TNO innovation for life

ESI
Powered by industry, academia and TNO

Compositional Performance Prediction for Tightly Coupled Manufacturing Systems

TNO Public ⟩ TNO 2023 R12451
8 December 2023

ICT, Strategy & Policy
www.tno.nl
+31 88 866 50 00
info@tno.nl

TNO 2023 R12451 – 8 December 2023

Compositional Performance Prediction for Tightly Coupled Manufacturing Systems

Author(s)	Jacques Verriet

Classification report	TNO Public
Title	TNO Public
Report text	TNO Public
Appendices	TNO Public
Number of pages	41 (excl. front and back cover)
Number of appendices	1
Sponsor	Canon Production Printing
Programme name	AIMS
Project name	AIMS 2023
Project number	060.55084/01.01

Our partners

The research is carried out as part of the AIMS program under the responsibility of TNO-ESI with Canon Production Printing as the carrying industrial partner. The AIMS research is supported by the Netherlands Organisation for Applied Scientific Research TNO.

Contents

1 Introduction

For a long time, the industry trend was (and to some extent still is) to improve and optimise individual systems to their extremes. However, the actual performance of a system does not only depend on the system itself, but also on the context in which it operates. System performance improvements (throughput, imaging, overlay, etc.) can still be increased by looking at the interactions of a system in its context by moving to a systems-of-systems performance engineering approach.

The AIMS program considers the performance of automated production systems of systems. These are facilities composed of multiple production systems connected via (manual or automated) transportations systems.

1.1 Approach

In 2023, AIMS has developed a domain-driven methodology to quickly and accurately predict the timing behaviour of tightly coupled production systems of systems. In these systems, production systems are connected via an automated transport system, e.g. a conveyor belt.

The methodology involves a modular domain model for the specification of the (tightly coupled) production system-of-systems and the work that is to be executed by it. This domain model involves four domain-specific languages, which are organised according to the Y-chart pattern [1]:

- The Material language specifies the materials to be handled by a production system and the operations that can be performed on these materials.
- The Equipment language describes the topology of a production system in terms of the processing stations and their connections.
  - Per station, the Equipment language specifies which operations it can perform and how much time is needed for the preparation and execution of these operations.
  - Per connection, the Equipment language specifies the transportation duration.
- The Job language specifies the batch of work to be performed by the production system in terms of products and their (sub)parts.
- The Allocation language specifies how the work is assigned to the available processing stations and the order in which the work is to be executed.

The specification of the production system and the work to be performed can automatically be translated into a *constraint graph*. This is graph-based analysis model, which specifies the relative timing constraints between events, and which can be used to predict the timing behaviour of the production system, e.g. the duration of the production of a batch of products.

1.2 Outline

This report is outlined as follows. In Chapter 2, the domain model to specify production systems-of-systems and their jobs is described using an illustrative example production system. Chapters 3 and 4 describe transformations from domain model instances to

constraint graph models and the corresponding constraint graph analyses. Chapter 5 describes how the tooling realising the domain model can be installed and used. A summary of the report and possible directions for future work can be found in Chapter 6. Detailed information on the domain model's languages can be found in Appendix A.

2 Domain model

In this chapter, we present the domain model used to specify (the timing behaviour of) manufacturing systems. Such a specification consists of four elements (the green blocks in Figure 2.1):

1. A *material model* describes the materials that are handled by a production system and the operations that can be performed on the materials.
2. An *Equipment model* describes the components of the production system and their capabilities and connections to other components. The component capabilities are described as timing information that is based on a Material model's materials and operations.
3. A *Job model* describes the work to be performed in terms of a Material model's materials and operations.
4. An *Allocation model* describes the assignment of the work specified in a Job model onto the components described in an Equipment model. This includes the order in which work is to be performed.

A full specification is used to generate a constraint model for timing prediction (see Figure 2.1). This generation is explained in Chapter 3.

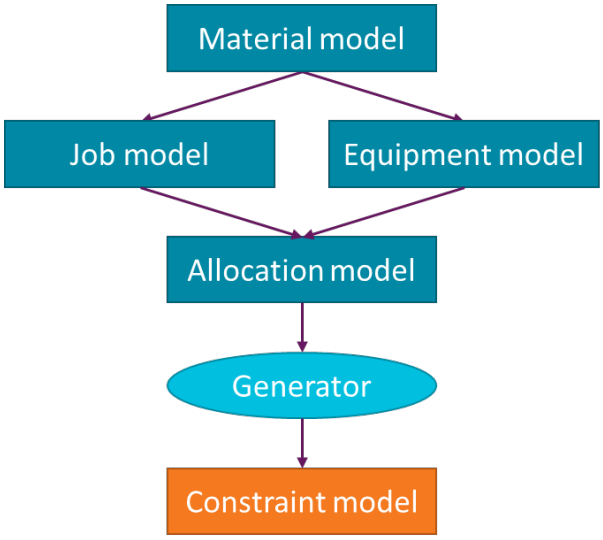

Figure 2.1: Production system domain model

To illustrate the domain modelling approach, we use a running example. This example is presented in Section 2.1. In Sections 2.2, 2.3, 2.4 and 2.5, we present the four elements of the specification of the running example. Details on the underlying syntax of the domain model's languages and the additional validation rules can be found in Appendix A.

2.1 Running example

The running example used to illustrate the domain model is a fictitious printer consisting of an input unit, a printer unit and a stapler unit. This fictitious printer is depicted in Figure 2.2.

- The *input unit* has two *input trays*, which are connected via a switch to the printer unit.
- The *printer unit* has a *printer* station which is capable of printing sheets on the fly, i.e. the sheets are printed as they travel through the printer.
- The *stapler unit* has a buffer where sheets to be stapled are collected, a *stapler* that can staple stacks of sheets, and an *output tray* where stapled stacks of sheets are collected.

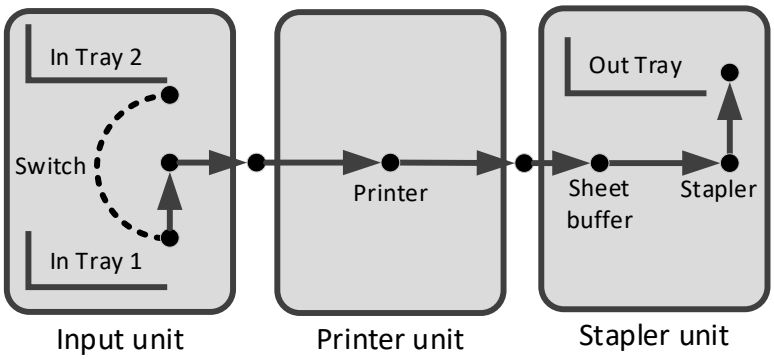

Figure 2.2: Fictitious modular printer example

Between the units, there are *handover points*, which can be used for modular timing analysis. This is explained in Chapter 4. Such an analysis may be particularly useful if modules are developed by different manufacturers.

2.2 Material model

A Material model consists of three elements, all specified in a single file with a .machine extension:

1. A specification of the *material types*,
2. A specification of the *material instances*, and
3. A specification of the *operations* on the materials.

These elements are described in more detail in Sections 2.2.1, 2.2.2 and 2.2.3.

2.2.1 Material types

Material types represent classes of similar materials. We distinguish *basic* and *composed* material types. The composed material types represent aggregates of other material types. The basic material types cannot be decomposed.

Figure 2.3 contains the material types for the running example. The purple text represents the keywords of the language. The black text is instance specific; these may involve user-defined names/properties (e.g. the first occurrence of the word `sheet` is the user-defined name of the material) or references to user-defined names (e.g. the second occurrence of the word sheet refers to the first occurrence). This material model specifies that the running example distinguishes two material types: the material type sheet is a basic material, and the material type stack is a composed material type containing sheets. Note that we do not distinguish between unstapled and stapled stacks of sheets.[1]

[1] The constraint graph generation and analysis explained in Chapters 3 and 4 do not use material composition; this may be used for future validation of specifications.

```
basic material type sheet

composite material type stack composed of sheet
```

Figure 2.3: Material types

2.2.2 Material instances

The material instances of a Material specification involve instances of material types. Like for the material types, we distinguish basic and composed material instances. Composed materials are assumed to be homogeneous, i.e. they consist of one material.[1]

Figure 2.4 specifies the materials of the running example. These are instances of the material types introduced in Section 2.2.1. We distinguish two instances of sheets, A4 sheets and A3 sheets. Moreover, the materials of the running example include two composite materials: stacks of both instances of sheets. This results in a total of four material instances. The specification in Figure 2.4 also shows the dimensions of the different materials.[2]

```
basic material A4 type sheet
dimensions
  length = 0.297 m
  width = 0.210 m

basic material A3 type sheet
dimensions
  length = 0.297 m
  width = 0.420 m

composite material stackA4 type stack composed of A4
dimensions
  length = 0.297 m
  width = 0.210 m

composite material stackA3 type stack composed of A3
dimensions
  length = 0.297 m
  width = 0.420 m
```

Figure 2.4: Materials

2.2.3 Operations

The last part of the Material specification describes the operations that can be performed on the specified material types and the corresponding material instances.

Figure 2.5 shows that the running example has two sheet operations, releaseSheet and collectSheet, and three stack operations, releaseStack, stapleStack and collectStack. It is assumed that operations do not change the material: e.g. a stapled stack is also stack.

[2] The constraint graph generation and analysis explained in Chapters 3 and 4 do not use material dimensions; these may be relevant for more refined timing analyses.

```
sheet operation releaseSheet

sheet operation collectSheet

stack operation releaseStack

stack operation stapleStack

stack operation collectStack
```

Figure 2.5: Material operations

2.3 Equipment model

An Equipment model describes the components of the fictitious printer. An Equipment model consists of three parts, all specified in a single file:

1. The *nodes* in the equipment and their capabilities,
2. The *segments* connecting the equipment nodes, and
3. Common *paths* through the equipment.

2.3.1 Nodes

The nodes in an Equipment model correspond the locations that material visits and where operations are performed. The equipment nodes of the running example are shown in Figure 2.6. The domain model distinguishes three types of nodes: *input nodes*, *output nodes*, and *internal nodes*. This example only includes internal nodes. The nodes in the model correspond one-to-one to the nodes visualised in Figure 2.2. If one would like to model the fictitious printer's units separately, then the handover nodes between them would become input nodes of the upstream unit(s) and output nodes of the downstream unit(s). This is explained in Chapter 4.

An equipment node can support multiple operations. The Equipment model specifies these operations; the keyword **`operation`** is used for this. The duration of the operation is specified for all relevant material instances. As performing an operation may require some preparation, the operation's recovery and setup times are also specified using the keyword **`setup`**. Note that these recovery and setup times may depend on the previous material instance and the current material instance. To define the *state* of a node's operation, the time and material instance of the last completion of an operation can be specified using the keyword **`committed`**.

Beside preparation for node operations, an equipment node may also need to prepare to receive a certain material. This is also specified using the **`setup`** keyword, but now used within the scope of an equipment node. An Equipment model specifies how much time an equipment node needs to change from one material instance to another. This duration is the minimum time between one piece of material leaving the node and another arriving at the node.

```
equipment RunningExample

node inTray1 type internal
  operation releaseSheet
    material A4 duration 0.5 s
      after A4 setup 0.5 s
      after A3 setup 60.0 s
    material A3 duration 1.0 s
      after A4 setup 60.0 s
      after A3 setup 1.0 s

node inTray2 type internal
  operation releaseSheet
    material A4 duration 0.5 s
      after A4 setup 0.5 s
      after A3 setup 60.0 s
    material A3 duration 1.0 s
      after A4 setup 0.5 s
      after A3 setup 60.0 s

node traySwitch type internal

node handover1 type internal

node handover2 type internal

node printer type internal
  setup
    material A4
      after A4 setup 0.5 s
      after A3 setup 0.5 s
    material A3
      after A4 setup 0.5 s
      after A3 setup 0.5 s

node sheetBuffer type internal
  setup
    material A4
      after A4 setup 0.5 s
      after A3 setup 60.0 s
      after stackA4 setup 0.5 s
      after stackA3 setup 60.0 s
    material A3
      after A4 setup 60.0 s
      after A3 setup 0.5 s
      after stackA4 setup 60.0 s
      after stackA3 setup 0.5 s

  operation collectSheet
    material A4 duration 0.5 s
      after A4 setup 0.5 s
      after A3 setup 1.0 s
    material A3 duration 1.0 s
      after A4 setup 1.0 s
      after A3 setup 0.5 s
```

```
  operation releaseStack
    material stackA4 duration 0.5 s
      after stackA4 setup 0.5 s
      after stackA3 setup 1.0 s
    material stackA3 duration 1.0 s
      after stackA4 setup 1.0 s
      after stackA3 setup 0.5 s

node stapler type internal
  operation stapleStack
    committed stackA4 finish time 0.0 s
    material stackA4 duration 0.5 s
      after stackA4 setup 0.5 s
      after stackA3 setup 1.0 s
    material stackA3 duration 0.5 s
      after stackA4 setup 1.0 s
      after stackA3 setup 0.5 s

node outTray type internal
  operation collectStack
    material stackA4 duration 0.5 s
      after stackA4 setup 0.5 s
      after stackA3 setup 1.0 s
    material stackA3 duration 1.0 s
      after stackA4 setup 1.0 s
      after stackA3 setup 0.5 s
```

Figure 2.6: Equipment nodes

2.3.2 Segments

The segments of an Equipment model describe the connections between equipment nodes. Figure 2.7 specifies the running example's segments using the keyword **segment**. We assume that two nodes are connected by at most one segment. For each segment, an Equipment model defines the segment's **source** and **target** node(s) and the transport **duration** of the segment. We assume that a segment's transport duration is independent of the material instance being transported.

The example system in Figure 2.2 involves a switch, which can alternate between the input trays. Switches are modelled with multiple **source** nodes and/or multiple **target** nodes. A switch represents multiple source-to-target paths, of which only one may be active. The time needed to switch between these paths is defined using the **switch time** keywords. We assume that the switch duration is independent of the switch's position. Moreover, we assume that switching happens when the switch is not occupied, i.e. all materials have left the switch.

```
segment traySwitch
source inTray1 inTray2
target traySwitch
duration 1.0 s
switch time 1.0 s

segment traySwitch_handover1
source traySwitch
target handover1
duration 1.0 s

segment handover1_printer
source handover1
target printer
duration 2.0 s

segment printer_handover2
source printer
target handover2
duration 2.0 s

segment handover2_sheetBuffer
source handover2
target sheetBuffer
duration 2.0 s

segment sheetBuffer_stapler
source sheetBuffer
target stapler
duration 1.0 s

segment stapler_outTray
source stapler
target outTray
duration 1.0 s
```

Figure 2.7: Equipment segments

2.3.3 Paths

A piece of material that runs through the equipment follows a path, i.e. a sequence of nodes connected by segments. We assume that such a path involves a single end-to-end activity, which is performed without being interrupted.

Figure 2.8 shows the specification of predefined paths through the equipment including the operations that are performed along the path. This example only specifies one operation per node on a path, but one can also specify a sequence of operations to be performed at a single node.[3]

[3] The equipment paths have been introduced for modelling convenience. By defining commonly used paths, the Allocation models (see Section 2.5) become more concise.

```
path stackPath
  sheetBuffer:   releaseStack
  stapler:       stapleStack
  outTray:       collectStack

path sheetPath1
  inTray1:       releaseSheet
  traySwitch:
  handover1:
  printer:
  handover2:
  sheetBuffer:   collectSheet

path sheetPath2
  inTray2:       releaseSheet
  traySwitch:
  handover1:
  printer:
  handover2:
  sheetBuffer:   collectSheet
```

Figure 2.8: Equipment paths

2.4 Job model

The Job model represents the work to be performed (by the equipment). The work is specified as a batch of products. Products are composed of parts, which are composed of subparts. The subparts are assumed to be atomic job elements. This means they are to be executed without interruption.

We assume that products are independent of each other. This means that they can be created in any order. On the other hand, we assume that the order of the parts and subparts in a specification is fixed. The specified order of the parts and subparts can be seen as a recipe to create the product.

Figure 2.9 specifies a Job model for the running example. It specifies two products, both consisting of two parts. For each part, the Job model specifies its subparts.[4] The first part of both products involves creating a stack of two (printed) sheets, the second part involves stapling this stack and collecting it. The only difference between the products is the involved material instance. The first product is created from (stacks of) A4 sheets, the second from (stacks of) A3 sheets.

[4] Note that the Job language does not have a subpart keyword.

```
batch RunningExample

product leafletA4
  part A4 sheets
    A4 sheetA4_1: releaseSheet collectSheet
    A4 sheetA4_2: releaseSheet collectSheet
  part stackA4 leaflet
    stackA4 stack1: releaseStack stapleStack collectStack

product leafletA3
  part A3 sheets
    A3 sheetA3_1: releaseSheet collectSheet
    A3 sheetA3_2: releaseSheet collectSheet
  part stackA3 leaflet
    stackA3 stack1: releaseStack stapleStack collectStack
```

Figure 2.9: Job model

2.5 Allocation model

An Allocation model assigns the work defined in a Job model to the nodes and segments specified in an Equipment model. The Allocation model is based on the Equipment model's path. Per subpart, it specifies which path through the equipment is to be followed including which operations are to be performed.

Figure 2.10 shows the Allocation model for the running example. The first leaflet uses sheets from the first input tray and the second leaflet takes sheets from the second input tray. The sheets are collected in the stapler unit's sheet buffer. From the sheet buffer, the collected sheets are handled as stacks; they are released from the sheet buffer, stapled at the stapling station, and collected at the output tray.

```
batch RunningExample
equipment RunningExample

product leafletA4
  part sheets
    sheetA4_1 path sheetPath1
    sheetA4_2 path sheetPath1
  part leaflet
    stack1 path stackPath


product leafletA3
  part sheets
    sheetA3_1 path sheetPath2
    sheetA3_2 path sheetPath2
  part leaflet
    stack1 path stackPath
```

Figure 2.10: Allocation model

In the running example, two sheets of A4 paper become a leaflet and two sheets of A3 paper become a leaflet. It is the specifier's responsibility that the paths followed by the materials matches this intention. The syntax and validation rules (see Appendix A) do not check whether the paths of the elements of a composite material, i.e. the individual sheets

in the running example, end at the same equipment node as where the path of the composite, i.e. a stack of sheets in the running example, starts. This is one of the possible extensions discussed in Chapter 6.

2.6 Summary

In this chapter, we have introduced the four elements of the manufacturing domain model using an illustrative running example. With the languages, we have described the underlying assumptions:

- Material language assumptions
  - Composed materials are homogeneous, i.e. they consist of one material.
  - Material operations do not change the material.
- Equipment language assumptions
  - Two nodes are connected by at most one segment.
  - A segment's transport duration is independent of the material being transported.
  - The switch duration of a switch segment is independent of the switch's position.
  - A switch may change position when it is not occupied, i.e. all materials have left the switch.
- Job language assumptions
  - The subparts of a Job specification are atomic job elements.
  - An equipment path involves a single activity, which is performed without being interrupted.
  - Products in a Job specification are independent of each other.
  - The order of the parts and subparts in a Job specification is fixed.

Most of these assumptions are used in the generation of analysis models, which is explained in Chapters 3 and 4. Some are there because of modelling convenience. More details on the languages' syntax and validation rules are described in Appendix A.

3 Monolithic constraint graph analysis

In Chapter 2, we have described the four languages to specify a production system. This chapter describes the generation of constraint graphs, which can be used to accurately predict the latency of work allocated to a production system. Section 3.1 introduces the constraint graph formalism. Section 3.2 describes the constraint graph generation using the running example introduced in Chapter 2. Section 3.3 describes how the constraint graphs can be used to predict the latency of allocated work.

3.1 Constraint graphs

Elmaghraby and Kamburowski [2] consider the analysis of generalised precedence relations. To describe these, they use graphs, which we call *constraint graphs*. Constraint graphs are constraint models containing relative timing constraints. A constraint graph is a directed graph $G = (V, A)$ with a collection of nodes V and a collection of arcs $A \subseteq V \times V$. The nodes in V represent events to be scheduled and the arcs in A represent the timing constraints between the events.

The timing constraints are described by a function $\Delta: A \to \mathbb{R}$. An arc $a = (n_1, n_2)$ from node n_1 to node n_2 specifies that event n_2 should follow at least $\Delta(n_1, n_2)$ time units after event n_1. The timing constraint defines a *release time* for event n_2. Note that the timing constraints are not limited to positive numbers; an arc $a = (n_1, n_2)$ with a negative $\Delta(n_1, n_2)$ specifies a *due time* for event n_1: it must happen at most $-\Delta(n_1, n_2)$ time units before event n_2.

The goal of scheduling is finding a function $\mathrm{T}: V \to \mathbb{R}$ that assigns an occurrence time to all events and that satisfies all timing constraints specified by a constraint graph's arcs. Typically, one wants to minimise the *makespan* or *latency* of a schedule, i.e. $\max_{n \in V} T(n)$.

Note that one can create constraint graphs for which no feasible schedule exists. This is when the constraint graph contains a *positive cycle*, i.e. a cyclic path whose summed arc weights is positive. The simplest example is shown in Figure 3.1; it involves a graph with two nodes n_1 and n_2 and arcs $a_1 = (n_1, n_2)$ and $a_2 = (n_2, n_1)$ with weights $\Delta(a_1) = 2$ and $\Delta(a_1) = -1$. In a feasible schedule for this graph, $T(n_2) \geq T(n_1) + \Delta(a_1) \geq T(n_2) + \Delta(a_1) + \Delta(a_2) = T(n_2) + 1$, which is impossible.

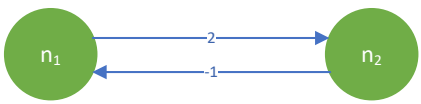

Figure 3.1: Infeasible constraint graph

3.2 Constraint graph generation

From a specification in our domain model, we can generate constraint graphs as introduced in Section 3.1.

The constraint graph that is generated from the running example introduced in Chapter 2 is shown in Figure 3.2. The nodes in this graph represent an event at an equipment node. There are three types of *event nodes*:

- *Start nodes* represent the start of a path of a piece of material.
- *Arrival nodes* represent the arrival of a piece of material at an equipment node.
- *Operation nodes* represent the completion of an operation of a piece of material at an equipment node.

The arcs in the generated constraint graph have a matrix shaped:

- Vertical paths represent the material view: the vertical paths are those that pieces of material follow through the equipment.
- Horizontal paths represent the equipment view: they contain arrivals and operations at one equipment node.

There are several types of arcs in the generated constraint graph. In Figure 3.2, they are distinguished by colour.

The *black arcs* between the event nodes represent the paths of a piece of material as specified in the Allocation model. One can see that one such path consists of sequences of groups consisting of one arrival node followed by zero or more operation nodes. Note that all the arcs on such a path are bidirectional, with a weight equalling either a segment's travel duration or an operation's duration. Bidirectional arcs indicate that the time between the connected events must equal exactly the weight of the arc.[5] This means that an arrival or the end of an operation coincides with the start of the next operation.[6]

The *blue arcs* in the constraint graph represent operation recovery and setup. These horizontal arcs are unidirectional, and their weights equal the minimum time between two consecutive completions of the same operation. Note that the weights of these arcs include the setup time between the first and the second operation and the duration of the second operation.[7] Both depend on the involved material instances as one can see in the Equipment model.

The *orange arcs* in the constraint graph represent the setup times for arrival of a piece of material at an equipment node. The orange arcs start at the last event for one piece of material at an equipment node to the first event for the next piece of material at the same equipment node. These constraint graph source nodes are arrival nodes, if there are no operation nodes, otherwise they are operation nodes. These constraint graph target nodes are either arrival nodes or start nodes.

The *yellow arcs* in the constraint graph represent constraints between consecutive events at an equipment node. The yellow arcs involve constraints between the end of the path of one piece of material and the start of the path of another piece of material. These are only included in the constraint graph if the one ends at the equipment node where the other path starts. In the example in Figure 3.2, the yellow arcs represent the aggregation of individual sheets into a stack of sheets.

[5] A bidirectional arc is a shorthand for two unidirectional arcs, one with a positive weight and one with a negative weight, equal to the weight of the bidirectional arc.
[6] This is the reason why we decided not to include events for the starts of operations.
[7] This could have been captured differently by using diagonal arcs to the start of the operation. This would create a multi-graph, i.e. a graph in which two nodes are connected by more than one arc.

The *grey arcs* in the constraint graph represent the delay from changing the position of a switch. There is one such arc in Figure 3.2, which represents switching from the first to the second input tray between the leaflets.

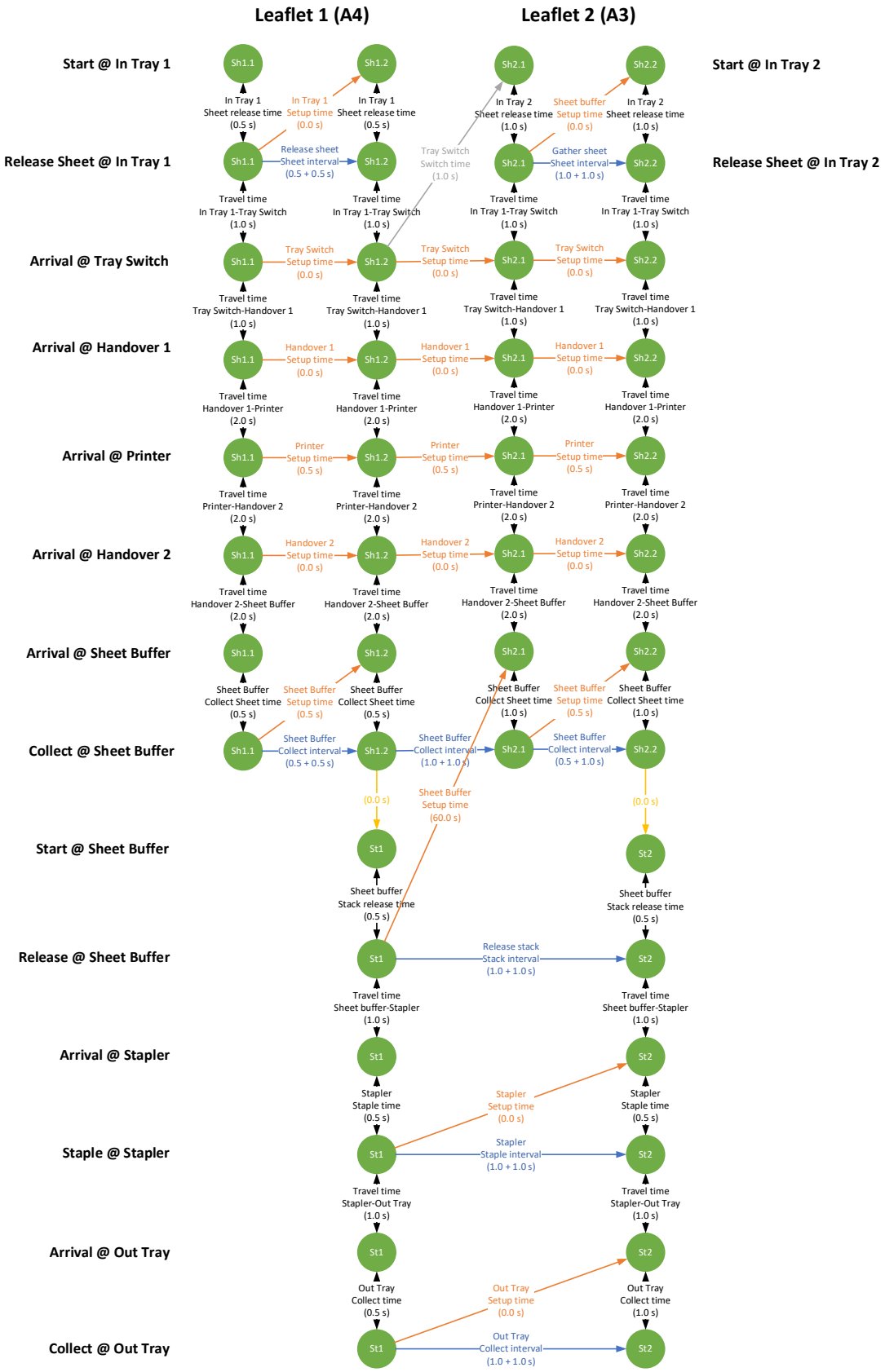

Figure 3.2: Constraint graph for running example

The creation of a constraint graph like the one in Figure 3.2, applies several rules. Every type of arc, indicated by the arc colours in Figure 3.2, has its own rule. These rules are based on the following assumptions:

- The durations of operations and transportation are constant.
- An equipment node can handle at most one piece of material at a time.
- Pieces of material do not overtake each other.
- The timing of an operation on a piece of material does not depend on future pieces of material.

The last assumption makes that generated constraint graphs do not contain arcs leading "from right to left": arcs follow the sequence of the pieces of material as specified in the Allocation model.

Other assumptions influencing the constraint graph generation are mentioned as underlined text in the explanation of the domain model's languages in Chapter 2.

3.3 Constraint graph analysis

In Section 3.2, we have illustrated how one can create a constraint graph from a domain model instance. This section explains how constraint graphs can be used for timing analysis.

A constraint graph contains relative timing constraints between events. To use a constraint graph for timing prediction or for optimisation, absolute timing information is needed as well. For this, an additional node is added to the constraint graph, the *zero node*. This node represents an event at the absolute time zero, which occurs before all events in the constraint graph. Zero-weight arcs are added between the zero node and all nodes that correspond to the start events of all pieces of material.

In the resulting constraint graph, the duration of the longest path from the zero node to an event in the constraint graph equals the earliest time at which this event can occur. From a constraint graph as the one shown in Figure 3.2, a schedule is created by computing the longest path tree from the zero node using the Bellman-Ford algorithm [3].[8]

[8] The Bellman-Ford algorithm is a single-source shortest path algorithm, which can compute longest paths by changing the sign of all arc weights.

4 Modular constraint graph analysis

In Chapter 3, we have considered the generation of one monolithic constraint graph and using that graph to create a schedule. Constraint graphs quickly become very large. As the Bellman-Ford algorithm is a quadratic algorithm [3], the running times of constraint graph analysis does not scale.

In this chapter, we extend the constraint graph generation and analysis in two manners. They are meant to make constraint graph analysis more flexible and more scalable:

1. *Incremental analysis*: When scheduling, not all work to be scheduled is known upfront. This means that one must perform constraint graph analysis multiple times. This can be achieved by a domain model specification that grows over time, but this is bad for scalability. We present an incremental method that keeps the constraint graph small.
2. *Modular analysis*: The running example introduced in Chapter 2 was a product line consisting of three units. Instead of generating a single constraint graph for the whole production line, one can also generate and analyse a constraint graph per unit and combine the analysis outputs. This reduces the constraint graph size and makes the method more scalable.

The extensions of the domain model for incremental and modular analysis are explained in Section 4.1. The updated constraint graph generation is explained in Section 4.2. Section 4.3 explains the updated constraint graph analysis.

4.1 Domain model

In Chapter 2, we have used a monolithic production line as a running example. In this section, we show the domain model's language concepts to describe incremental and modular scheduling and timing prediction. Section 4.1.1 explains the modular analysis using the running example. Section 4.1.2 and 4.1.3 explains the concepts of the Equipment language and the Allocation language needed to facilitate modular timing analysis.

4.1.1 Running example

We assume a single controller that is responsible for deploying work in a production system. In the running example introduced in Chapter 2, the input unit is responsible for the deployment: it decides when sheets are released from the input trays. Figure 4.1 shows the running example with the corresponding information flows. There is a flow of scheduling decisions downstream, i.e. from the input unit to the printer unit, and from the printer unit to the stapler unit. These decisions specify the times at which events occur.

Figure 4.1 also shows an upstream information flow: the timing constraints of the stapler unit are communicated to the printer unit, and those of the printer unit are communicated to the input unit.

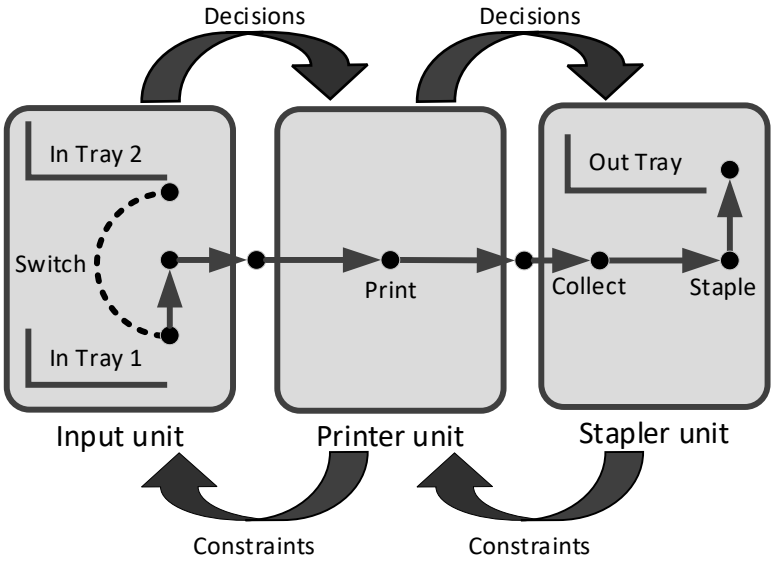

Figure 4.1: Running example with information flows

To communicate (scheduling) decisions downstream, we need concepts to fix event timings. These concepts are introduced in the Equipment model; these concepts are explained in Section 4.1.2. To communicate timing constraints upstream, we need additional concepts in the Allocation model. These concepts are explained in Section 4.1.3.

The Material and Job model do not require additional concepts for modular and incremental constraint graph analysis. The Material and Job language concepts as introduced in Chapter 2 are sufficient.

4.1.2 Equipment model

The Equipment model as introduced in Chapter 2 specified the timing constraints of pieces of equipment with respect to arrivals of pieces of material and operations on pieces of material. The Equipment model example did specify the state of the equipment based on what the equipment has done in the past. To define the *state* of a node with respect to the last handled material, the time and material of the last node departure can be specified using the keyword **`committed`**.

An example of the keyword **`committed`** is shown in Figure 4.2, which shows the specification of the stapler node of the stapler unit in Figure 4.1. The third line in Figure 4.2 specifies that the last stapling operation of the stapler node involved the material stackA4 and finished at time 12.0, i.e. 12 seconds after the zero node event.

```
node stapler type internal
  operation stapleStack
    committed stackA4 finish time 12.0 s
    material stackA4 duration 0.5 s
      after stackA4 setup 0.5 s
      after stackA3 setup 1.0 s
    material stackA3 duration 0.5 s
      after stackA4 setup 1.0 s
      after stackA3 setup 0.5 s
```

Figure 4.2: Node specification with committed work

What the example does not show is that the keyword **committed** can also be used to specify the material instance that last left the node and the time at which this material instance has left. Furthermore, the keyword **state** is used to specify the initial position of a switch. An example for the input unit is shown in Figure 4.3.

```
segment inTray_traySwitch
source inTray1 inTray2
target traySwitch
duration 1.0 s
switch time 1.0 s
state inTray1-traySwitch
```

Figure 4.3: Switch node specification

To allow modular constraint graph analysis, we create domain model specifications per unit. For this, we need to define the boundaries of the unit. These boundaries are represented by nodes that are shared by neighbouring units. An example for the stapler unit is shown in Figure 4.4. The keyword **input** indicates that node handover2 is an input for the stapler unit. It is also an output of the printer unit. To specify outputs, the keyword **output** is used.

```
node handover2 type input
```

Figure 4.4: Input node specification

4.1.3 Allocation model

To communicate timing constraints upstream, the Allocation model needs additional concepts. These concepts specify the constraints from downstream units. This is done using timing constraints for the arrival of pieces of material at output nodes, which are input nodes of downstream units.

An example of such timing constraints is shown in Figure 4.5. Some constraints are between arrivals at consecutive pieces of material, e.g. the constraint between the arrivals of sheetA4_1 and sheetA4_2. Others are between non-consecutive nodes, e.g. the constraint between the arrivals of sheetA4_1 and sheetA3_1.

```
between sheetA4_1 @ handover2 and sheetA4_2 @ handover2 delay 1.0 s
between sheetA4_1 @ handover2 and sheetA3_1 @ handover2 delay 62.0 s
between sheetA4_1 @ handover2 and sheetA3_2 @ handover2 delay 63.5 s
between sheetA4_2 @ handover2 and sheetA3_1 @ handover2 delay 61.0 s
between sheetA4_2 @ handover2 and sheetA3_2 @ handover2 delay 62.5 s
between sheetA3_1 @ handover2 and sheetA3_2 @ handover2 delay 1.5 s
```

Figure 4.5: Output node constraint specification

4.2 Constraint graph generation

From a modular specification as introduced in Section 4.1, we also generate constraint graphs. The generated modular constraint graphs are similar to the generated monolithic constraint graphs as explained in Chapter 3. Figure 4.6 shows the generated constraint graph for the stapler unit of the running example.

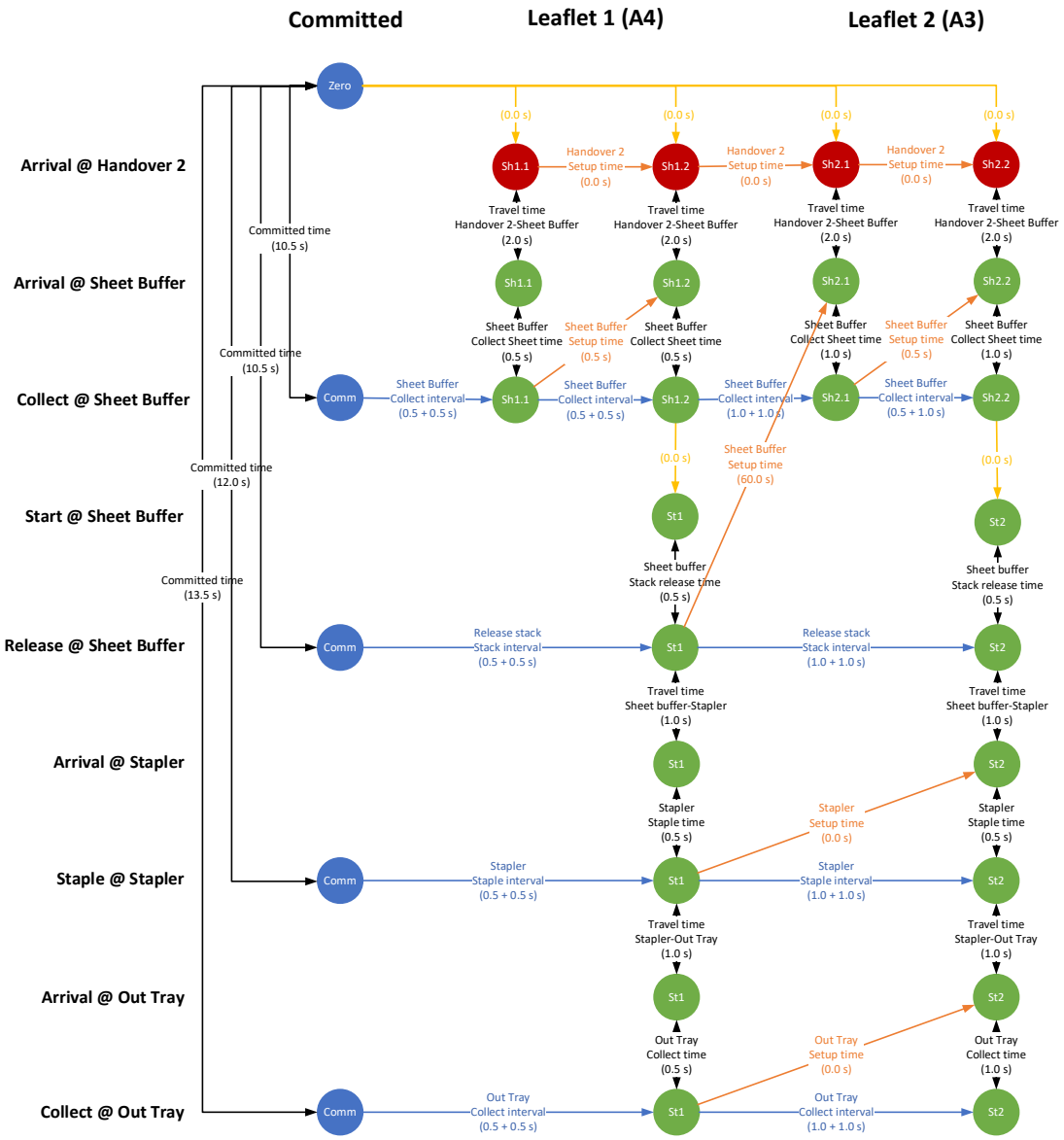

Figure 4.6: Generated modular constraint graph

The modular constraint graph structure differs from the monolithic constraint graphs in several ways:

- There is a (blue) zero node, which represents an event occurring at time zero.
- There are (yellow) unidirectional arcs from the zero node to every start node of a subpart path. These arcs have weight zero.
- There are (blue) nodes representing the events that having been committed.
- There are (black) bidirectional arcs between the zero node and the committed event nodes. The weights of these arcs equal the times at which the events occur.
- There are (blue) unidirectional arcs from the committed event nodes to the first uncommitted arrival and operation nodes. The weight of these arcs is calculated in the same way as the weights between uncommitted operation and arrival events.

Note that the example in Figure 4.6 does not include any committed arrival event nodes and the corresponding arcs to uncommitted arrival nodes.

4.3 Constraint graph analysis

Using a constraint graph as explained in Section 4.2, we can create a schedule by computing the longest paths from the zero node to all events in the constraint graph. This is identical to the (monolithic) analysis explained in Chapter 3.

The modular analysis that we aim for involves computing the constraints between the input nodes of a constraint graph. These are the red nodes in the constraint graph in Figure 4.6. These are computed by applying the Bellman-Ford algorithm [3] for the zero node and every input node. The result is a matrix containing the lengths of the longest paths between all pairs of input nodes. The matrix for the constraint graph in Figure 4.6 is shown in Table 4.1.

Table 4.1: Longest paths lengths between input nodes

	Zero	Sheet 1.1	Sheet 1.2	Sheet 2.1	Sheet 2.2
Zero	0.0	8.5	9.5	70.5	72.0
Sheet 1.1	-∞	0.0	1.0	62.0	63.5
Sheet 1.2	-∞	-∞	0.0	61.0	62.5
Sheet 2.1	-∞	-∞	-∞	0.0	1.5
Sheet 2.2	-∞	-∞	-∞	-∞	0

The cells of the table contain the minimum duration between arrival events at the input node. For example, the delay between the second sheet of the first leaflet and the first sheet of the second leaflet is at least 61 seconds. This long delay is because of the setup time needed between A4 sheets and A3 sheets.

The value -∞ indicates that there is no constraint between two events. Table 4.1 confirms that a sheet's arrival timing does not depend on future sheets' arrivals: all values below the matrix' diagonal equal -∞.

This analysis collapses all constraints in a constraint graph into the constraints between its input nodes. The input nodes of one unit are the output nodes of another unit. The constraints between these output nodes can be included in the constraint graph for the upstream unit. An example of this inclusion is shown using the Allocation language in Figure 4.5. Note that the timing of Figure 4.5 corresponds to that in Table 4.1.

5 Domain model usage

This chapter describes how to get started with the domain model. Section 5.1 describes the installation needed to get started. Section 5.2 describes the actions needed before one can start modelling and the actual modelling. Section 5.3 explains the model transformations that are performed when creating a domain model instance.

5.1 Installation

This section describes the steps needed to install the software needed. Section 5.1.1 explains how to clone the GitLab environment of the domain model. Section 5.1.2 describes how to install the software needed to obtain a working environment. Section 5.1.3 explains how to prepare the working environment for modelling.

5.1.1 Repository

Clone the repository https://ci.tno.nl/gitlab/aims1/aims-production-line-domain-model.git in a folder of choice, e.g. using TortoiseGit.[9]

5.1.2 Installation

The following steps are to be taken to install the software needed to set up a working environment:

- Download and install the latest *Java* 17 version from https://adoptium.net/.
- Download *Eclipse IDE for Java and DSL developers* version 2023-09 from https://www.eclipse.org/downloads/packages/ and unzip it in a folder of choice.[10]
- Download and install *GraphViz* from https://graphviz.org/ or download and install *Microsoft Visual Studio Code* from https://code.visualstudio.com/ and install its *Graphviz (dot) language support for Visual Studio Code* plugin.
- Download *JGraphT* from http://prdownloads.sourceforge.net/jgrapht/jgrapht-1.5.2.zip?download. Copy the jar files in this zip file to the Languages/nl.tno.esi.aims.allocation/lib folder in the cloned repository.

5.1.3 First use

To create a working environment, one needs to generate Java classes from the Xtext syntaxes of the domain model. This involves the following steps:

- Start Eclipse by double clicking the eclipse.exe executable in the folder where you unzipped the Eclipse IDE.
- When asked, specify a workspace folder; next select "*Launch*".
- From the menu bar, select "File -> Import...". The "Import Dialog" opens.
- Select "*General -> Existing Projects into Workspace*" and click "*Next*".
- Click the "*Browse...*" button and select the folder "Languages" in the cloned repository and press "*OK*".
- Click the "*Select All*" button and select "*Finish*".

[9] TortoiseGit can be downloaded from https://tortoisesvn.net/.
[10] A later version of the Eclipse IDE will probably also work.

- In the Eclipse package explorer, navigate to "*nl.tno.esi.aims.material -> src -> nl.tno.esi.aims.material*".
- Right-click "*material.mwe2*" and select "*Run As -> MWE2 Workflow*". Messages stating that the project contains errors may be ignored.
- In the Eclipse package explorer, navigate to "*nl.tno.esi.aims.equipment -> src -> nl.tno.esi.aims.equipment*".
- Right-click "*equipment.mwe2*" and select "*Run As -> MWE2 Workflow*". Messages stating that the project contains errors may be ignored.
- In the Eclipse package explorer, navigate to "*nl.tno.esi.aims.job -> src -> nl.tno.esi.aims.job*"
- Right-click "*job.mwe2*" and select "*Run As -> MWE2 Workflow*". Messages stating that the project contains errors may be ignored.
- In the Eclipse package explorer, navigate to "*nl.tno.esi.aims.allocation -> src -> nl.tno.esi.aims.allocation*".
- Right-click "*allocation.mwe2*" and select "*Run As -> MWE2 Workflow*". Messages stating that the project contains errors may be ignored. After this step, the project should not have any errors anymore.
- In the Eclipse main menu select "*Run -> Run Configurations...*". The "*Run Configurations*" dialog opens.
- In the list on the left, right-click "*Eclipse Application*" and select "*New*".
- In the "name" text box type a name for this configuration.
- Click the "*Apply*" button.
- Click the "*Close*" button.

5.2 Modelling

This section explains how one can create instances of the domain model. Section 5.2.1 explains how to open the modelling environment. Section 5.2.2 describes how to import existing project into the Eclipse modelling environment. Section 5.2.3 explains how to create a new project and how the create language instances within the project.

5.2.1 Starting

The following steps should be taken to start the modelling environment:

- From the Eclipse menu bar select "*Run -> Run Configurations...*"; the "*Run Configurations*" dialog opens.
- Select the run configuration that you created in section "*Run configuration*".
- Click the "*Run*" button.

After the steps, one can import and adopt existing projects (see Section 5.2.2) and create new projects (see Section 5.2.3).

5.2.2 Existing project

To import existing modelling projects into the modelling environment, the following steps can be taken:

- From the Eclipse menu bar, select "*File -> Import...*". The "*Import dialog*" opens.
- Select the "*General -> Existing Projects into Workspace*" import wizard and click "*Next*".
- Click the "*Browse...*" button and select the folder "*Instances*" in the cloned repository and click "*OK*".
- Click "*Select All*".
- Click "*Finish*".

After these steps, the project is visible in the project explorer. One can adapt the language instances in the imported projects by double-clicking on the relevant file and making changes in the text editor that will open.

5.2.3 New project

To create a new project with a domain model instance, one should perform the following actions:

- From the menu-bar select "*File -> New -> Project...*". The "*New Project*" dialog opens.
- Under "*Wizards*" select "*General -> Project*" and click "*Next*".
- In the "*Project name*" text box enter the name for your project.
- Click the "*Finish*" button.

The following steps are used to create a language instance in the newly created project. These must be repeated for every file.

- Right-click the newly created project folder and select "*New -> File...*". The "*New File*" dialog opens.
- In the "*File name*" text box enter a file name with an extension dependent on which kind of model you want to make, either *.material*, *.equipment*, *.job* or *.allocation*.
- Click the "*Finish*" button. If the "*Configure Xtext*" dialog pops up, click the "*yes*" button.
- Edit the newly created file in the text editor and save the file afterwards.

Editing a language instance file may trigger one or more model transformations. These are explained in Section 5.3.

5.3 Model transformations

After saving a (valid) language instance, one may automatically trigger one or more transformations from the language instance to another format. These transformations are explained in this section. Section 5.3.1 explains the generation of a visual representation of an Equipment model. Section 5.3.2 explains the generation of a visual representation of a constraint graph corresponding to an Allocation model. In Section 5.3.3, we describe the different types of constraint graph analyses that are performed on Allocation models and their output files.

5.3.1 Equipment visualisation

After saving a valid Equipment model, a transformation from the Equipment to a GraphViz model is triggered. This transformation shows the nodes and segments of the equipment. The nodes are labelled with their name and their type. The segments are labelled with their duration, length, and velocity. The equipment visualisation file is generated in the corresponding project's *src-gen* folder, and it is named *<equipment name>.dot*, where *<equipment name>* is the name specified in the first line of the Equipment model.

5.3.2 Constraint graph visualisation

From a valid Allocation model (including the underlying Material, Equipment and Job models), we also create a GraphViz model. This GraphViz model contains a visual representation of the constraint graph, which is used for all timing analyses. The constraint graph file is called *<batch name>_<equipment name>_constraintgraph.dot*, where *<batch name>* equals the batch name specified in the Job model and *<equipment name>* is the

name specified in the first line of the Equipment model. The generated constraint graph visualisation is placed in the project's *src-gen* folder.

5.3.3 Constraint analyses

The constraint graph created from a valid Allocation file is used for multiple timing analyses. The results of these analyses are written in CSV files in the project's *src-gen*. The following CSV files are written:

- File *<batch name>_<equipment name>_schedule.csv*, where *<batch name>* equals the batch name specified in the Job model and *<equipment name>* is the name specified in the first line of the Equipment model, contains the schedule in which each event occurs as early as possible. The timing of the event equals the length of the longest path from the constraint graph's zero node. This schedule is the results of the analysis explained in Chapter 3.
- File *<batch name>_<equipment name>_inputpathlength.csv*, where *<batch name>* equals the batch name specified in the Job model and *<equipment name>* is the name specified in the first line of the Equipment model, contains the aggregated timing constraints between the input nodes of the equipment. This is the outcome of the analysis presented in Chapter 4.

6 Conclusion

This chapter provides a summary of this report in Section 6.1 and it lists possible directions for future work in Section 6.2.

6.1 Summary

In this report, we have described a domain model for the timing behaviour of production systems consisting of multiple connected units. The domain model consists of four domain-specific languages:

- The Material language specifies the materials to be handled by a production system and the operations that can be performed on these materials.
- The Equipment language describes the topology of a production system in terms of the processing stations and their connections.
- The Job language specifies the batch of work to be performed by the production system in terms of products, parts and subparts.
- The Allocation language specifies how the work is assigned to the available processing stations and the order in which the work is to be executed.

The report describes a transformation from instances of the domain model to constraint graphs, a constraint model involving relative timing constraints between events. These constraint graphs can be used to derive a schedule for the allocated work. Constraint graph analysis can also be used for a modular timing analysis: the constraints of one system unit are combined into constraints for the arrivals of work at the unit's inputs and these constraints are added to the constraints of the upstream unit as constraints on the arrival at the upstream unit's outputs.

6.2 Directions for future work

This section lists some directions to extend (the expressiveness of) the domain model or to extend its analysis capabilities beyond the current scheduling/timing prediction.

6.2.1 Disassembly operations

The domain model introduced in this report has been applied to several production systems covering assembly operations. In the running example, sheets of paper become a leaflet. Production systems may also have disassembly operations. For instance, a wooden log may be cut into planks and a large sheet of paper may be cut into business cards. A Job model (implicitly) assumes that a piece of material does not fundamentally change. This is not true for inline disassembly, i.e. disassembly while material is moving: the disassembled pieces of material may follow separate paths. The Equipment language's input and output nodes are used to model subpart paths with a moving start or end. A similar principle may be needed to capture the start and end of material paths involving inline disassembly.

6.2.2 Language parameterisation

Equipment models may be very verbose. If an equipment node can handle many different material instances, the timing of the node's operations must be specified for each material instance separately. Often the timing of an operation depends on the characteristics of the material on which the operation is performed. In such cases, operation durations can be specified in terms of the parameters of a material type instead of listing the durations for each instance of that material type. This could greatly shorten an Equipment model.

Moreover, operations may depend on parameters that are not related to a material instance. These could be introduced in the Equipment model to increase expressiveness of the domain model.

These types of parameterisation require changes in the Material language and the Equipment language. Both the Material and the Equipment language need to be extended with user-defined (material or operation) properties and the Equipment language must be extended with expressions using these properties.

6.2.3 Language element reuse

The Job language is used to specify the operations that need to be performed on pieces of material. Such a list must be specified for every subpart separately. Similarly, an Allocation model defines a path to each individual subpart. It is however not uncommon that multiple subparts of one part require the same operations and even must follow the same path to realise a certain production intent. For instance, this holds for the running example introduced in Chapter 2. Job and Allocation models would be much shorter if one could define the operations and path for a sequence of identical subparts. Job and Allocation models can also be shortened by defining several identical products or identical parts at once instead defining them individually.

The same applies to Equipment models. Production systems may have multiple instances of the same machine. Instead of defining each machine individually, the Equipment language could be extended by a machine template which can be instantiated multiple times.

6.2.4 Part/subpart dependencies

The Job language defines the work to be executed in terms of operations to be performed on subparts. As explained in Chapter 2, the Allocation language does not provide any support to check whether an allocation of a job matches a specifier's intent. To allow such checks, this intent should be captured. A first step in this direction would be to define dependencies between parts and subparts. For the running example, individual sheets of paper are collected and they continue as a stack of sheets. This could be introduced to the Job language by specifying the subparts of a composite piece of material.

6.2.5 Node (cluster) capacities

The constraint graph generation has several assumptions. These main assumptions are the following:

- There is at most one piece of material being handled at an equipment node.
- The number of pieces of material that can be stored at an (internal) equipment node is unbounded.

- There may be multiple pieces of material on an equipment segment, but they do not overtake each other.

These assumptions may be very restrictive. There may be equipment nodes that can handle multiple pieces of material simultaneously. Moreover, any equipment node has a finite storage capacity. These capacities could be added to the Equipment language by introducing operation and storage capacities per node.

Adding these capacities will significantly influence the constraint graph construction. For instance, an equipment node that can handle two pieces of material simultaneously can be modelled by a constraint between an arrival of one piece of material and the arrival of the piece of material after the next. This means that constraints are needed between non-consecutive pieces of material. This also implies that the state of the equipment can no longer be captured by the last material leaving/last operation finished: a history of multiple arrivals/operations is needed.

6.2.6 Constraint graph analysis speedup

Recently, a single source shortest path algorithm has been developed with an expected near-linear time computational complexity [4] [5]. Using this algorithm instead of the quadratic Bellman-Ford algorithm could greatly improve the scalability of the timing analysis using constraint graphs. Whether the new algorithm indeed improves scalability is a topic for follow-up research.

6.2.7 Loosely coupled systems

Not all equipment in a manufacturing facility is mechanically coupled. Some pieces of equipment are decoupled by buffers and manual or AGV-based transportation. With changes to the timing specification in the Equipment language, such loosely coupled systems can also be described by the domain model.

However, the constraint graph analyses presented in this report are not suited for the full facility. Especially, the no-overtake assumption is too restrictive for loosely coupled transportation. Instead, one can generate a discrete event simulation (DES) model from a specification of a loosely coupled system.

If a production facility consists of a mixture of loosely and tightly coupled systems, a hybrid model consisting of a DES model and constraint graph models can be generated. The DES model sends material arrival events to the inputs of constraint graph models, and the latter send material arrival events at their output node to the DES model. Here the assumption that a piece of material can be allocated in a tightly coupled system without having to consider future materials is an important strength of the constraint graph analysis.

6.2.8 Optimisation

The domain model described in this report consists of four domain-specific languages. The analyses presented in Chapters 3 and 4 use instances of all languages to create constraint graphs for scheduling/timing prediction of a fully specified scenario. A partial specification can be used as a basis for optimisation:

- Given a specification of the material, the equipment, the job and the allocation, one can optimise the order in which products should be allocated.

- Given a specification of the material, the equipment, and the job, one can find the optimal allocation of products to equipment. This should consider the limitations addressed in Section 6.2.4 and preferably have dealt with them before going for this kind of optimisation.

Ideally, one would like to generate optimum Job and Allocation specifications from a specification of the material and the equipment and a specification of the products to be created. This is out of scope for the languages described in Chapter 2; these do not capture sufficient information about the functional aspects of operations to derive recipes to create desired products. Such recipes correspond to sequences of operations for the products' parts and subparts. A first step in this direction is possible by introducing available recipes per product; then optimisation involves selecting recipes for all products and allocating them to the equipment.

Constraint graph analysis, possibly combined with simulation, provides a prediction of the timing behaviour of a production system. Other formalisms are required for optimisation. The most promising concept seems constraint programming, which can capture all aspects of (tightly and loosely coupled) manufacturing systems.

6.2.9 Gantt chart visualisation

The analyses presented in Chapters 3 and 4 produce a schedule, i.e. an assignment of a timestamp to every event. This schedule is represented as a CSV file, which does not (directly) provide insight in the schedule's timing. To allow this insight directly, we propose an additional transformation: from an Allocation specification and generated schedule, one can generate a Gantt chart which shows the transports and operations of all subparts over time and the dependencies between these actions.

References

[1] B. Kienhuis, E. Deprettere, K. Vissers en P. Van Der Wolf, „An approach for quantitative analysis of application-specific dataflow architectures," in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, Zurich, 1997.

[2] S. E. Elmaghraby en J. Kamburowski, „The Analysis of Activity Networks under Generalized Precedence Relations (GPRs)," *Management Science,* vol. 38, nr. 9, pp. 1245-1263, 1992.

[3] R. Bellman, „On a routing problem," *Quarterly of Applied Mathematics,* vol. 16, pp. 87-90, 1958.

[4] A. Bernstein, D. Nonangkai en C. Wulff-Nilsen, „Negative-Weight Single-Source Short Paths in Near-Linear Time," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, Denver, CO, 2022.

[5] K. Bringmann, A. Cassis en N. Fischer, „Negative-Weight Single-Source Shortest Paths in Near-Linear Time: Now Faster!," *arXiv,* vol. 2304.05279, 2023.

Appendix A

Domain model language syntax and validation

In Chapter 2, we have introduced the languages of the domain model using an example. This appendix shows the underlying syntax of the languages and the validation rules to restrict the allowed language instances.

A.1 Material language

The syntax of the Material language is shown in Figure A.1. The white nodes in the syntax graph represent the language's keywords. The grey nodes in this syntax graph correspond to the language's non-terminals and terminals. Each non-terminal has a corresponding part in the syntax graph.

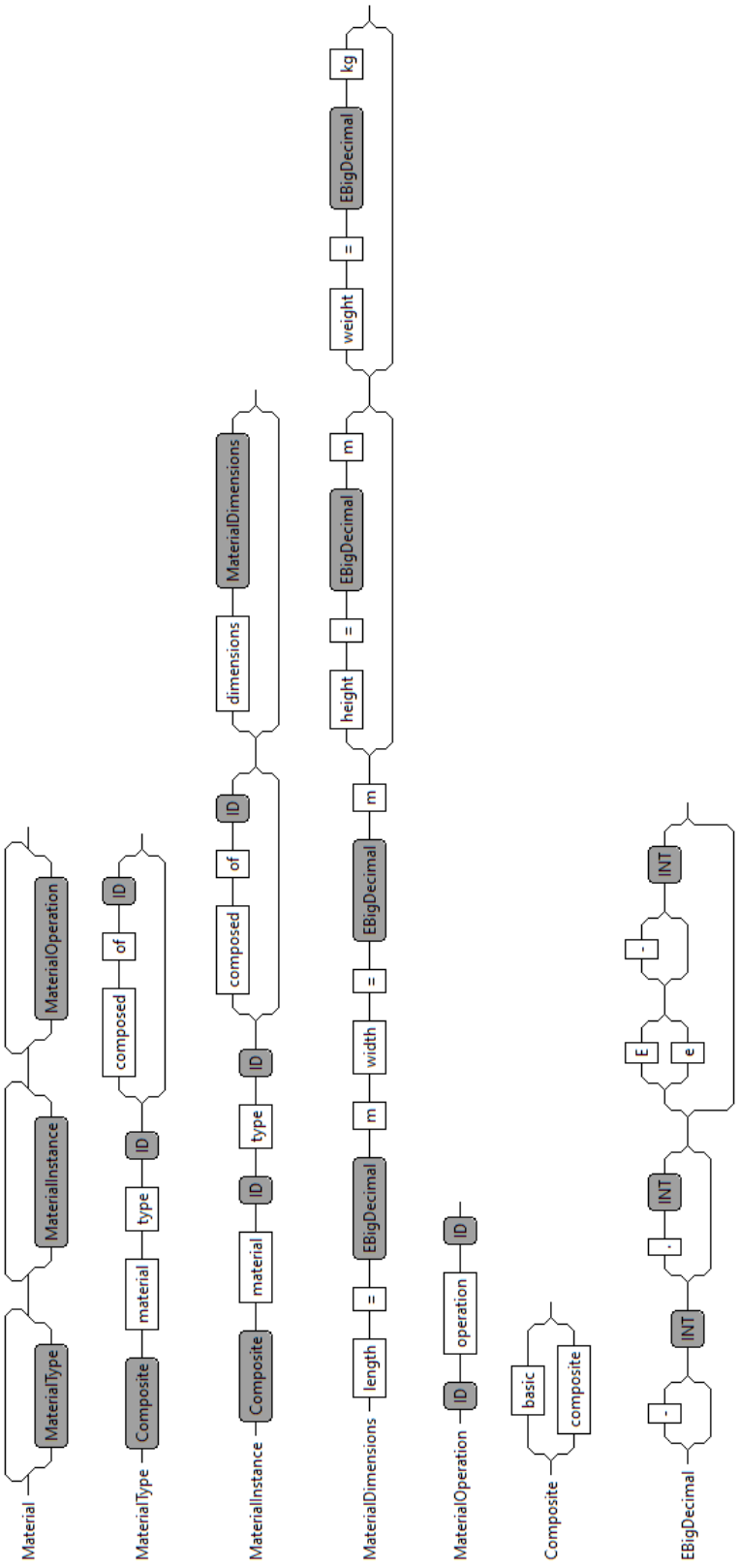

Figure A.1: Material language syntax

The syntax shown in Figure A.1 allow undesirable material definitions. The following validation rules have been implemented to avoid these:

- Every material type has a unique name.
- Every material instance has a unique name.
- Every operation has a unique name.
- The dimensions of every material instance are positive.
- Basic material types do not have composite material types.
- Composite material types have composite material type.
- Basic material instances do not have composite material instances.
- Composite material instances have composite material instances.

A.2 Equipment language

The syntax of the Equipment language is shown in Figure A.2.

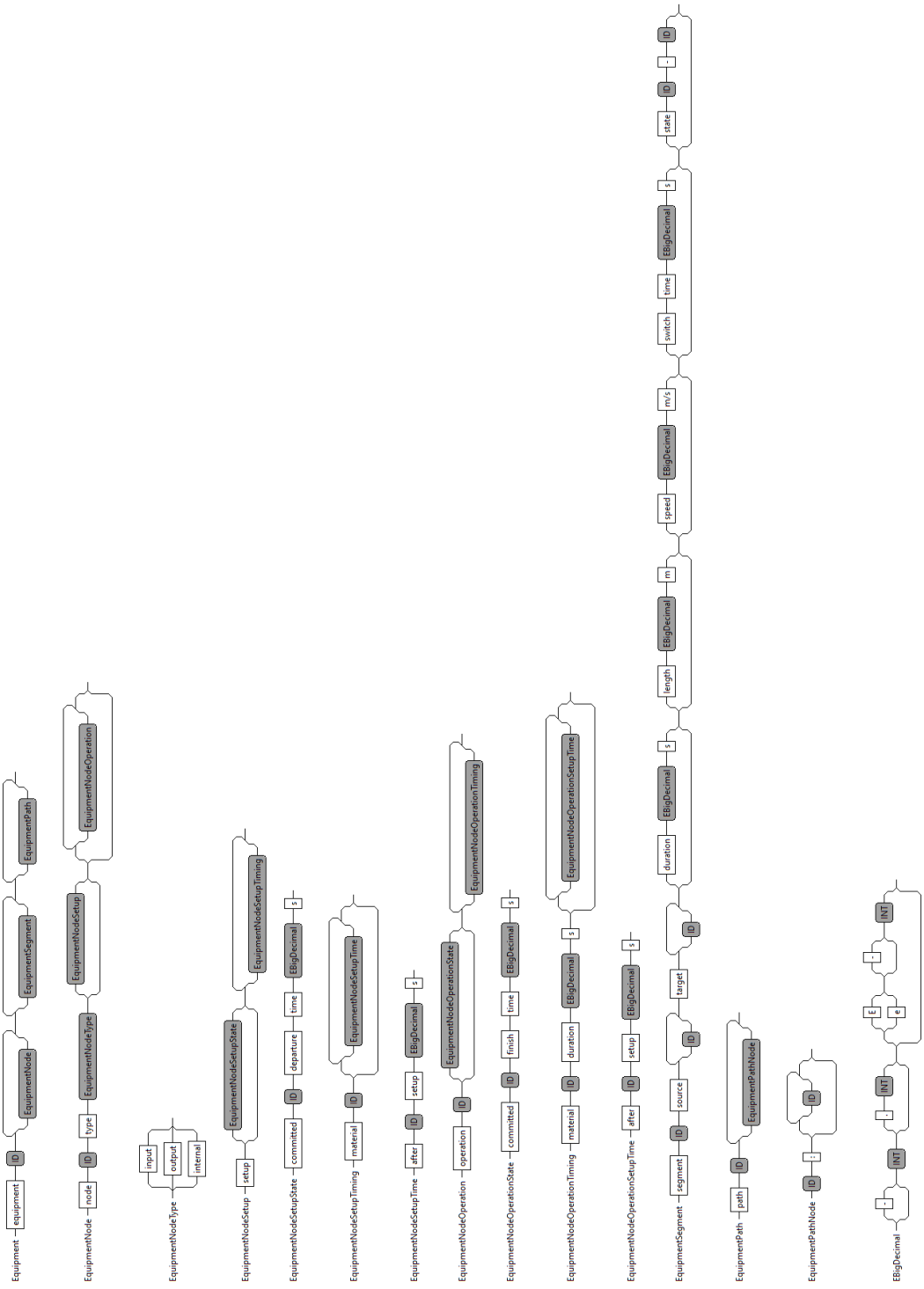

Figure A.2: Equipment language syntax

The following validation rules forbid Equipment specifications that the syntax in Figure A.2 allows:

- Each node has a unique name.
- Each segment has a unique name.

- Each path has a unique name.
- The state of the equipment does not have negative timestamps.
- All operation durations are positive.
- All switch durations are positive.
- All segment durations are non-negative.
- All setup delays are non-negative.
- Arrival setup timing is defined at most once for each pair of material instances.
- Operation setup timing is defined at most once for each pair of material instances.
- Switch states are only defined for switch segments.
- Switch durations are only defined for switch segments.
- The duration of a segment must match its length and speed (if defined).
- Switch segments do not have duplicate source nodes.
- Switch segments do not have duplicate target nodes.
- There are no segments from a node to itself.
- Input nodes do not have incoming segments.
- Output nodes do not have outgoing segments.
- There is at most one segment between every pair of nodes.
- Every path is connected.
- The operations on a path can be performed by the specified nodes.
- The equipment is fully connected.

A.3 Job language

The syntax of the Job language is shown in Figure A.3.

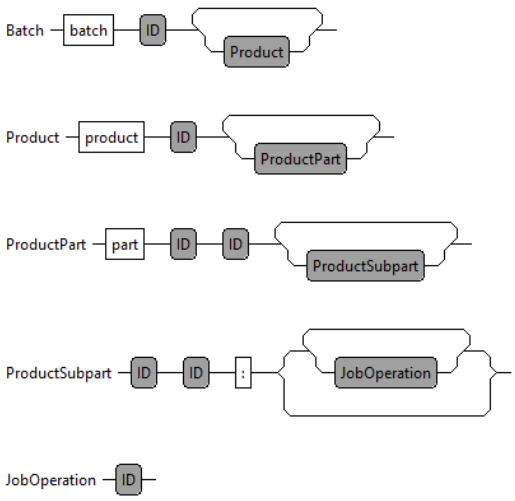

Figure A.3: Job language syntax

In addition, the following validation rules have been implemented for Job models:

- Each product has a unique name.
- Each part of a production has a unique name (within the scope of the product).
- Each subpart of a part has a unique name (within the scope of the part).

A.4 Allocation language

The syntax of the Allocation language is shown in Figure A.4.

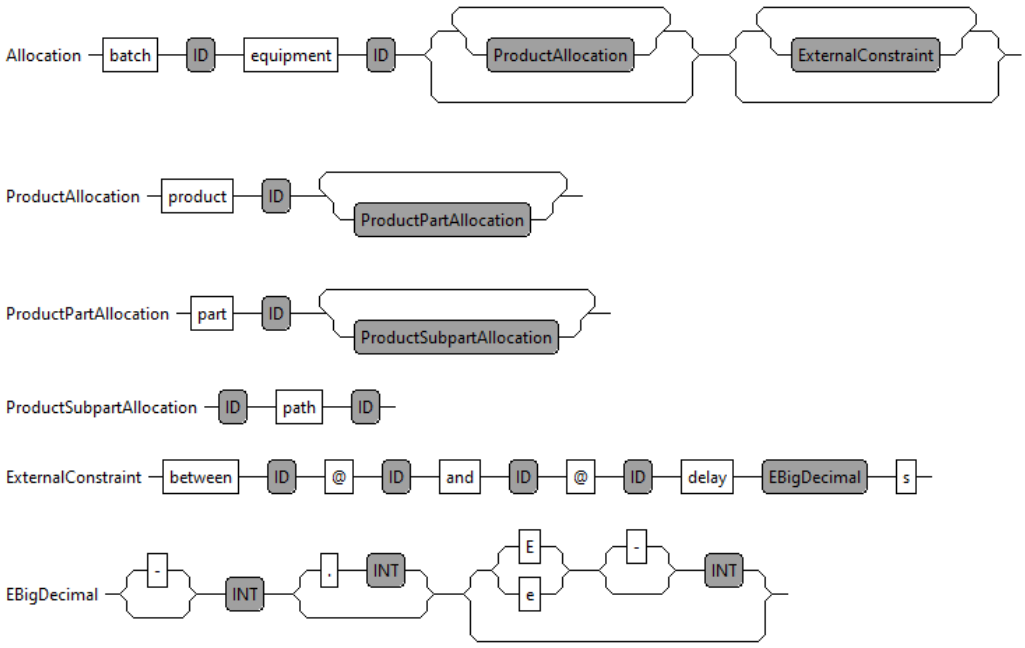

Figure A.4: Allocation language syntax

In addition, the following validation rules have been implemented for Allocation models:

- Every product is allocated at most once.[11]
- If a product is allocated, all its parts and subparts are allocated exactly once, and their order matches the order in the corresponding Job model.
- The equipment path to which a subpart is allocated provides the operations specified in the Job model in the order specified in the Job model.
- Both events of an external constraint correspond to an arrival of a subpart at an output node.
- There are no externals constraints between an event and itself.
- There is at most one external constraint between a pair of events.
- The value of an external constraint may not be negative (and should be positive).

[11] It is allowed to not allocate all products.

ICT, Strategy & Policy

High Tech Campus 25
5656 AE Eindhoven
www.tno.nl

TNO innovation for life