ISSN 0280-5316
ISRN LUTFD2/TFRT--5786--SE

Integrating Exception Handling in Machine Development

Johan Henricson
Daniel Allansson

Department of Automatic Control
Lund University
February 2007

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
February 2007

Document Number
ISRNLUTFD2/TFRT--5786--SE

Author(s)
Johan Henricson and Daniel Allansson

Supervisor
Mattias Wallinius at Tetra Pak D&E in Lund
Karl-Erik Årzén at Automatic Control in Lund

Sponsoring organization

Title and subtitle
Integrating Exception Handling in Machine Development (Integrerad händelsehantering och maskinutveckling)

Abstract

In modern batch plants, alarm floods overwhelming the operator is a common problem of ever increasing severity. As plant hardware increase in complexity and performance, measures such as downtime and meantime between failures, increase in importance and the need for a well-functioning exception handling routine therefore grows critical. In order to minimize downtime and hence boost production efficiency, it is important that faulty or unexpected behaviour is noted early and diagnosed accurately. If it is, the machine or its operator can deal with it fast and accurately, perhaps even while production continues. If it is not, however, it basically means two bad things, the first being unnecessary production stops and the second a flooded alarm list. The alarm list is the screen on which all exceptions are listed and which is meant to tell the process operator what is wrong and what he should do about it. If no care is taken about it, the list will be flooded since one key exception will cause several others, leaving the operator with the quite unpleasant task of identifying what went wrong and how he should fix it. This master's thesis, carried out at Tetra Pak in Lund, Sweden, presents a well-structured exception handling method and a way of linking it to a workflow.

It uses advantages of reusability, modularisation and linking to almost any structural model such as a Function Means Tree (used in WCE) or a UML-model. The linking has several advantages minimizing the work and improving evolutionary possibilities. Examples are being made for illustrational purposes but no implementational efforts or issues are addressed. A well-defined structure for information flow is also suggested to provide modular thinking that, for example, prepares the possibility to collect data for statistical analysis and such.

Keywords
Alarm management, Error handling, Alarm handling, Root Cause Analysis, Modularisation, World Class Engineering, Function Means Tree

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
70

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through:University Library, Box 3, SE-221 00 Lund, Sweden Fax +46 46 222 42 43

Acknowledgements

The work on this master thesis was carried out at Tetra Pak Research and Development in Lund in cooperation with the Department of Automatic Control at the Lund Institute of Technology (LTH), Lund University. The authors wish to express great gratitude to supervisors Anders Sundberg at Tetra Pak and Karl-Erik Årzén at LTH and to Mattias Wallinius, co-supervisor at Tetra Pak. All three are men of impressive knowledge, thank you for sharing some of it with us.

The job was done in parallel with two other jobs performed by fellow LTH students Maja Arvehammar, Johanna Byrlind and Kajsa Karlsson. Most of the work was done in our cosy basement room together with Maja's ridiculously noisy test rig and Mattias' "HMI Task Force" including special agents Daniel, Mattias and Lama. You've all been great, thank you for letting us be a part of your bunch and for the Friday breakfasts!

We also want to say a collective thank you to all other Tetra Pak staff members that has been nothing but helpful and patient with us and our stupid questions.

Malmö, early February 2007
Johan Henricson and Daniel Allansson

Contents

Table of figures

Nomenclature

Abbreviations

CED	Cause-and-Effect Diagram
CET	Complete Exceptions Table
ETA	Event Tree Analysis
FME(C)A	Failure Mode Effect (and Criticality) Analysis
FMT	Functions Means Tree
HMI	Human Machine Interface
ID	Interrelations Digraph
MFM	Multilevel Flow Modeling
PLC	Programmable Logical Controller
RCA	Root Cause Analysis
RET	Root Exceptions Table
RPN	Risk Priority Number
TAA - SCAP	Tetra Aptiva Aseptic - Second Carton Aseptic Platform
UML	Unified Modeling Language
WCE	World Class Engineering

Definitions

Complete Exceptions Table
A *Complete Exceptions Table* is a table where all identified Exceptions will be stored by the Exception Handler.

Component
A *Component* is a part of the machine (or rather a part of the batch flow in the machine) with a clear and specific goal and likewise clear requirements.

Dependency Arcs
Dependency Arcs are the internal connections, each describing a downstream dependency and upstream Exception propagation, meaning that the source block must fulfil its goal in order for the receiving one to be able to do the same.

Dependency Structure
A *Dependency Structure* is a structure with Goal, Functions, Subfunctions, interconnections and requirements.

Exception
An *Exception* is an indication of a state or an occurrence in the system that requires the operator's attention. Exceptions range from errors and failures to preventive management assistance.

Function

A *Function* is a block inside the Component having concrete Goals to be fulfilled. The Function block may contain both Machine and Process Exceptions. Every Function may, and typically will, have dependencies of its own and will therefore require input from other Functions, Subfunctions or other Components directly. The Functions in the set may hence be interconnected with Dependency Arcs in various ways to describe the internal dependencies. It is crucial that these interconnections are carefully drawn as they play a large part in the actual real-time Root Cause Analysis during runtime.

Function Modules

A *Function Module* is a part of the machine.

Goal

A *Goal* is a brief description of the component’s purpose and should answer the questions “What is the purpose of this Component?”, “When does it have to be fulfilled in order to ensure proper process operation (often equal to one or more operational states)?” and “Where, physically, is it supposed to deliver its results?“.

Indication

An *Indication* is a symptom of some unexpected event in the machine. An indication will typically refer to a sensor reading exceeding some predefined value but does not have to be that rudimentary. Activation of an *Indication* is referred to as ‘triggering’ it and the states of an indication are hence ’triggered’ and ‘untriggered’.

In

An *In* of a Component is a link to the Goal of another function that is located before the current in the Component cascade that is the plant.

Root Exceptions Table

A *Root Exceptions Table* is a table where the Exceptions that are judged to be Root Causes are listed by the Exception Handler.

Subfunctions

A S*ubfunction* is a special case of a function that relates directly to purely physical Components in the machine. Their purpose is to provide prerequisites for the functions to work properly. The Subfunction block contains only Machine Exceptions.

Triggered Indications Table

A *Triggered Indications Table* is a table where all the triggered Indications will be listed.

1. Introduction

This thesis is part of a large project currently going on at Tetra Pak with the overall goal of evaluating new approaches to constructing the software parts for new generations of filling machines. The evaluation aims to find out what, if anything, could be won from implementing control system, operator interaction and such in an object-oriented open-source programming environment rather than the traditional Programmable Logical Controller (PLC)-languages of the IEC61131-3 standard and how development efficiency could be improved by re-using solutions from earlier projects and tests. This thesis is one of three linked together as a part of this evaluation. The others are “Object Oriented Automation System” (Arvehammar 2007) focusing on the object oriented PLC-implementation and “New Web based HMI portal for Tetra Pak Equipment” (Karlsson, Byrlind, 2007) that focuses on the foundation of, and a tag-lib for, a web-based Human-Machine Interface (HMI). The objective of this particular thesis is explained in Section 1.1 below.

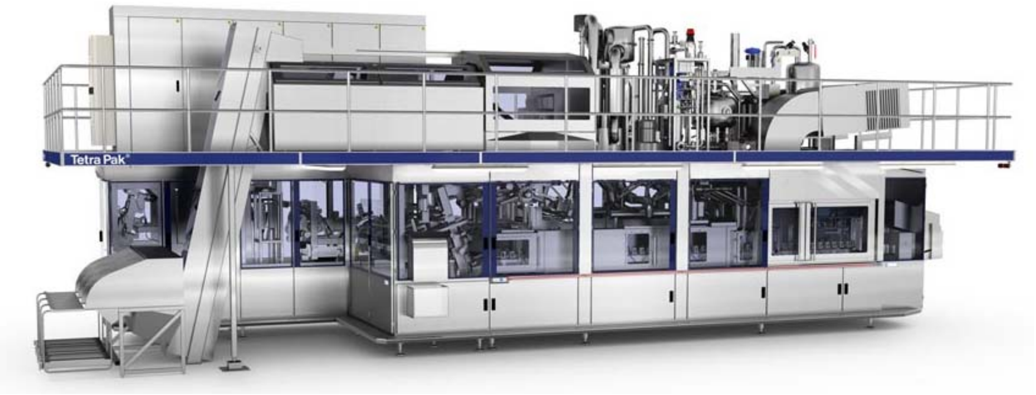


Fig. 1.1 The Tetra Aptiva Aseptic packaging machine (From Tetra Pak).

While the projects are focused on future concepts and generations of machines still to come it is useful to have existing hardware to study and relate to. For such reference this thesis uses the Tetra Aptiva Aseptic (TAA) machine showed in figure 1.1 above. The TAA is one of Tetra Pak’s latest plants for package forming and filling. The bottle it produces is a laminated paper bottle with a plastic top used for milk and still drinks e.g. juice. The TAA is further explained and discussed in Chapter 2.

1.1 Aim of Thesis

The main goals of this thesis are to find new ways to increase efficiency in construction of new generations of packaging machines and decrease both the number of process production halts and the mean time for each such halt in order to minimize downtime and hence boost production efficiency.

Possibility for minimizing downtime is provided by finding out a Root Cause Analysis method, and suggesting actions to take in order to obtain a suitable set of alarms for it. The suggested method also includes functionality to keep up to date when changes are being done to the machine. The solution can be extended with e.g. a Diagnosis block for more predictive methods or statistical analysis to be added later on if it should be desired.

The thesis also suggests ways to improve both development and run-time efficiencies by integrating their respective developments and linking it to existing workflows such as the World Class Engineering (WCE) described in Section 1.4.

1.2 Product Development and Run-Time Efficiency at Once

Evolution in industrial manufacturing production in general and in the packaging business that Tetra Pak is a part of in particular is characterized by two things. Firstly, the development process is required to be faster and cheaper and, secondly, the resulting machines are required to be more robust and perform better.

The demands are to some extent commonly viewed as each others opposites – a machine could be developed faster but it will be marred by poorer performance figures or vice versa. In this thesis it is shown that it is not necessarily so.

Experts and common experience agree that the earlier in the process a flaw or malfunction can be detected, the less expensive it is to handle. The difference is in the common case estimated to approximately ten times between two adjacent development phases such as design-construction or construction-prototype (Johansson, 2003). This is illustrated in figure 1.2.

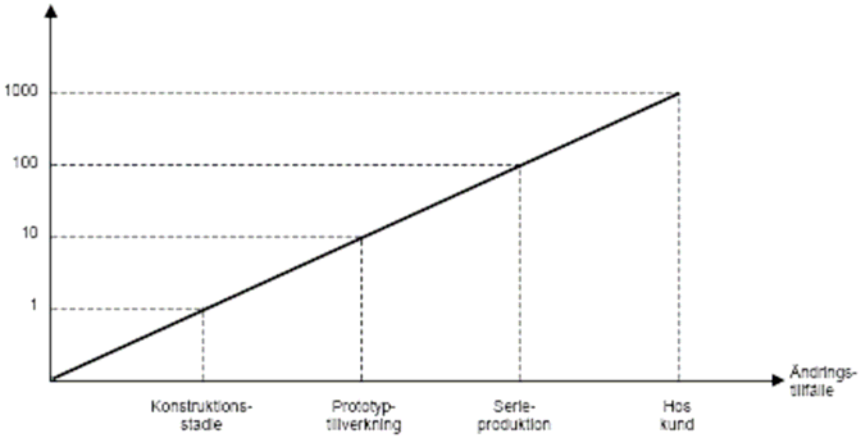


Fig. 1.2 Handling cost as a function of the time it is handled, (From Johansson, 2003)

A reasonable conclusion that could be drawn from this it that effort should be made to be able to spend as large amount of the correctional patching work as possible in the early, inexpensive part of the curve. That also goes well together with modern work flow ideas and manufacturing methods such as the “Toyota Way” and Lean Enterprising described in (Morgan, Liker, 2006) and WCE (described in Section 1.4) that is currently being implemented at Tetra Pak. None of these methods specify the error or exception handling concept or its proceedings as they are very loose on technical details.

One way of identifying possible failure modes at an early stage in order to be able to fix or handle them, and at the same time integrate the development of an exception handling system with the Lean concept “Do more with less” (Morgan, Liker, 2006) and the WCE methodology “Think first, then act” is presented in this thesis.

The suggested approach uses the advantages of re-usability (to avoid inventing the same thing twice), modularisation (to avoid revalidations and allow evolution) and using one and only one central model to represent an entire machine. Together they form a powerful and effective system for identifying hazards at an early stage and for exception handling during run-time. The method merges process development power with run-time effectiveness in a convenient way meaning increased cost efficiency throughout the production chain.

1.3 Process Modelling and Modularization

It is becoming common practice in industrial product development of today to build some sort of software model. The increasingly complex constructions need to be modelled in order for the work to be arranged and evaluated, ensuring that construction is done in a structured and effective way. It has also been common to use software models for simulations and evaluation prior to the actual construction.

Splitting the large model structures into smaller, more manageable, modules is also useful for exchanging parts of machines and allowing them to evolve. Also the visualization and handling of models and plans are simplified by the modules being studied separately from its surroundings. Reasons for modularization also include that some industrial machines manufactured today, within areas such as food processing and drug handling, have demands on them to be validated in order to ensure customer safety. Modularizing models and machines allow parts and functions to be validated separately, before adding them to a machine. Another reason is that this makes it possible to change, exchange and add modules afterwards as the models grow. Since the modularization makes the model more useful this is what development is heading for.

The implementation of a modern exception handling routine is often based on a model of the machine: that makes it possible to use the existing model or, even better, integrate the exception handling with the existing model. This way it is possible to represent the entire machine and its components and functions, such as the exception handling, as different views of a single structure. That makes it remarkably easier to keep that single structure up to date as well as re-using and exchanging solutions or components.

There are different methods for using modularization for the software system; two used methods are the Unified Modelling Language (UML) and the Functions Means Tree (FMT) (Malmqvist 1995), (Nordqvist 2003). Both these methods, as well as most other models, describe the smaller pieces of the machine from what means they fulfil. Regardless of whether it is describing the direct means to fulfil or a class in the software system, the general description will be the same.

The UML-model is one of the best-known software modeling techniques. Essentially, the UML-model is a complex model system to represent the software of projects or machines. It includes several different views such as the most basic, the class diagram, representing the classes in the project and their connections to each other. This is the view referred to when UML is mentioned in this thesis.

The FMT approach differs slightly in focus from the UML in the model system. This is a well used method when designing and a development is done in projects. It is used for development because it does not focus directly on the real software classes as the UML do, but rather the task that each piece has. When brainstorming about how the project should work the FMT works as a good model to put the different thoughts together to get a structure.

The FMT approach starts from something that is to be done, like a task to fulfil, called a function. The function can be implemented in different ways, and therefore each function should be given different possibilities to be implemented, called means (how to do it).

Every function has certain conditions that need to be fulfilled in order for the function to be considered successful. A mean can have other functions or means as conditions to itself and hence the function-means tree is built as shown in figure 7.4.

1.4 World Class Engineering

World Class Engineering (WCE) is a concept suggesting how the product development process should be carried out. It is an internal Tetra Pak concept and is described only in internal documentation.

The work method consists of several steps and is complete from customer requirements to a final product. The key ideas are to increase the desktop work and decrease the actual testing on built machines and test rigs, and to use knowledge gained from earlier machine developments in effective ways comparing them to newly developed approaches.

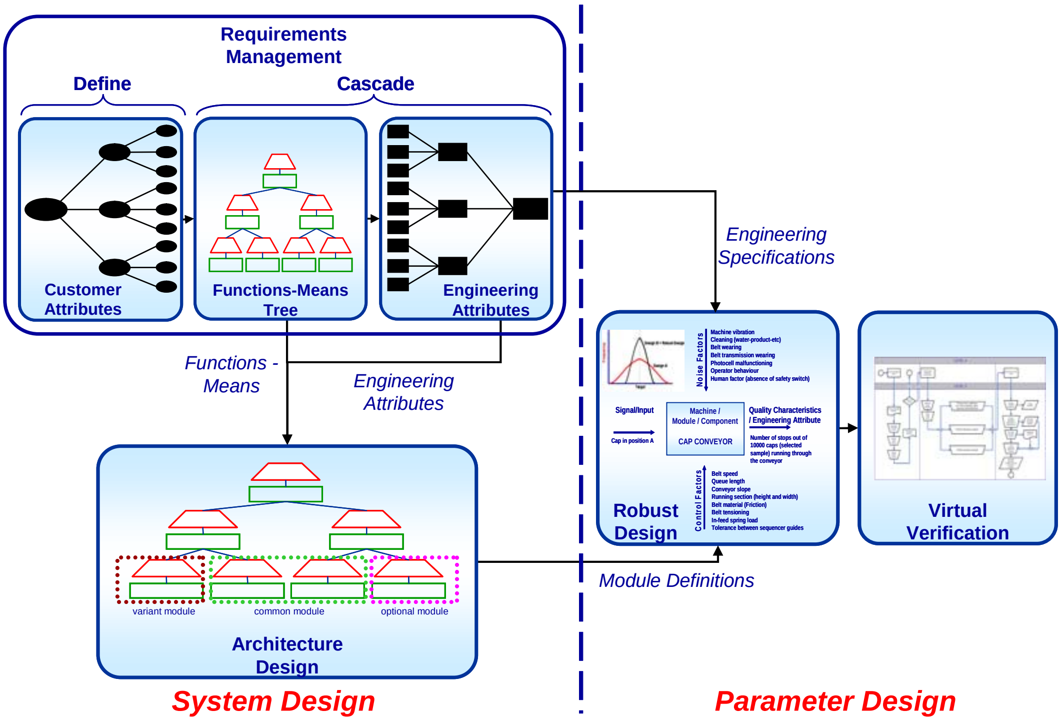


Fig. 1.3 A view of the WCE scheme, (From Tetra Pak internal material).

The work in WCE is done stepwise, see figure 1.3. From the customer requirements a Function-Means Tree is set up where the main parts needed to fulfil the requirements are specified. With the FMT and knowledge of earlier used or constructed means and functions, which have been documented in the Engineering Attributes archive, a first architectural design of the machine is done. By simulation, or some other way of evaluation, every part of the machine is being investigated and tested. When it fulfils the requirement from both the customer and the Engineering Attributes a virtual Function Module is built. With this, the robustness is investigated to find out what variables are affecting the machine robustness and which of them are possible to tune to make the module as robust as possible. Knowledge from the engineering attributes is used along the results from the architectural model that was first built.

When the parameters have been set and the robustness has been optimized there is only one more thing to do before building a real machine. The remaining step is to do a Virtual Verification of the machine, if this fulfils the requirements from the customer and is working well, the real machine can be built.

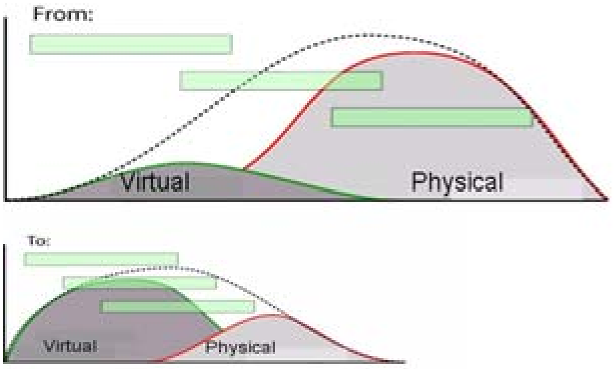


Fig. 1.4 Time and cost reducing because of WCE,
(From Tetra Pak internal material).

The result of using this method is that the simulations and calculations prior to putting the machine together increase in relation to the running-in time that will decrease drastically and therefore shorten the total development-time (fig. 1.4)

1.5 Outline of Thesis

In the first section, chapters one and two, an introduction to the thesis and the problem is given. The current conditions and the hardware that the work is performed on are presented.

In chapters three and four some initial thoughts and definitions are formulated and the strategy is set. Different methods and procedures for performing parts of the work are evaluated and presented in Chapters five and six leading up to the majority of the actual work that is done and presented in Chapter seven and eight. Chapter nine sums up the work and concludes the thesis.

2. The Tetra Aptiva Aseptic

2.1 Brief Description of the Tetra Aptiva Aseptic Filling Machine

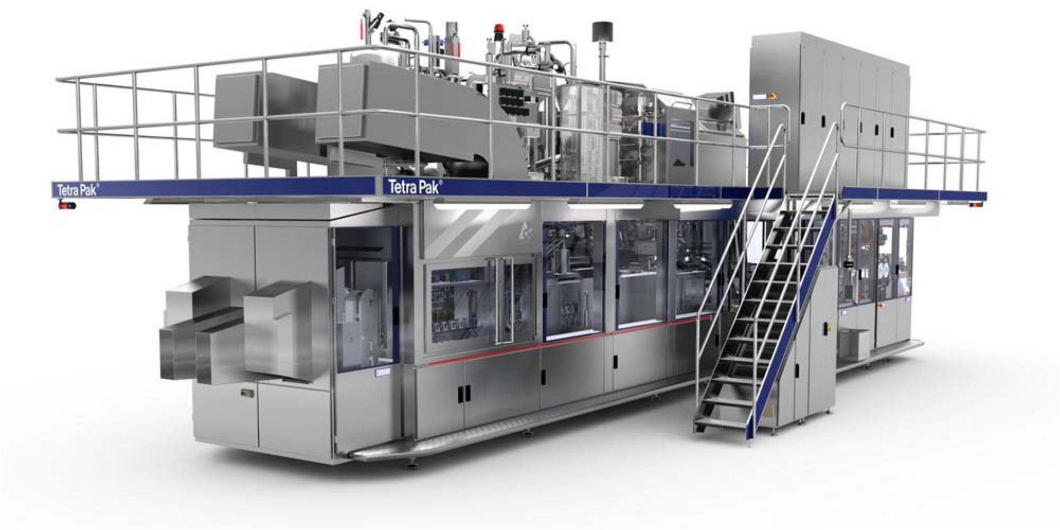


Fig. 2.1 The Tetra Aptiva Aseptic packaging machine (From Tetra Pak).

The Tetra Aptiva Aseptic (TAA) machine (fig. 2.1), also known as the SCAP, is one of Tetra Pak's latest plants for package forming and filling. It produces a bottle from a laminated paper sheet that forms the body and a prefabricated plastic top that is attached to it. The tops for the machine are moulded in a separate plant and come attached in pairs, the machine separates them and attaches them to the package as a part of the bottle making. The bottle can be used for still (non-carbonated) drinks such as milk, juice or still water.

The TAA machine creates complete, filled and sealed bottles from the special paper and the plastic tops. The machine forms, sterilizes, fills and seals the bottles as they advance down the production line. As there is several different production steps it is possible to break down the production flow in Function Modules (fig. 2.2.).

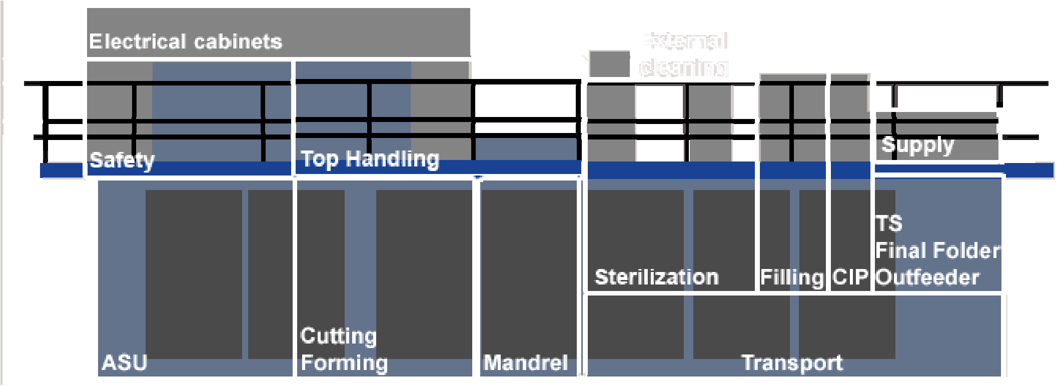


Fig. 2.2 A view of the Function Modules of the TAA, (From Tetra Pak HMI).

The machine has four production lines in the major part of the machine. To have four production lines is equal to having four identical machines working parallel with each other. They are, however, dependent of each other since the physical machine in practice has two lines working in parallel, each line producing two bottles at a time. The final number of effective production lines is hence four. The four-line production begins when the sheets have been cut and the bottle begins to be shaped. The paper preparation before this is single-lined. In order for the paper preparation to keep up with the four-lined part it has to work at a four times higher pace. Between the one-lined and four-lined parts of the machine there is a buffer connecting them. This buffer is called the Magazine and will be described in further detail later in this chapter since it is used throughout the thesis for tests and references.

2.2 Alarm Handling on the TAA

It is in practice impossible to avoid unwanted or unexpected events from occurring when a machine is running. Such events could include the paper roll running empty during production or something going wrong with the machine e.g. a motor that stopped. To know when something has happened and what has gone wrong the machine has sensors that check whether process states are within specified boundaries or not. If not, then the operator handling the machine should be notified of this and, depending on the severity of the event, the machine should possibly be stopped automatically.

When a machine is built the alarm implementation with necessary sensors and software traditionally tend to come second. The consequences of this, rather optimistic, method for building machines are many redundant alarms and alarms originating from sensors placed where there was room for them rather than where they were actually needed. Like many recent industrial plants the TAA machine has very many alarms and sensors, partly due to poor Alarm Handling.

When an error occurs in some part of the process it is hardly ever the only one, it will often influence its surroundings causing other secondary errors elsewhere. If, for example, the paper roll is empty eventually the whole machine will be affected because there is no more paper to make bottles from. Therefore, when an error occurs, there will often not be only the corresponding alarm, but several more as a consequence of the error. The notification of these alarms that have been raised is in fact correct but the cause of them is a single root error and not all of the triggered ones.

The operator presentation of what alarms have been raised is very simple on the TAA, all raised alarms are shown in chronological order. When the operator is notified of the alarms he has to figure out the root cause and take the appropriate corrective actions to solve it himself. Nowadays the operator usually assumes that the first alarm time-wise is the Root Cause or, if it is an frequently occurring alarm flood, the operator might know by experience what the Root Cause usually is. This method is highly unreliable since the root error it is not always noted first, neither is the operator knowledge fully reliable. It is, however, in practice the most frequently used method as it is the best working one up to date.

2.3 Alarm Structure

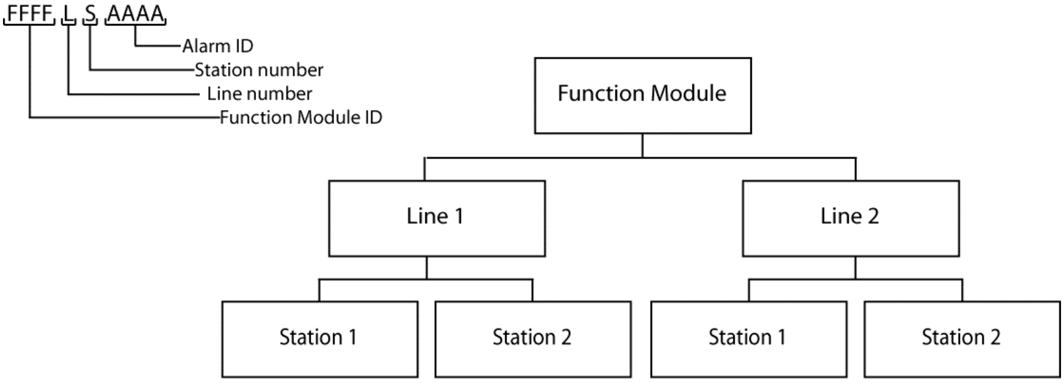


Fig. 2.3 A Function Module overview.

To identify an alarm with regards to its nature and origin every alarm has a ten-digit identification number. As mentioned earlier in this chapter the machine is divided into function modules. The first four digits (from the left) in the id represent what function module that the alarm belongs to. The next digit represents which of the machine's two physical lines the alarm comes from. Each line is also divided in two parts, called stations, which are represented by the sixth digit in the id. This structure for a function module is shown in figure 2.3. The last four digits in the id represent the alarm's unique alarm description.

Different alarms affect the process in different ways and with different severity; therefore each alarm is assigned a colour code representing how severe it is. In the TAA machine, and in general, the colours are blue, yellow and red. The blue alarms are mere notifications to the operator and not really a fault. The yellow alarms represent events severe enough to cause the machine to stop in a controlled way and require the operator to take some action before starting it again. The most severe alarm group is the red one. If a red alarm occurs the machine emergency stops immediately. This group contains alarms corresponding to situations where human danger and severe fault in the machine are at stake.

All alarms from a machine are today stored in a database from which the operator receives them. It is also possible to access the alarm history and other data from a net-based client.

2.4 The Control System of the TAA

To have a machine like the TAA running requires a large and complex control system. The control system masters all steps in the production and synchronizes them with each other. The control system is implemented on a PLC and communicates with the operator through an HMI. The HMI is the interface to the operator. In the control system the beginning of the Alarm Handling takes place as that is where the alarms are raised and possibly stops the machine. The rest of the Alarm Handling is done in the HMI.

As mentioned in Chapter 1 work is currently being done to evaluate the possible advantages of changing the PLC-system into an object oriented one. If it is implemented it is supposed to handle a larger part of the Alarm Handling than today because the Alarm Handling resources will become a part of the design procedure.

When the TAA machine is running it passes through several phases or running modes. First there are some phases preparing the machine for production, such as ”Sterilize” and ”Tank Fill” in which the filling section of the machine is sterilized and the tank for the liquid that is to be bottled is filled up. Then there is a production phase and finally there are some closing phases in which, among other things, the machine is cleaned.

In each of the phases mentioned above there are states that are always gone through. When an event occurs, the machine enters certain stop-states holding it. Depending on how severe the event is different stop-states are entered, some of them allow the machine to finish its current cycle halting it under controlled circumstances while other make it stop immediately in order to avoid serious harm to humans or machine parts.

2.5 A Closer Look at the TAA Magazine

The Magazine works as a buffer between the single-line paper preparation and the four-line forming and filling parts of the machine. This is, as mentioned earlier in this chapter, needed in order for the parts to work together. The advantage with the Magazine is that it can build up a buffer of sheets to use e.g. when a roll of paper is empty and it has to be exchanged with a new one. The splicing takes some time and during that time no sheets are being cut. If there is a buffer of sheets in the Magazine the four-line production might not have to stop because of an empty paper roll.

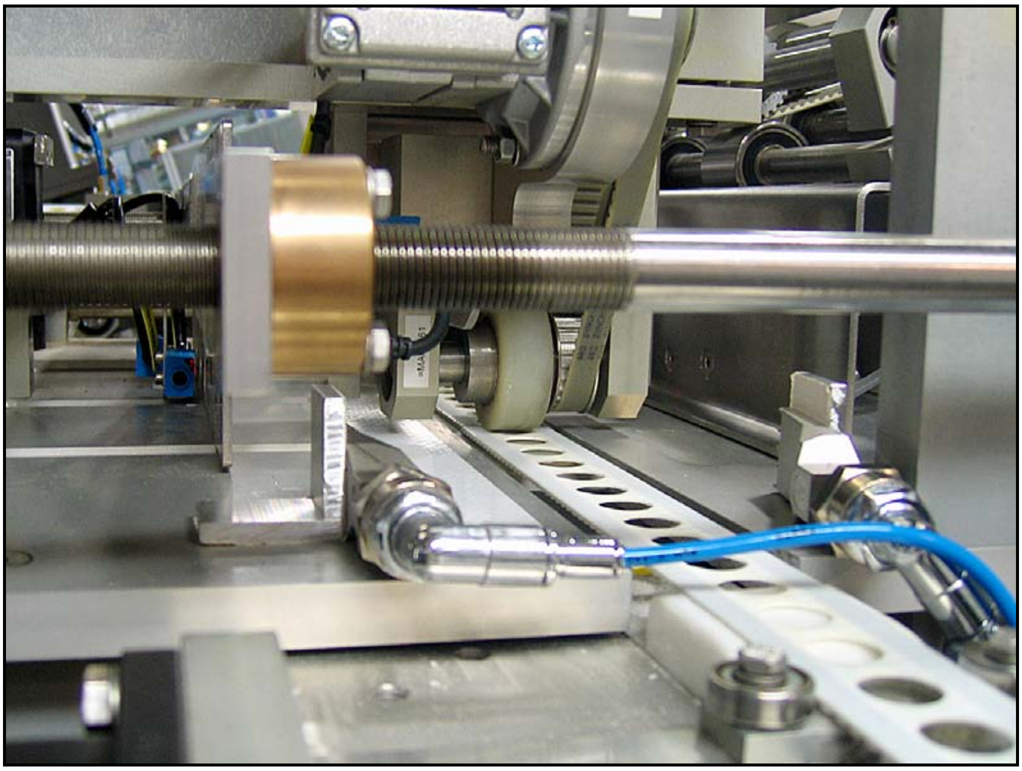
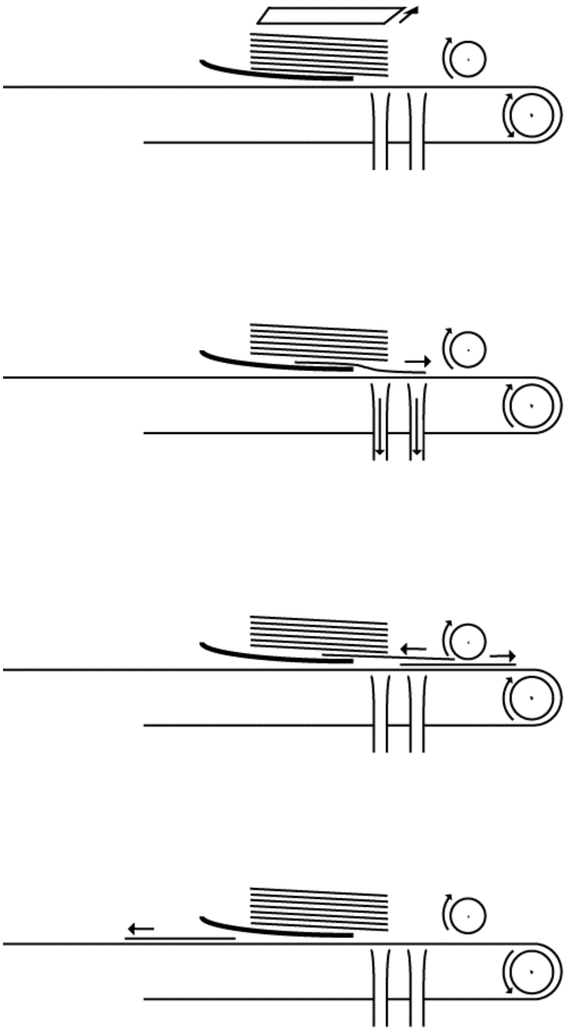


Fig. 2.4 The transport belt and wheel in the Magazine

When sheets leave the cutting machine they are stacked on top of each other in a buffer that is filled from the top and release sheets from the bottom. The bottom sheet is pulled out of the buffer by a conveyor belt. In order to make it stick to the belt, that has holes in it (fig. 2.4), a vacuum pump sucks the paper down onto the belt, enabling it to pull the paper out of the buffer. To assist the belt there is a wheel above it that spins in the opposite direction of the belt in order to prevent multiple sheets from coming out of the buffer at the same time. When the sheet is out of the buffer, the conveyor belt changes direction and pulls the sheet back the other way, under the buffer, and into the four-line part. The process is illustrated in figure 2.5 below.



1. Cut sheets of paper arrive from the cutter section and are stored in the buffer

2. Vacuum pulls a sheet to make it stick to the conveyor belt. The belt pulls the sheet out of the buffer

3. In case more than one sheet was pulled out, the surplus is pushed back in the buffer by the sheet separator

4. The conveyor belt changes direction and the single sheet is delivered to the next component

Fig. 2.5 TAA Magazine function

3. Introduction to Exception Handling

In this thesis *Exceptions* will be the common denotation for events that occur asynchronously in the machine. Exceptions are either unexpected or expected but not predictable in time, and may require operator or control system attention. The exceptions set include error notifications and alarms but is somewhat wider and also include messages on occurrences of a more informal nature. Such exceptions could be used for operator notifications, data logging or as base for a statistical analysis for predicting and preventing future malfunctions.

The research in the area focuses mainly on the alarm and error handling part, as that tends to be the most critical and challenging subject. A large part of the exceptions originate from undesired behaviour that either has occurred or is about to. As a result of this, exception handling could essentially be seen as equal to what is usually referred to as Alarm Handling in the literature but with a few additions. All results from this thesis could therefore be used in any Alarm Handling solution as well as in the one presented here.

In exception handling a triggered sensor is not equal to an exception being raised. Triggered sensors cause *Indications*, equal to what is usually referred to as alarms, which in turn may cause exceptions. More than one Indication may originate from a single sensor as Indications are to be viewed as conclusions being drawn by studying a sensor's output. A temperature sensor may, for instance, be associated with one "temp. high"-Indication and one "temp. low"-Indication. From one or more raised Indications a conclusion on what exception has occurred can be drawn.

3.1 The Benefits of Exception Handling

Deciding on implementing, or even evaluating the implementation of, a sophisticated alarm or exception handling is a large and often rather tough step to take for any industrial machine manufacturer. As it is a new discipline, which in many cases has not even been considered before, it is accompanied by new costs. This could be hard to motivate as the machines have in fact been working in some manner even before, without the extra expense. The way things are heading, though, with more complex equipment, it must be done sooner or later. Considerable gains could be obtained, though, both in terms of minimizing downtime and increasing system robustness. Combined with the fact that a major part of the structure for the implementation of such a handling could in fact be obtained rather easily through integration with a modern workflow such as the WCE (described in Section 1.4) this makes it the authors' complete conviction that now is the time to take the step.

The largest advantage that an exception or alarm handler can provide to a system is saving of time and money by making production halts fewer and shorter. There are often exceptions raised without reason and when a real exception is raised multiple other exceptions are also raised and shown to the operator. If only correct alarms are raised and only the root alarm is shown when they occur the down-time would decrease.

Flexibility, or lack thereof, regarding both functionality and machine changes is generally a problem for the exception handler when new requirements are put on the machine and the human interaction/data presentation. The handling (if there is one) is based on the original configuration and is often very non-flexible. With the implementation of a well-structured system, of which the exception handler is a large and central part, this flexibility can be obtained without loss of the original functionality or usability. Say, for example, that the exception handling should be expanded with a diagnostics feature or that some major or minor part of the machine needs to be modified. A well-structured exception handler can handle such changes and they can be implemented without any major effort.

3.2 Removing or Handling Exceptions

The designer of a machine obviously has large influence on how well the machine will work when it is finished. This is also true regarding the exception handling. A risk analysis as described in Chapter 5 is performed to identify possible hazards and errors. Every identified error is then worked with and it is evaluated how to handle it.

There are three different actions to take for handling the identified errors. The first and most efficient way, if possible and financially reasonable, is to make changes to the machine so that the error won't occur at all. The second alternative is to build the machine so that it handles the error by itself. The last alternative is to allow the error to occur, stop the machine and then correct it manually.

To build the machine in away so that the Exception won't occur at all obviously seems like a good alternative, but it is not optimal in all cases. It has to be considered how often the Exception occurs and how much effort is needed to make a design that eliminates it completely. If the exception is rarely occurring, it might be more efficient to accept it when it occurs and handle it manually.

For frequent exceptions that are hard to prevent completely, it is generally more effective to extend the machine to accept the exception but handle it on its own. An example of this is a disposal of detected faulty sheets from the paper cutting that is done before the approved sheets are stored in the Magazine. With this method there will be some waste, but it might still be more efficient than having the machine stop and the exception fixed manually.

Which of these methods is most appropriate to use depends on how often the exception occurs, how difficult it is to change the machine so that it can handle it and how hard it would be for the operator to take some corrective action to solve it. For each of the identified exceptions in the machine this optimization has to be carried out. If this is done properly, much work and time can be saved.

So far it has been assumed that the machine is still to be built. If the machine already exists and there is an interest to implement exception handling on it, it can be done but preferably it should be done during the first design phases.
Changing an existing machine sets some rather strict limitations on how extensive changes can be made; the machine might not be able to change as much as one would have wanted.

4. Information Flow

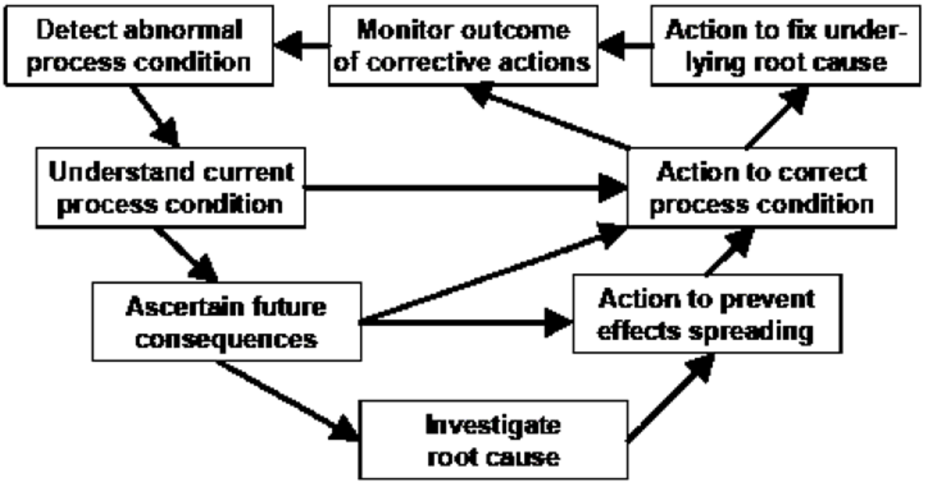

Detect abnormal process condition
Monitor outcome of corrective actions
Action to fix underlying root cause
Understand current process condition
Action to correct process condition
Ascertain future consequences
Action to prevent effects spreading
Investigate root cause

Fig. 4.1 A view over how operator reacts on incoming alarms,
(From Smith, Howard, Foord, 2003)

The size of a control system in a packaging machine (or any industrial machine) is often very large, and they don't tend to get any smaller over time. To have a structured implementation and overview of the Exception Handling for a control system of this size the control system needs to be well structured. The exception handling can often be separated in different parts or steps in the flow.

In (Smith, Howard, Foord, 2003) it is discussed how an operator is reacting and thinking when alarms are noticed. This is a good thing to keep in mind when suggesting a stepwise exception handling. Figure 4.1 shows how **(**Smith, Howard, Foord, 2003) describe the flow of an operators mind when the alarm is noticed and from this figure the model for the exception handling in this thesis is based, showed in figure 4.2. The suggested flow is in three steps, one preparation step, where the first filtration is done, one Root Cause analysis step and finally one block where action is taken.

The exception handling starts with one, or more, incoming triggered indications from different sensors in the machine symbolizing that something is not as it should be. A first selection of the indications are made, if a severe fault has occurred (a red exception/indication) the machine should be stopped immediately. If there are indications coming up twice the doublet is neglected. The steps mentioned up to this point are performed in block A in figure 4.2 above. This is a kind of preparation for the Root Cause Analysis that is done in the B block. This block focuses only on finding the Root Cause. The last steps in the exception handling is in the third and last of the mandatory parts in the flow.

In this block, C, some action should be taken, depending on the properties of the identified Root Cause. Either it should be presented to the operator or the machine should take some action of its own. Along with these steps in the procedure, logging should be made. The incoming indications, the raised Exceptions and also the Root Cause should somehow be logged, e.g. in databases.

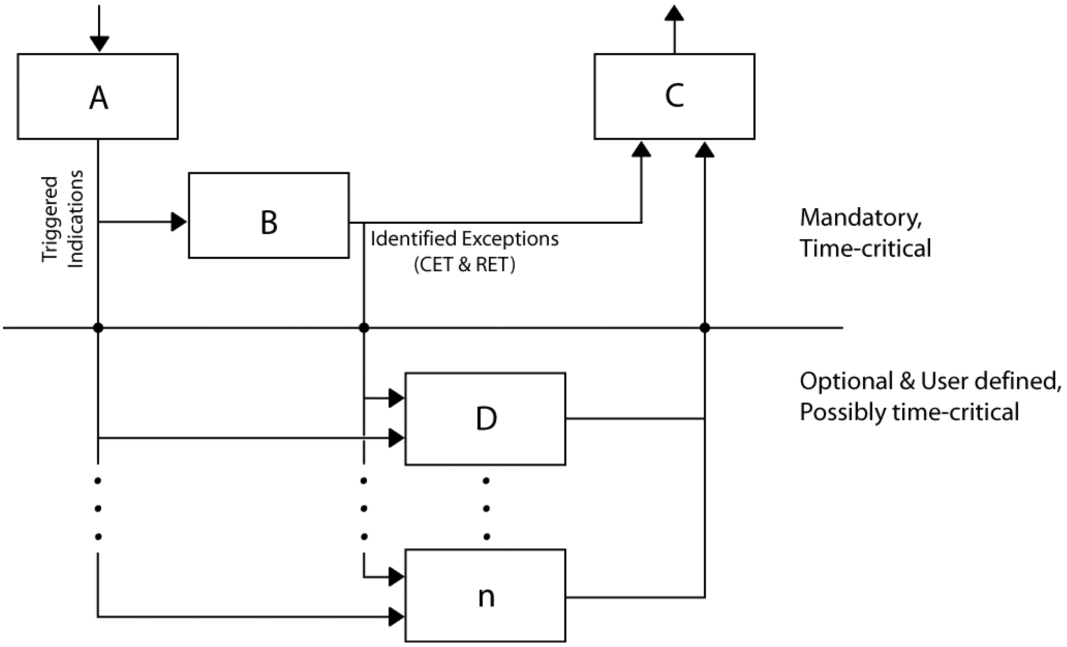

A
C
Triggered Indications
B
Identified Exceptions (CET & RET)
Mandatory, Time-critical
D
Optional & User defined, Possibly time-critical
n

Fig. 4.2 A view over the suggested information flow

These three parts are the mandatory parts that are required for the system to work as it should, but some day it may be of interest to add some more advanced analysis block. Examples of this could be a more advanced Root Cause Analysis or a Diagnosis block where statistical analysis is used. A Diagnosis block could tell the operator that something is going to happen in a while or that something is wrong very often and probably some underlying fault to this based on statistical analysis of the indications and other interesting data from the system.

5. Risk Analysis Methods

A large part of the exceptions that are to be identified in the system are indications of things going wrong. They indicate either some more or less severe machine hardware malfunction or some process disturbance that is due to such malfunctions, material failure or some behaviour overlooked in the construction and design. Ideally one would list and implement identification possibilities for every such possible exception at every location in the machine. Obviously, that is impossible as all states and quantities are not measurable and as it would be extremely costly to implement such a solution. Costly both in terms of sensors and other required hardware and in the virtually infinite development cost that would be associated with performing an analysis thorough enough to identify almost every exception.

The trade-off between detail in exception identification and economy should be done with care and it is important to identify what parts of the machine or production chain are performance hotspots. That is, areas that are crucial for machine function and where unexpected behaviour is costly in terms of machine downtime or production disturbance. There are many ways of doing the analysis (Larsson, Svensson, 2000) but classical risk analysis methods such as the Failure Mode Effects (and Criticality) Analysis (FME(C)A) and Failure Tree Analysis (FTA) prove quite useful when performed at component level and are briefly described below. Both are inductive, qualitative methods meaning that they promote the “Learn and Prevent rather than Find and Fix”-approach and that they can be performed at an early stage, before anything is actually built.

The first step should be to identify what components are most critical to machine functionality and then the actual analysis should be performed on every component with the level of detail depending on the result from the first step. A first, rather ‘hand wavy’ analysis could and should be conducted already in the planning stage (in WCE meaning the FMT construction) where different means are compared to each other.

The analysis methods are rather straightforward but are ultimately based on a “structured form of brainstorming” (Doggett, 2004) and should therefore be performed by a group of people that share good and thorough knowledge of the component in question (Johansson, 2003), but come from different backgrounds as that generally stimulates thinking in a creative way. Breaking down the component in as small pieces as possible is often useful for seeing details that could fail. It takes longer time though and may make it difficult to see higher-level connections that are easily overlooked. It is, hence, important to find a suitable level.

5.1 FME(C)A

An FMECA is carried out in order to gain knowledge of what the risks are in processes or plants or as is the case in this context what exceptions may arise in a given function or machine part and how serious impact they may have on system performance. The method is based on a table being filled out column by column for each exception that can be thought of. If a column for Exception Criticality is included the method is usually called FMECA, otherwise it is a form of basic

FMEA. The list of headings that are to be filled out vary somewhat depending on who specified them but a rather general version, appropriate for the application in question is presented by (Johansson, 2003) and shown in figure 5.1. According to that, the headings are:

1. Possible Failure Mode. Brief description of an exception that might occur, including where it is located and how it failed.
2. Possible Failure Effect. What possible effect(s) the failure might have.
3. Possible Cause: List of possible causes for the failure. May be more than one.
4. Existing Tests: List of what tests are being done that might reveal the exception in question.
5. Probability of occurrence: Index describing how frequent the exception might be if untreated. 1 means barely any probability and 10 very likely to occur.
6. Criticality: Index describing how severe the exception would be if it occurred. 1 is equivalent with not at all severe and 10 with very severe – apparent risk for serious personal or machine injury.
7. Probability of detection: Index describing how probable it is that the exception is *not* noted before it is too late. 1 corresponds to very unlikely that is goes by unnoticed, 10 very likely that it is missed.
8. Risk Priority Number: The indexes in columns 5, 6 and 7 are multiplied, resulting in an Risk Priority Number (RPN) in the range from 1 to 1000. Higher RPN mean the exception should be handled more carefully.
9. Handling method: How the exception should be handled. Based on the RPN in column 8 one of the three actions described in Section 3.2 is chosen.

The analysis is divided into three separate phases, the first ranging from columns one through three. During this first phase every possible and impossible exception thought of is written down and stored for later – no thought is too extreme here.

In the second phase the result from the previous is sorted and arranged in groups that have something in common, it is not crucial that the groups are very well defined or closely related, the step is just a way of structuring the results from step 1, somewhat simplifying the following. Once the grouping is done, every exception is put on a separate row in columns 1-3; Possible Failure Mode, Possible Effect and Possible Cause(s).

Phases 1-2			Phase 3				Analysis	
Possible Failure Mode	Possible Failure Effect	Possible Causes	Existing Tests	Prob. of Occurrence	Criticality	Prob. of Detection	Risk Priority Number	Handling method

Fig. 5.1 FMEA headings

Step three is scoring in columns 5, 6 and 7. The list should be gone through a few times to ensure consistency in the judging. It is hard to be fair and consistent but it is important for the analysis to be as accurate as possible.

The above is merely a brief description of headings translated freely from Swedish (Johansson, 2003). Before analysis begins, a more in-depth study should be made.

5.2 ETA

Another analysis method is the Event Tree Analysis (ETA), like the FMEA it is inductive, meaning that possible failure events are listed first, and then their possible effects are analysed, as opposed to conductive methods that work the other way around. (Rausand, Høyland, 2004) describe a method for constructing an Event Tree in seven rather simple and straightforward steps. As the analysis is going to be used for risk identification and back-tracking exception propagations rather than traditional risk management it is not necessary to follow them thoroughly but the structure might prove quite useful.

The resulting structure is not that different from the Fishbone pattern of the Ishikawa diagram described in Section 6.1. First of all possible exceptions are established trough a brainstorm as explained in the previous section. Within the completed exception list, every exception is analysed with regards to what consequences it might have on its surroundings if it occurs. For each possible consequence, that may or may not occur, a probability is assigned that help identifying what risks are critical to handle. An example structure from (Rausand, Høyland, 2004) is shown in figure 5.2 as it explains the procedure quite well. Probabilities in every fork are multiplied to get an end probability for every outcome. The resulting structure could be used to step trough in a way similar to the Ishikawa or the end probabilities could be used to estimate what exceptions should be handled in what way (described in Section 3.2)

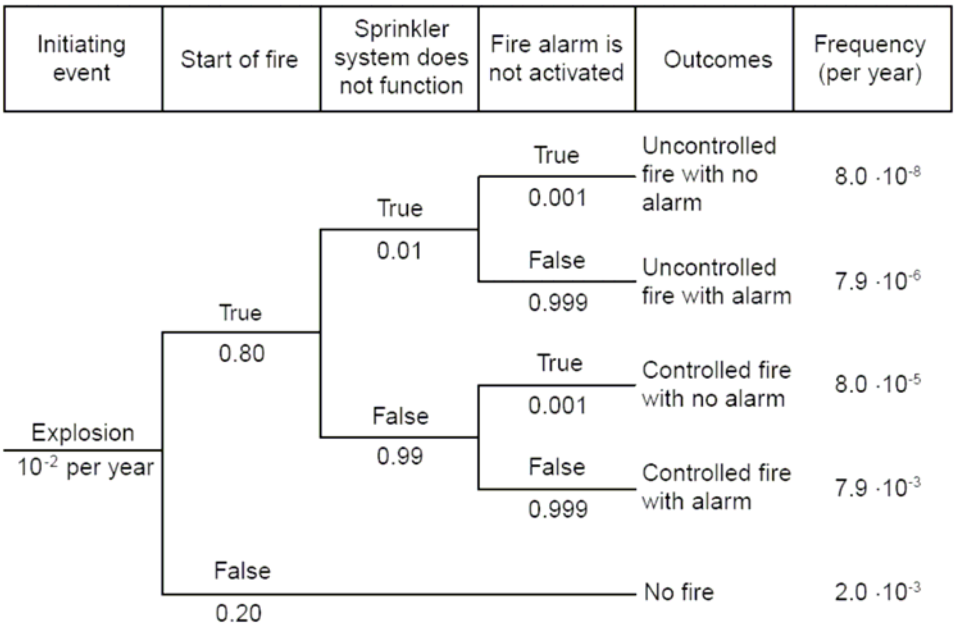


Fig. 5.2 Event Tree Analysis (From Rausand, Høyland, 2004).

6. Root Cause Analysis Methods

The need for a thought-through strategy for cleaning up the exception list presented to the operator or service technician was made obvious in Section 2.2. It might seem to be an easy and straight-forward task to construct a set of rules for identifying what exception is the root one. Constructing a general and fool-proof algorithm that at the same time is reliable, reusable and effective to implement proves to be an unsolvable equation, though. The result will always be a trade-off.

There are numerous approaches to take in this discipline commonly known as “Root Cause Analysis” or simply RCA. Most of the approaches originate from philosophical rather than scientific theories and are often used for optimisation and efficiency boosting in companies and organizations or to analyse issues in the community. The theories are, hence, not at all limited to the automation field but are applied in many different areas of research and development. This means that they are very versatile and non-application specific in their ‘out-of-the-box’ editions and therefore must be adapted to the specific application before they can be used.

A demand that follows from the modularisation “think” that runs through the whole project is that the structure used for construction and modelling of machine and machine parts can be broken down in smaller pieces that each can be tested and validated separately. It would be desirable that these modules also included Exception Handling objects as it would mean that they could be included in the validation process as well as it would contribute to the ideal of having only one model that the whole process and all steps in its development revolved around.

In the following sections a few investigated methods that are used to some extent in the suggested solution are presented briefly along with their respective advantages and disadvantages.

6.1 Cause-and-Effect Diagram (CED)

The Cause-and-Effect-Diagram, also known as the Fishbone or Ishikawa diagram (after its inventor, Japanese professor Karou Ishikawa), might be the most intuitive and simple approach to RCA. It has been given the rather suiting denotation “structured form of brainstorming” (Doggett, 2004) in his comparative studies of CED, ID and CRT (CRT being an abbreviation for Current Reality Tree, a method not further mentioned in this thesis).

The basic concept is to list all possible direct causes for a given symptom. Then list these causes’ possible causes and so on until all cause chains are as long as possible. The actual analysis is performed by putting the observed unexpected behaviour (indication) as the root node in a tree with its possible causes as branches in turn branching to their possible causes. Starting in the root node and moving on up, logical tests are performed to trace the way back to the Root Cause (Doggett 2004). As can be seen in figure 6.1 the graphic representation often resembles the classical fishbone pattern, hence the alias “Fishbone Diagram”.

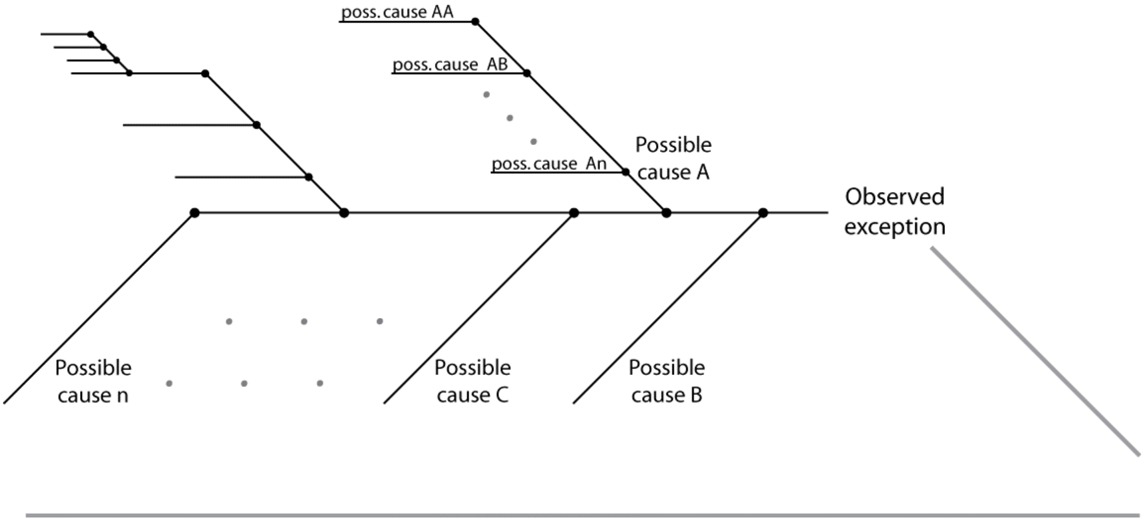


Fig. 6.1 Cause-and-Effect Diagram

6.2 Process Matrix (PM)

A development from the Fishbone method is the Process matrix (Schippers, 1999). This method is a way of representing one or multiple Fishbone Diagrams. Advantages in using a Process Matrix instead of a Fishbone Diagram include that it can handle larger diagrams through easier analysis of the characteristics. In a Fishbone Diagram there is no possibility to differentiate the strength of the causes and effects; in the Process Matrix there is. Implementation of a Process Matrix is significantly easier than an original Fishbone Diagram.

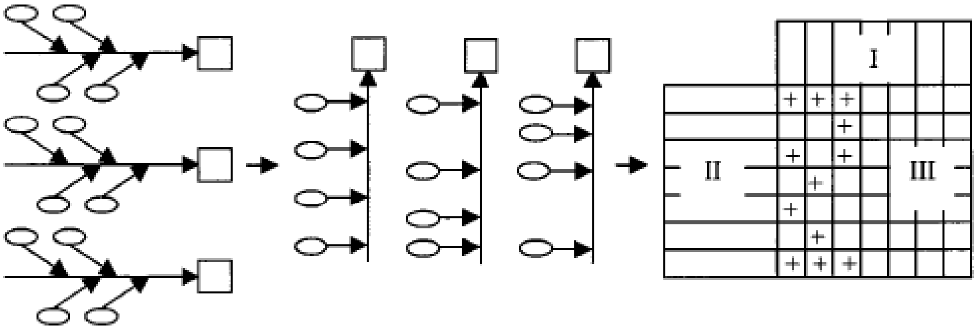


Fig. 6.2 Transition from Fishbone diagrams to Process Matrix, (From Schippers, 1999)

A Process Matrix is a table of the causes and the effects, see figure 6.2. In the first row (I in the figure) the effects are listed and in the first column (II) the causes are given. By putting relations in the table the connection between causes and effects appear. If one effect is caused by three of the causes these three rows in the column for the effect should be marked (III). Doing this for all the effects the table grows up.

6.3 Interrelations Digraph (ID)

The Interrelations Digraph (ID) approach may be a bit less intuitive structure-wise than the Fishbone Diagram but nevertheless has a few useful points. The basic idea, as described by (UVM 2001) and (QLR 2004), is to put down every relevant exception on a sheet of paper. Then, for each one of them it is analysed what other exceptions might have caused it and what others it could possibly cause itself. Dependencies are illustrated by arrows leading from the cause to the effect (fig. 6.3) and are important during runtime analysis.

During runtime, when exceptions are raised, the system looks at what alarms are raised and how they are interconnected. For each raised indications the number of incoming and outgoing arrows are counted and stored. Alarms with many outgoing arrows are classified as drivers and those with a lot of incoming arrows are classified as causes. By studying the relation between ins and outs the Root Cause(s) are identified and presented to the operator.

The method has good points and is quite useful as a stepping stone towards new approaches. In practice though, it is rather inefficient in its basic form and is not always able to identify the actual Root Cause (Doggett 2004). It also shares the problems of vulnerability to missing alarms and inconsistencies in the alarm database with the Fishbone method as well as most other methods even if it is not quite as sensitive as the Fishbone to missing alarms, in theory it only means that too many Root Causes are identified.

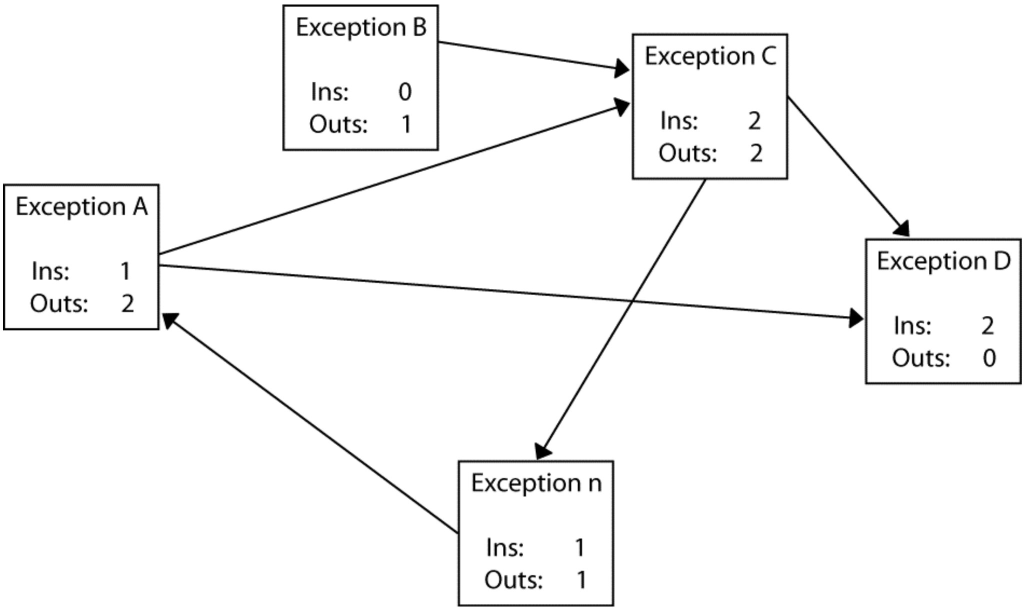


Fig. 6.3 Interrelations Digraph

6.4 Digraph Procedure

The Digraph Procedure described by (Kelly, Bartlett, 2006) uses a set of nodes connected by directed graphs (digraphs) that “represents the propagation of inputs through a system”. The nodes correspond to exceptions and sensors and the arcs to possible propagation routes. The key idea is that an error or an exception affects one or more given process parameters such as mass flow or temperature in some predefined way, either positive or negative. Tracing of the root cause is conducted by moving backward in the chain of parameter deviation in order to identify the triggering failure or exception. In the example displayed in figure 6.4 below M1, M2 and M3 represent mass flows at three different locations in the process, Fault One, Two and Three are exceptions that affect the mass flow in positive, negative and positive ways respectively. High flow in the location corresponding to M1 affects M2 but not as strongly as it is itself affected by Fault One. A node may have any number of arcs leading to or from it. Effects can be described as –10, -1, 0, +1 or +10 in turn meaning “High low, moderate low, normal, moderate high and large high” (Kelly, Bartlett 2006).

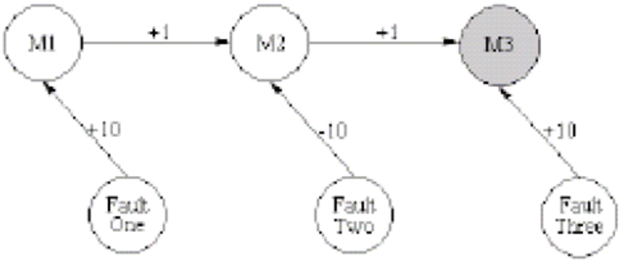


Fig. 6.4 Digraph Procedure, (From Kelly, Bartlett, 2006)

If it is established that the mass flow M3 is above normal it may be caused by one of two factors. Either Fault Three has occurred causing the increase in flow or fault one has occurred caused increase in M1, which has propagated through M2, and then causing the established increase in M3. Fault two has not occurred, as it would have meant a decrease in M2.

The largest advantage of the Digraph Method is that faulty flows can travel through locations corresponding to non-triggered exceptions meaning that shared resources or mutual effects are not issues. The disadvantage is that the system rather quickly grows complex and hard to overview. The disadvantage could be worked around by grouping items in a hierarchical groups but it still means that it is tough to keep the structure up to date, evolve it and add new functionality. Finding a delimited module with appropriate features for validation is hard as no general interface can be defined to connect the modules to each other.

6.5 Multilevel Flow Models (MFM)

One of the Alarm Handling methods used for industrial processes is Multilevel Flow Models (MFM). Originally the Danish professor Morten Lind (Lind, 1987) introduced the MFM concept. After that several extensions have been made. One such is (Larsson, 1992).

The approach with MFM according to Larsson is based on flow modelling. There could be different flows, but these can be represented by three different types of flows; energy, mass and information flows. Although these three flow types are different they have many characteristics in common. The MFM model is built by small components called functions. The functions are chosen so that they are as general as possible. The most common functions are Source, Transport, Barrier, Storage, Balance, Sink, Observer, Decision Maker and Actor. With these functions most of the flows and processes can be described. The concept of MFM is to build sublevels with separate goals to fulfil. The different sublevels are dependent of that other sublevels have fulfilled their goals. By making these sublevels the model will be fairly simple to overview. A figure of an engine running on gasoline and oxygen is shown in figure 6.5. The model to the right is sublevels with specific goals to fulfill. In each sublevel there are functions building a flow depending on lower levels and affecting the levels above. To the left a sketch of the engine is shown.

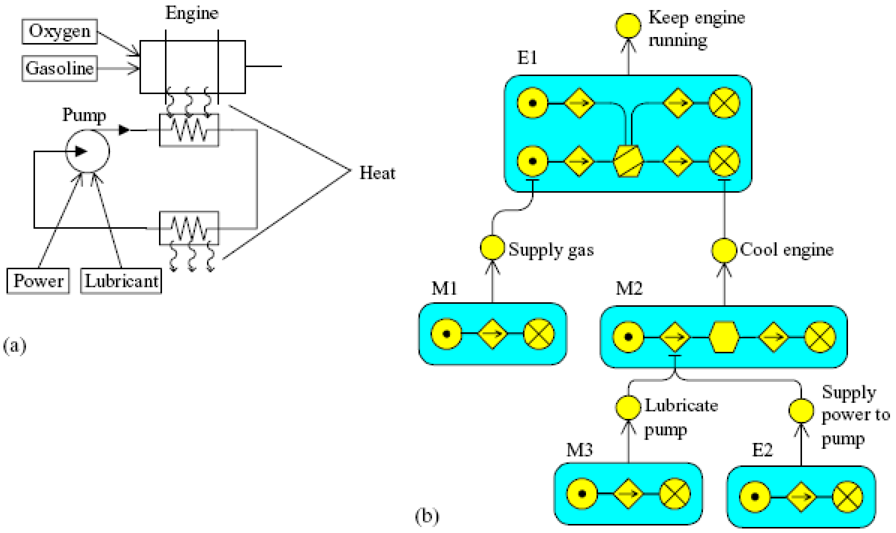


Fig. 6.5 A MFM of an engine running on gasoline and oxygen. (From Öhman, 2002)

7. Exception structure and handling

Keeping things as simple as possible for as long as possible structure-wise is often, if not the best way to go, at least a superior way to get started. Structural simplicity and intuitiveness in a methodology is the best way of minimizing errors and misinterpretation when applied to real applications. Therefore, the suggested approach is based on what resembles a blend of the Ishikawa Diagram and the Interrelations Digraph which are described in Chapter 6 but with a few decisive differences. Elements of the far more advanced MFM and Digraph philosophies, also explained in Chapter 6, are also used to some extent

The Magazine described in Section 2.5 will be revisited and referred to for illustrational purposes throughout the chapter.

A walkthrough of all steps during both construction and run-time is available in Appendix D.

7.1 Components and Structures

Machine functionality and behaviour are highly dependent on several internal Components and their respective functionality that must be achieved in order to reach the desired end results. The Components' delimitations can be identified in a variety of ways, including construction of a Function-Means Tree (FMT) as proposed in the WCE workflow concept described in Section 1.4. Other ways are studying a UML Class Diagram or using the well-known Failure Mode Effect and Criticality Analysis (FMECA) that is described in Chapter 5.. Regardless of which method is used they tend to give identical structures and always provide the same results and robustness.

A Component is defined as a part of the machine (or rather a part of the batch flow in the machine) with a clear and specific Goal and likewise clear requirements – basically a black box with one or more ingoing and a single outgoing connections.

The *Component* is in turn broken down into a set of items corresponding to its functional Goal, a set of internal Functions and Subfunctions that are needed to reach the defined Goal and a set of zero or more external requirements (Ins) that correspond to Goals of other Components located before the current Component in the batch flow. The blocks are interconnected by directed arcs as is illustrated in figure 7.1.

Modules

The structure with its Goal, Subfunctions, interconnections and requirements is hereby denoted *Dependency Structure* and will furthermore be referred to as such. The internal connections are denoted *Dependency Arcs* and each describe a downstream dependency and upstream exception propagation, meaning that the source block must fulfil its Goal in order for the receiving one to be able to do the same.

Every Component such as the Magazine is required to have exactly one Goal, that is a brief description of the component's purpose and should answer the questions "What is the purpose of this component?", "When does it have to be fulfilled in order to ensure proper process operation (often equal to one or more operational states)?" and "Where, physically, is it supposed to deliver its results?". These answers are often used when identifying modules in a structure and are chosen in a way that help intuitive linking of the Components and simplify re-use of Components and Subfunctions. Each component can have one and only one goal for the back-tracing described in Chapter 9 to be able to tell what routes the propagation has taken. The Component internally consists of Functions and Subfunctions as defined below

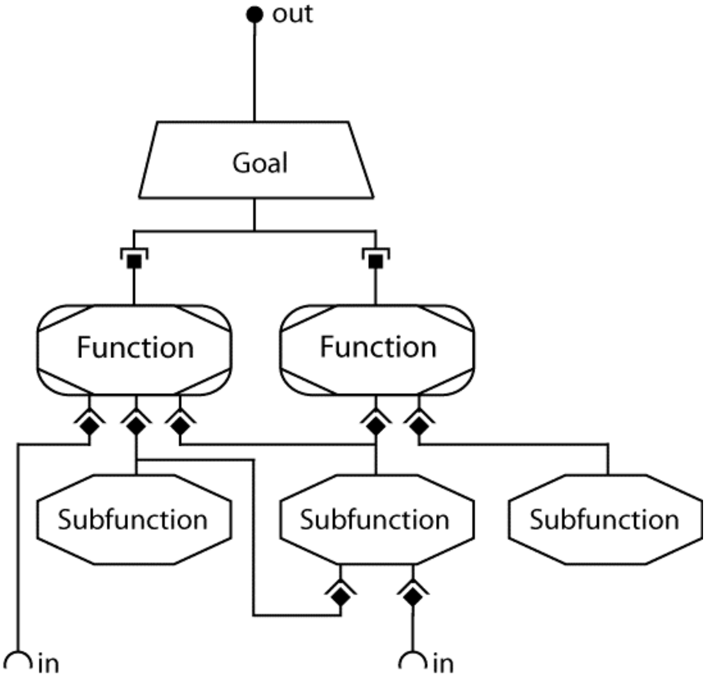


Fig. 7.1 General Dependency Structure

Functions resemble Components in that they have sole purposes and outputs but may have multiple input requirements. The primary differences are that Functions have more concrete Goals that are closer related to actual functions in the machine or the means in the FMT. Typically the Component Goal for the Magazine would be "Deliver satisfactory sheets one at the time during Run-mode", while the Function objective has to be more specific and reveal how that should be done, such as "Grab one sheet at the bottom of the pile and pull it out". Each Function internally consists of a structure for identifying raised Exceptions as explained in Section 7.3. The Function block may contain both Machine Exceptions and Process Exceptions as they are described in Section 7.3.

Every Function may, and typically will, have dependencies of its own and will therefore require input from another Function, Subfunction or another Component directly. The Functions and Subfunctions in the set may hence be interconnected with dependency arcs in various ways to describe the internal dependencies. It is crucial that these interconnections are carefully drawn as they play a large part in the actual real-time Root Cause analysis during runtime.

Subfunctions are special cases of Functions that relate directly to purely physical components in the machine, their purpose is to provide prerequisites for the Functions to work properly. They could be viewed as means for fulfilling the Functions, as the Functions "Grab and pull" requires a Grabbing to be made but is not concerned whether the grabbing is done by vacuum suction or a mechanical arm grabbing the sheet. This feature also means that the Subfunctions are useful and re-useable items that represent a way of performing a specific task. That and the fact that Functions and Subfunctions are treated differently in the runtime RCA makes it meaningful to define Functions and Subfunctions as different types of objects. Subfunctions may contain only Machine Exceptions and, hence, may not affect the actual process flow directly but only indirectly. The only ingoing connection possibilities of a Subfunction are the Ins of the Component, that is the Goals of other Components.

The *Ins* of the Component are links to Goals of other Functions that are located before the current in the Component cascade that is the plant.

A summary of the modules types and their respective connection possibilities are displayed in table 7.1 below.

Table. 7.1 Module types and allowed connections

Module type	Allowed requirements	Contains
Component	Components	Dependency structure
Goal	Functions	Goal description
Function	Functions*, Subfunctions, Ins	Process Exceptions, Machine Exceptions
Subfunction	Ins	Process Exceptions

*Only functions that are not in turn depending on the current one, resulting in dependency loops

The Dependency Structure and its linkage to the FMT provide the possibility of validating and re-using each Component, Function and Subfunction by itself and viewing the entire machine and its requirements and dependencies as one large tree structure, grouped in a hierarchical structure of Components while it still retains the possibility of isolating a single Root Cause even when it means tapping into more than one Component.

Delicate care must be taken on how the requirements are formulated and above all, how to connect them to each other in a way that provides and ensures the desired behaviour

A Component is externally defined by its one or more Ins and its one and only out. All other elements such as the Dependency Structure, requirements, indications and variables are purely internal states and must not be directly addressed by other Components in order for the module validation principle to be valid.

Exceptions

An *Exception* is defined as an unexpected or unpredictable event that may require operator or control system attention. Depending on the severity of the exception one of three different approaches is chosen as previously mentioned in Section 3.2. The severity and internal priority order is dependent on the machine’s operational state and hence has as many values as the machine has operational states. The values are represented by set of properties referred to as the exception priorities each ranging from zero to nine where indexes 1-3 correspond to the lowest severity (equal to the blue alarms from Section 2.3), 4-6 medium (yellow) and 7-9 severe (red) exceptions.

Priority zero is used to indicate that the exception has no priority at all and should not be presented to the operator. It applies to the somewhat abstract exceptions that are links to other Components' outs rather than located in the current one. Those exceptions cannot be identified as roots. It is also specified for every exception whether it is a *Process Exception* or a *Machine Exception,* that is whether it describes an exception such as a machine malfunction or something unexpected happening to the actual item (in the TAA case meaning the bottle). The reason for making the distinction between Process and Machine Exceptions is that Process Exceptions refer to an object that (normally) will travel through the machine possibly causing subsequent Process Exceptions, but not Machine Exceptions. Machine Exceptions may cause Process Eexceptions to occur indirectly by failing to fulfil the Subfunctional Goal but Process Exceptions (always located in Functions) may not cause Machine Exceptions.

7.2 Module Interconnection

The reason Functions and Subfunctions are connected to each other is that when an exception occurs at some location in the machine (equal to a function or Subfunction in the Dependency Structure) it may cause exceptions in other Functions as well in the way described in Section 2.2. The connections are a way of specifying possible propagation routes in order to provide backtracking possibility when multiple exceptions are identified in adjacent Functions. A dependency arc, as it is defined in Section 7.1, begins in a Function or Subfunction whose exceptions may propagate and ends in a Function that might be affected by the first. Arcs may be drawn from any Subfunction to any Function, between Functions but only in one direction, from any in to any Subfunction or Function and from any Function to the Component Goal. Allowed arcs are described in table 7.1.

Through the linking, Dependency Chains are formed and will be used in the Root Cause Analysis to match against exception chains. During runtime, Process Exceptions in a Function will be masked out if an exception of any type is identified in a Function or Subfunction that is connected to the first Function which means that the connections are of outmost importance.

A connection through an arc is active regardless of what exceptions are identified in the Functions. Process and Machine Exceptions are separated but apart from that it does not matter whether the exceptions seem to be related or not. This is an effect of the modularisation that provides the possibility to exchange Functions or Subfunctions without rebuilding the entire Component. As long as the sheet transport requirement is not fulfilled, the separator function is not regarded as to whether it is because a conveyor belt that has broke or a human courier that has taken a break.

On the downside is the risk of masking out exceptions that are actually unrelated and hence should have been reported as two separate errors. As it will never be possible to tell the case "A caused B" from "A and B occurred simultaneously" and as the issue is rarely a problem since priorities are inherited (see Sectiton 8.1) and all raised exceptions should be available to the operator at request (mentioned in Appendix C.3), this is accepted for now.

The suggested approach in its final form shares some structural thoughts with the likes of the Digraph Procedure and the MFM-method each explained in Chapter 6. It differs though, primarily in that no specific physical or abstract quantities such as the MFM's mass, energy and information flows are predetermined and that indications are considered to be Boolean (triggered or non-triggered) rather than continuous or chosen from a larger discrete set like that of the Digraph Model's {-10, -1, 0, +1, +10}. A sort of indication severity grading may still be present in form of high/high-high indications and such, but those are viewed as separate triggered or non-triggered indications rather than the same indication with different severity. Also, the MFM and Digraph methods allow flows to propagate both upstream and downstream and pass through layers where no exceptions have been identified, the suggested one does not.

The fact that the suggested method is unable to pass indications or flows through layers that correspond to non-triggered events mean that it is limited to complete exception chains, or rather: it will interpret all chains as complete. This could be worked around by implementing a simple probabilistic pattern recognition algorithm (such as a simple m-out-of-n rule or similar) if testing should prove it useful but it is omitted for now.

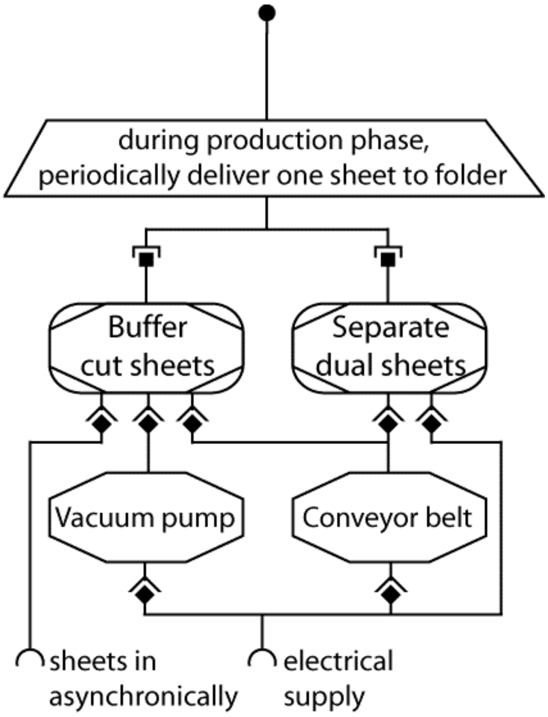


Fig. 7.2 Dependency structure for the Magazine

7.3 Inside the (Sub-) functions

A macroscopic view on the Dependency Structure and its elements is presented in Section 7.1. In this section a more microscopic look, focusing on the inside of the structure's Functions and Subfunctions, is shown. The linking of the Subfunctions provide tracking of exception propagations and help identifying in what function the actual root exception was raised as shown in Section 7.2. In order to decide more specifically what has occurred, every Function and Subfunction has a corresponding mechanism for mapping indication patterns to occurred exceptions. The mechanism has strong connections to the approach of the Process Matrix described in Section 6.2 and could be seen as an Ishikawa diagram on its own.

The pattern mapping is represented by a table as the one seen in Appendix A and 8.3 below. Every Exception judged to be important enough to be made identifiable is assigned a column in the table and every indication available is put on a separate row. Note that an indication is not equal to a sensor but rather an interpretation of a signal from one, meaning that one actual temperature sensor may correspond to multiple indications in the matrix, such as the temperature being very low, low, high and very high. Combinations of one or more indications that must be, must not be or may be triggered for a decision on an exception are represented by assigning 1:s (indication must be triggered) 0:s (indication must not be triggered) and x:s (one or more of the indications must be triggered) to the corresponding cells. Leaving the cell blank mean it does not matter whether or not the indication is triggered. The decision table can be realized as a logic sequential net with 1 meaning the indication is connected to an AND-gate, 0 a NAND-gate and x:es to OR-gates. Blanks are not connected at all. A such table for the Magazine is shown in Appendix A.

Unidentified combinations

Special care must be taken to ensure that no unidentified indication combinations, that can jeopardize the analysis, appear. A possible scenario could be that two or more indications that should never appear at same time actually do by some unforeseen reason (maybe a sheet is stuck at an angle causing both the empty and full indication of the buffer to trigger at the same time). As they never should appear together no exception has been defined for the combination. In the best case it leads to the presentation of two separate Exceptions that each matches a part of the pattern, in the worst case though it means that no Exception at all is triggered as no pattern condition is fulfilled. The first is bad and rather annoying but the latter could prove to be catastrophic.

Both issues could be eliminated fairly easy by running a check that gather all unused combinations and set them as conditions for an Exception that say “Unidentified exception in function x” or such. The operator would then either be presented with a list of exceptions with criteria that were ‘almost’ matched or a view of all raised exceptions. That is if the Root Cause was tracked to the Function in question, if it was not the unidentified exception will not be a problem as it will be masked out by lower level exceptions.

Indications

As mentioned above, indications do not correspond directly to sensors but rather to interpretations of their data. This means that if the Function is to be re-used in another context the indications can be inherited and only the mapping between indications and actual sensors has to be remade. Indications are allowed to appear at more that one place in a Process Mmatrix and in more than one Process Matrix within the same component. It is however not adviced to allow sharing of indications between components as it is hard to keep track of what sensor affects what software component (even though the linkage should be made as obvious as possible). Sharing an indication between functions means typing it twice on two separate rows as it has to be part of both Process Matrixes. In practice, all Process Matrixes belonging to the same Component are typically merged together to form one larger matrix.

Exception Grouping

Even within every Process Matrix, exceptions are grouped together in clusters representing exceptions that may not appear at the same time. It may typically involve exceptions that either indicate the same physical quantity but of different severity (i.e. hot and very hot) or exceptions within the same physical component that each could cause one another but that does not appear in the same order every time. Examples of the latter are the motor warm and motor stopped exceptions – either something stopped the engine and therefore it has run warm or the engine was worn out, grew warm because of increased friction and ultimately stopped. In the graphical representation of the Process Matrix a such group is represented by a thick line as seen in Appendix A and only one Exception per group may be presented in the default operator view.

If more than one Exception is identified within a group, the one with the highest priority number is the one that will be masking out the others, as it is defined as being more severe. Should more than one triggered Exception share the same priority number the first one to appear is displayed as it is the most likely to be the one causing the other(s).

7.4 Versatility and Evolution

One of the most important strengths and strongest arguments for the suggested approach and structure is that it allows re-use of solutions and structures and improving and tweaking of old solutions. The re-use and improvement features are demonstrated with an example tied to the now well-known Magazine component from Section 2.5. The example describes a workflow as it could have been carried out with the suggested method rather that what was actually done as the machine was designed and constructed using old manners.

In the FMT of the WCE standard, in addition to the Function description, criteria are specified that state how often a Function must succeed in order to be considered working. Magazine criteria for the function “Store sheets that arrive asynchronously and deliver them one at the time” could typically be formulated as “Allow a maximum of one double sheet to continue from the Magazine into the next Component per ten thousand sheet deliveries”.

When the Component is designed, different approaches are considered. Old, used techniques are evaluated against new conceptual ones. Either complete solutions with Goals, Functions, Subfunctions and Exceptions can be re-used in new solutions or old Subfunctions such as a conveyor belt or a storage module can be used to make the development of completely new ones more effective. In this case the solution described in Section 2.5 with a vacuum pump sucking hold of sheets in the buffer and a belt pulling them out of it, changing direction and delivering them to the next Component, is the one decided on. The Component is equipped with its Functions and Subfunctions such as “Grab”, “Transport” etcetera.

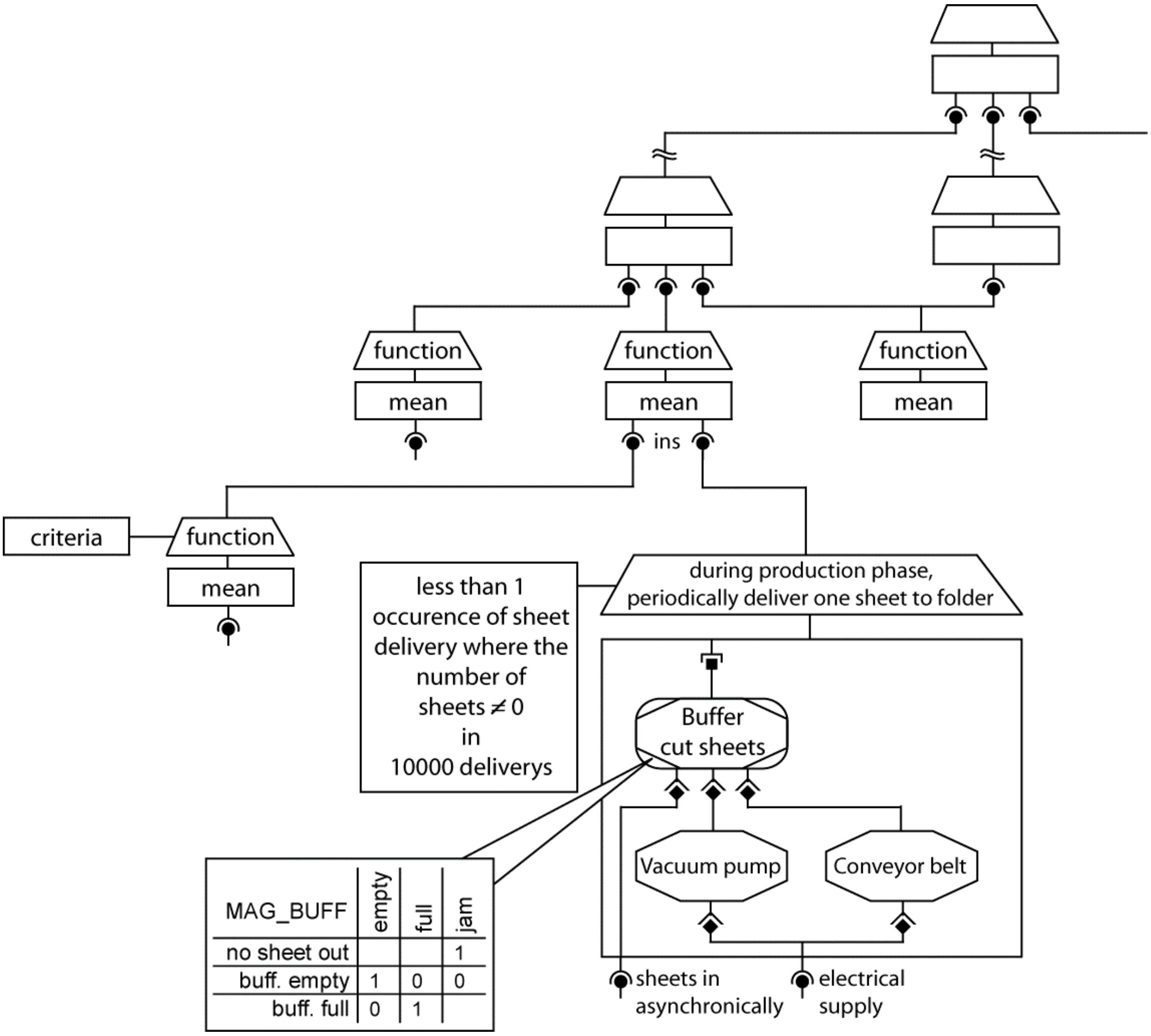
Down the line, maybe months or even years later, closer analysis or actual hardware tests on test rigs prove that the specified criteria cannot be met by improving the ‘grab and move’-technique. It is then decided that a Function should be added that separates double sheets within the Component. The Function “Separate dual sheets” is added under the goal. Then a clever way of implementing it (a wheel spinning in the opposite direction of the sheets) is figured out and it is considered whether any new Subfunctions will be needed. It is analysed what new Dependency Arcs are needed to represent dependencies on or for the new Functions and Subfunctions. It is identified what internal exceptions that should be handled and a Process Matrix is set up for the

(Sub)function(s). If a Function or Subfunction has been used previously in some context, it can of course be re-used.

As the only things that are changed structure-wise are the arcs and the new (Sub)function(s), none of the previously tested and validated Functions needs to be revalidated. Only the “Separate dual sheets”-function and the new arcs have to be gone through in order for the updated Component to be considered valid again.

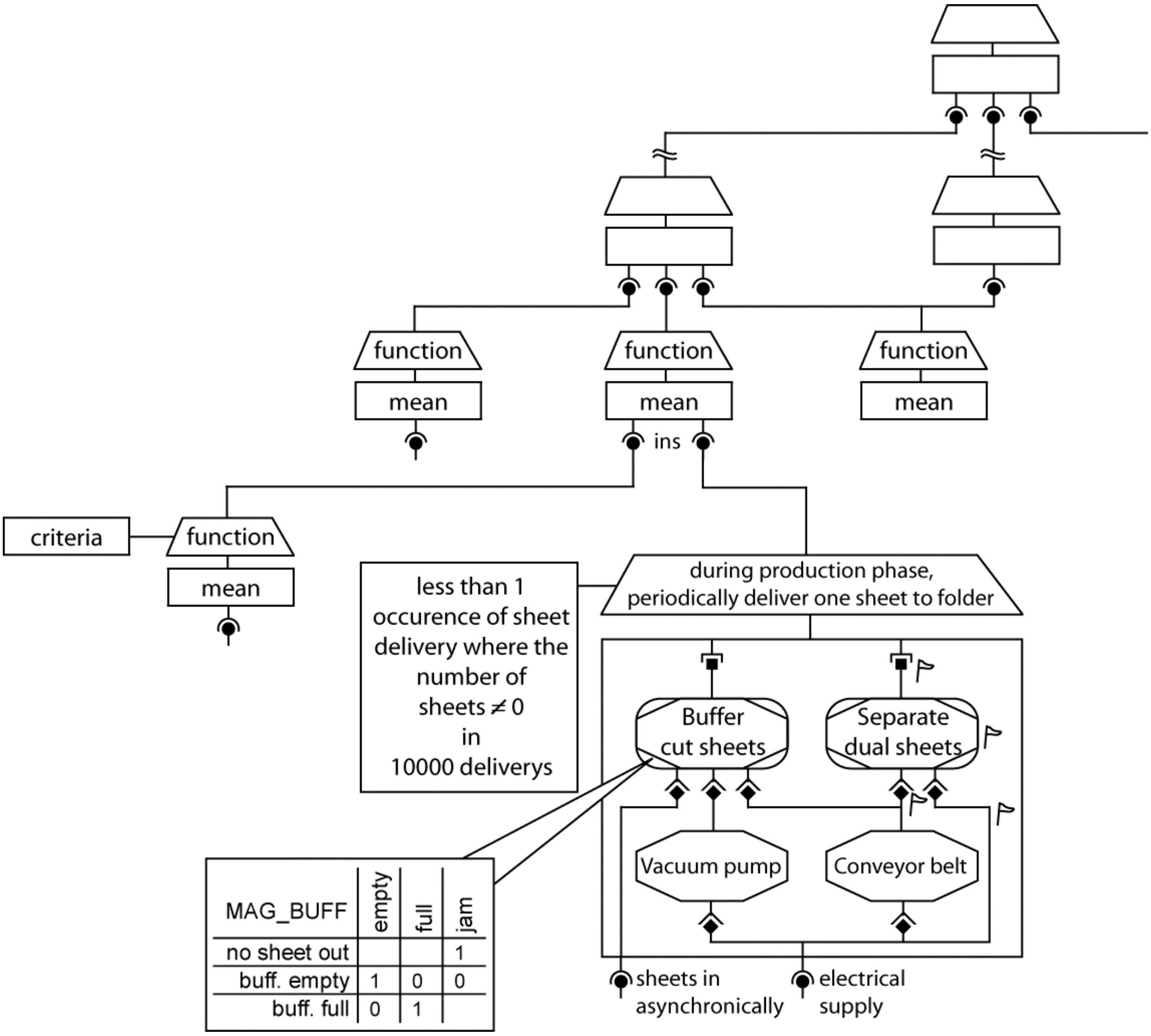
If criteria still cannot be met by working on the tweaking of Functions, more Functions and Subfunctions still can be patched onto the structure. Such Functions may include the waster that throws away the dual sheets before they get the chance to move on to the next Component and cause a clog there. The procedure with new Functions, Subfunctions, Arcs and partial validation is then repeated once more.

The steps are illustrated in the figures below (7.3 and 7.4). In the first (7.3) the imagined design for the Magazine without a sheet separator is shown and the second (7.4) shows the Function and Dependency Arcs that are added to represent the addition of the separator wheel. The flag icons indicate what Functions and arcs need to be validated in order for the Component to be considered valid once again.


function
mean
function
mean
ins
function
mean
criteria
function
mean
less than 1 occurence of sheet delivery where the number of sheets ≠ 0 in 10000 deliverys
during production phase, periodically deliver one sheet to folder
Buffer cut sheets
Vacuum pump
Conveyor belt
sheets in asynchronically
electrical supply

MAG_BUFF	empty	full	jam
no sheet out			1
buff. empty	1	0	0
buff. full	0	1	

Fig. 7.3 Design for the Magazine without a sheet separator.



MAG_BUFF	empty	full	jam
no sheet out			1
buff. empty	1	0	0
buff. full	0	1	

Fig. 7.4 The Function and Dependency Arcs that are added to represent the addition of the separator wheel. Flagged items represent items that need to be (re)validated

8. The method in practice

8.1 The RCA Procedure

The effort of developing and using the structure and Components and the linkage between them that is specified in the Chapter 7 is justified by its two effects; boosting efficiency in product development and increasing runtime performance by decreasing downtime and increasing time between failures. The long-term product development effects are described in WCE documentation and Section 1.4. Below is a summary of how the runtime Root Cause Analysis, that help achieve the second effect, is handled.

The first thing that happens is that one or more indications are triggered. Every triggered indication is stored in a *Triggered Indications Table*. From the indications it is then identified to what Components they belong. The table is scanned and the indications are matched against the Components' Process Matrixes to identify a set of exceptions. That is: estimations of what unexpected events have occurred. The results are stored in a second table referred to as the *Complete Exceptions Table* (CET). This list should on operator request be made visible in the HMI.

That table is also the starting point for the RCA-algorithm. For every Function and Subfunction it is noted whether or not any of its internal Machine and Process Exceptions respectively are triggered or not. For every item with at least one of them being flagged, the internal groups are scanned one by one to ensure that only one exception is reported from every such group. The exception that remains from each subgroup is put in a third table known as the *Root Exceptions Table* (RET). Then, for every one of the functions that were flagged to contain a Process Exception, it is checked whether there are exceptions, of any type, in any of the Functions or Subfunctions that are connected to it through incoming Dependency Arcs. If so, then the Process Exceptions are eliminated from the RET.

It is important to note that when an exception masks out another from any table it always attains the highest one of the masked and its own priorities. This is important in order to make sure that no severe event can pass as being harmless even if it is in fact the result of another, fairly innocuous event.

When the steps above have been completed, the RET holds the exceptions that are identified as roots and is, hence, presented to the operator and control system before the cycle starts over when new indications are triggered. As indications may appear in an almost arbitrary order, mainly due to inexact sensor trigger levels and delays in the information flow, it is important that the implementation is made iterative in order to function properly.

The issue with root exceptions being masked out by non-roots that was mentioned in Section 7.2 is in most cases resolved with the separation of Process and Machine Exceptions. Because Machine Exceptions are not allowed to cause other Machine Exceptions to occur other than indirectly they are never masked out

8.2 Work Procedure

Early in the planning and designing of a new packaging machine it has to be analyzed what internal functions and tasks need to be implemented. When the functions are specified, one or more solutions for them are being considered and evaluated. For evaluation assistance, one model for each considered solution is built and fitted into the process structure, such as an FMT or a UML-model depending on the development strategy. The model is typically rather basic, identifying Functions and Subfunctions as defined in Chapter 7 but not going into detail with exceptions or indications.

Completing these steps significantly simplifies identifying pros and cons in the different approaches, making evaluation and deciding on either of them easier. All solution candidates, whether they are used or discarded, are stored in an archive of some sort together with their identified pros and cons. The archiving step is important as rejected solutions either may prove useful in another context later or may be immediately rejected without having to perform the evaluation all over again the next time the same function has to be implemented.

Once all means-implementation approaches have been decided on, the next step is designing them. After completing the first design phase, a risk analysis is performed as described in Chapter 5. All exceptions that are considered needed and their corresponding indications are identified and included in the design.

During the following software design step priorities are assigned and Process Matrixes built and tests are being made as to whether or not the FMT functions criteria described in Section 1.3 can be met. If not, then the design can be completed with additional functions as described in Section 7.4.

9. Concluding Discussions

9.1 What was Meant to Be and What Actually Became of it

Starting out, the main objectives were to find a solution for the problem with alarm bursts and to investigate if the currently used alarm structure is really optimal with regards to the classification of alarms and how they are handled. It was also to be found what could be won if the Alarm Handling was a part of the process design rather that an additional feature created when the machine is already built.

The solution was meant to be general rather than specific for any machine. For references though, the TAA should be used and if there was time enough, perhaps the solution should be tested on some part of the TAA. Parallel work was done on evaluating object oriented programming in the control system and if both works proved ready there were loose plans of trying to integrate them. It was not required though that an implemented and tested solution was created, however there should be a suggestion for a roadmap on how to implement a more efficient Alarm Handling routine.

The issue of alarm bursts was solved by a non machine-specific RCA approach, in which advantages of the proposed alarm structure were used. A way of integrating the solution in a workflow such as the WCE was proposed in order to make the Alarm Handling part of the design process.

It was never an actual option to implement the solution neither on the TAA or the object-oriented work.

As time went by and the RCA approach was maturing, it got obvious that in order for it to be really efficient, it had to be constructed from a structure of some sort. At the time big changes were being made to the product development routine at Tetra Pak and research was therefore done to see what could be won from integrating the construction of such a structure with the new workflow routine.

9.2 Is the RCA Working?

A few tests, of which results are shown in Appendix B, have been performed, showing that the method is accurate a vast majority of the times. In some test cases where more than one root exception occurred, root exceptions are in fact masked out by others. It is an almost impossible task to notice these, and the masked exceptions are physically close to the shown. The priority of the displayed exception is always correct though (and it is always in fact a root exception) and the correct action is therefore always taken.

As all tests have been done manually and only on the TAA Magazine it is not possible to draw any major conclusions but the tests and the structures are enough for the authors to consider the method stable and useful. The answer to the headline question would hence be, “yes”.

9.3 Connection to WCE/FMT/UML

The WCE approach is built of several steps in the development; it starts with building an FMT from the customer requirements. With this FMT the development continues. A good thing about the suggested RCA is that it can start from either an FMT or a UML-model. If the WCE is used it is very simple to use the created FMT as a start to the RCA. If the WCE is not used and there is a UML-model available instead this will be the starting point to the RCA.

9.4 Connection to the Two Other Theses

As mentioned in the beginning of Chapter 1, this thesis is one in a group of three works being done together at Tetra Pak. The main idea in these theses is to create new ways of thinking regarding the control system and other software of the packaging machines. Instead of working with old programming languages and basic or no Exception Handling, it is supposed to give a new creative point of view towards object orientation and adaptable HMI that could be shown in any web-browser.

In this thesis, regarding the Exception Handling, it is assumed that an object oriented control system is present. This is a valid assumption as the work of (Arvehammar, 2007) is done parallel to this work, trying to implement the control system in an object oriented programming language.

The third work in the group is (Byrlind, Karlsson, 2007) regarding the new HMI that should work in any kind of web-browser and separate the technical and graphical developing stages. It also provides possibility for a good presentation of exceptions and the ability to subscribe to different types of information such as the RET and the CET from the RCA, or the output from a future diagnosis or prediction block.

9.5 Further Work:

Different Phases in The Process

In this thesis it is assumed that the machine is in its production phase. The Exception Handling works the same way in the other phases such as Calibrate, Sterilize, Tank Fill, etc. but the exceptions have other priorities.

One thing that is suitable for further work is to investigate how the transitions between the phases should be handled. What happens with the RET? Should it be cleared, should the machine stop because of an old exception etc?

Diagnostics

As mentioned earlier (Chapter 4) an interesting part for an Alarm Handling system would be a Diagnostics part. This could look at process data and exception history in order to predict exceptions before the actually occur.

10. References

ARVEHAMMAR, Maja (2007): “Object Oriented Automation System” Master’s Thesis. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

BYRLIND, Johanna and KARLSSON, Kajsa (2007): “New Web based HMI portal for Tetra Pak Equipment” Master’s Thesis. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

CAMPBELL BROWN, Donald and O’DONNELL, Manus (1997): “Too much of a good thing? – Alarm management experience in BP Oil” Stemming the Alarm Flood (Digest No 1997/136), IEE Colloquium on.

HSE – Health and Safety Executive (2000): “Better Alarm Handling” Chemical sheets no 6, HSE information sheet. Available at: http://www.hse.gov.uk/pubns /chis6.pdf (Read 2006-09-11)

JOHANSSON, Per (2003): “Feleffektanalys – Failure Mode and Effect Analysis”. Division of Machine Design, Linköping University. Available at: http://www.machine.ikp.liu.se/publications/fulltext/fmea.pdf (Read: 2006-12-04)

LARSSON, Jan-Eric (1992): “Knowledge-Based Methods for Control Systems” Doctor’s thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

LARSSON, Jan, SVENSSON, Jesper (2000): “Formella Metoder - En studie om Riskanalyser”. Department of Information technology and Media, Mid Sweden University

LIND, Morten (1987): “Multilevel Flow Modeling - Basic Concepts” Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lungby, Denmark.

LIU J et al. (2001): “Intelligent alarm management through suppressing nuisance alarms and providing operator advice” Chemical & Process Engineering Center

KELLY E.M, BARTLETT L.M (2006): “Application of the Digraph Method in System Fault Diagnostics”. Aeronautical and Automotive Engineering Department, Loughborough University, Loughborough, Leicestershire.

MALMQVIST, Johan (1995): “A Computer-based Approach Towards Including Design History Information In Product Models and Function-Means Trees”. Machine and Vehicle Design, Chalmers University of Technology, Göteborg, Sweden

MORGAN, James M, LIKER Jeffrey K. (2006): “The Toyota Product Development System: Integrating People, Process, and Technology”. Productivity Press, USA, 200604 (ISBN 1563272822).

NORDQVIST, Carl-.Johan (2003): “UML – del 1 av 3”. Available at: http://iwtjanster.idg.se/webbstudio/pub/artikel.asp?id=207 (Read: 2007-02-12)

QLR - QUALITY LEADER RESOURCES INC. (2004): “Framework-Tools, Interrelationship Digraph”. Available at: http://www.weimproveschools.com/tools/Tools040321Interrelationship.pdf (Read: 2006-10-17)

RAUSAND M, HØYLAND A (2004):”System Reliability Theory; Models, Statistical Methods and Applications (Second Edition)”. Wiley, 2004

SCHIPPERS W.A.J (1999): “The process matrix, a simple tool to analyse and describe production processes” Quality and Reliability Engineering International 15: 469-773, 1999

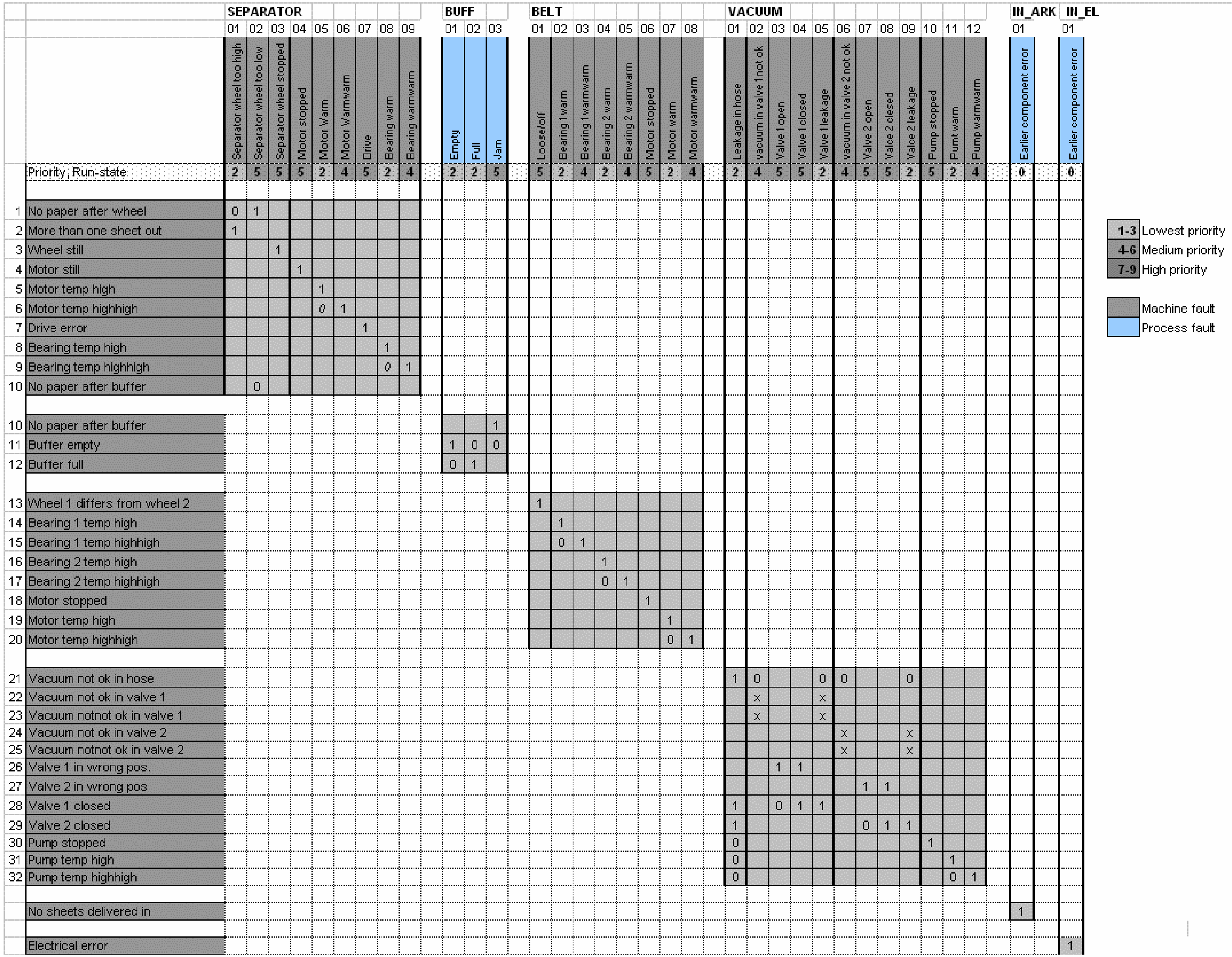
SMITH, W H, HOWARD, C R, FOORD, A G (2003), “Alarms Management – Priority, Floods, Tears or Gain?” 4-sight Consulting.

UVM - University of Vermont (2001): “Inter-relationship Diagram”. Available at: http://www.uvm.edu/~auditwww/tools/?Page=interrel.html (Read: 2006-10-13)

ÖHMAN, Bengt (2002): “Discrete Sensor Validation with Multilevel Flow Models” IEEE intelligent Systems, vol 17, no 3 pp 55-61 May/June 2002.

A. Process Matrix for the TAA Magazine

To give an example of how the Components, Exceptions, indications and their linkage to each other this table below were made. It is a simplified representation of the Magazine in the TAA-machine. The priorities on the Exceptions are shown, as well as which are Process Exceptions and which are Machine Exceptions.



		SEPARATOR									BUFF			BELT								VACUUM												IN_ARK	IN_EL
		01	02	03	04	05	06	07	08	09	01	02	03	01	02	03	04	05	06	07	08	01	02	03	04	05	06	07	08	09	10	11	12	01	01
		Separator wheel too high	Separator wheel too low	Separator wheel stopped	Motor stopped	Motor Warm	Motor Warmwarm	Drive	Bearing warm	Bearing warmwarm	Empty	Full	Jam	Loose/off	Bearing 1 warm	Bearing 1 warmwarm	Bearing 2 warm	Bearing 2 warmwarm	Motor stopped	Motor warm	Motor warmwarm	Leakage in hose	vacuum in valve 1 not ok	Value 1 open	Value 1 closed	Value 1 leakage	vacuum in valve 2 not ok	Value 2 open	Valce 2 clesed	Valce 2 leakage	Pump stopped	Pumt warm	Pump warmwarm	Earlier component error	Earlier component error
	Priority, Run-state	2	5	5	5	2	4	5	2	4	2	2	5	5	2	4	2	4	5	2	4	2	4	5	5	2	4	5	5	2	5	2	4	0	0
1	No paper after wheel	0	1																																
2	More than one sheet out	1																																	
3	Wheel still			1																															
4	Motor still				1																														
5	Motor temp high					1																													
6	Motor temp highhigh					0	1																												
7	Drive error							1																											
8	Bearing temp high								1																										
9	Bearing temp highhigh								0	1																									
10	No paper after buffer		0																																
10	No paper after buffer												1																						
11	Buffer empty										1	0	0																						
12	Buffer full										0	1																							
13	Wheel 1 differs from wheel 2													1																					
14	Bearing 1 temp high														1																				
15	Bearing 1 temp highhigh														0	1																			
16	Bearing 2 temp high																1																		
17	Bearing 2 temp highhigh																0	1																	
18	Motor stopped																		1																
19	Motor temp high																			1															
20	Motor temp highhigh																			0	1														
21	Vacuum not ok in hose																					1	0			0	0			0					
22	Vacuum not ok in valve 1																						x			x									
23	Vacuum notnot ok in valve 1																						x			x									
24	Vacuum not ok in valve 2																										x			x					
25	Vacuum notnot ok in valve 2																										x			x					
26	Valve 1 in wrong pos.																							1	1										
27	Valve 2 in wrong pos																											1	1						
28	Valve 1 closed																					1		0	1	1									
29	Valve 2 closed																					1						0	1	1					
30	Pump stopped																					0									1				
31	Pump temp high																					0										1			
32	Pump temp highhigh																					0										0	1		
	No sheets delivered in																																	1	
	Electrical error																																		1

1-3	Lowest priority
4-6	Medium priority
7-9	High priority
▨	Machine fault
🟦	Process fault

B. Evaluating Tests

To be able to verify the suggested model in some way a table, shown in Appendix A, is set up, corresponding to the Components, Exceptions and indications of the Magazine in the TAA. With this table some tests, similar to real behaviour, are done to determine how well the method is working. These were done through some steps:

- Decide on what has gone wrong in the machine.
- Mark the indications in the table that will be triggered because of the(se) wrong(s)
- From the indications, find what exceptions that are triggered.
- From the triggered exceptions, find the Root Causes
- Compare the Root Causes in point 1 and point 4 and evaluate.

B.1 Belt Stopped

Table B.1 The indications, Exceptions and Root Exceptions.

Triggered Indications Table	Complete Exceptions Table	Root Exceptions Table
01 10 17 18	MAG:BUFF:03 MAG:BELT:05 MAG:BELT:06	MAG:BELT:06 (Engine stopped)

A case where the conveyor belt stops because of the bearing being faulty, table B.1. Because of this Exception there won’t be any paper out of the buffer and because of that the Jam-exception occurs. The reason that the engine stopped-exception is showed instead of the faulty bearing is that the engine stopped is the main alarm even if the bearing caused the stop. By controlling the Complete Exceptions List and there seeing the bearing-exception you get the cause of the stop.

B.2 Vacuum Pump Stopped

Table B.2 The indications, Exceptions and Root Exceptions.

Triggered Indications Table	Complete Exceptions Table	Root Exceptions Table
1 10 21 22 23 24 25 30 31 32	MAG:BUFF:03 MAG:VAC:10 MAG:VAC:12	MAG:VAC:10 (Pump stopped)

A case where the vacuum engine stopped and became warm, table B.2. This causes bad pressure and several indications. When the vacuum isn't working it's not possible to get any paper out of the buffer and then an exception of jam in the buffer occurs.

B.3 Separator Engine Hot and Leakage in Vacuum Hose

Table B.3 The indications, Exceptions and Root Exceptions.

Triggered Indications Table	Complete Exceptions Table	Root Exceptions Table
1 5 10 21 28 29	MAG:SEP:05 MAG:BUFF:03 MAG:VAC:01	MAG:SEP:05 (Engine warm) MAG:VAC:01 (Leakage in hose)

In this case two Root Causes occur at the same time, table B.3. The separator engine becomes hot and the vacuum hose has pressure fault. When the vacuum is faulty the jam from the buffer is noticed. The vacuum is a Subfunction to the separator but because the separator alarm is a machine related alarm it is always noticed and therefore two correct Root Causes are identified.

B.4 Belt Bearing and No Sheets from Previous Function Module

Table B.4 The indications, Exceptions and Root Exceptions.

Triggered Indications Table	Complete Exceptions Table	Root Exceptions Table
1 10 11 14 19 No sheets in	MAG:BUF:01 MAG:BELT:02 MAG:BELT:07 MAG:IN_ARK:01	MAG:BELT:02 (Bearing 1 warm) MAG:IN_ARK:01 (Earlier component error)

In this case there are also two Root Causes, table B.4. While there are no sheets coming from the previous parts of the machine there is a bearing in the conveyor belt that is faulty. The reason that it was the bearing-exception that was identified as the Root Cause and not the engine hot is the time that was noticed. Because the bearing-exception came first it is considered that this is causing the engine hot, and not the other way around.

B.5 Bad Pressure in One Valve and Jam in the Buffer

Table B.5 The indications, Exceptions and Root Exceptions.

Triggered Indications Table	Complete Exceptions Table	Root Exceptions Table
10 24	MAG:BUFF:03 MAG:VAC:06	MAG:VAC:06 (Vacuum in valve 2 not ok)

In this case two exceptions are noticed, from these two the Root Cause is the vacuum-exception because the buffer is dependent on the vacuum, table B.5. But in this case both exceptions are Root Causes and this is not noticed. This will be noticed when the thought Root Cause is corrected and when start running again the next invisible Root Cause will be displayed.

C. Implementation Considerations

In this appendix there are some sections containing different things to have in mind when implementing the Exception Handling, but are not really connected to the thesis as it turned out.

C.1 Alarm Groups

All exceptions in a machine don't share the same priority. To establish a priority structure, exceptions are divided into three different priority groups. The groups are labelled by colour, with blue exceptions being mere notices to the operator and not really anything wrong, yellow alarms causing the machine to stop in a controlled way and require the operator to take some action. The most severe exception group is the red one. If a red exception occurs the machine emergency stops immediately. In the red exception group the ones that mean human danger and severe fault on the machine end up. Two suggestions on how to distribute the priorities has been made by (Campbell, Brown, O'Donnell, 1997) and the (HSE 2000) and they are very similar. Campbell, Brown, O'Donnell suggests a distribution of 10/20/70% of high/medium/low priority. HSE suggest 5/15/80% for the same priorities. As they are similar it is a good suggestion to try to approximately achieve these distributions in order to optimize performance.

Since there are several phases during the production the alarms is varying in importance. Some of the alarms are only interesting in some phases and therefore the alarm priorities may be changed between them.

Besides the colour groupings it is possible to do other groupings. One of these is to divide the alarms into alarms from the process and alarms from the machine. The optimal situation of this is to only have process alarms as root alarms, but in practice this is not likely.

C.2 Nuisance Indication Removal

In order for an effective Exception Handling to be implemented it is a prerequisite to have suitable indications that are behaving as expected. According to (Liu et al., 2001) two of the most important things that affect the ability to create good Exception Handling are the placement and the limits of the sensors.

The placement of a sensor is very important to get a good Exception Handling. If there are sensors triggering exceptions measuring things that actually aren't interesting for the Exception Handling, it could be appropriate to remove them and only place sensors where you want a measurement and an indication (unless, of course, the sensor is needed for the control system). Optimizing the number of indications will improve the conditions for a good Exception Handling. This action is favourable to machines that are being designed. When the machine is made and working it is more difficult to move and change the existing exception.

To do an inspection of the indication limits is also a good preparation for the Exception Handling. Often the sensor limits are far from the reasonable causing unnecessary indications or no indications at all. Some sensor limits are far too generous and cause no indication when it in fact should occur. Sometimes the issue is the opposite; the sensor limits are to tough and cause indications far more often than it should. A sensor limit inspection is often useful to do both on a machine being designed and to a working machine.

C.3 Operator Interaction

With a good Exception Handling the possibilities of what to show to the operator increases drastically. The first thing the operator wants to know when the machine stops is what the Alarm Handling system thinks is the Root Cause of the problem. This is the key problem that the system has to solve, but sometimes it can be good for the operator to have some more information about what really happened. As an option for the operator the system should therefore be able to show more information about what really happened, when the operator asks for it.

There are two more views to choose from. The first is viewing all of the identified exceptions, not only the Root Causes. Helped with these, the cause of the stop might become clearer. The second possibility is viewing every triggered indication.

If the diagnostics have been implemented and this provides a suggestion to solve the problem this has to be presented to the operator. The result from the statistical diagnostics is going to be shown if the diagnostics is implemented. What could be shown here is what exceptions might be coming up and perhaps a suggestion on how to prevent this. The diagnostics part may also be able to present the frequency of the alarms, and tell if the frequency is remarkably high for any exception.

D. Walkthrough of suggested solution – The Windshield Vision example

In order to illustrate the structures, workflow and run-time behaviour of the suggested methodology this appendix will provide a complete walkthrough of an every-day problem that is fairly easy to understand.

The example will include the construction of a dependency structure with all of its bits and pieces and a step by step walkthrough of a root cause analysis case during assumed run-time.

The example viewed will be part of an FMT for an arbitrary automobile, focusing on the function that makes sure the driver has sufficient view through his windshield. It is assumed that the FMT is already constructed according to figure D.1

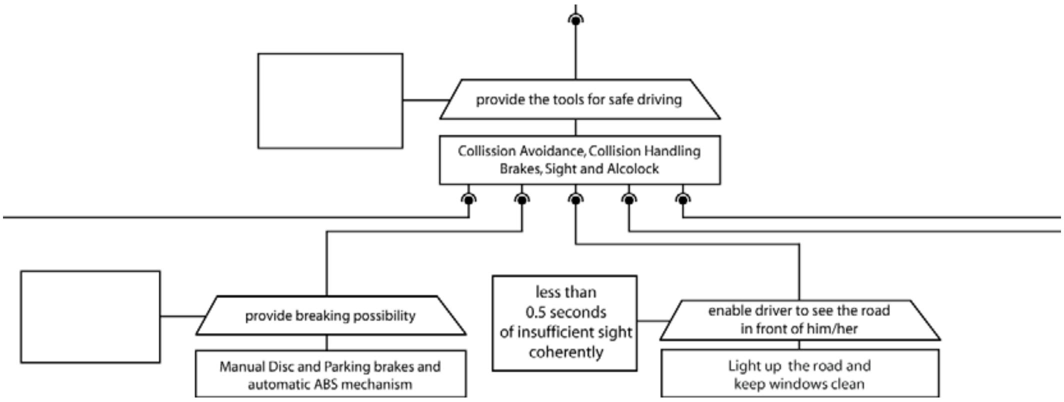

provide the tools for safe driving
Collission Avoidance, Collision Handling Brakes, Sight and Alcolock
provide breaking possibility
Manual Disc and Parking brakes and automatic ABS mechanism
less than 0.5 seconds of insufficient sight coherently
enable driver to see the road in front of him/her
Light up the road and keep windows clean

Fig. D.1 Detail from the Function-Means Tree of the automobile.

D.1 Building the Dependency structure

A function in the FMT represents a task that needs to be fulfilled. When realizing its mean as a dependency structure, the structure's goal will be to fulfil that mean. In practise, this means that the dependency structure goal will coincide with the function description from the FMT. The first step in the construction of the dependency structure is hence taken by putting "Enabling driver to see the road in front of him" as the structure's goal.

In the FMT from figure D.1 it has already been decided that the mean for enabling the driver to see in front of him, should be "Light up the road and keep windows clean". The next step taken is therefore realizing the mean by constructing the rest of the dependency structure below the goal.

First of all the functions needed to achieve the goal are defined, for the defined goal it is pretty straightforward to define two functions, "Light road" and "Clean Window" put it is not always intuitive to find the best structure. The functions are put under the goal, directly connected to it, see figure D.2 below.

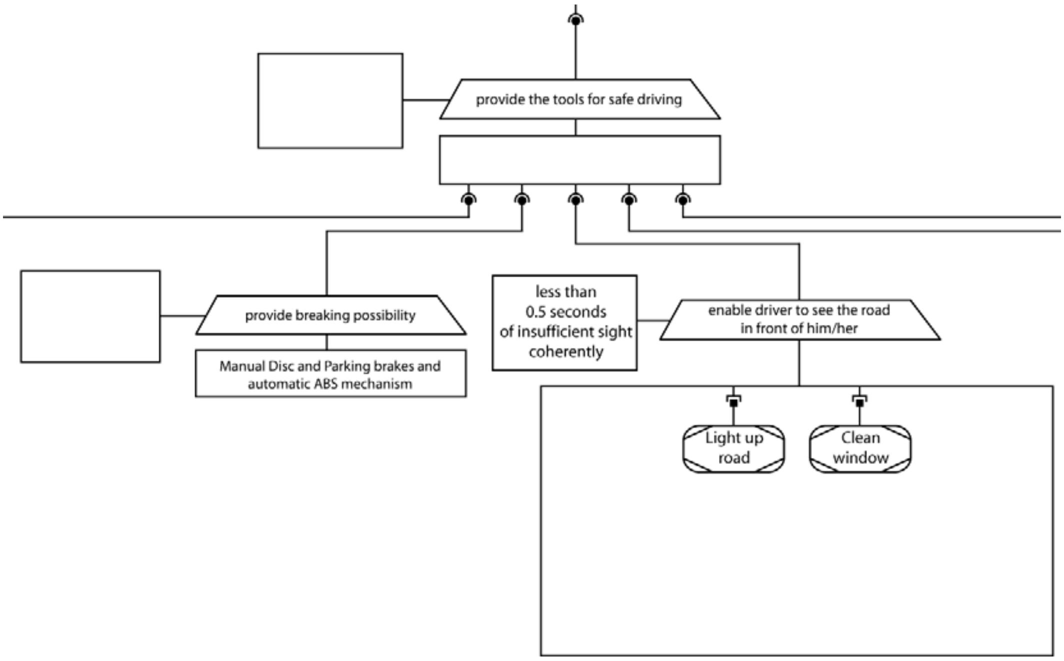

provide the tools for safe driving
provide breaking possibility
Manual Disc and Parking brakes and automatic ABS mechanism
less than 0.5 seconds of insufficient sight coherently
enable driver to see the road in front of him/her
Light up road
Clean window

Fig. D.2 The dependency structure taking form.

Once the functions are in place, their respective requirements are analysed and defined as ingoing connections to the respective functions. In order to obtain inputs for them, subfunctions and ins are needed. Subfunctions correspond to physical modules performing a specific task such as the car's headlights or the magazine's vacuum pump. Ins are links to more complex systems that are defined as components on their own. The subfunctions that are assigned to represent the structure's function requirements are: Headlights, Washer System (tank, pump, hoses and nozzles for the washer fluid), Windshield Wipers and Defogger Fan. These are added and are connected to one or more functions as seen in figure D.3.

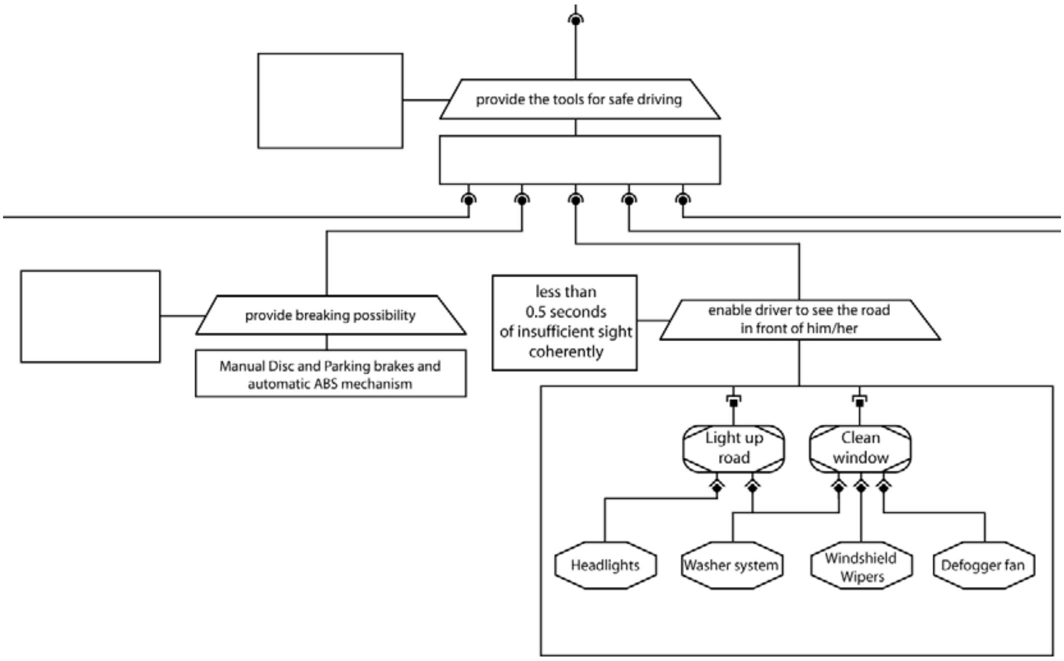

provide the tools for safe driving
provide breaking possibility
Manual Disc and Parking brakes and automatic ABS mechanism
less than 0.5 seconds of insufficient sight coherently
enable driver to see the road in front of him/her
Light up road
Clean window
Headlights
Washer system
Windshield Wipers
Defogger fan

Fig. D.3 Further development of the structure.

The subfunctions might have requirements of their own, either other subfunctions or ins. These requirements are analysed and the needed parts are included. All four of the subfunctions require the Power Supply to be able to deliver the DC they need to run and the Washer System require that there is washer fluid available in the tank. Both are modelled as Ins, arcs are connected to represent dependencies. The dependency structure is then complete as seen in fig. D.4 below.

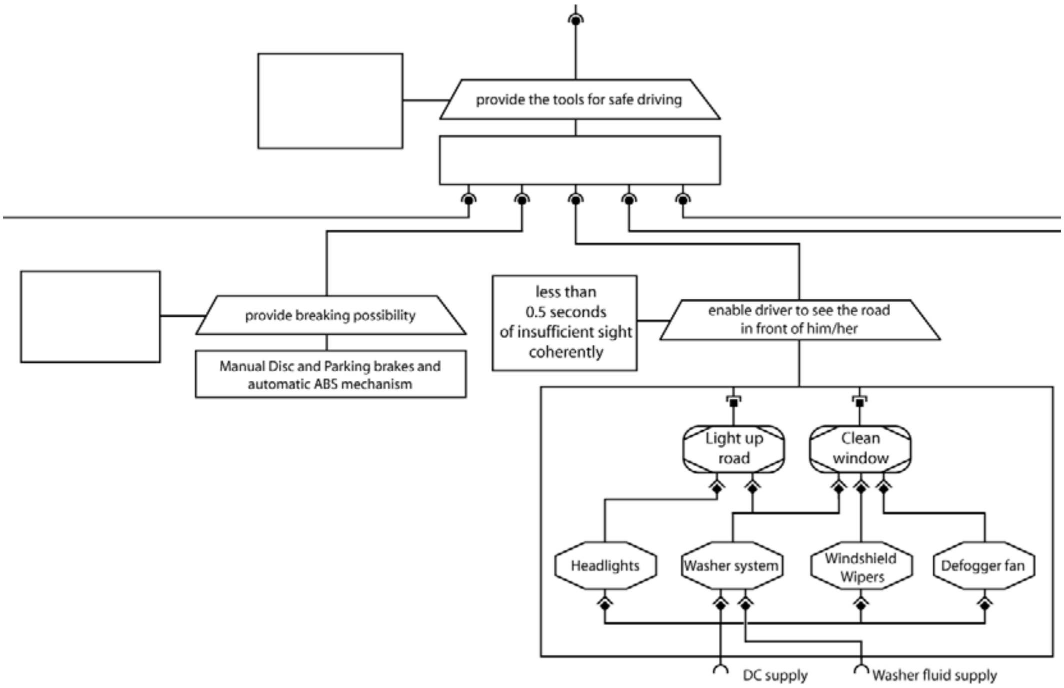

provide the tools for safe driving
provide breaking possibility
Manual Disc and Parking brakes and automatic ABS mechanism
less than 0.5 seconds of insufficient sight coherently
enable driver to see the road in front of him/her
Light up road
Clean window
Headlights
Washer system
Windshield Wipers
Defogger fan
DC supply
Washer fluid supply

Fig. D.4 The complete dependency structure and its close surroundings.

Propagation possibilities can be seen from the complete Dependency Structure (fig. D.8). It illustrates both how exceptions may propagate and the linkage between the FMT and the dependency structure. As each function-mean couple will in practice correspond to a component and each mean to a dependency structure, the dependency structure can in fact be seen as part of the FMT. The FMT will then be illustrating the hierarchic dependency of the components.

D.2 Constructing Process Matrixes

Once the dependency structure is complete, each function and subfunction is in turn realized with a Process Matrix specifying what exceptions it should handle and what indications to use to identify them.

A rather rough overview risk analysis (as described in Chapter 5) should be performed in order to decide what of the component's functions/subfunctions are the most crucial to process performance. Once that is done, analysis of each of the functions and subfunctions are carried out, focusing mostly on the parts that were identified as most critical in the overview. Those of the identified exceptions that are judged important enough to be made identifiable are entered in the columns of the table that will form the process matrix. This is shown in figure D.5 below.

	Defogger Fan				
	Fan Motor Broken Down	Fan Motor Bearing Worn Out	Motor Coupling Disconnected	Air Pipe Blocked	Heater Element Broken

Fig. D.5 The first step towards a process matrix.

Every exception is assigned a type (Outer/Inner influence) and the exceptions are grouped into bunches of exceptions out of which only one per group must be presented to the user. For each operational state (for the automobile assumed to be “Running” and “Parked”), priorities in the range 0-10 are assigned to the exceptions. The most severe exception in the defogger fan subfunction is “Fan Motor Broken Down”. As the “Fan Motor Broken Down” and “Fan Motor Bearing Worn Out” correspond to the same physical component, they are grouped together. Progress so far is illustrated in figure D.6. The groups are identified by thick lines and the types by background colour. As all Defogger exceptions are Inner influence exceptions, they all share the same colour.

More exceptions can be seen in the complete process matrix in figure D.9.

	Defogger Fan				
	Fan Motor Broken Down	Fan Motor Bearing Worn Out	Motor Coupling Disconnected	Air Pipe Blocked	Heater Element Broken
Priority driving	3	2	2	2	2
Priority parked	1	1	1	1	1

Fig. D.6 Exception priorities, types and groups.

Once exceptions are listed it is time to find ways of indicating them. Required indications are found and are then listed on the rows of the matrix.

When both columns and rows are filled out the only part remaining is to connect them with logic as described in Section 7.3. The final result for the defogger fan –subfunction is shown in figure D.7.

	Defogger Fan				
	Fan Motor Broken Down	Fan Motor Bearing Worn Out	Motor Coupling Disconnected	Air Pipe Blocked	Heater Element Broken
Priority driving	3	2	2	2	2
Priority parked	1	1	1	1	1
Motor stopped (but should be running)	1		0	0	
Motor temp. High		X			
Motor temp. Very High		X			
Low airflow from fan			1	0	
Low airflow from vent			1	1	
Airtemp. not ok					1
Fan stopped	1		1	0	

Fig. D.7 The complete Process Matrix for the Defogger Fan subfunction.

To conclude, the steps are:

Dependency Structure.
1. Define the Goal.
2. Define functions.
3. Define needed ins and subfunctions.
4. Connect them with dependency arcs.
5. Define additional ins and subfunctions.
6. Connect them with dependency arcs.

Process Matrix.
7. Overview risk analysis for component.
8. Risk Analysis for each function/subfunction is carried out.
9. Exceptions are entered in the process matrix.
10. Exceptions are grouped and assigned types and priorities.
11. Find required and suiting indications and enter them in the matrix.
12. Construct logic connecting indication combinations to exceptions.

D.3 Run-time behaviour

To illustrate the run-time behaviour an example is made up. The scenario involves wear, and eventually breakdown, of the electrical motor that drives the defogger fan.

The first event to notice is the triggering of the “Motor Temp High” indication. The matrix shows that one exception, “Fan Motor Bearing Worn Out”, has its conditions fulfilled and is hence raised. As it is the only exception it is also considered the root one. As the driver/operator is not worried by the condition, it gets worse over time. The “Motor Temp Very High” indication is triggered and the pattern matching is repeated. No new exception is detected, though, as only one exception was assigned the failure mode.

Time goes by and eventually the motor has had it and comes to a permanent halt, meaning that indications for “Motor stopped” and “Fan stopped” trigger. The conditions for the “Fan Motor Broken Down” exception are fulfilled and two exceptions are thus raised. As they are both within the same subfunction and group only one must be a root. The “Broken Down” has a higher priority and therefore is determined to be the root cause and masks out the other.

The stopped fan leads to fogging of the windshield, in terms of indications meaning the triggering of “Insufficient vision through Window” and “Inside Window Humidity high”. The process matrix identifies and raises the “Windshield Very Foggy” exception located in the “Clean Window” function. The exception is not judged to be root, though, as an exception is still raised in the “Defogger Fan” subfunction that poses a requirement for “Clean Window”. “Windshield Very Foggy” is masked out but as it has a higher priority the “Fan Motor Broken Down” inherits it.

As is shown in figure D.8 below, the procedure has used the six triggered indications, used them to identify three exceptions and correctly judged one of them to be the root of the others.

Triggered Indications Table
Motor temp. High
Motor temp. Very high
Motor stopped
Fan stopped
Insuff. Vision through wind.
Inside window Hum. High

Complete Exceptions Table

Pr.	Descr.
2	Fan Motor Bearing Down
3	Fan Motor Broken Down
5	Windshield foggy

Root Exceptions Table

Pr.	Descr.
5	Fan Motor Broken Down

Fig. D.8 TIT, CET and RET for the example

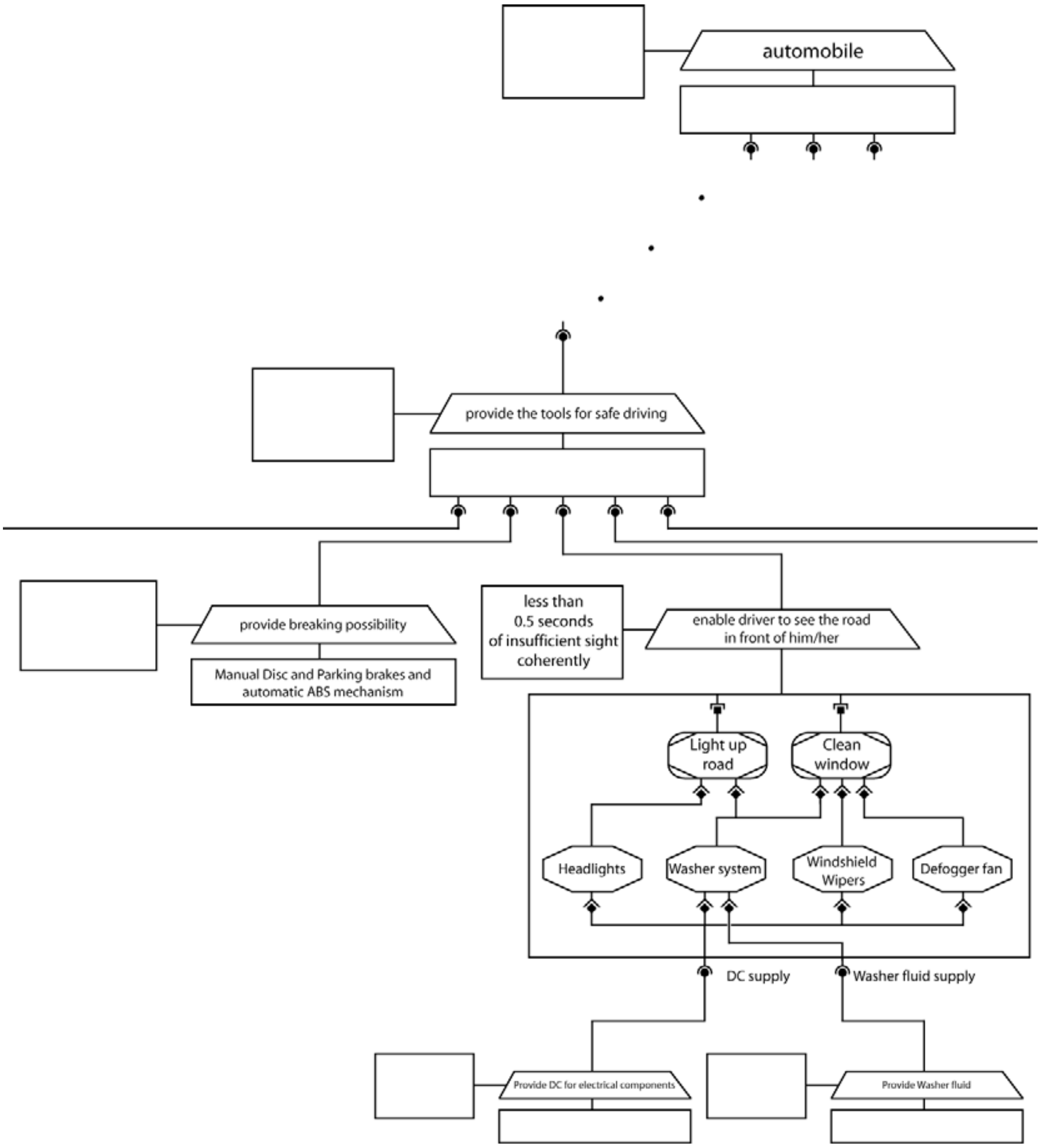
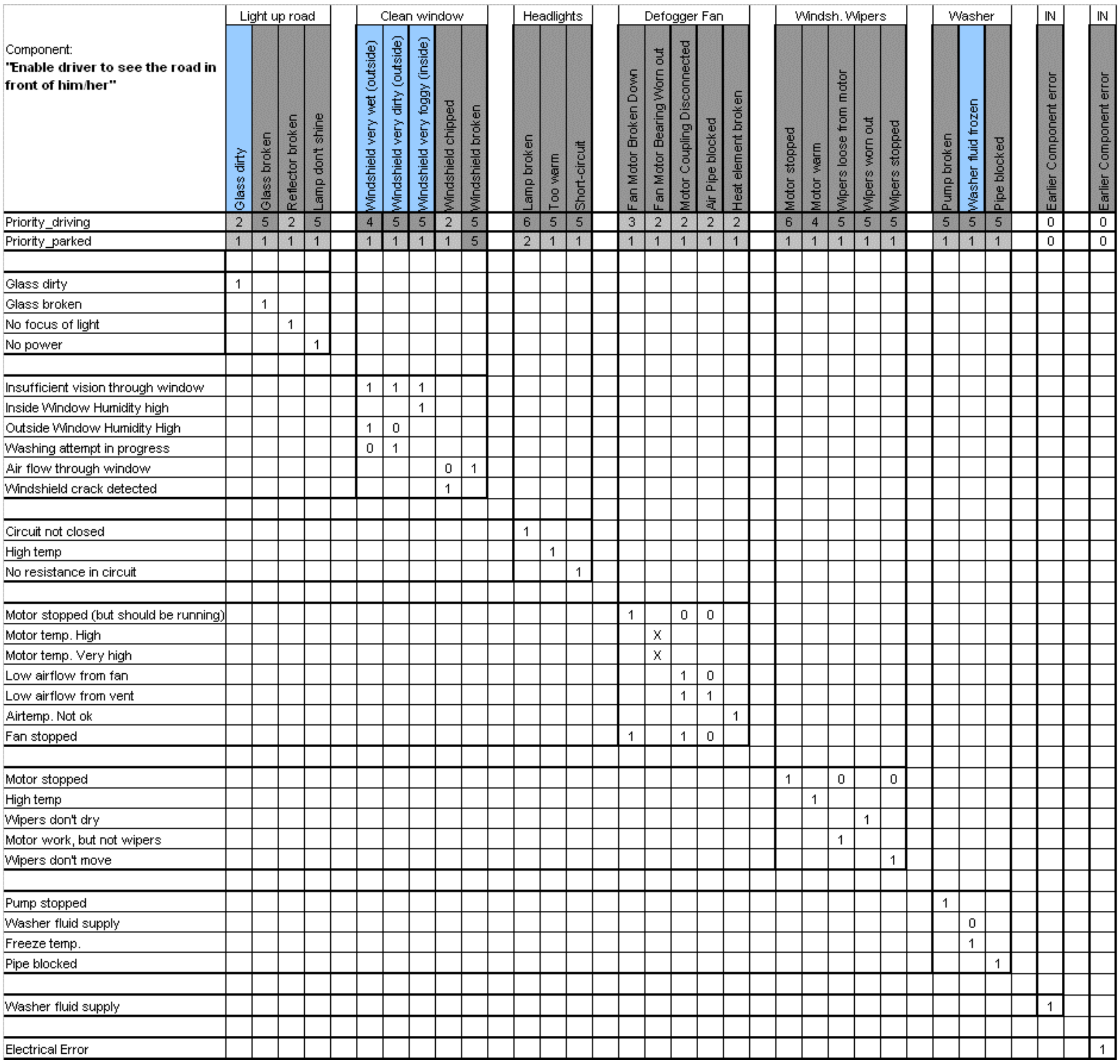

automobile
provide the tools for safe driving
provide breaking possibility
Manual Disc and Parking brakes and automatic ABS mechanism
less than 0.5 seconds of insufficient sight coherently
enable driver to see the road in front of him/her
Light up road
Clean window
Headlights
Washer system
Windshield Wipers
Defogger fan
DC supply
Washer fluid supply
Provide DC for electrical components
Provide Washer fluid

Fig. D.9 Detail from the complete FMT/Dependency Structure.



Component: **"Enable driver to see the road in front of him/her"**	Light up road					Clean window						Headlights				Defogger Fan						Windsh. Wipers						Washer				IN		IN
	Glass dirty	Glass broken	Reflector broken	Lamp don't shine		Windshield very wet (outside)	Windshield very dirty (outside)	Windshield very foggy (inside)	Windshield chipped	Windshield broken		Lamp broken	Too warm	Short-circuit		Fan Motor Broken Down	Fan Motor Bearing Worn out	Motor Coupling Disconnected	Air Pipe blocked	Heat element broken		Motor stopped	Motor warm	Wipers loose from motor	Wipers worn out	Wipers stopped		Pump broken	Washer fluid frozen	Pipe blocked		Earlier Component error		Earlier Component error
Priority_driving	2	5	2	5		4	5	5	2	5		6	5	5		3	2	2	2	2		6	4	5	5	5		5	5	5		0		0
Priority_parked	1	1	1	1		1	1	1	1	5		2	1	1		1	1	1	1	1		1	1	1	1	1		1	1	1		0		0
Glass dirty	1																																	
Glass broken		1																																
No focus of light			1																															
No power				1																														
Insufficient vision through window						1	1	1																										
Inside Window Humidity high								1																										
Outside Window Humidity High						1	0																											
Washing attempt in progress						0	1																											
Air flow through window									0	1																								
Windshield crack detected									1																									
Circuit not closed												1																						
High temp													1																					
No resistance in circuit														1																				
Motor stopped (but should be running)																1		0	0															
Motor temp. High																	X																	
Motor temp. Very high																	X																	
Low airflow from fan																		1	0															
Low airflow from vent																		1	1															
Airtemp. Not ok																				1														
Fan stopped																1		1	0															
Motor stopped																						1		0		0								
High temp																							1											
Wipers don't dry																									1									
Motor work, but not wipers																								1										
Wipers don't move																										1								
Pump stopped																												1						
Washer fluid supply																													0					
Freeze temp.																													1					
Pipe blocked																														1				
Washer fluid supply																																1		
Electrical Error																																		1

Fig. D.10 The Process Matrix for the Seeing-component.