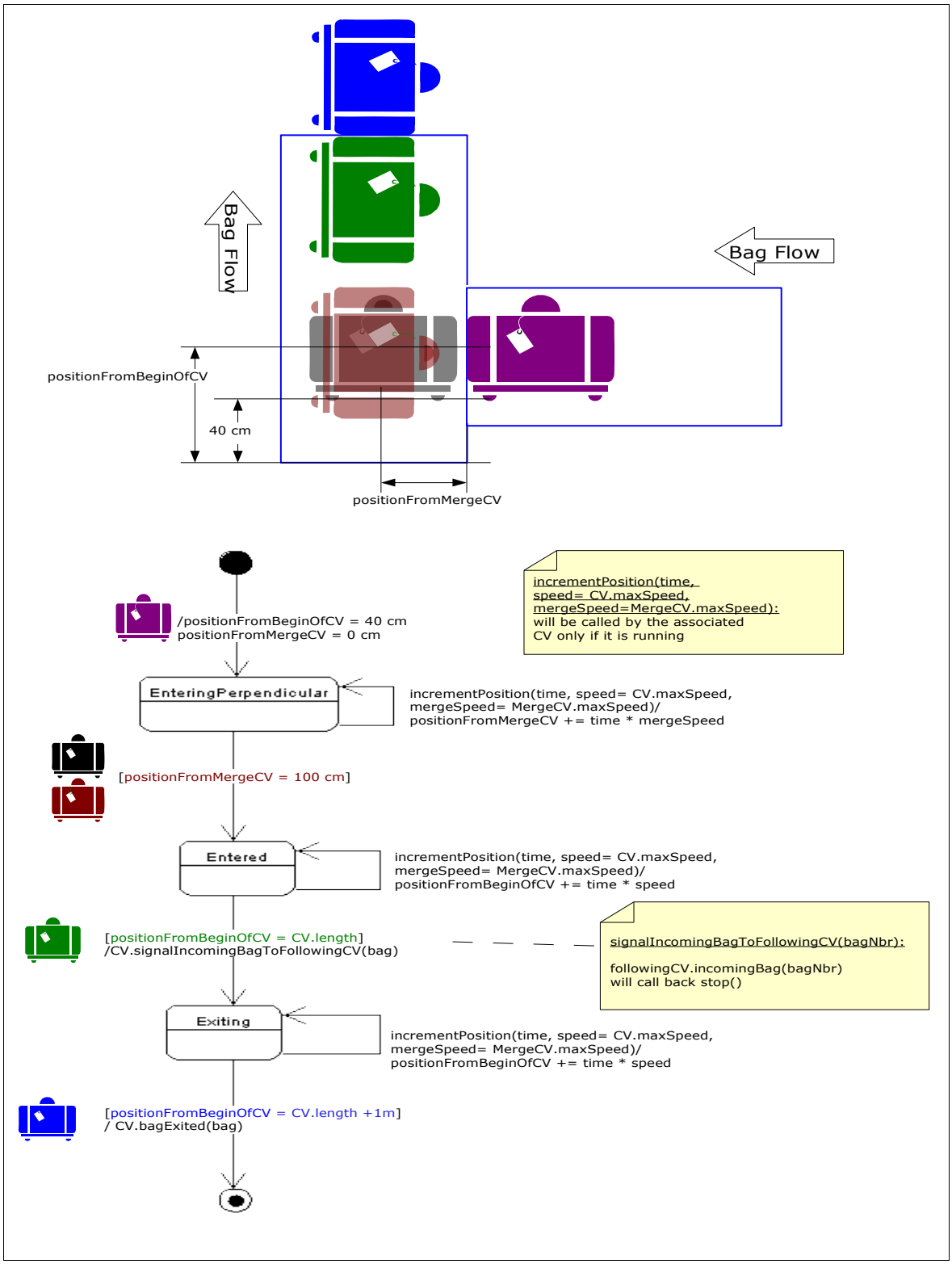


3.4.2.4. FSM8: Induction Conveyor



3.4.3. Merge

3.4.3.1. FSM9: BaggageItemPerpendicularFirstPosition



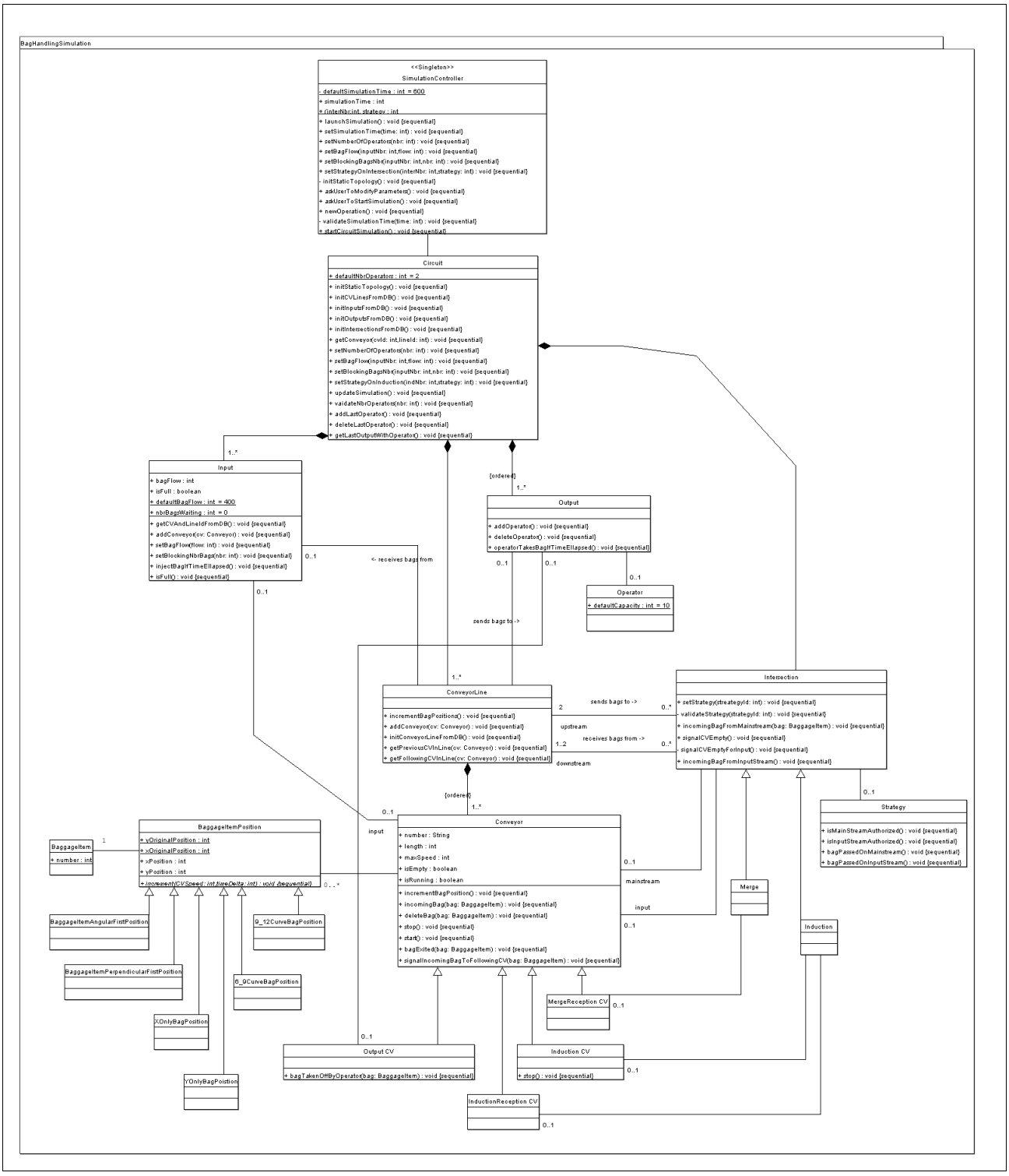
3.4.3.2. FSM10: Merge

Same as #3.4.2.2. FSM5: Induction, except that the three influenced conveyors are:

3.4.3.3. FSM11: MergeReceptionConveyor

Same as ,#3.4.2.3. FSM6: InductionReceptionConveyor with createBagPerpendicularFirstPosition() in place of createBagAngularFirstPosition().

3.4.4. Updated class diagram



3.5. N+1 views

3.5.1. Logical view

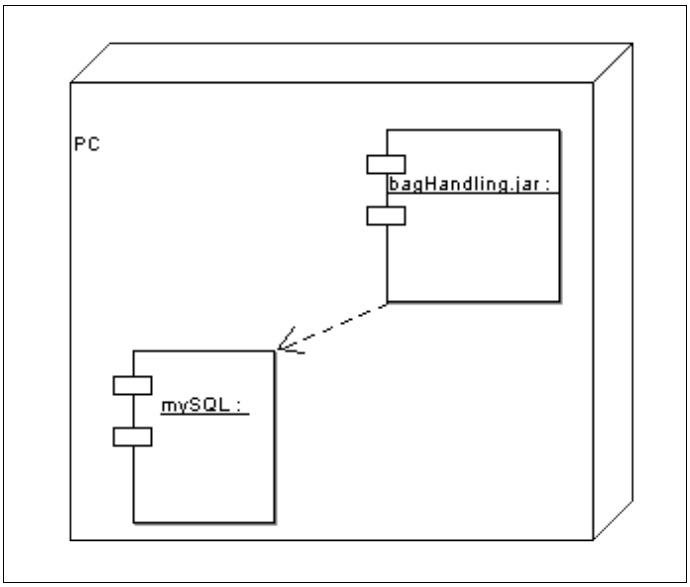
See #3.4. State diagrams (with updated class diagram).

3.5.2. Process view

Up to now, we only have one process with one thread => no need for Process view.

3.5.3. Deployment & Implementation views

Up to now, we only have one application (bagHandling.jar) running on one Node:

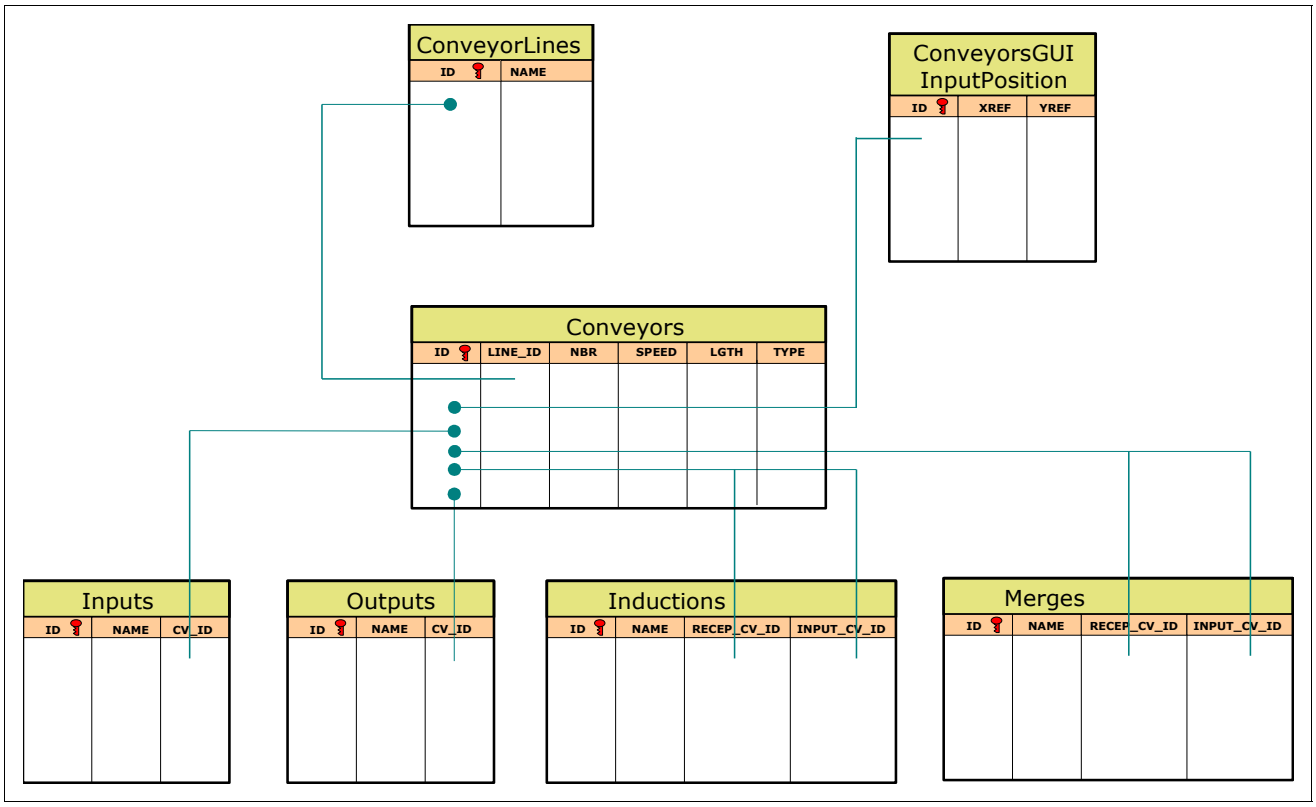


3.5.4. Use-case view

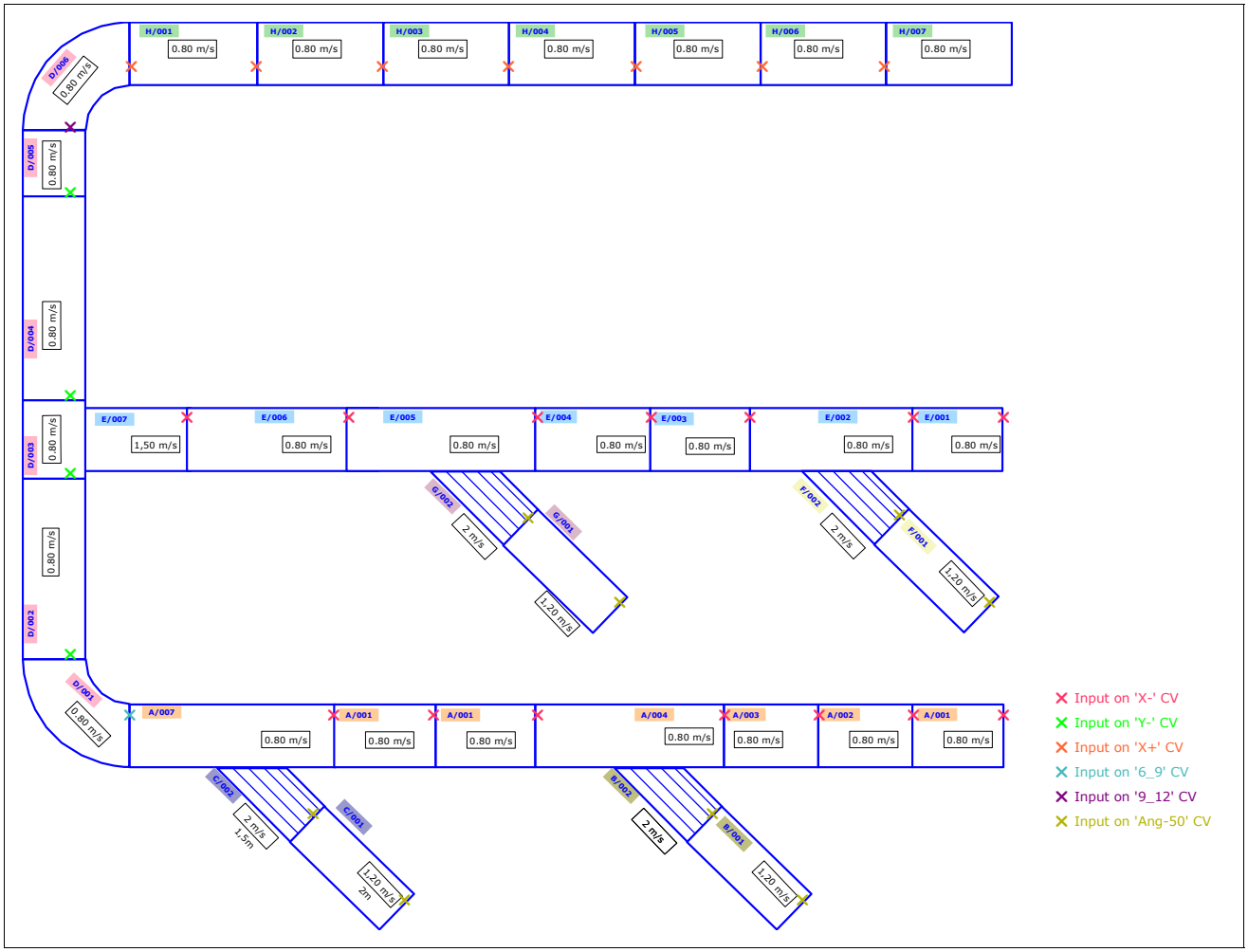
See #3.3. Collaboration diagrams (Interaction diagrams).

3.5.5. Database view

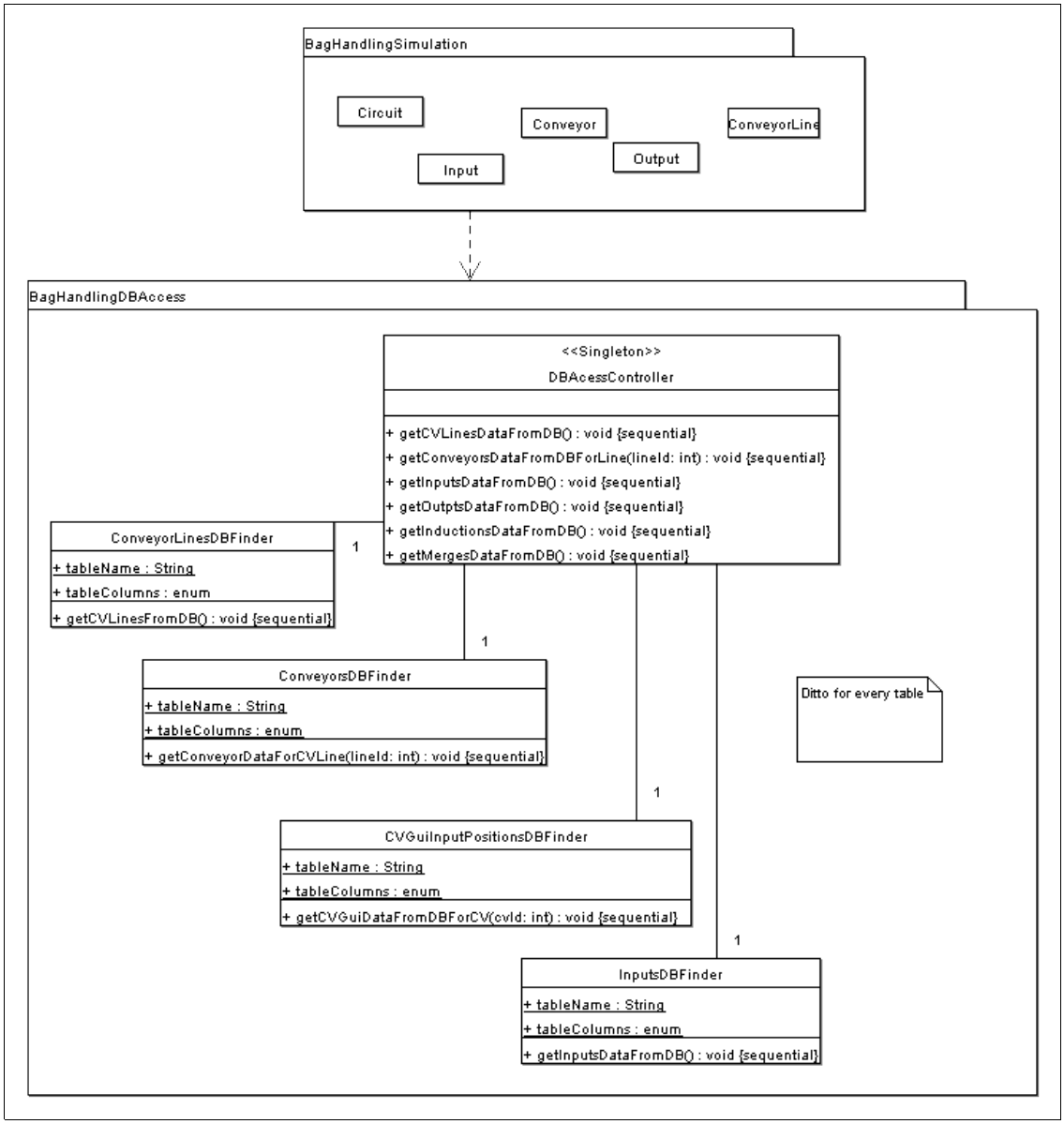
3.5.5.1. Table model



With the Xref, Yref positions taken from:



3.5.5.2. *Updated logical view*



3.6. Basic OO Design Applicability

Information hiding, Demeter's law:

SimulationController (Facade)

Cohesion, responsibility assignment:

Circuit creates and manages ConveyorLines, Inputs, Outputs (Controller, Creator)

ConveyorLines create and manage their conveyors (Controller, Creator)

Intersections manage the intersection state diagram (Expert)

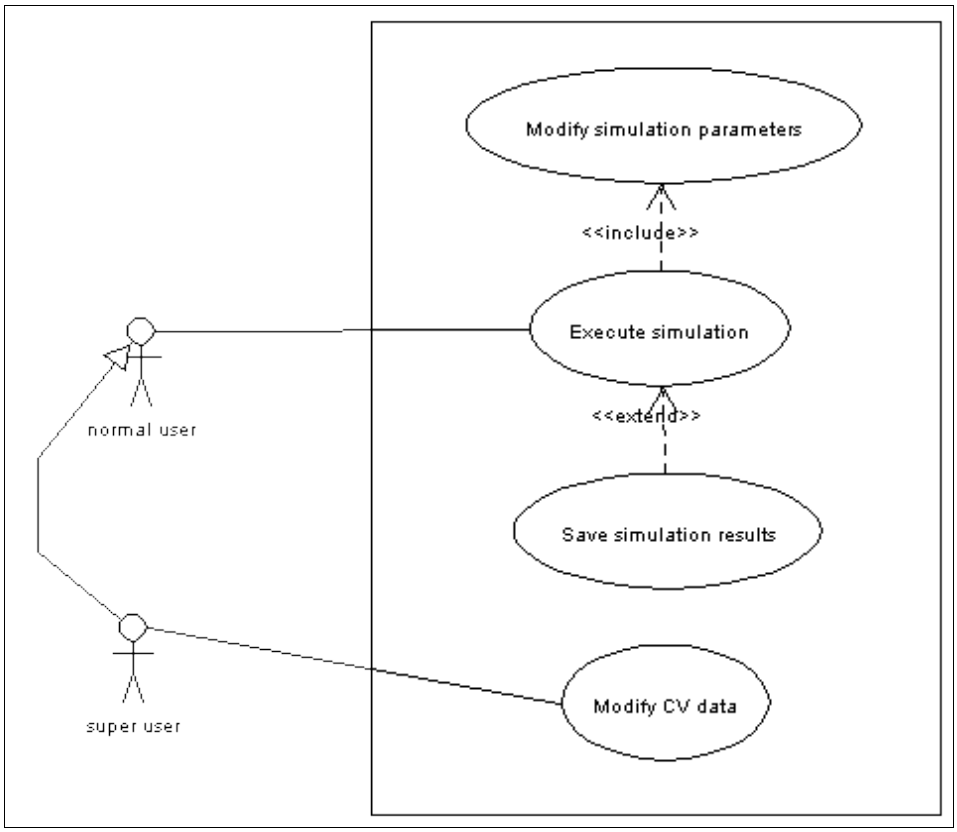
Inputs manage the input of bags (Expert), and create bags (Creator)

Conveyors create the BagPositions (Creator), and manage their state diagram (Expert)

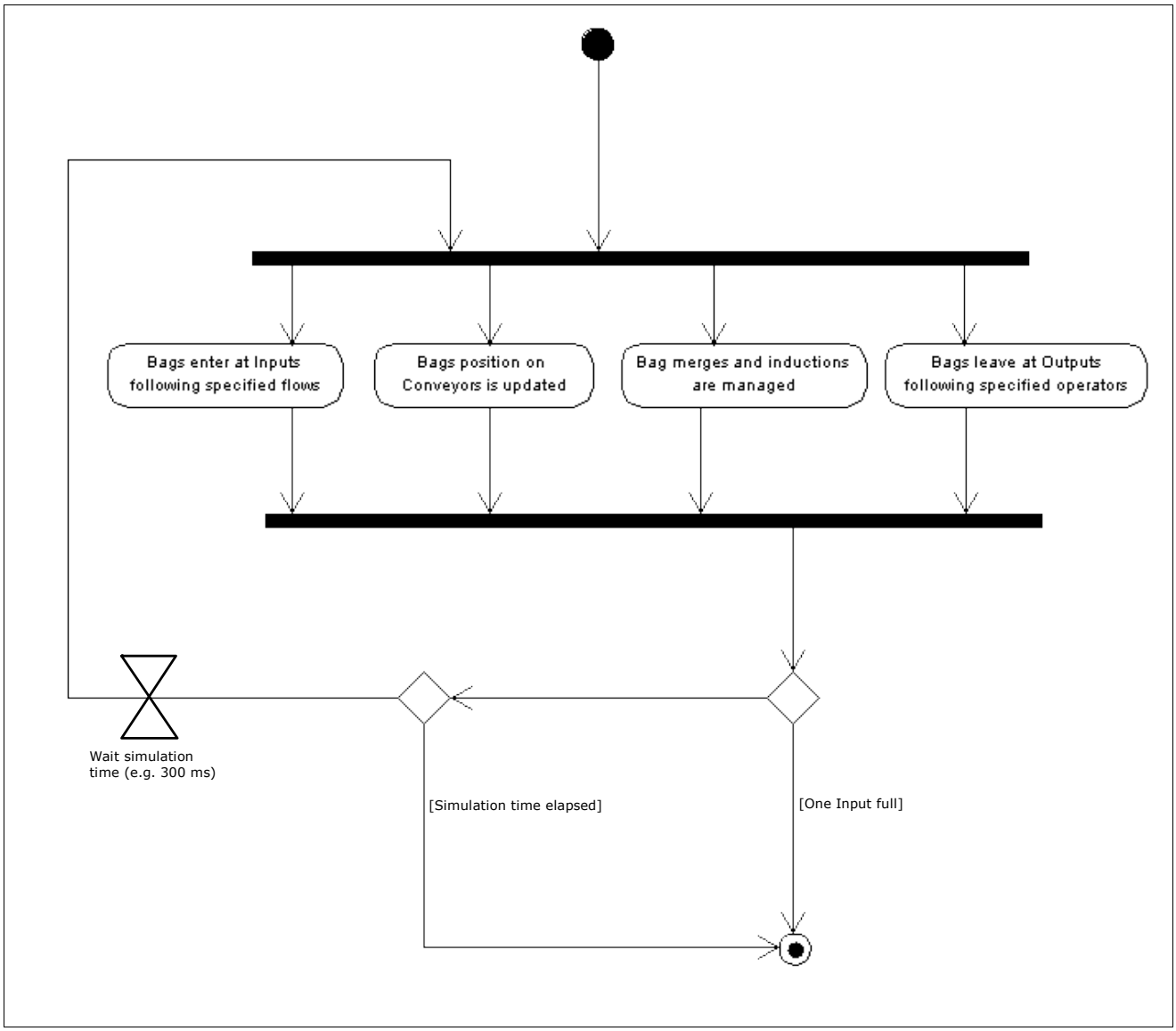
...

4. Elaboration iteration 2

4.1. Refined use-case model

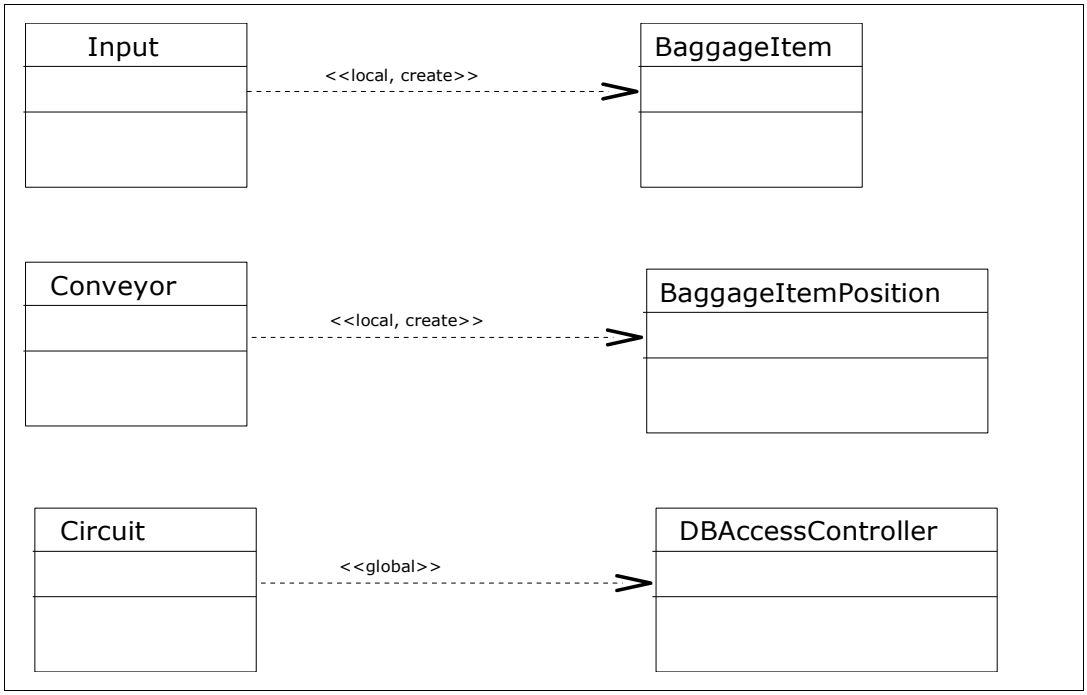


4.2. Activity diagram

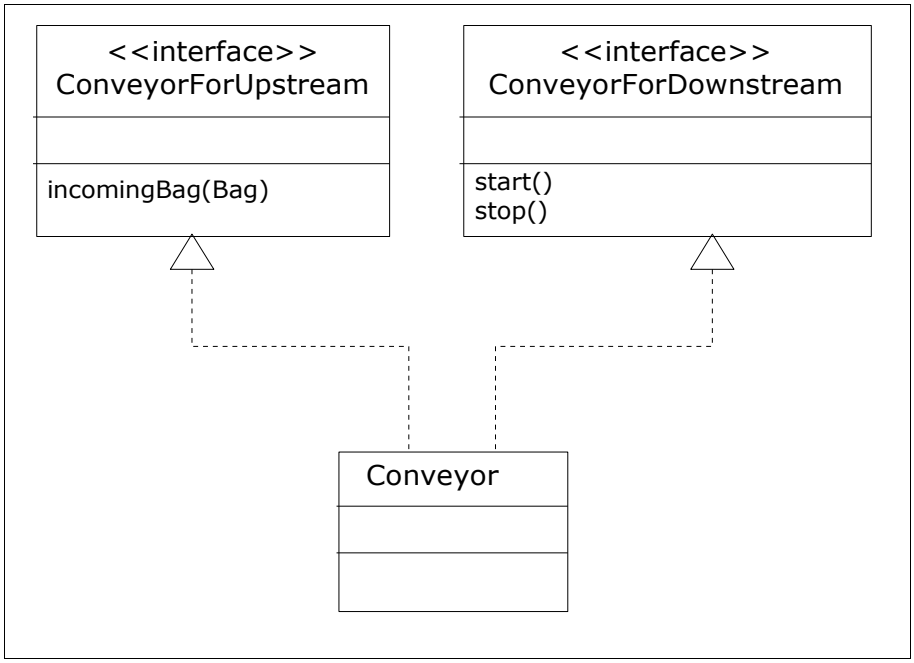


4.3. About class model

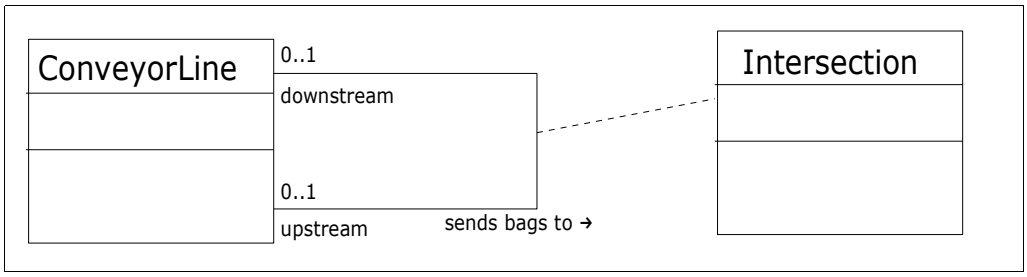
4.3.1. Attribute dependencies



4.3.2. Interfaces and abstract classes



4.3.3. Association class



4.3.4. Qualifier

We may reduce the multiplicity of all the composition associations '**Circuit-Input**, **Circuit-Output**,...) by adding a Qualifier specifying the unique identifier of each composite (respectively Input id, output id,...).

4.3.5. Derived attributes/associations

The **Conveyor** associated with a Input is derived from the fact that it is the first **Conveyor** of a **ConveyorLine** which has no upstream Intersection:

```
{Input.Conveyor = ConveyorLine.Conveyors.firstElement() if ConveyorLine.upstreamConveyorLine = null}
```

Ditto for **Output** conveyors, and **Intersection** managed **Conveyors**.

4.3.6. Constraints

The **ConveyorLine**s have {ordered} lists of **Conveyor**s. We may also specify that the conveyor inheritance is {complete, disjoint}.

4.4. OO design – Main sequence

4.4.1. GUI

C_a = 0

C_e = 3 (SimulationController, Circuit, Conveyor)

→ I = 1, S = 0

A = 0

→ D = 0

4.4.2. Business

C_a = 3 (given as hypothesis)

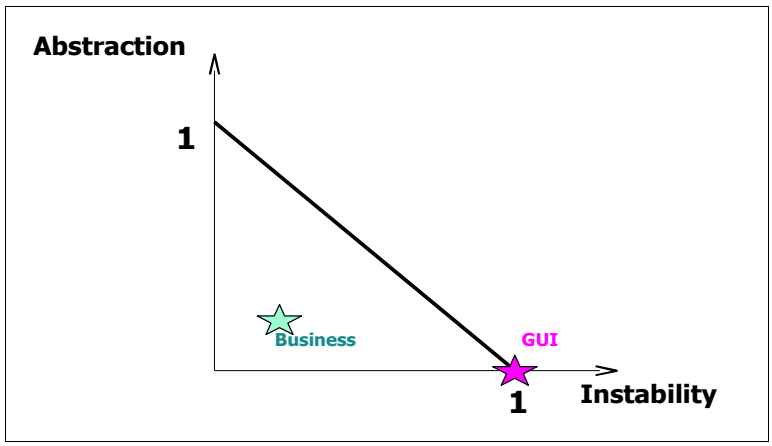
C_e = 1 (DBAccessController)

→ I = 11/4 = 0,25; , S = 0,75

A = 5 (Conveyor + 2 Interfaces, Intersection, BagItemPosition)/23 = 0,22

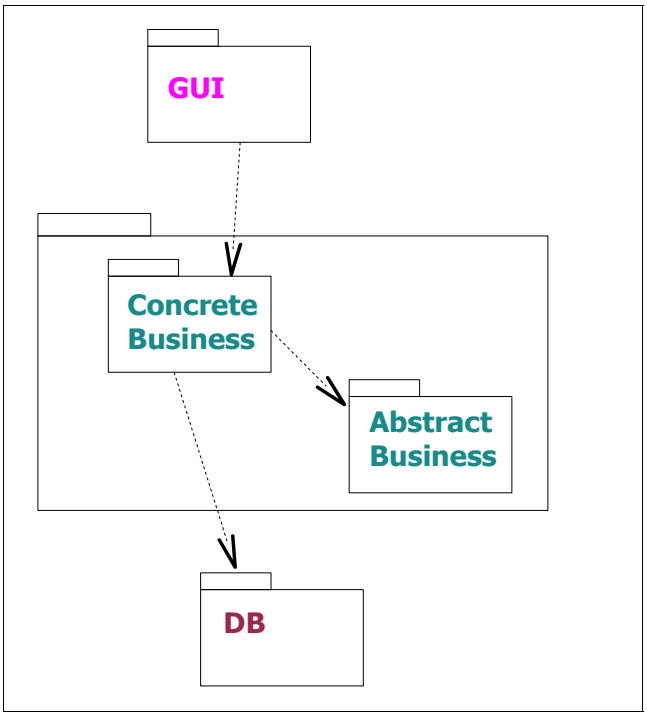
→ D = 0,67

4.4.3. Main sequence



GUI packages are the ultimate example of Instable and less abstract package, because they depend on all the others in the application.

The business package is not very stable nor very abstract. To make the things better, we could add an abstract business package, containing all the stable notions (Circuit, Conveyor,...), and another one less stable and less abstract:



Remember through that this way of doing thing is desirable for Frameworks, and librairies aimed at being reused. This could be “over-kill” for normal/small applications, like our Case Study for example.

4.5. Design patterns applicability

Patterns to be found in the code:

- Singleton (SimulationController, DBAccessController)
- Abstract Factory (BaggagePositionFactory, for BaggageItemPosition types)
- Template Method (reuse of the state machine between the InductionReceptionConveyor and MergeReceptionConveyor)
- Observer (SimulationControllerListener)
- Adapter (Conveyor interfaces on Intersections)
- Strategy (Intersection strategy)
- possibly State (for Intersections)