A handbook supporting model-driven software development (a case study)

Jelena Marincic, Angelika Mader, Roel Wieringa

March 26, 2009

Contents

1 Introduction

The systems in the scope of our research in MOCA project [4] are embedded control systems. To prove their correctness one can use models for formal verification. Researchers have been investing a lot of effort in formal languages and tools development. However, not much attention was directed to the process of design and construction of these models. This process is non-formal, creative and rational at the same time, and not easy or useful to formalize.

Our goal is to extract repeatable, generalizable, rational elements of the model design process. Those elements constitute a method we propose, for systematic model construction and verification.

We started our work with literature study and a small but non-trivial modelling example we performed in our lab [3]. The result is the first version of our modelling method. Its ingredients are as follows.

- Describing the plant, separating plant and control early in the design of the entire system model
- Non-monotonic refinement
- Collection of assumptions
- Steps to perform as modelling guidelines

We described the modelling example in detail, as well as the first method version in technical reports [3] [2].

The next step was to validate (and improve) our method on a real life system. This report describes the case study [1] we performed in a company that produces embedded systems of complex mechanics and software.

[1] What we really performed was action research, but in computer science community often the term 'case study' is used for both case study and action research.

1.1 Case study protocol

The high level structure of our case study research is as follows.

- Problem investigation
  - Stakeholders
  - Problematic phenomena
    - interviews
  - Goals (including constraints)
- Solution specification
  - Our method, customized to the company
- Solution validation
  - Interviews
- Solution implementation
  - Use our method
- Implementation evaluation
  - Focus groups

After identifying problems, we designed the solution - our method accustomed to the actual problem. After that, we validated the solution through interviews with the stakeholders. We will describe each of these steps in details in the sections that follow.

1.2 Neopost corporation

Neopost [5] develops, produces and distributes different types of mailroom equipment and document systems. Its customers are companies that send a lot of paper mail on a daily basis, like insurance companies, post offices, banks etc. Folding papers, putting documents into envelopes, closing, addressing and weighing them, and paying the right amount for stamps can be automated using document systems. Some of these systems are shown on Fig. 1. These systems do one or more of the following: folding papers, inserting documents into envelopes, addressing and franking envelopes and other things related to sending mail and manipulating papers and envelopes.

The head office is located in Bagneux, near Paris, but the company is decentralized - each subsidiary individually sets annual targets and is evaluated on its ability to achieve them. This localization makes each subsidiary close to local customers and local national postal authorities. Neopost has reorganized R&D in three centers worldwide: one in Bagneux (France) for franking machines, one in Shelton (USA) for infrastructure and networks and one in

Figure 1: Neopost machines: Intelligent document systems - they fold documents and insert them to envelopes.

Drachten (the Netherlands) for document systems. As for production, there are two worldwide assembly centers: one in Lude (France) for mid-range to high-end franking machines, and one in Drachten for mid-range to high-end document systems (machines that fold and insert documents into envelopes). The assembly of entry-level franking and document systems has been outsourced to Asian countries.

1.3 Neopost Technologies

The organizational structure of Neopost Technologies [6] in Drachten is shown on Fig. 2. This case study was performed in R&D department, so its structure is also shown on the figure.

Another division of R&D department is a division to projects. There are three different types of development projects:

1. Advanced Development: development of new technologies and pioneering concepts.

2. New Products: a platform that develops completely new machines, for which the breakthroughs from Advanced Development provide important added value.

3. Further Development: current products get new features, quality improvements, cost price reductions, etc.

The department leaders are responsible for allocation of the teams to different projects. Project leaders are responsible for projects implementation. The multidisciplinary project teams are put together using a project matrix (Fig. 3). The heads of departments and leaders of projects do not report to each other. They have the same level of responsibility and they both are responsible to the CTO in the head office.

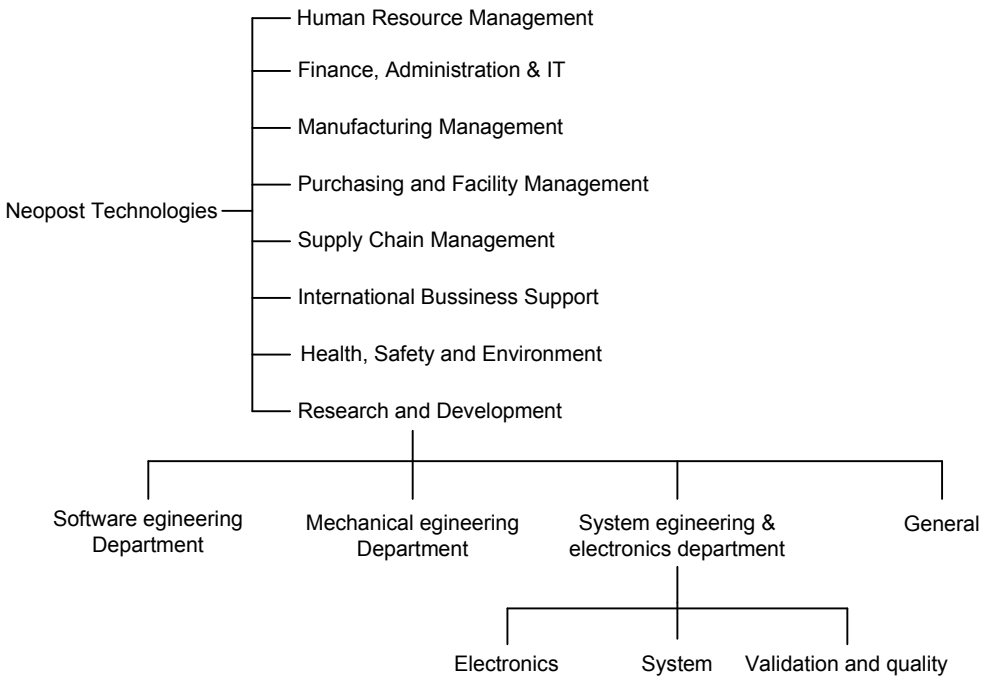


Figure 2: The organogram of Neopost Technologies and the R&D department.

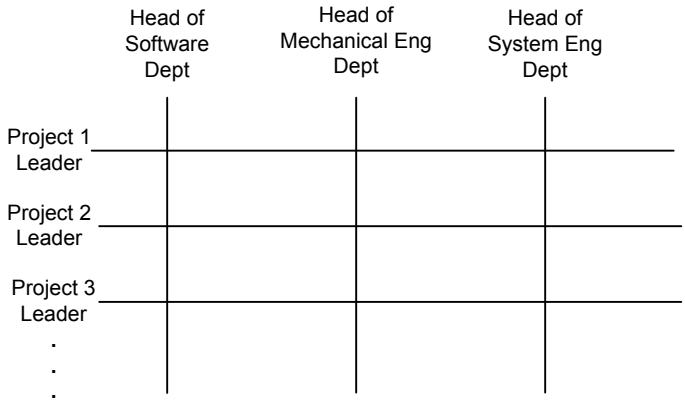


Figure 3: Matrix hierarchy - project leaders and heads of departments have the same level of responsibility and report to the head office CTO.

2 Problem Investigation

2.1 Project cycles

Each project for a new product development (type 2 mentioned in Par. 1.2) has the following cycles: requirements identification, building or improving the mechanics and software, testing and verifying the system. The diagram on Fig. 4 shows three main phases: feasibility study, after which a prototype should be ready, phase 1 after which a product for mass production should be ready, phase 2 after which a product is maintained.

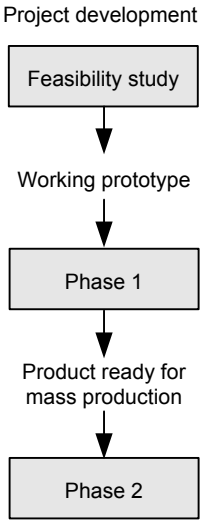


Figure 4: Phases of a new product development.

The R&D department in Neopost is divided into the groups as shown on Fig. 2.

Problem 1: Communication between mechanical and software departments

The separation to software and mechanical engineering is on one hand natural and necessary. On the other hand this can lead to the well known problem of hardware and software being developed separately. People on one 'side' do not have knowledge about the issues from the other side that are also relevant from their own development. This costs a lot of time in the integration phase, when software and mechanics are put together.

2.2 The timeline of mechanics and software development

Mechanics is the mechanical part of the machine.The table on Fig.5 shows the development phases. Total development time and time to finalize every phase varies from project to project and it depends on many factors, like for example the size and type of the machines being build.

The goal of the new project development is to set up a factory. This means two things: the right tooling for the production exists, and the plant is tuned and ready for a mass production.

The mechanical department starts with 3D drawings that are basis for a prototype. Their 3D model is not perfect, so after the prototype is ready, they make improvements of the model and tune the prototype. The next thing is to make tooling for production, e.g. molds. After constructing a 2D model, the molding is produced and improved. Once the tooling is tuned, the mass production can start, so the factory is set.

Mechanical Engineers Activities	Software Engineers Activities
1. Make a 3D drawing 2. Make a prototype 3. Tune the prototype	
4. Make a 2D drawing 5. Make tooling for production (e.g. molds) 6. Make a tooling prototype (2-3 pieces of the plant)	Start developing software
7. Tune 8. Update tooling 9. Produce (100 pieces)	Develop software

Figure 5: Mechanical and software development in the past.

Problem 2: Integration Time

In Neopost, the plant and the software development were isolated from each other. In fact, the software development would start only after the mechanics was developed. Once both the software and the plant were developed, they would be put together and tested. At this late integration phase, it took too much time to fix the software to make the whole system working as required (Fig.6.)

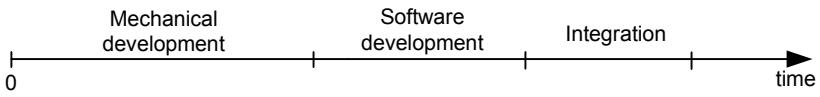


Figure 6: The timeline of the development and the integration, without using the emulator.

2.3 Testing

Once the mechanics is ready, the software can be tested. There are three main types of the tests run.

Testing user scenarios

The user interface offers different tasks to be performed by the machine. An example of this is: For all the documents stacked in the feeder 1, fold each document in 'z' fold and insert it together with a business reply envelope from BRE feeder into envelopes stacked on the envelope feeder. There are a lot of these scenarios and it would take too much time to test them manually. The user scenarios are stored in testing scripts and they are automatically tested.

Regression testing

As software develops, there can be different versions, updates, changes etc. Experience shows that bugs discovered at one point may reappear in later software versions. Therefore, once a bug is located and fixed, tests are run for next software versions to check whether it reappeared.

System imperfections or faults, border conditions

Even though the papers have standard sizes, they differ in sizes by couple of millimeters. How much they can differ from the size the control software expects is determined by doing testing. Also, the maximum speed of the paper, although determined and calculated in advance, has some tolerance. It is tested once the mechanics exists.

2.4 The emulator

Neopost products are typical examples of real-time reactive systems. The software running on them interacts with a user via user interface and controls the plant via sensors and motors. Embedded software controls the plant by sending signals to motors, and monitors the plant via sensors. Testing or verifying only the software means that sensors will not sense anything, since there is no plant and there will be no paper to move or pass along the sensors. To enable concurrent engineering and to shorten the integration time, Neopost developed the mechanics emulator for one of their new products.

To verify the software while the mechanics is still being developed, the model of the mechanics is made. If realized in software it is called a simulator, if realized in hardware it is called an emulator. In both cases the software is 'plugged in' into the model and then the system requirements are tested in the model.

Building such a model enables testing the software and the system requirements and it also encourages the communication between mechanical and software engineers.

Verifying the software with the mechanics model is expected to shorten the integration time significantly as shown on Fig. 7 and Fig. 8.

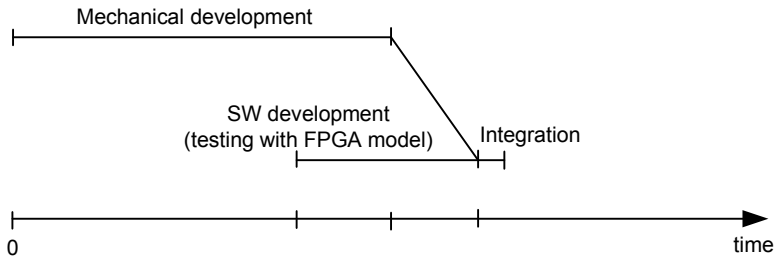


Figure 7: The timeline of development and the integration, when the emulator is used.

Mechanical Engineers Activities	Software Engineers Activities
1. Make a 3D drawing 2. Make a prototype 3. Tune the prototype	Design the software using the emulator (model), update the model Integrate sw and prototype
4. Make a 2D drawing 5. Make tooling for production (e.g. molds) 6. Make a tooling prototype (2-3 pieces of the plant)	Continue developing software and update the model if necessary
7. Tune 8. Update tooling 9. Produce (100 pieces)	Integrate the product and the software Develop software

Figure 8: Mechanical and software development now.

2.5 The system and the model

The system is the new inserting machine being developed. As shown on Fig. 9 it consists of the following components: (1) system controller (implemented on a processor) that contains user interface and forwards user wishes to the embedded controller; (2) the embedded controller (implemented on another processor); (3) sensors and actuators; (4) the plant (the mechanics).

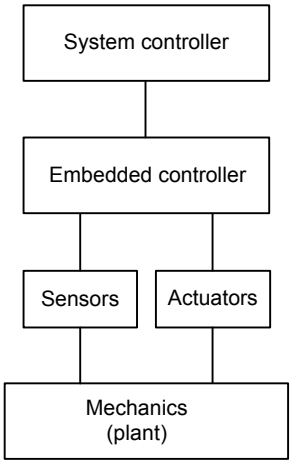


Figure 9: The system architecture

The plant consists of the following modules: feeders for documents, envelopes and business reply envelopes; a collator for collating papers; a folder where papers can be folded in five different ways; a moistener for moistening the envelope flap; an inserter where an envelope is opened, documents inserted into it and finally closed; an exit part which moves filled envelopes to the exit. Figure 10 shows a diagram representing mechanics modules. Arrows on the figure represent the flow of papers and envelopes through the inserter machine.

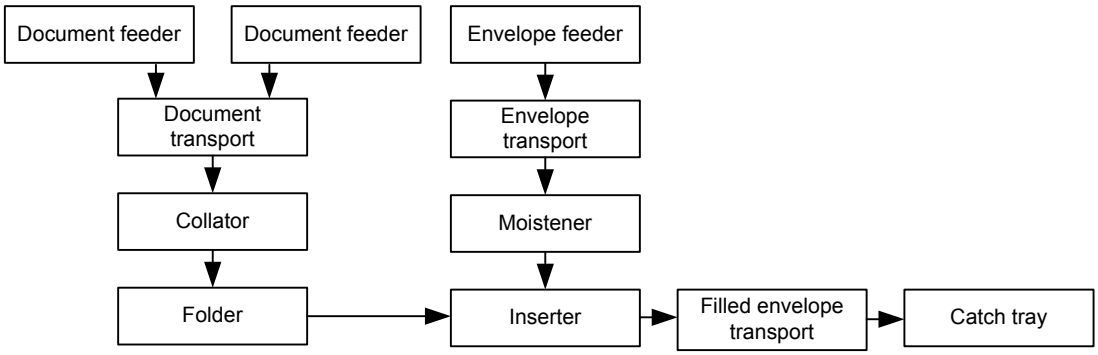


Figure 10: Diagram of the mechanical plant parts and path of paper and envelopes

One AC motor and one DC motor are responsible for the movement of papers and envelopes. Rollers rotate and move papers and envelopes through the system. The rollers are connected to clutches. If a clutch is turned on, it transmits the motor drive to its rollers, which causes them to rotate. If the clutch is off, it disconnects the motor drive from the rollers, so they do

not rotate. Both motors and clutches are actuators. As papers and envelopes move, they pass along sensors. By reading sensors values, the control software observes the position of papers and calculates their length as function of time they spend passing along them and the motor speed. Other relevant details about the system will be given later, for now it is important to understand that the control monitors a number of sensors and sends signals to motors and a number of clutches.

Figure 11 shows the idea of replacing parts of the system (shown on the left side) with their models (shown on the right side). The system control (on the left side) contains user interface through which a user expresses his wishes about how the documents will be handled. The system control translates user wishes to protocol messages and forwards them to the embedded controller. The system controller module is replaced by testing scripts, as shown on the right side of the figure. Testing scripts contain all possible messages expressing all possible user wishes that the user interface offers.

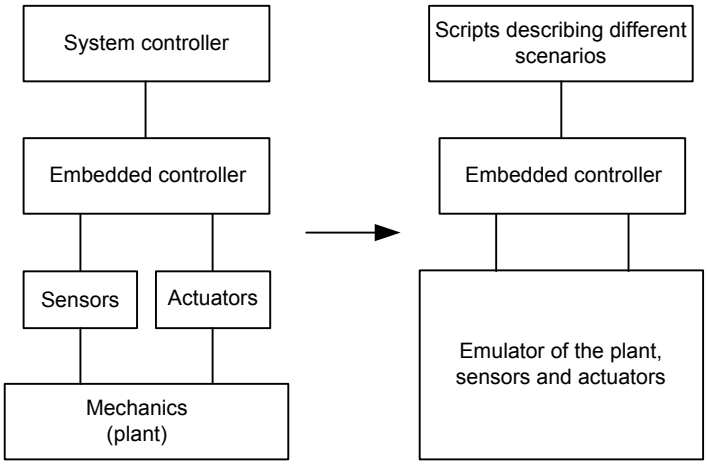


Figure 11: The concept of the emulator.

The mechanical plant (left side on the Fig 11) is represented by the emulator (right side on the Fig 11). The emulator describes the mechanics and papers and envelopes moving through it.

The emulator is a computer controlled device that accepts signals from the embedded controller output (actuators interface) and sends signals to the embedded controller input (sensors interface). These signals have the same behaviour as the signals from the sensors and actuators of the mechanics. The control software does not 'know' whether it is connected to the mechanics or the emulator. (Ideally, it does not know. In practice there are some parameters and variables in the software that are changed for working with the emulator, for example some initializations.)

In the ideal case, the model (i.e. the emulator) would describe all possible

mechanics behaviours, but as we will show later, a subset of possible behaviours is described, with the assumptions on the software behaviour. Deciding what idealisations and abstractions to make is determined by the following factors:

(1) For some parts of the embedded software it is known how it will behave and what it will not do.

(2) Some compromises are made because of limited hardware resources of the emulator, in such a way that the results obtained from the model are not compromised.

(3) Requirements tested and model purpose allow some abstractions and idealizations.

Figure 12 shows the implementation of the idea presented on the Fig. 11.

The emulator is a specialized computer with programmable hardware - a field programmable gate array (FPGA). An FPGA is programmed with a hardware description language (HDL). There are tools that allow programming in other language and that compile it to a HDL. One such tool is the LabView tool in which one can program in G-language, using diagrams. This was the choice of Neopost as software engineers already had experience with LabView and not with HDLs.

With the LabView it is possible to visualize changes of values of variables describing sensors and actuators. Also, the model has parameters which describe paper lengths, distances between papers and some mechanical properties of the plant that can be changed. These can be changed while the model is running, via a LabVIEW diagram.

About FPGA

FPGA is an integrated circuit board with digital inputs and programmable digital outputs. On the board there are logic components and reconfigurable connections between them.

Logic components perform logic functions like for example AND, NOT and XOR. A logic component can also contain memory elements that 'remember' output values for the next one or more time units, so the output is a function of the inputs and previous output values. (An example of such a function is a flip-flop which output y at the time $t = nT(n = 1, 2, 3...)$ is: $y(nT) = NOTy((n-1)T)$).

Logic components outputs are connected to other logic components inputs to realize more complex functions. The connections between logic components are reconfigurable, i.e. reprogrammable. An FPGA can be programmed using either hardware description language or the logic circuit diagrams.

About LabView

LabView is a development environment from National Instruments. LabView uses "G" language, which is a dataflow language. A program in LabView is a

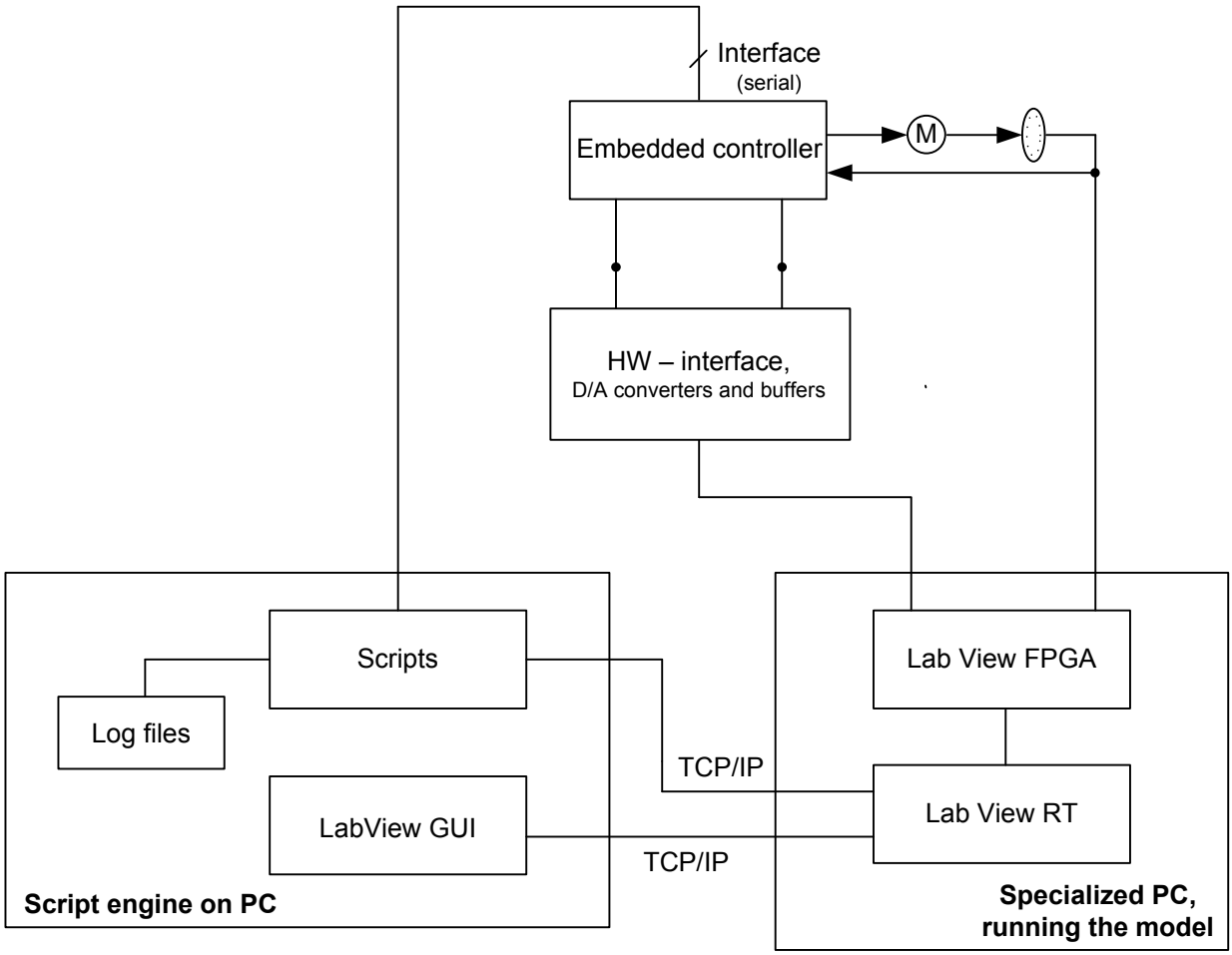


Figure 12: The model

data-flow like diagram. Functional blocks are connected with wires that represent data flow. Fig. 13 shows the snapshot of a LabView diagram.

LabView can compile programs to different targets. The LabView FPGA add-on module compiles the block diagram to VHDL code, which is then compiled to FPGA bit file. National Instruments developed specialized PC with FPGA and LabView FPGA running on it.

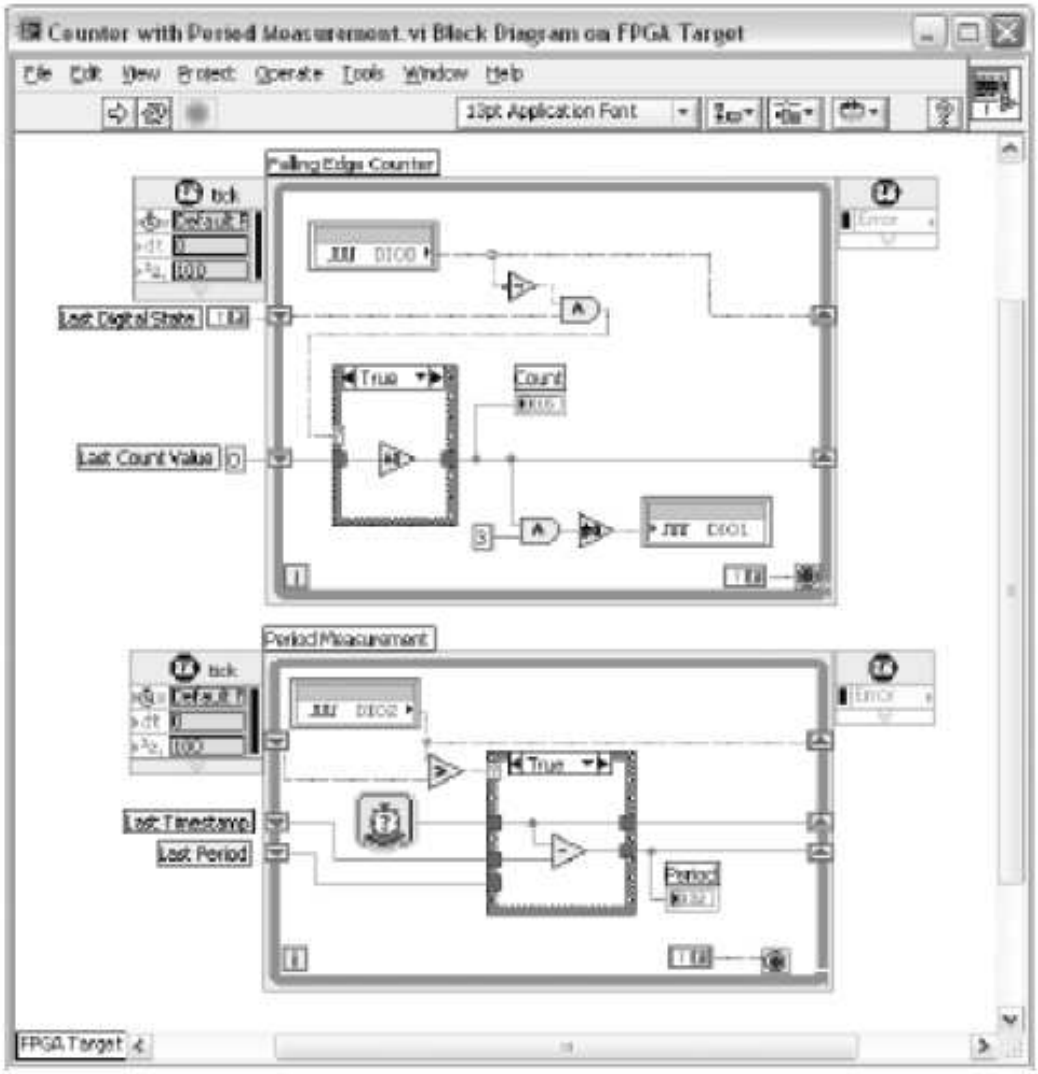


Figure 13: The snapshot of a LabView diagram.

3 Problems analysed in the case study

3.1 Stakeholders

The interested parties or the stakeholders in the emulator (model) development are the following people or job roles:

- Head of the system department
- Head of the software department
- The modeller
- Programmers (developers, testers)
- Mechanical department
- The verification and validation engineer.

The table on Fig 14 shows the goals the stakeholders want to achieve with the model (G), constraints for achieving them (C) and properties of their roles or departments related to the goals (P).

For the heads of departments, as well as project leader the goal is to deliver the new system in the shortest possible time, with as least engineering hours invested as possible. To do this, it is necessary to shorten *time to market*. One of the things to improve is the integration time. They estimated building an emulator as the optimal solution for this problem.

The project leader is focused on finishing project phases on time. Although the heads of departments think about the future projects, too, their first priority are finishing current projects.

Because of the constraints in money, time, people, their knowledge of different tools, hardware resources, the model cannot be 'perfect'. Some compromises are made, but they should not invalidate the results obtained from the model. This means that once the system is ready, additional improvements will have to be done with the software as a result of model's 'imperfection'. So the model has to be good enough to shorten the integration time to a certain extent.

3.2 Problems and goals identified for this case study

Problems mentioned in Section 2 are the problems that had already been solved by constructing emulator. The problems identified for this case study are related to more efficient and systematic construction of the emulator (model) in the future. In interviews with stakeholders we identified the following goals of our action case for them.

- P1: Make the modelling process independent from the modeller.
- P2: The model construction should take as least time as possible.

Stakeholders	Goals (G) and Constraints (C)
Head of System Engineering Dept.	G: Shorten time to market by shortening integration time. G: The model has to be cheap in time, cost and human resources. G: Make modelling faster for future projects. G: Make better models in future projects.
Head of Software Engineering Dept.	G: Shorten time to market by shortening integration time. G: The model has to be cheap in time, cost and human resources; G: Model has to be good enough - to represent the plant so that the results obtained from it are meaningful.
Project leader	G: Finish each phase on time as planned. G: Design the model as soon as possible.
Modeller	G:Make a model of the plant with tools good for the chosen kind of model. C: The plant does not exist yet; C: Modelling some parts would take too many hardware FPGA resources.
Software Developers	G: Develop the software and test it. P: Not experienced in language (VHDL) used for programming hardware.
Validation and verification eng.	G: Test and verify the software. P: Not experienced in language (VHDL) used for programming hardware.
Mechanical dept.	G: Deliver the mechanical part. P: Not concerned about the models used for software verification. P: They are the source of knowledge for the modeller.

Figure 14: Stakeholders, their problems and their constraints

- P3: It would be good to be able to assess the quality of the model, to what extent it describes the system, how perfect it is.
- P4: Evaluate whether it would be better to ask mechanical department to build the model.

The modeller spent two years (not full time) thinking how to make the model, trying out different solutions and choosing the optimal ones. Not only that he learned a lot about the system, but he had a lot of insights what is important to think about, to decide, to take into account etc. before and while modelling. If he is not the person who will make the model next time, someone else will have to spend a lot of time learning things the first modeller already learned.

Therefore, the need for a modelling handbook was identified. This consists of the method we designed, customized to the company's needs.

4 Solution specification - modelling handbook

The modelling handbook consists of the following parts:

- Part I: Questions and steps before and while constructing the model.
- Part II: Facts about models and modelling process that a modeller should be aware of.
- Part III: Re-usable modules, re-usable principles, alternatives that did not work well.
- Part IV: Other modelling concerns.

In this section we will describe each of these parts in detail. Each part talks about different aspects of modelling process and involves different roles. The first part is a collecition of the decisions and steps made while modelling. These we have already identified in earlier work. In this modelling handbook we accustomed them to the problem in hand by giving some of the possible answers to these questions. The second part describes other steps that accompany modelling process, like for example collecting modelling assumptions and testing the model. In the third part we extracted modelling solutions that could be re-used for a new machine model, in form of modelling templates. They are specific to Neopost machines, but we kept them independent of the modelling language used. We described them informally, as state machines. In the forth part we identified issues that are relevant for Neopost machines models, like for example having certain variables as parameters.

4.1 Part I: Questions and Steps

The first part contains questions to answer and steps to perform before and while modelling. They are about different decisions about the model and these decisions involve the following roles: system architect, verification engineer, validation engineer, testing engineer, modeller, and system integrator. Answers to these questions assist in finding out which system properties and aspects should be described in the model and how. We also collected possible answers to these questions. In the future, some new possible answers may arise, but we have now only the questions, the answers relevant for current projects and estimation what in the future possibly might be the answer. We think that having the questions and possible answers is better than having nothing. They shape the process of the model design and make stakeholders aware of different model aspects. The questions are expected to be useful as means to systematically design the model.

The order of the steps (answering the questions) is not as they are listed. They are intertwined, and the answers to them can change in time.

We started from the list of questions we made in our previous work [1], some of them we elaborated into more questions and identified a new one (Q4).

Q1: What will the model be used for? What goals does it have to fulfil?

Answering these questions assist in deciding what parts of the system to describe, how much time to spend on the model, whether to document it and how detailed, whether to make it more understandable and for whom.

The overall and most obvious and highest-level goal is that the model has to support the software design and system verification within the R&D department.

When identifying possible answers to this question, we identified different groups of goals (or subgoals). They also might be seen as different aspects. These aspects have existed as verification aspects of previous projects and they will exist in the future ones.

We found the following aspects (or classes of verification problems): (1) Verification of different parts of the system, (2)Time aspect, (3) Social aspect (4) Economical aspect

(1) Within the verification goal, we identified different subgoals (or answers to the question "What are we doing with the model?")

- Software testing
  - Testing all possible scenarios of user requests and paper lengths
  - Regression testing of software functionality over different software releases
  - Endurance testing
  - Fault tolerance - how will the software react to faults in the mechanics
  - Fault tolerance - how will the software react to some irregularities of the material (e.g. different length of the paper)
  - Testing the impact on the software behaviour after changes in the emulator (model). For example - what will happen if a sensor is added?
- Mechanics verification
  - Testing limitations and limits of the specification
  - Feedback from software engineers to mechanical engineers
- System verification
  - Fault tolerance - how will the software react to faults in the mechanics
  - Fault tolerance - how will the software react to some irregularities (e.g. diff length of the paper)
  - Fault tolerance - reproduce difficult fault scenarios
  - Endurance test
  - Testing limits/limitations
  - Visualization

  - Communication with other models
  - System performance
  - Functional or non-functional requirements

(2) When the model will be used? Within this aspect we identified the following phases relevant for the model's purpose:

- Feasibility study - the mechanics does not exist yet or is not stable
- End of feasibility study - the prototype is there, but a lot of people want to work with it
- Phase 2 - more mechanics are there: regression tests, reproduce difficult fault scenarios - for some things it is easier to test with the model than with the emulator
- Maintenance phase - model not necessary any more.

(3) Social aspect of the model goal. A model can serve for communication with other departments, experts, managers, companies. We identified the following important groups of people with whom it might be necessary to communicate different ideas using the model. They are:

- Colleagues within the department
- Other departments
- Other companies.

Q2: What are the constraints for satisfying model's goals?

Related to model's goals, there are constraints that imply some compromises and trade-offs. Constraints can be:

- Reducing costs,
- Reducing time,
- Reducing number of people involved
- Constraints of the tools, hardware and software resources
- Limited knowledge of tools, software, programming languages

Also, one should be aware of short-term and long-term goals. For example, having the current phase finished as soon as possible is a short-term goal and having the processes in the future projects more efficient is a long-term goal.

Different people's work is evaluated in relation to different goals, some of them are evaluated according to the short-term goals and some of them are evaluated against long-term goals. That is why it is good that all stakeholders of the modelling process answer these questions.

Q3: What is the expected model's lifecycle?

Answering these questions helps deciding the following: If and how detailed to document the model, how much attention to pay on understandability of the model, how much attention to pay on re-usability of the model.

It is possible that the model will be

- Re-used in a future project
- Some model parts will be re-used in a future project
- Maintained over the whole product design and maintenance phase.

Q4: What requirements are verified?

Answers to this question determine what parts and aspects of the system have to be described with the model. One possible classification of the requirements is to functional and non-functional requirements.

Functional requirements (requirements for the system behaviour, says what the system should do). They can, for example, be: "Paper moving through material path, paper folded in the folder the way the user specified etc.", timing requirements etc. Non-functional requirements describe properties of the behavior of the system, for example quality features like security, performance etc.

Another useful classification is to software and system requirements. Software requirements define how the software should behave, so the specifications are for the signals on the software interface. System requirements are the requirements for the whole system and refer to the mechanics parts.

Answering these questions helps deciding what parts and aspects of the system to describe with the model. For example if the functional timing requirement is to be tested with the model, then the speed of the paper is for sure relevant to be described as part of the model.

Q5: Is it necessary to decompose (split) models into more models for different purposes and requirements?

What goes into the model depends on the requirement verified with the model. If there are different purposes and different requirements that we want to test with the emulator, it can happen that the model will contain different system descriptions for different purposes. If, as a result it becomes too complex, it makes sense to split it into more models.

For example we might want to test the software tolerance to different faults in the mechanics. For that we have to identify the parts who can wear out or break and describe them. But if we want to use the model to verify requirements about the paper movement, we do not need a complex model describing much more details.

The disadvantage of this that then more models have to be maintained.

Q6: Who will design and who will use the model?

Model designer can be

- Someone from the company who has already designed the similar model
- Someone from the company familiar with the model, but didn't design the previous model
- Someone from the company not-familiar with the model
- Someone outside the company

Users of the model can be

- Verification and validation engineer
- Developers, software engineers
- Mechatronics engineers
- Testing engineers
- Mechanical engineers

Answering this question helps estimating the time necessary to make the model (if the modeller is the same person who designed the previous model, the time will be shorter), whether to document it and how detailed, to what extent the model has to be modularised and understandable (if the model will be made by more than one modeller, it has to be modularised. The modellers will inevitably have to communicate with each other.)

If the model is used for communication, it has to be understandable to all parties communicating via model, not just the modeller. For example other software engineers use the model to test their software, if there is an error they will have to determine whether an error occured in the software, mechanics or if there is a bug in the model. To determine this, a software engineer will have to understand the model.

Q7: What are the quality criteria for the model?

Some of these criteria we mentioned in the previous papers [1] and one we found while looking at the concrete model (degree of software independence).

These criteria are not independent of each other. There can be a relationship or an inverse relationship between them.

The first quality is **truthfulness**. Model mimics some aspects and behavior of the system. The model is only an abstraction of the system which has to include enough descriptions to make the results obtained from emulation meaningful. It is up to an expert decision to estimate when the model is good enough to serve as the emulator.

For example, the modeller might decide not to model the slip of each roller on the paper path (this means that roller is moving but it does not move a

paper, it slides over it), but he will describe it as a slip on the segment where more than one roller can exist. As a result of a slip, the software calculates the wrong paper length. This model is good enough to test how the control will overcome the problem of not having the correct paper length. The model is good enough for this requirement. If we want to test the possibility of having paper wrinkled on a segment due to different slips of different rollers, we will have to put more details about rollers slips in the model.

The second quality is **degree of software independence**. The more we want to describe the mechanics independently of the software, the model becomes more complex. It is always a trade off.

Both sides in this case study started work on this handbook thinking that maybe the mechanical engineers should be the ones to design the emulator because, after all, it describes the mechanics. The big insight for both us and the model stakeholders in the company, was that the model is made with a lot of knowledge about how the software is, or will be, designed.

It is not practical nor easy to model *all* behaviours of the mechanics. This would have as a result, a model to complex to review, too difficult to maintain, it would increase the time necessary to design the model and it would require more hardware resources; the gain would not be worth all that effort. Actually this might make the model not usable.

There are some software design decisions that are fixed and will not be changed.

Having said that, a modeller must not forget that the emulator describes the mechanics. Sometimes, for software engineers it is difficult to forget about what the software does and difficult to focus only on the phenomena and behavior in the mechanics that reacts to actuators signals and turns on and off the sensors. Also, a modeler should be aware that mechanics does not 'know' things that the software 'knows', like for example the length of the paper and number of papers in a set.

Where to take into account how the software is designed and where to focus on describing mechanics only - we did not give an answer to this.

Model has to be **complete**. What goes in the model depends on what we want to test and/or verify. All aspects of the system that influence the property we are checking should be present in the model. This is related to the question about the purpose of the model and requirements that are tested. (It is actually difficult to know whether the model is complete. After a lot of work with the model and the system, one is more convinced that everything is there.) For example, the current model does not test timing requirements. Therefore the speed is not described, because even if it is modelled, the results obtained with the model describing it, and with the model not describing will be the same. For the requirements defined, the model is complete.

How much it is necessary to have the model that is **understandable** is inseparable from the question to whom it has to be understandable. For peer-reviewing it is good that the model is understandable to other engineers using the model or to those who provide relevant system information for designing the model. The example of a software engineer mentioned with the question Q6

is also an example of the situation in which it is necessary that a user of the model understands what is what in the model.

Wherever possible, it is good to keep the model **traceable**. This means that the model structure can be mapped to the system structure. This helps to: (1) Justify modelling decisions, because it is easy to see that what we wanted to describe about the model exists somewhere in the system. If something is changed in the mechanics, it is easy to make changes in the model. (2) Trace the source of problems in the system if emulator shows an error of a model component. For example, if the control software shows that a paper becomes 'longer' while moving along one of the sensors, it is easy to look at the model of that sensor and see whether there is a modelling bug, or a sensor is not working properly. If the sensor is part of the module where it is not easy to trace the movement of the paper along that particular sensor, it is difficult to find out what the source of the problem is.

Good decomposition and traceability increase model **maintainability**. Again here, the question is the purpose of the model, who will use it and for how long.

A model's **simplicity** depends on the modelling language, tools and hardware components being used. In the case of the emulator we were looking at, the modeller had one state machine per model with maximum of three states. The state machine can be hierarchical with maximum one level depth. This way it was possible for the modeller to keep track of the module and to visually validate that the model is correct.

Q8: What is the structure that will be modelled?

It is possible to decompose a system according to different criteria. We can identify the system structure based on physical components, or functions performed by the system or, in this case, processes performed on papers and envelopes.

Before starting to construct the model, it is good to decompose the system into modules and to design a model in such a way that the structure of the system corresponds to the structure of the model. This way parts of the system are mapped to the parts of the model and if something changes in the system, it can be easily changed in the model. Also, if there is an error condition, it can be traced back to the source of error in the system. If there is a bug in the model, it is easier to find it, by tracing back to the part of the system modelled. Some of the possible decompositions are:

- Decompose the system into functional modules (e.g. transporting papers and envelopes, collating, folding, inserting paper into an envelope, moistening envelope).
- Decompose the system into physical modules (e.g. feeders, collator, folder, inserter).
- Decompose the system from paper point of view - design all possible paper paths (this coincides with the functional decomposition given above).

- Identify parts that will have more variants (e.g. feeders of different capacities and paper lengths).
- Different configurations of the system (inserter machines are highly modular, they can be connected with a franking module, for example).

Usually the decomposition of such complex systems already exists and it is not pure functional or pure physical decomposition. It is always a mixture of different decompositions.

What does this mean for the modeller? It is good to have mapping of the system parts to the parts of the model, but the modeller has to decide for each modules what to (explicitly) describe - the physical parts or the functions or something else.

In the current model the paper path modelled the flow of the paper, so for this, the so called 'workpiece decomposition' was made. For other parts like collator, folder, inserter, it was difficult and unnecessary to model physical parts, but it was decomposed to sub-functions that were then modelled. In practice, an already existing decomposition contains different decompositions mixed, so the modeller has to decide what to take from it.

Q9: What modelling tools and languages will be used?

In interviews with people involved in emulator modelling we identified the following factors that determine this decision:

- What is already accessible in the company?
- What tools modelers already know or like?
- What is the cost of certain development package?
- What formalism describes the mechanics best?

Q10: What abstractions and idealizations will be made?

Abstractions and idealizations that will be made follow from the answers to previous questions. They depend on the purpose of the model, requirements to be verified and constraints in resources. Sometimes there has to be a trade-off between different things we want from the model. Examples of different abstractions have already been given in illustrations of previous questions.

One more example is: When modelling movement of folding knives, bringing them to the folding position takes time, and putting them back is instantaneous in the model (moving a real folding knife of course takes some time in both directions). This is done because in the model it is important to emulate the moment of paper being folded, as this has to be synchronized with other processes in the system. On of the reasons that the movement of the knife into non-folding (starting) position is instantaneous in the model is that while moving back into starting position there is no danger of colliding with any other mechanical part.

If, however, it would turn out that the knife might be stuck between the starting and folding position, and if the control would be able to observe this (or consequences of this), this would have to be in the model. Of course the requirement that we are verifying would also have to be fault tolerance requirement.

4.2 Part II: Model validation and modelling assumptions

Rather then having model validation and modelling assumptions implicit in the modelling process, we suggest to plan and think of the model validation and to collect the assumptions. This should increase the confidence in the model.

Testing, debugging, validating the model

Although the final goal is that the model verifies the system, it is necessary to be convinced that the model represents the system. For this, it is necessary to test, validate and debug the model itself. We proposed this after noticing that when parameters of the model are changed, it is difficult to trace what is going on in the model, by just looking at it. LabView, for examples, offers a simulation and debugging tool in which it is possible to execute step by step and to set breakpoints.

Assumptions

As part of our method we propose to collect the assumptions on the system. They are the conditions under which the model represents the system. For the current model, we collected number of assumptions and presented them as an example. However, we didn't go further with this, by giving guidelines how to collect them. Our experience is, that once we became aware of them, collection assumptions becomes an inseparable part of the modelling process.

4.3 Part III: Concrete Modelling Solutions

Sections in Part III give practical advice on how to design the model (emulator). The main model components are:

- The material path
- Collator
- Folder
- Inserter
- Sensors and actuators

As already mentioned, it is not possible to predict features of the future inserter and folding machines, but for some parts it is possible to predict whether they will (significantly) change.

Our modelling guidelines vary a lot for the material path and the rest of the modelling components.

The reasons are as follows.

(1) The paper path will most probably not change significantly when it comes to those parts that went into the model. The other parts may change a lot so that completely different elements execute collating, folding and inserting. (2) The modeller spent a lot of time exploring different ways to model the path. For modelling the rest of the machine, no alternatives were explored.

In the Appendix we give a part of the handbook with reusable modelling solutions for the paper path. Here we analyse what templates, soulutions and modelling decisions we extracted.

4.4 The material path

The appendix [1] shows the guidelines to describe the material path. The path is divided into tracks, and there can be a sensor on each of the tracks. The modelling guidelines look at the following parts:

- Track and possible roller slips
- Sensor
- Combining track and sensor
- Two papers merging
- Switch that determines which route the paper will take
- Part of the path where material from different feeders can arrive

These elements already exist, and for each we extracted the following elements

- The description of the machine elements (physical parts) that went into the model component
- Modelling assumptions
- State machines implemented by the model components
- The table with the dictionary and definitions that for each element of the state machines says what it describes in the machine
- Possible faults and errors of the machine component that the model component describes
- The rationale for current modelling decision
- Previous tries and why they did not work
- Alternative solutions and the discussion on them

As some more general questions we identified the following questions to answer when modelling the paper path:

- Is the shape of the material path important?
- Is it important at what angle towards horizontal surface it stands?
- What parts move and what parts are fixed?
- What parts move dependently on each other?
- Is it necessary to model all the parts of the path or only movement along/through/on/under/between them?
- Are their dimensions relevant?
- Are their dimensions related to the paper dimension?
- What parts are connected and how?
- What can go wrong with them? What are the consequences?
- What can go wrong with material - papers and envelopes?
- What requirements are verified? What is relevant for the timing? How does this result in implementation space?
- What is the direction of the movement? Does it change or it is the same?
- What edges are detected?
- Is there a role in the project that monitors the paper path?
- Is the horizontal position of the paper relevant for the material path emulator? Would it mean to model it for the mechanical engineers?
- Intertwining with other modules?
- Which mechanical part does move the paper and how?
- Types of motors and their characteristics?
- Types of sensors and their characteristics?
- What can happen in the system and is not represented in the model?

Collating, folding, inserting

Here we had more difficulties to extract repeatable model components. The paper path is estimated not to be changed dramatically, but the parts that collate, fold and insert papers can be designed in a completely different way. Here, the creativity of mechanical engineers plays a big role. In order to have different dimensions of the machine, or different features, they invent completely new parts and concepts.

However, the functions and subfunctions that these parts execute do not change. These are the processes that have to be done with papers and they stay the same. Papers will always have to be collated, envelopes flaps have to be moisturized etc. So, instead of having concrete, reusable principles as we gave for the material path, we identified

- Functions and subfunctions that were described in the model
- Order of executing them
- Questions and issues that are important for modelling these parts

The decomposition to functions and subfunctions already exist in the documentation, so we only extracted those that went into the model. For example, for the folding function different fold types and calculation of the position of the first and second fold knife are identified as reusable.

For all the functions, we identified steps, questions and issues when modelling them. They are:

- Divide system functions to sub-functions.
- Identify physical parts that perform the functions. This requires decomposition of each module to its sub-components.
  - What are the initial properties (e.g. speed and the initial position at the moment of the paper arrival) of those sub-components and are they relevant?
- How are (physical) sub-components moved?
  - One clutch per component or one clutch for more components?
  - Are only their end positions relevant - like for example knife on and knife off
  - Does moving from one position to another that takes time has to be modelled?
- Initial and end positions of components

  - Are the initial and end positions of a mechanical component relevant? (If so, are the positions between initial and end position relevant or the component can be modelled as a two-state component?)

- Speeds and speeds ratios of the components
  - Is the speed ratio of two mechanical components relevant?
  - Is the component speed relevant and is it constant?
- Identify relevant physical properties of the components (time, temperature, elasticity etc.) model?
- Does one mechanical part perform two functions at the same time?
  - If there is a mechanical part taking part in sub-functions of two different modules, can it be that the sub-controls do two opposite actions on it?
- Should the following work in synchronization?
  - Different sub-functions within one function (What is the sequence diagram?)
  - Sub-functions performed in different modules
  - Sub-components of different modules?
- Is the function performing in the mechanics only or the software also takes part? (Things that happen in the mechanics and that cannot be observed by the control can be just states in the model that take some time, without going into details of what happen in the mechanics.)
- Identify interfaces between components
  - How is the paper delivered to the module? It can be:
  - The activity of two components, the one who delivers the paper and the one who is receiving it or
  - Only one of them.
- Can the interface be part of the component model?
  - It can be that there is a switch or a component that belongs to
  - Two modules at the same time
  - Can be a component on its own (In the latter case, its model can be part of one of the models modules.)
- How will the model observe the beginning and end of some actions? For paper arriving to a certain point or paper entering a module, sensors can be used (either models of sensors that exist in the system or a virtual sensor - a sensor in the model that does not exist in the system and that does not send signals to the control). For waiting that mechanical actions finish, timers can be used.

- Material
  - Can a paper buckle?
  - Can a paper or envelope be lost or take the wrong path?
  - Is it relevant to put these in the model?
  - Can it happen that only part of material cannot arrive (e.g. a paper is being transferred from one component to the next, the sensor senses its leading edge, but something goes wrong and the part stays in the previous component)?

4.5 Part IV: Other modelling concerns

The model layout

This is about system elements into the model elements. Sometimes it is possible to map the system parts into the model components. Sometimes this requires more complex model and it is better not to have this mapping.

An example of this modelling of the points where papers from different feeders can join together. If there is mapping of all four feeders as they are, the adder component has to have four virtual tracks (see appendix). However, papers and envelopes will never be pulled from the feeders at the same time. This allows to have them mapped as the feeders are placed on different sides of the machine. The model is simpler, and it still describes the system truthfully for the requirements that are verified.

Parameters The model has to allow change of parameters. For example, in the document feeders, papers of different sizes can be stored. Also, the speed of the motor can be changed. Furthermore, when modelling faults, different faults should be injected, while the model is executing. This makes the model more complex and difficult to validate.

The Fig.15 shows the model as a hierarchical state machine consisting of the state when parameters are refreshed and the other that actually describes the system. The reset event is the event that occurs when the model is reset, and also when some of the clutches are turned on. The latter one means that some of the emulator submodules can refresh the parameters that were changed until a clutch is turned on. In this case turning on the clutch triggers the state machine to describe the paper arrival. Before that the state machine is in the wait state, writing for the paper to arrive. This prevents that a paper for example changes its length or thickness.

Implementation Platform Different ways to calculate or model something have as a result the same model behaviour but takes different amount of memory space, hardware inputs and outputs. It can happen that one modelling decision or solution spreads all over the model, so if it is noticed that it takes too much resources, it takes too much time to correct it. For example comparison takes a

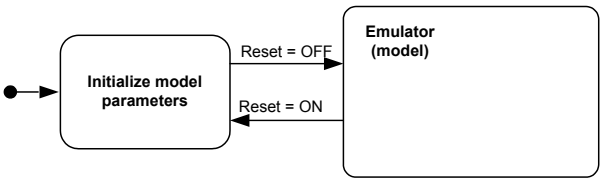


Figure 15: The model has the state where it resets and refreshes the parameters.

lot of FPGA space. So, if possible it is better to compare Booleans. Comparison functions treat the Boolean value TRUE as greater than the Boolean value FALSE. These are the things that have to be thought about in advance.

Modeling Faults

The most of the testing time goes to finding out what can go wrong. For the current emulator, the faults in the mechanics were simplified. Faults were classified to track slip (rolling the pinch without paper moving), and three different sensor faults. It is yet to see whether this was good enough for fault tolerance testing.

Testing Scripts Before modelling it is good to answer the following question: How we will run the model and where the results will be sorted? Testing scripts have the following functions: (1) They emulate user wishes expressed through user interface and forwarded through the system control. All possible scenarios are defined there. (2) They automatically set parameters of the model that describe the system, like paper length, track slip etc. (3) They, together with the model, determine what will be tested. If we want to see the source of an error, there will be log that will have recorded the time and place of an error occurrence.

Testing scripts are connected with the model, so that test and verification engineers do not have to manually test the scenarios and change parameters.

5 Validation of the method proposed

There is an internal validation and external validation. Internal validation is validation for this particular case, i.e. for the company and its machines. External validation is the validation of the solution we propose to general case of verification of an embedded control system. In this report we only give the internal validation.

5.1 Internal validation

As we already said in Sect. 1, we started this case study with the method with following ingredients in mind:

- Describing the plant, separating plant and control early in the design of the entire system model
- Non-monotonic refinement
- Collection of assumptions
- Steps to perform as modelling guidelines

Describing the plant, separating plant and control early in the design of the entire system model

The nature of this emulator is that it describes the plant, so this separation was done from the start. Here we confirmed once again that describing the plant completely independently from the software can result in unnecessarily complex model. When designing a model for a posteriori verification, this is an easy to do.

Non-monotonic refinement

Non-monotonic refinement describes how to structure the knowledge about the system. Some of its elements exist as part of the rationale for modelling decisions and as alternative modelling decisions. However the form of documenting modelling decisions in the form of a theorem was not used here.

Collection of assumptions

For the existing model, we collected the modelling assumptions. Some of them are the knowledge about the system that 'everyone knows'. We were told that those things that 'everyone knows' can be very dangerous, because it can happen that not everyone knows them. Collecting assumptions should prevent having a model that does not really describe the system.

Steps to perform as modelling guidelines

The steps we identifies were considered as useful and important things to be aware of when modelling.

We asked the modeller whether the first part would be useful, since this is the list of questions without a tool to support it. His opinion is that no tool is necessary to facilitate doing the first part, as it was clear and short enough.

5.2 Re-usable elements

The new element we added is a collection of modelling decisions and some of the solutions. They are the things that already exist in the current model, and in the head of the modeller. By putting them explicitly int eh handbook, we expect that they make the modelling process (1) repeatable and (2) not-dependent on one person.

We will stay in contact with the company and will interview them once they use the handbook in one of their next projects.

Bibliography

[1] A. H. Mader, H. Wupper, M. Boon, and J. Marincic. A taxonomy of modelling decisions for embedded systems veri?cation. Technical Report TR-CTIT-08-37, 2008.

[2] J. Marincic, H.Wupper, A. Mader, and R. Wieringa. Obtaining formal models through non-monotonic refinement. Technical report TR-CTIT-07-33, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, May 2007.

[3] J. Marincic, A. Mader, and R. Wieringa. Capturing assumptions while designing a verification model for embedded systems. Technical report TR-CTIT-07-03, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, Jan 2007.

[4] MOCA. Modelling control aspects of embedded systems. *http://moca.ewi.utwente.nl/*. NWO Project 600 065 120 241420.

[5] Neopost. *http://www.neopost.com*.

[6] Neopost Technologies. *http://www.neopost-technologies.nl/*.

6 Appendix

(starts at the next page)

1 Material path

1.1 Relevant elements of the material path

In this subsection the engineering knowledge that went into the model is given. It might happen that in the future, in the next project, it is necessary to describe some elements with more details.

Material, inserts, documents and BRE's
Documents and envelopes are called *material*. Material that goes into the envelope is called *inserts*. *BRE (Business Reply Envelopes)* are small envelopes that are inserted into envelopes.

Figure 1-1 shows those definitions on a diagram.

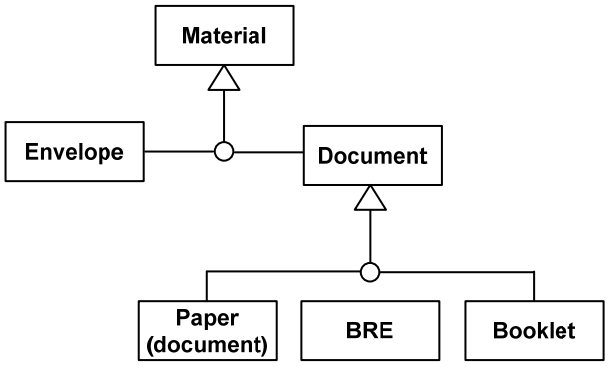


Figure 1-1: Definitions of material and insert.

Path
It is the path through which papers or envelopes pass. There can be more paths in the system. One is the path through which papers move. Another is the one through which envelopes move. These two can merge into one path. The paths go through all system modules. As the material moves through the inserter, different system functions are performed on it.

Rollers
Rollers move a paper or an envelope. Figure 1-2 shows the principle of paper movement between a pair of rollers. If the paper is in the feeder, there is one roller moving it.
Assumptions: Rollers always move in one direction.

(a) **(b)**

Figure 1-2: A 2d sketch of rollers that move the material. Either a pair of rollers rotates in one direction causing the movement of the paper between them; or, a paper is lying in the feeder and only one roller moves it.

Every pair of rollers is connected to a clutch. When the clutch is on, the rollers rotate. When the clutch is off, they do not rotate.
Faults:
It can happen that rolling does not move the paper for a while, so the paper moves slower.
Rollers can wear out.
The speed of the roller touching the paper is different than the speed of the roller on its other side not touching the paper (this is not modelled).

Clutches
Clutches are coupled to the elements that can move or rotate. When turned on, a clutch transmits motor drive to the elements to which it is connected causing those elements to move or rotate. Clutches are used to rotate rollers, to reverse tracks, to trigger the knives. One clutch can move one or more rollers.
Clutches introduce delay.

Motors
A motor can move more rollers and other parts via drive transmission wheels and clutches. One motor is connected to all clutches and rollers on the paper path. Another motor is the stepper motor in the collator and it moves the stopper (this is the situation in X model).

Pulse disk
Motor speed can be calculated with the pulse disk. It is known how many pulses are counted during which a paper moves 1 meter. However, it can happen that the speed is not constant through the whole machine. Figure 1-3 is a diagram of the motors connected to rollers through clutches.

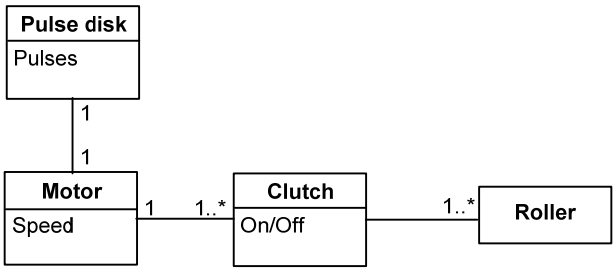


Figure 1-3: The motor is connected to one or more clutches and to the pulse disk. A clutch moves one or more rollers.

Sensors
There are digital and analogue sensors. Digital sensors are less accurate and therefore not good for absolute measurement. Their main parts are a lever and a spring; turning off takes more time, so after a paper passed it can be that the sensor is not yet in the Off position. This time is longer as the sensor ages. Analogue sensors are photocells and the signal they produce is a continuous signal that is not easy to model. Right now DFC sensors are used.
Sensors are modelled as a digital sensor with only On and Off states.
Faults
Sensor can be broken and then it is always off. Sensor can also be stuck in the On position.

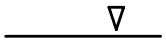


Figure 1-4: A 2D sketch of a paper moving along the sensor

Switches
A switch determines continuation of a path on a ‘crossroad’ of different tracks. Switches are mechanical elements that have two different positions, taken when the switch is on and when the switch is off. As shown on Figure 1-5, each position determines where the path continues.

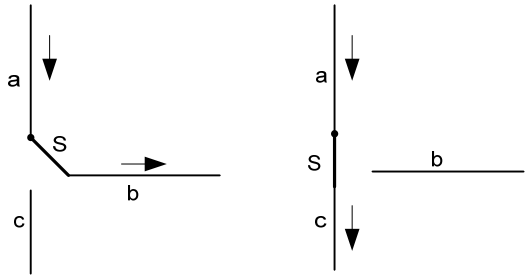


Figure 1-5: The switch can be in two possible positions. Depending on the position (switch on or off) the coming material follows either segment *b* or segment *c*.

Feeder
Feeder is the place where papers or envelopes are stored at start. Its mechanical construction determines the distance that papers will have between each other *through the whole path*, till the collator (for papers) and inserter (for envelopes).

Constants

Maximum thickness of inserts is determined as maximal of the two values: maximum thickness between rollers and maximum thickness between the folder rollers. The maximum thickness is calculated for z-folded papers (worst case).
Envelopes move one by one. Inserts can consists of a number of sheets. Maximal thickness is determined by the characteristics of the folder and pinches.

1.2 Material

Material = paper, set of papers, an envelope, a booklet or a BRE (business reply envelope).

A single paper or a single envelope moving through the system is considered as one object characterized with length and thickness.

If two papers join, they become one object characterized with length, maximum thickness and the length of the longer paper. If the deviation parameters equals 0, that means that the paper is moving alone or that two equal sheets are perfectly aligned.

The material data is modelled as data that flows through the components.

The material is described with the parameters given below, in the table on Figure 1-6.

Material characteristics	**Data**
Length	Length of a paper or two papers joined together or envelope length.
Thickness	Thickness of a paper or two papers joined together or an envelope.
Deviation	- equals 0, if the material is moving alone; or two equal sheets are perfectly aligned - equals the length of a longer paper, if two papers joined

Figure 1-6: Parameters of an object describing material

1.3 The path model

The path model consists of the following components:

- Track
  - Simple track
  - Track with a slip
- Sensor
- Track and sensor
  - Special case: feeder
- Adder
- Switch
- Selector

Also, an important element is the **flow of data** that describes material flow[1]. The parameters of papers and envelopes are described as parameters of the feeder, which is the point where their path starts. In one of the following subsections, these parameters will be described with more details.

All the lengths and distances are expressed in number of pulses generated by the pulse disk – either only leading or both edges of pulses are counted. Note that some machines can have different parts with different edges/m parameters – this means that the speed is different on different segments.

It would occupy too many FPGA resources to model the **motor.** Instead, the motor is connected to the emulator via the pulse disk. In the model, there is a **pulse counter** that counts pulses from the pulse disk.

[1] In this section whenever the word ‘paper’ is mentioned it refers to material, i.e. it can be both paper or envelope or two papers moving together and therefore seen as one piece of a material.

1.4 *Track*

A path is decomposed (divided) into tracks.

A track describes the following:

- A piece of physical space (a path segment) through which material passes.
- Movement of the material through this segment
- Rollers (if there are any on the chosen segment)

A track can contain zero, one or more rollers. The rollers are not modelled separately; they are implicitly part of the track. Also, if there is a slip, then it is the same for all the rollers on the same track.

Assumptions:
The shape of the path is not relevant.
Only the movement in the forward direction is relevant. It is assumed that material cannot move to the left or right.
The vertical or horizontal position is not relevant, gravitation does not pay role here.
If the clutch is off, right after the trigger out, the track component will continue to move the paper.
The slip is constant on one track for all the rollers and for all the papers.
A roller has two wheels; however it is described with one slip.

There can be a track with a slip or without it (see the explanation later.)

1.4.1 Data flow diagram of the track

Figure 1-7 shows the track component and the data flow through it. Arrival of a paper or set of papers is modelled with the TriggerIn event. As soon as the leading edge of the paper leaves the track, an event TriggerOut is generated. TriggerOut is TriggerIn for the next component.
Data describing the length and other paper properties comes in DataIn signal and is forwarded to the next component as DataOut signal.

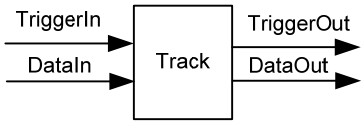


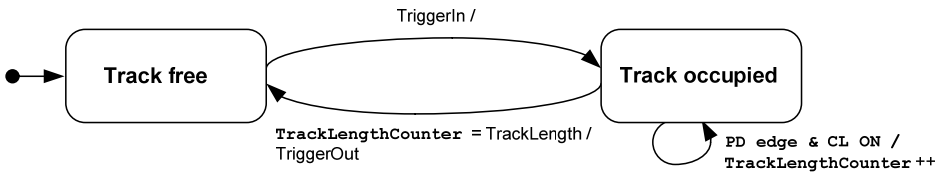
Figure 1-7: Track behaviour is described with a state machine. From the data-flow point of view, track is the component that receives data about material on the trigger and it generates trigger when it has to forward data to the next component.

1.4.2 Simple track - state machine

The track state machine describes a path segment and the movement of *material leading edge* through it.

On Figure 1-8 the track state machine is given (together with the dictionary). If the clutch is on and TriggerIn arrived, a local variable, TrackLengthCounter will increment its value. The counter counts the pulse disk edges; after the number of edges that correspond to track length, triggerOut is generated. For example, if it takes 1000 pulses for a paper to move 1m, then if the track is 10cm long, the TrackLength will be represented in number of pulses, in this case 10.

The track length is chosen so that there can be maximum 1 leading edge of a paper present at the same time on the same track. Therefore in the model it cannot happen that the TriggerIn arrives while TrackOccupied state is active.



Phenomena in the system	Their representation in the model
	States
Track empty or leading edge passed the track	TrackFree
Leading edge of a material on the track.	TrackOccupied
Clutch that moves the roller on the corresponding track is turned on.	CL ON
	Events
Leading edge of a paper arrives into the track	TriggerIn
Leading edge of a paper leaves the track	TriggerOut
Time passing between leading edge arriving and leaving	Number of pulses necessary to move the distance that equals the track length (plus the slip time)
An edge of a pulse going from 0 to 1 or from 1 to 0 from the pulse disk.	PD edge
	Variables
Track length	The length of the track

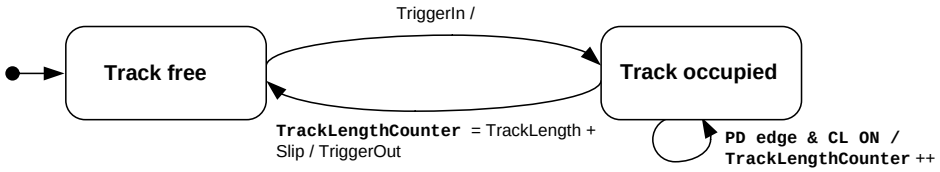
Figure 1-8: Track state machine

In the model, the path is represented as a sequence of tracks. Each paper or envelope is an object with its length, thickness and separation distance from preceding or succeeding paper or envelope. Once a paper enters a track, it will not change its length (it can merge with another paper only on specific places, which will be described later).

1.4.3 Track with a slip - state machine

To model the slip of rollers, the parameterised track needs additional parameter that determines for how many pulses material is slowed down. The only difference between the models on Figure 1-8: Track state machine and Figure 1-9: Track with slip state machine is that in the latter model *slip* parameter is added. The condition to go back to TrackFree state is now: TrackLengthCounter = TrackLength + Slip

Note that slip does not affect object parameters.



Phenomena in the system	Their representation in the model
	States
Track empty or leading edge passed the track	TrackFree
Leading edge of a material on the track.	TrackOccupied
Clutch that moves the roller on the corresponding track is turned on.	CL ON
	Events
Leading edge of a paper arrives into the track	TriggerIn
Leading edge of a paper leaves the track	TriggerOut
Time passing between leading edge arriving and leaving	Number of pulses necessary to move the distance that equals the track length (plus the slip time)
An edge of a pulse going from 0 to 1 from the pulse disk.	PD edge
	Variables
TrackLength	The length of the track
Number of pulses that pass during which paper does not move	Slip

Figure 1-9: Track with slip state machine

1.4.4 The rationale for having a simple track and a track with a slip

Instead of having all the tracks with a slip that can take any value from zero to N, a simple track does not have the slip parameter. This saves the memory space in FPGA emulator.

1.4.5 Faults and errors in the track

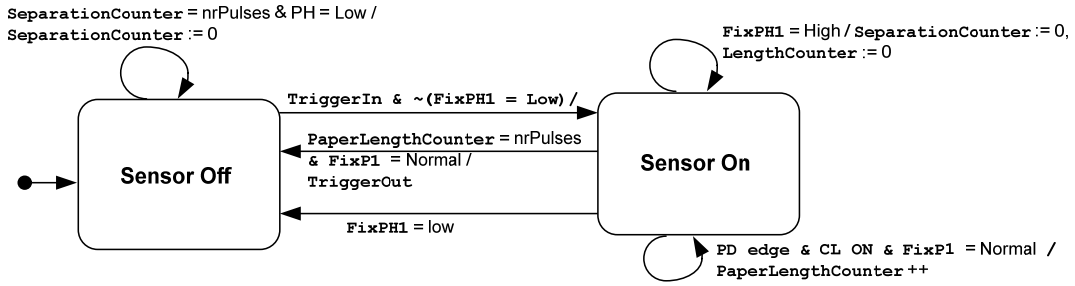
Error is detected if the new triggerIn comes before the current paper leaves the track. (The second sheet arrives sooner than expected).
Depending on the requirement, this is an error in the model (something that is not possible in the system happened in the model) or an error in the system (paper arriving too early).

1.5 Sensor state machine

Assumptions: Turning on and off is not delayed and it takes no time.
Sensor is modelled as a point in space that is 'long' one pulse. The place where the sensor is a place to divide a path segment into tracks, so that sensor is placed at the beginning of a track, as shown on Figure 1-11.

A sensor can be modelled without a track.
If there is no physical sensor at the start of the track, there can be a virtual one, if necessary.

The sensor is, in most of the cases, coupled with a track! Therefore the sensor and the track slip that are put together should be the same.



Phenomena in the system	Their representation in the model
	States
Nothing in front of the sensor.	SensorOff
Something in front of the sensor.	SensorOn
Clutch that moves the roller on the corresponding track is turned on.	CL ON
	Events
Paper leading edge arrives in front of the sensor	TriggerIn
Paper trailing edge of a paper passed the sensor	TriggerOut
Time passing between leading edge arriving and leaving	Number of pulses necessary to move the distance that equals the track length (plus the slip time)
	Variables
State of the sensor Sensor working Sensor broken and off Sensor broken – stuck in on position, like there is a paper always in front of it	FixPH FixPH = Normal FixPH = Low FixPH = High
Time that has to pass for a whole paper to pass along the sensor, calculated in number of pulses	nrPulses
SLIP	The number of pulses that express slow down of paper on the track. It has to be the same as the slip of the track to which the sensor belongs (this has to be done manually)

Figure 1-10: Sensor state machine

On Figure 1-10, the sensor state machine is given together with the dictionary. It starts in Sensor Off state. When the leading edge of a paper arrives, there is a transition to state Sensor On and the counter starts counting the pulses. The counter is counting till the value corresponding to the paper length. After that, it goes back to the SensorOff state. If there is a slip, the paper moves slower, so the counter has to count to the maximal value plus the slip. If the sensor is broken it will stay in Sensor On or SensorOff state.

Figure 1-12 shows sensor and track state machines running in parallel when describing the path segment shown on Figure 1-11.

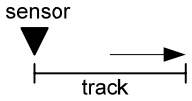


Figure 1-11: Sensor positioned at the beginning of a track.

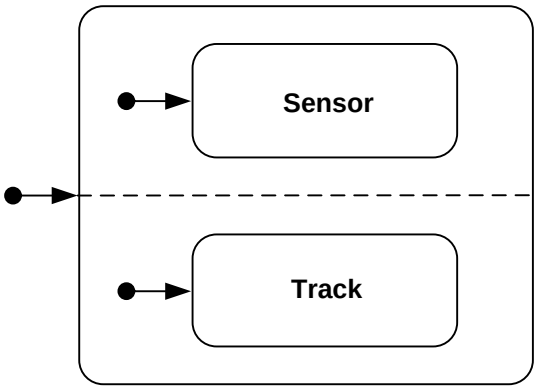


Figure 1-12: Parallel state machines describing movement along the sensor and track. The sensor stays On for the number of pulses that correspond to the paper length. The track stays occupied for the number of pulses that correspond to the track length.

1.5.1 Faults and errors in the sensor

A sensor can be stuck in On position, or broken and therefore always in Off position.

The distance between sensors is the minimal length of the paper (insert).
In the future: slip should be a % of the paper length.

1.5.2 DFC sensor

At the moment only digital sensors are modelled. Modelling DFC sensors would require having an analogue waveform for slip plus length duration.

1.5.3 Feeder

Feeder describes the following:

- The feeder module storing material
- The separation distance between material
- The material itself is represented by its length and thickness
- Movement of the material

The feeder component consists of a track component and a sensor component As it models the starting point in the system, it is the starting point in the data flow diagram, and does not get triggerIn impulses from any other components. TriggerIn is generated by the sensor module.

A counter is counting number of pulses for a paper to pass along the sensor, adds the slip if necessary and a separation distance. Separation distance is the distance between two papers.

Assumptions:
Mechanical characteristics of the feeder determine the separation distance which stays the same all the way through the feeder module.
There is always paper in the feeder.

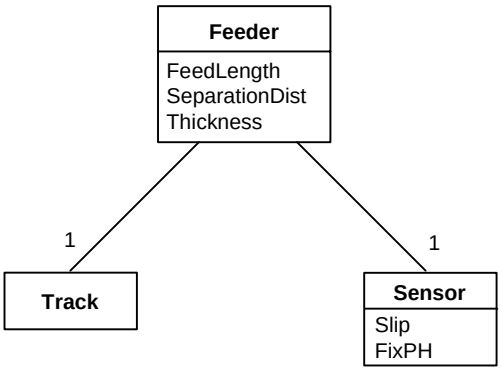
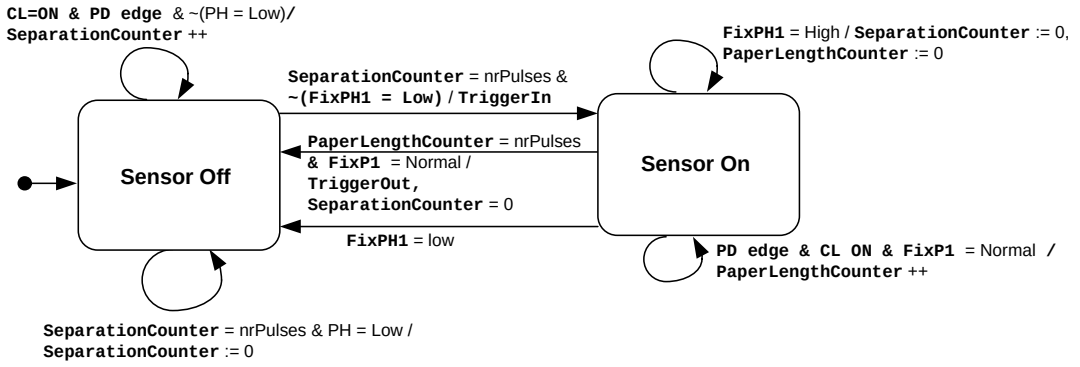


Figure 1-13: Feeder consists of a track and a sensor. It also contains the data about material stored in it.

On Figure 1-14 the state machine of the sensor in the feeder, together with the dictionary in the table, is given. The only difference with the ‘normal’ sensor’ is that it does not wait for the TriggerIn event. It goes to On state, every time the counter counts the number of pulses that correspond to the separation distance.



Phenomena in the system	Their representation in the model
	States
Clutch that moves the roller on the corresponding track is turned on.	CL ON
	Events
Leading edge of a paper arrives into the track	TriggerIn
Leading edge of a paper leaves the track	TriggerOut
Time passing between leading edge arriving and leaving	Number of pulses necessary to move the distance that equals the track length (plus the slip time)
An edge of a pulse going from 0 to 1 from the pulse disk.	PD edge
	Variables
Length of material in the feeder	feedLength
Thickness of a material in the feeder	feedThickness
Minimal distance between two pieces of material in the feeder	separationDist

Figure 1-14: The state machine of the sensor in one the feeder. Instead of waiting for TriggerIn, the component generates it after the paper leaves the feeder and after number of pulses corresponding to the separation distance.

1.5.4 Faults:

Slip in the feeder is always bigger than in the other tracks in the system.

1.5.5 Sensor at the end of a track

At the end of a path there can be a sensor. Therefore, in the model, it should come right after that track. The sensor in the model will not count pulses, but it will just observe arrival of the leading edge of the paper and the leading edge leaving (when another component moves the paper away).

1.5.6 Track with the length that can change?

If the paper is coming to the track, and there may be possibility that another paper joins it, there might be useful to have the track within the input length can increase while the paper is moving along it.

1.6 Deciding about the track length – alternative solutions

There are two modelling strategies – having tracks as long as possible – for example from the feeder to the collator; or divide paths to shorter tracks.

In the case of a long track there can be more than one sheet of paper or envelope occupying the track. If (1) there is no slip and (2) the rollers move with a constant speed, papers arrive to a long track with the separation distance between them preserved. There will not be collision. The distance between them that was on the feeder exit will remain the same.

If, for example, there can be two sheets on the long track, or to be more precise, two leading edges of two papers, then the state machine on Figure 1-15 can be used to describe this solution[2].

The meanings of states and events are the same as in the table on Figure 1-8. Additional variables are:
trackLengthcounter_1 - counts the pulses passed while the first paper is moving,
trackLengthcounter_2 - counts the pulses passed while the second paper is moving.

If the first paper leaves the track, trackLengthcounter_1 should copy the value of trackLengthcounter_2 and keeps counting (this way a FIFO queue principle is implemented). If the track capacity is three leading edges, it is easy to model it by adding additional state and additional counter.

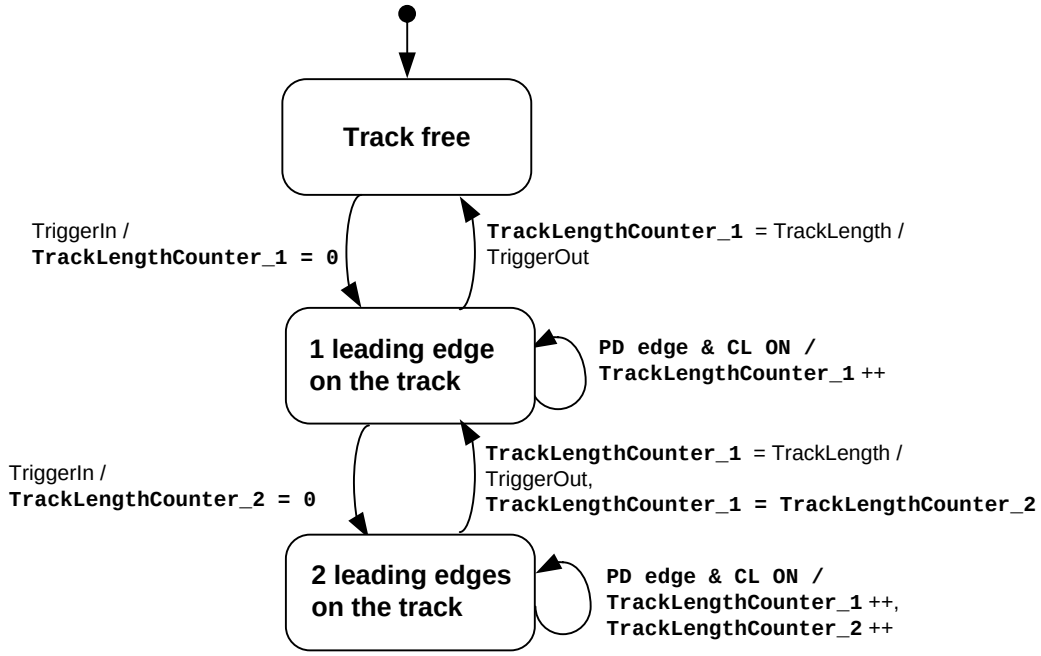


Figure 1-15: State machine describing a long track

[2] This solution is not the part of the existing model. It explains the principle discussed with the modeller of X emulator.

If there is a slip on a long track, there has to be an additional state machine modelling the material progression along the start of the track. This parallel state machine can be a sensor or a virtual sensor. Figure 1-16 describes this solution. Again, this solution is not the part of the existing model, but it explains the principle discussed with the modeller of X emulator.

In the solution presented, the sensor state machine is a virtual sensor – it does not exist, but we put it in the model to keep track of paper movement. If, however there is a real sensor, than the sensor state machine should also include possibility that sensor is broken (off state) or stuck in the On state. In this case the counter should continue counting pulses even if stuck in On or Off state and should keep track of paper movement anyway.

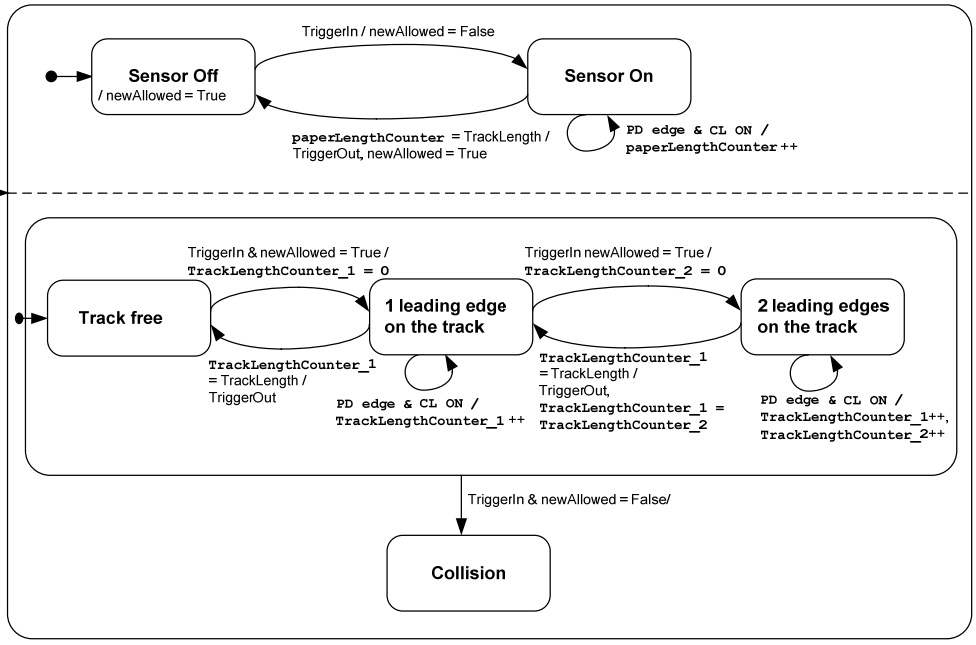


Figure 1-16: The track state machine describing a long track- it can contain two leading edges of two separate papers. In the state '2 leading edges on the track', two leading edges are present. As soon as the first one leaves the track, counter 1 should have the value of counter 2 as it will continue to track the second leading edge that will become the first leading edge in the track.

Another solution is to have two parallel state machines for each paper – to model a paper as a state machine rather than a track, as shown on Figure 1-17.

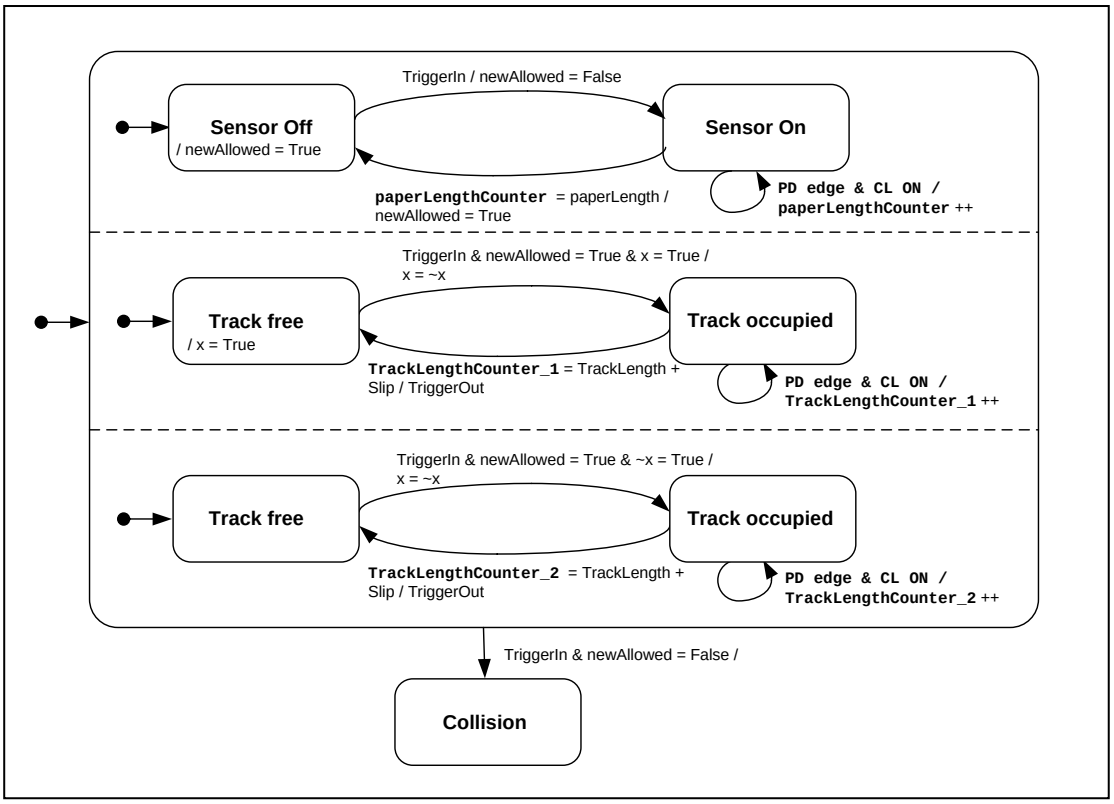


Figure 1-17: Two parallel state machines describe movement of two pieces of paper on a long track. Sensor

This solution turned out to be too difficult to maintain in LabView – it is difficult for the modeller to inspect the model and keep track of the events.

1.6.1 Short tracks – alternative solutions

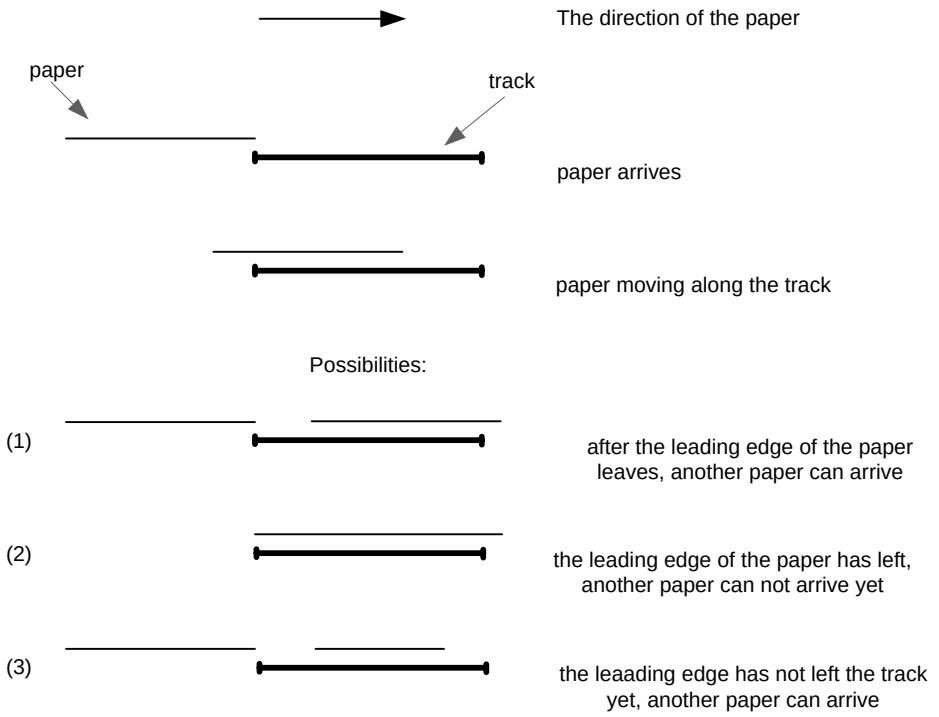


Figure 1-18: Different possible lengths of a track

Currently, the lengths are chosen so that outcome (1) shown on Figure 1-18 is possible. The maximum track length is 100mm.

Note that 'track free' means two possible things: either track is empty or the leading edge of papers has left it. As soon as the leading edge has left it, a new paper can arrive.
This is because the track length, l > paperLength.

There cannot be two papers on the track at the same time, the track length is chosen so that Track length < Paper length + Separation distance.

However, there *are* different paper lengths, so the model is not a realistic representation of what can happen.

If there was no slip, and if the speed of the paper is the same through the transport path, the simple track is enough to describe the real track. But if there is a slip, it can happen that the separation distance changes and that a new paper merges with a paper that precedes it.
So, if a new paper arrives after the track is in the trackFree state, it might happen that there was a paper collision (or merging) in the system, not observed in the model

Possible solution needs monitoring what happens on both sides of the track (Figure 1-19). The track state machine observes the moment when the leading edge arrives and moves to the next track. On the start of the track, a sensor state machine will detect the arrival of leading edge and passage of trailing paper edge.

Until the paper leaves the track, nothing else can arrive.

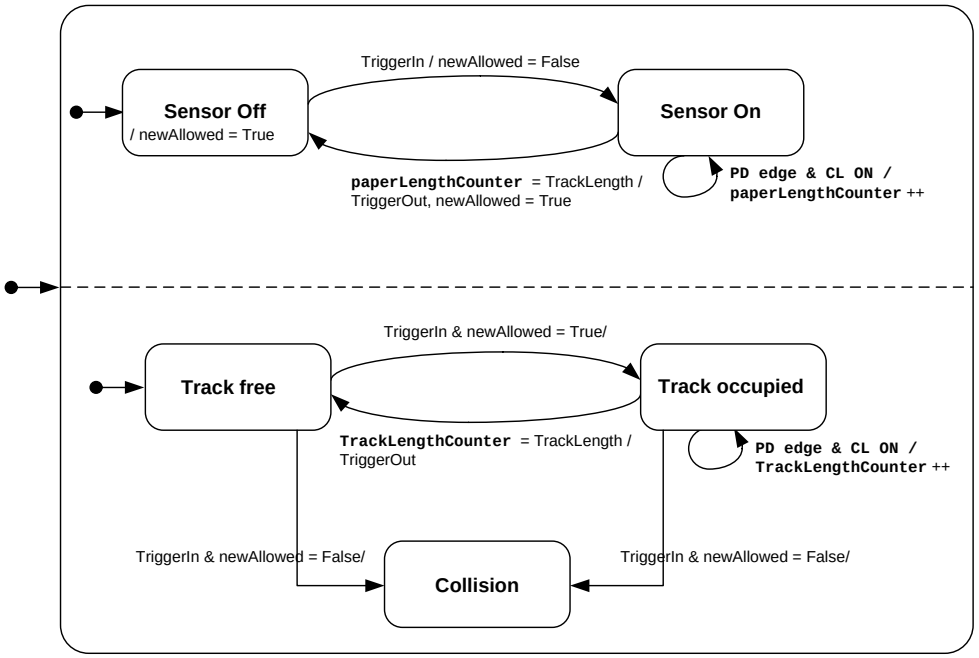


Figure 1-19: Possible solution of a short track with a slip

If the first paper suddenly starts slowing down after the new one arrived, this will not be detected.

The sensor state machine is the case of a sensor without faults. In case we want to model sensor faults, the counter should keep counting even if the paper is stuck in off or on position.

Slip in the feeder causes separation distance is larger. This does not cause collisions.
Slip later can cause collisions.

Assumption: track slip does not change for a track
.

The track state machine describes the movement of a paper leading edge.
Track is always coupled with a sensor (real sensor model or a virtual sensor) that stands on the beginning of the track. The sensor state machine emulates leading edge arrival and trailing edge passing.

Track lengths are chosen so that there can only be one leading edge of a paper present on it. The possibilities (1) and (2) on the Figure 1-18 describe this. On the same figure, possibility (3) describes the situation of having 2 leading edges of two papers at the same time.

In this case track state machine should have three states, as shown on
Figure 1-20.

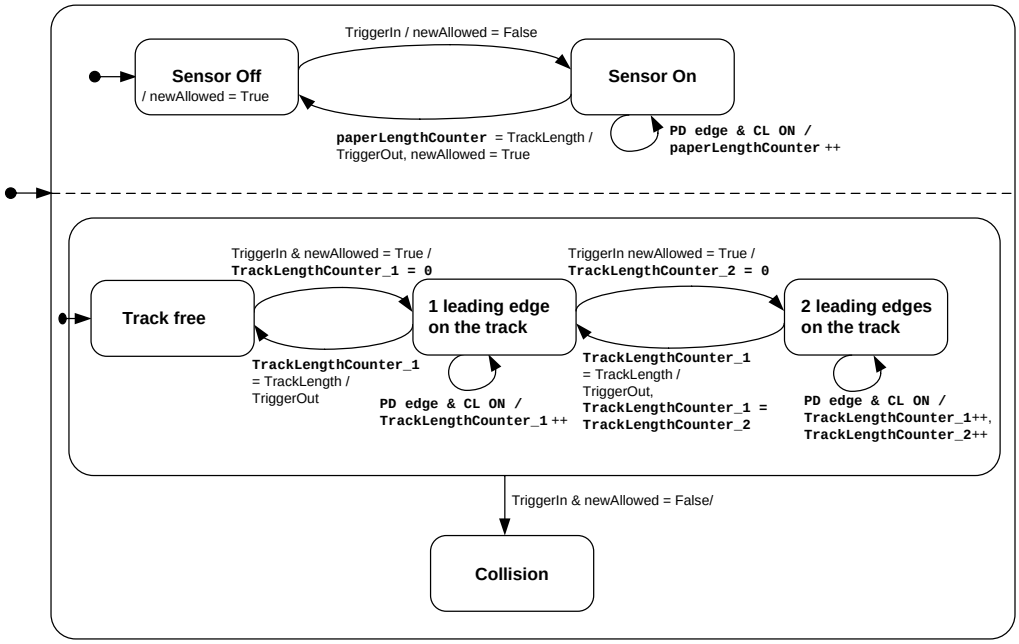


Figure 1-20: The track state machine describing a track that can have two leading edges of two separate papers present at the same time. In the state '2 leading edges on the track', two leading edges are present. As soon as the first one leaves the track, counter 1 should have the value of counter 2 as it will continue to track the second leading edge that will become the first leading edge in the track.

Longer track gives less freedom for describing different slips of different rollers.

Alternative idea: to calculate the new length of the paper and not look at this as a collision.

1.7 Paths branching and merging

Physical paths can branch, or merge. Papers can join together and form one set that is being transported. These are described with the elements in the following sections: adder, selector, and distributor.

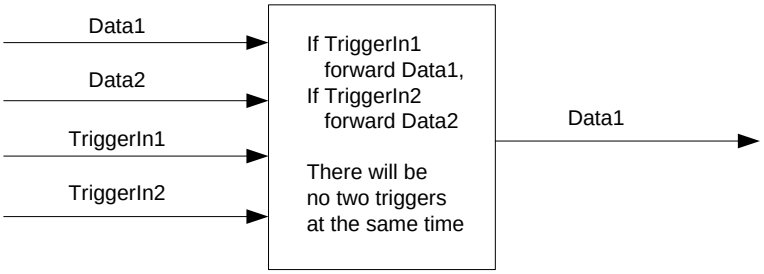
1.7.1 Selector

Selector describes

- A point on the path where materials from different feeders can arrive, but not at the same time, for example a BRE envelope or a paper.

The selector is connected to two different components that describe movement of material from two different feeders. Figure 1-22 shows the sketch of the two segments in the system. In the model, depending on the trigger, the selector component forwards the data about one of them, as shown on Figure 1-21.
Note that selector is not a state machine.

Assumptions: Here we know that there won't be two triggers at the same time.
If this however happens, the model will recognize only one of them.
There is not a physical component that selects between two pieces of material, this is calculation in the model which data to forward.



Phenomena in the system	**Their representation in the model**
Material arriving from direction 1 (for example feeder 1), represented by its length, thickness and separation distance	DataIn1
Material arriving from direction 1 (for example feeder 1), represented by its length, thickness and separation distance	DataIn2
	Events
Leading edge of a piece of material coming from direction 1 arrives.	TriggerOut
Leading edge of a piece of material coming from direction 1 arrives.	TriggerOut

Figure 1-21: Selector component

Direction 1 is assigned to one of the coming directions, and direction 2 to another. The way it is done is arbitrary, but it is important not to mix directions.

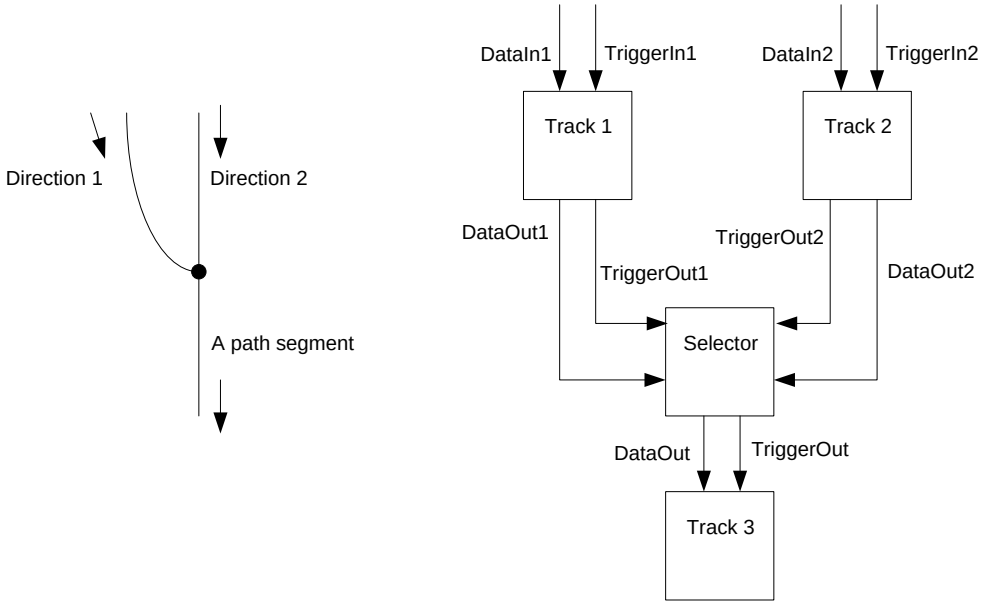
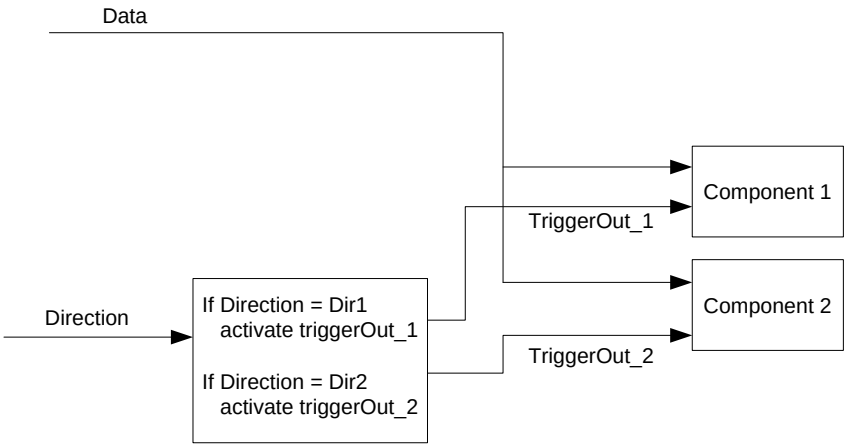


Figure 1-22: Left side: sketch of the two paths joining together. Right side: their representation in the model, where selector forwards the data about the material that arrived.

1.7.2 Distributor

This component models the switch that can redirect coming material to one of two possible paths. Figure 1-23 and Figure 1-24 show the model component and the sketch of path segments that it represents.

Assumptions: Switch on and off are two only positions and changing from one position to another does not take time.



Phenomena in the system	Their representation in the model
Material arriving from direction 1 (for example feeder 1), represented by its length, thickness and separation distance	DataIn1
Material arriving from direction 1 (for example feeder 1), represented by its length, thickness and separation distance	DataIn2
	Events
Leading edge of a piece of material coming from direction 1 arrives.	TriggerOut
Leading edge of a piece of material coming from direction 1 arrives.	TriggerOut

Figure 1-23: The distributor and a table with the dictionary

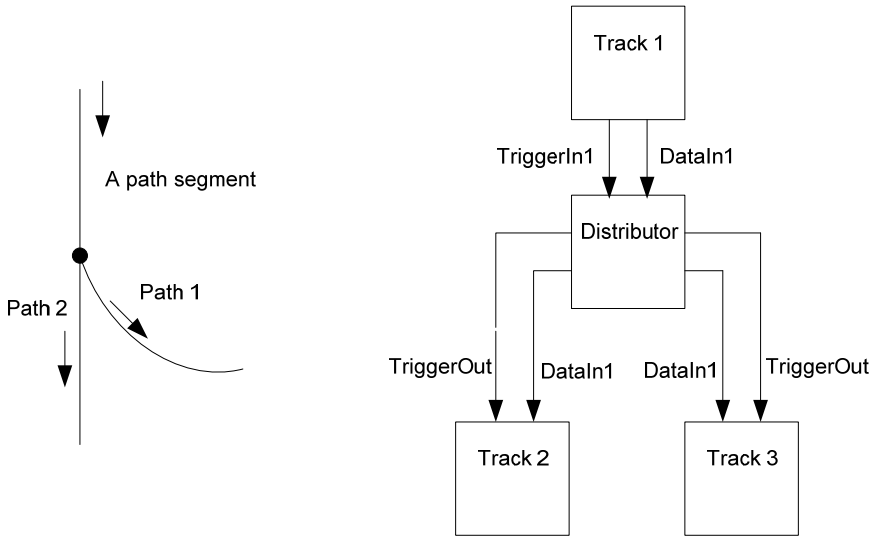


Figure 1-24: Left side: sketch a path branching into two paths. Right side: their representation in the model, where distributor forwards the data to one of the paths.

1.8 Adder state machine

Adder

- Describes the movement over a potential joint point of two papers
- Describes the movement over non – existing virtual track
- Calculates length and thickness of papers that joined

Assumptions:
There is no possibility to combine 3 papers, e.g. smth from feeder1, feeder 2 and next paper from feeder 2. Theoretically this would be possible (neoflow model showed this), but the collator belt cannot handle this.

The control will make sure that two papers have a minimal overlaping length.

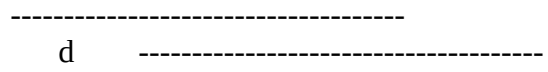


Figure 1-25: Two papers joined, wit the non-overlapping length d.

As shown on Figure 1-26, papers from two different directions can join together. If there is a tower configuration, point x will be the exit point of a lower feeder. After point x, there is the next path segment. If we model it as a track, we will have to provide the length of the coming paper, as soon as the paper enters the track. This is because tracks in the model are given the length once at the entrance and this length will not change.

To overcome this problem, there is a virtual track in the model. While moving over the virtual track, there is a chance that another paper arrives. Once the leading adge passed the virtual track, there won't be another paper sent (*assumption*).

The consequence of this trick is that it is not possible to check timing requirements for the system with this model because in the model moving over virtual track will take time and this will not happen in the model.

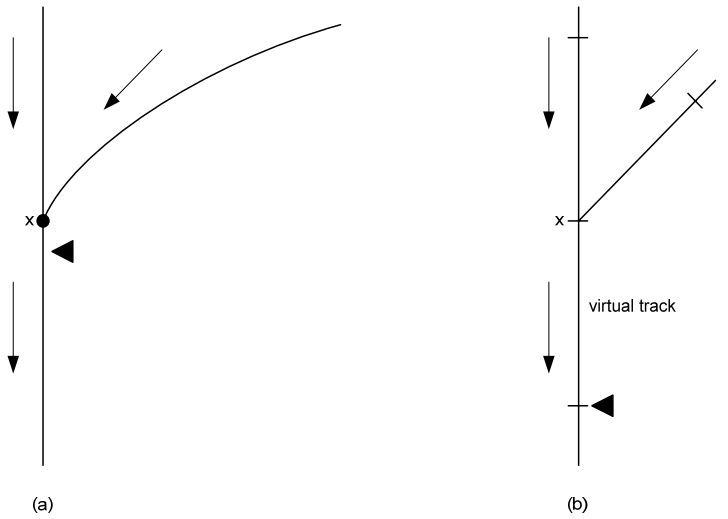
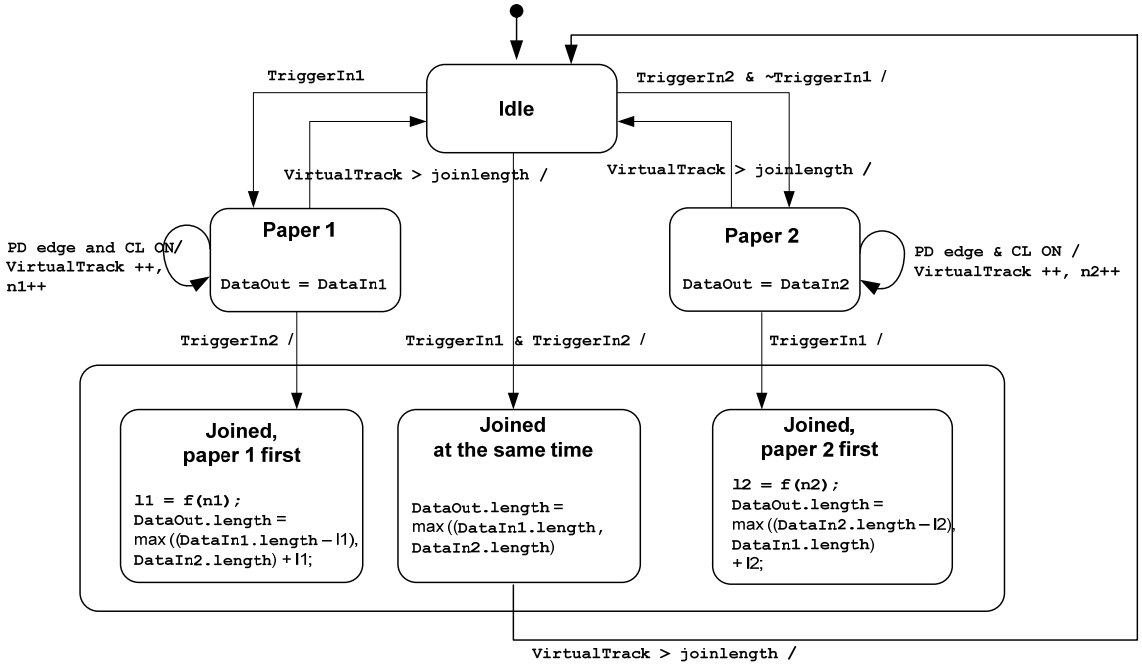


Figure 1-26: (a): Joining area in the inserter. (b) Joining area in the model

Figure 1-27 shows the adder state machine.



Phenomena in the system	Their representation in the model
	States
Set1 has not arrived Set2 has not arrived	Idle
Paper (material) from the first feeder moving over joint point, paper from the second feeder has not arrived.	Paper 1
Paper (material) from the second feeder moving over joint point, paper from the second feeder has not arrived.	Paper 2

Paper joined, paper 1 is first	Joined, paper 1 first
Paper joined, paper 2 is first	Joined, paper 2 first
Paper leading edges joined together	Joined at the same time
Clutch that moves relevant rollers is on.	CL ON
	Events
Paper from feeder 1 arrived at the joint point	TriggerIn1
Paper from feeder 2 arrived at the joint point	TriggerIn2
Virtual track > joinLength	A paper passed the virtual track.
Pulse edge detected	PD edge
	Variables
n1	The distance that paper 1 passed before paper 2 joined, expressed in number of pulses
n2	The distance that paper 2 passed before paper 1 joined, expressed in number of pulses
	Constants
joinLength	The length of a segment on which papers can join together

Figure 1-27: Adder state machine and the dictionary

DataIn1, DataIn2 carries the information about papers from feeder 1 and 2, respectively. It says something about the length, thickness and deviation.

1.9 Layout of the model vs. the inserter layout

There are two possibilities:

(1) Tracks in the model can be mapped to physical tracks in the inserter
(2) Tracks in the model are ordered differently in the model than in the inserter, but the model still represents the system for the given requirement.

Sometimes the requirement forces us to not to have the mapping or the model is simpler if we do not have the mapping and rely on how the control is designed. Here is an example.

Figure 1-28 shows the machine where envelopes and papers follow the same path. The model that follows the system structure is shown on the same figure. A virtual track describes merging of a paper and an envelope. This track does not exist in the system and it adds additional time that it takes for material to go through it.

If we know that the control will never allow papers and envelopes to merge (although the mechanics allows this), we can use the model shown on Figure 1-29. Now the tracks T1 and T2 describe the same physical track in the system, but the time (number of pulses) needed for material to reach the switch is now more precise than in the previous model.

The components of the model on Figure 1-29 would also describe the machine shown on Figure 1-30, if the machine with such configuration existed. In this case the model would follow both the system layout and would have correct description of timing properties.

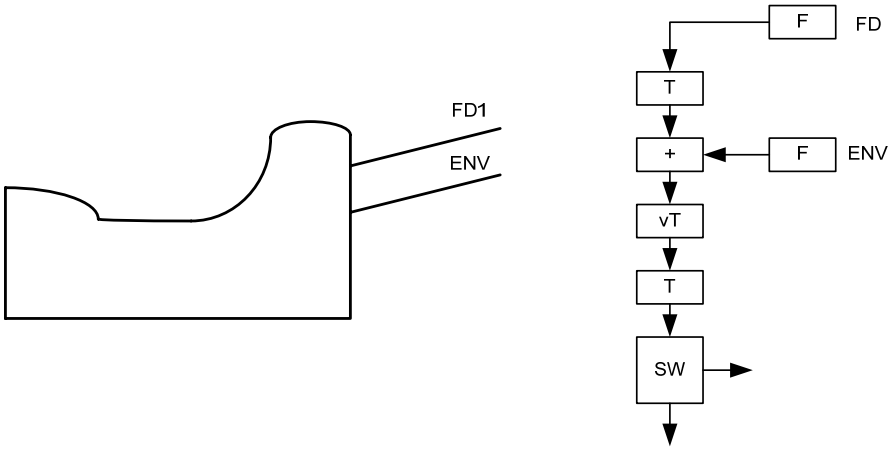


Figure 1-28: An inserter and the model in which there is a mapping of tracks to the model tracks.

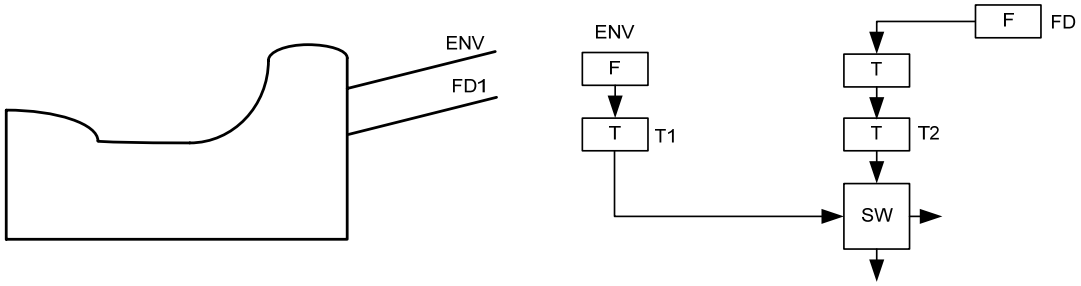


Figure 1-29: The model of the paths that is used for machines with different feeder configurations.

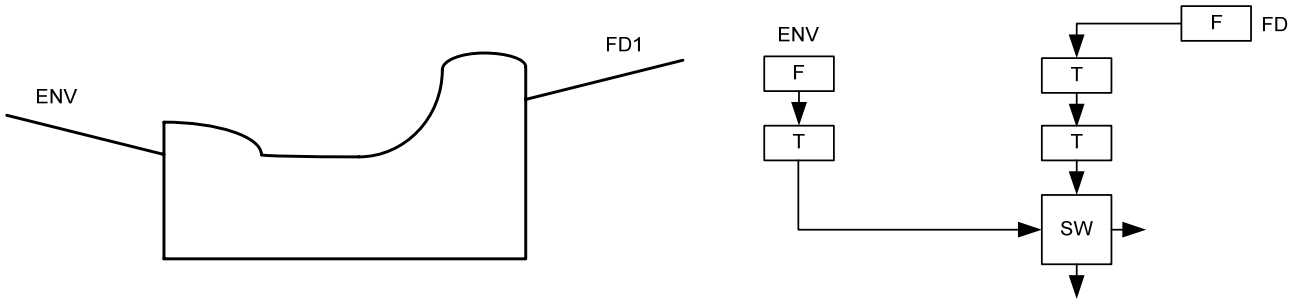


Figure 1-30: The model of the paths that is used for machines with different feeder configurations.