

Assignment 2

Production System Modelling in AToMPM

Bentley James Oakes

Updated October 16th.

1 Practical Information

The goal of this assignment is to design a domain-specific modelling language (formalism) and subsequently to model production systems (a factory) in that language in the visual modelling tool **AToMPM**.

The different parts of this assignment:

1. Implement the abstract syntax of your language in AToMPM.
 - The formalism used will be
`/Formalisms/LanguageSyntax/SimpleClassDiagram`
2. Enrich the abstract syntax with constraints so that you can check that every model is well-formed.
3. Create a concrete syntax, and generate a modelling environment by compiling the metamodel and the concrete syntax model. Do this incrementally.
 - The formalism for this part will be
`/Formalisms/LanguageSyntax/ConcreteSyntax`
4. Create some production system models that are representative for all the features in your language. The requirements for two valid models are specified below, and there should be a third invalid model to show that your constraints detect invalid models.
5. Write a report that includes a clear explanation of your complete solution and the modelling choices you made.

This assignment should be completed in groups of two if possible, otherwise individually is permissible.

Submit your assignment as a zip file (report in pdf, commented abstract and concrete syntax models, and example models) on Blackboard before **Thursday, October 22nd, 23:59h**. Contact Bentley Oakes (bentley.oakes@uantwerpen.be) if you have any issues.

2 Requirements

This section lists the requirements of the production system domain-specific language and the report. The language requirements are split into two sections: one on abstract syntax, and one on concrete syntax. Make sure to test each requirement with test models!

2.1 Abstract Syntax

There are no modifications in this section from Assignment 1.

The *abstract syntax* of the DSL captures its *syntax* and *static semantics*. The requirements for the abstract syntax are:

1. A production system consists of the infrastructure with *conveyor belts* running between *machines*. *Workers* operate the machines while *items* are transferred on the belts. *Items* are processed and are either assigned by a quality check to be *accepted*, *rejected*, or *fixed*.
2. The belt network consists of a number of interconnected belt *segments*. The language must support the following segments:
 - **Straight** - A trivial belt segment which allows an item to move straight. Has one incoming and one outgoing segment.
 - **Split** - Allows an item to go straight, or to split off onto another belt. Has one incoming segment and two outgoing segments.
 - **Join** - Joins two segments. Has two incoming segments and one outgoing segment.
 - **Machine** - Similar to *Straights*, but can also be at the beginning or end of a belt. Depending on type has zero or one incoming segments and zero, one, or three outgoing segments, but is always connected to at least one segment.
 - Each *Machine* has a unique name, consisting of a single upper case letter, followed by zero or more lower case letters, ending with zero or more numbers.
3. Although this will not be allowed at run-time, the language should support more than one item to be present on a belt segment at a time.
4. There are two types of *Items* in this production system: *Cylinders* and *Cubes*. An *Item* must be on only exactly one segment.
5. There are a number of *Machines* which exist in this production system
 - *Arrival* - The *Arrival Machine* produces either *Cylinders* or *Cubes*. *Items* are produced when the *Machine* is operated.

- *Assembly* - The *Assembly Machine* is actually two linked *Machines*. One for the *Cylinders* and one for the *Cubes*. These *Machines* are different segments, and are linked by a bi-directional association. An *Assembly Machine* handling *Cubes* must be linked to exactly one *Assembly Machine* handling *Cylinders* (and the reverse of course). The output of an *Assembly Machine* is an *AssembledItem*, which is itself an *Item*.
 - *Inspection* - The *Inspection Machine* inspects the *Item* (including *AssembledItem*), and determines whether the *Item* is to be accepted, fixed, or destroyed.
 - An *Inspection Machine* is still a type of *Segment*, but it must also have one output belt for the *Items* to fix, and one output belt for the *Items* to destroy.
 - *Receiver* - The *Receiver Machine* takes any incoming *Items* off the belt, for further processing.
 - *Fixer* - The *Fixer Machine* attempts to repair any defects in the *Item*.
 - *Incinerator* - The *Incinerator* destroys the *Item* on the belt.
6. Each of these *Machines* requires an *Operator* to operate. *Operators* have a *name*¹. Each *Machine* can have at most one *Operator* be present, and the *Operator* must be present for the *Machine* to function.
7. These *Operators* also need a *schedule*, which will be defined in a second domain-specific language. This is so that each operator can have a different schedule in the production system. The requirements for this second language are:
- A schedule is associated to an *Operator* by referring to the name of the *Operator*. Each *Operator* must have a schedule, and a schedule must have an *Operator*.
 - The schedule of an *Operator* tells them which *Machines* to operate, and for how many time steps. The *Operator* will start at the first *Machine* in the list, and operate them in order until the end of the list in which case the schedule will repeat. There must be at least one step where a machine is operated in each schedule.
 - Whenever the operator moves between two different machines (including when the schedule is repeated), there must be a step (of duration one) which represents the movement of the worker within the physical space. During this movement step, that operator will not operate any *Machine*.

¹And hopes, dreams, fears, and rich social lives. But these qualities won't be modelled here, only their name.

2.2 Concrete Syntax

Notations in production systems modelling are not standardized. Therefore you will have a lot of freedom coming up with your own notation. The only requirements are that:

- Your notation does not need to be beautiful, but it must be clear and understandable.
- The license for downloaded images must be respected. For example, `flaticon.com` requires textual attribution which can be placed in your report.

Figure 1 shows an example production system from the first assignment, and one representation of it. Note that your solution should be somewhat more clear than this.

As well, the actions, mappers, and parsers of AToMPM must be used to improve the user experience of modelling the production system:

- Display the percentage chance of acceptance, rework, and failure on inspection machines, controlled with attributes on the machine instance.
- Display other useful information as you see fit (such as the name of machines and operators).
- Model an action that automatically “snaps” a segment when it is connected to another segment.
 - As in, when two segments are connected, one moves directly adjacent to the other.
 - `..Examples../Formalisms/Traffic/` has an example of this in the `Positionable` class.

3 For the Next Assignments

The next assignments will all utilize AToMPM for various model transformations. Therefore:

- Spend time becoming familiar with AToMPM concepts and interface
- Report issues, annoyances, and suggestions to `bentley.oakes@uantwerpen.be` and/or propose code changes at `https://github.com/AToMPM/atompm`.
- Think carefully about your solution, and spend extra time improving the concrete syntax.
- Look at the AToMPM documentation for how to use transformations, and if possible begin experimenting.
 - The next assignment will use transformations to implement the operational semantics of the production system.

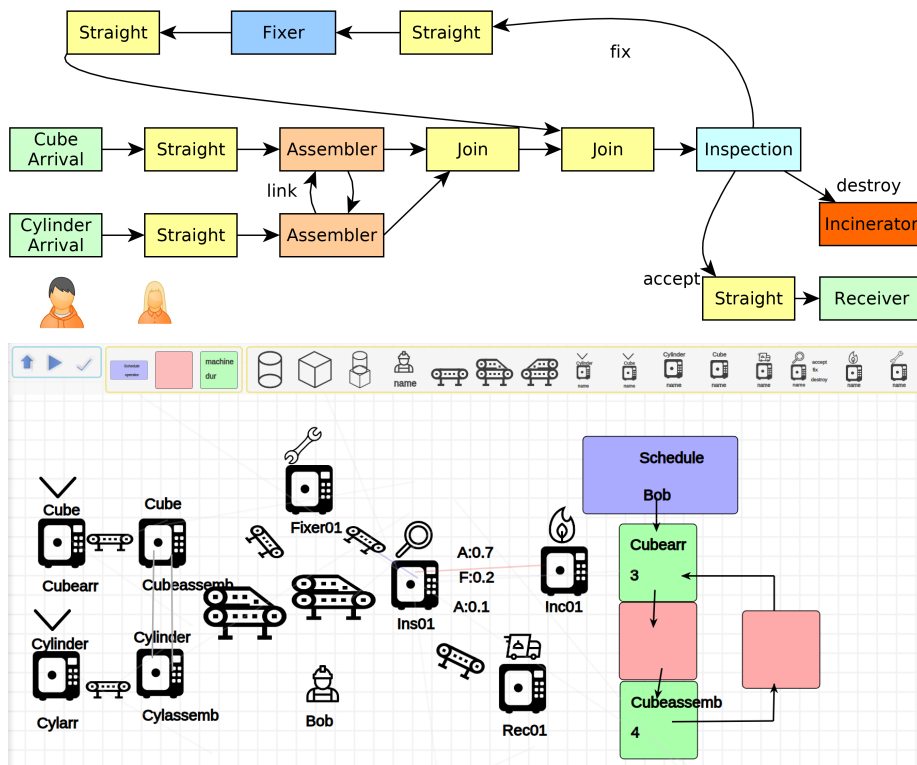


Figure 1: An example production system and its model in AToMPM.

4 Report

There are a number of requirements for the report. Above all, the **marker must be able to read the report and have a clear understanding of all aspects of the assignment, without having to investigate the model files.**

Specifically, the report must contain:

- A brief outline of how the abstract syntax, concrete syntax, and example models meet the requirements of the assignment
 - This may include metamodels, diagrams, (pseudo-)code, etc. as needed to provide the essential details of the assignment.
- A discussion of any interesting decisions made.
- A discussion of possible improvements to the abstract/concrete syntax.
- A brief description of the constraints present in your languages.
- Three example production systems.
 - Two valid, one invalid (doesn't meet the constraints).
- For each production system, show:
 - A figure of the production system within AToMPM.
 - The results of constraint checking on the invalid production system, and which constraint fails.
 - These production systems should be *in medias res* (in the middle of things). This means that there should be items on belts and at machines, and operators at machines. This shows that all appropriate links are modelled and given appropriate concrete syntax.

5 Useful Links and Tips

- AToMPM main page: <https://atmpm.github.io/>
- Download and code: <https://github.com/AToMPM/atmpm>
- Documentation: <https://atmpm.readthedocs.io/en/latest/>

Acknowledgements

Based on an earlier assignment by Simon Van Mierlo.

Icon authors from www.flaticon.com:

- Cylinder - <https://www.flaticon.com/authors/kiranshastry>
- Cube - <https://www.flaticon.com/authors/smashicons>

- Belts, Machine, Inspector, Incinerator, Receiver - <https://www.flaticon.com/authors/freepik>
- Arrival Machine - <https://www.flaticon.com/authors/catalin-fertu>
- Fixer - <https://www.flaticon.com/authors/srip>