

ISSN 0280-5316  
ISRN LUTFD2/TFRT--5811--SE

# The Application of Agent-Based Technology to Packaging Line

Johannes Alenius  
Mats Millnert

Department of Automatic Control  
Lund University  
January 2008



<b>Lund University</b> <b>Department of Automatic Control</b> <b>Box 118</b> <b>SE-221 00 Lund Sweden</b>		<i>Document name</i> MASTER THESIS	
		<i>Date of issue</i> January 2008	
		<i>Document Number</i> ISRN LUTFD2/TFRT--5811--SE	
<i>Author(s)</i> Johannes Alenius and Mats Millnert		<i>Supervisor</i> Claudio Donati at Tetra Pak, Modena Italy Anders Rantzer Automatic Control in Lund (Examiner)	
		<i>Sponsoring organization</i>	
<i>Title and subtitle</i> The Application of Agent-Based Technology to Packaging Line (Att utnyttja agent-baserad teknologi för en förpackningslina)			
<i>Abstract</i> Control systems used by manufacturing companies today are often centralized. In such a system, the controller is concentrated to one location. As the production lines in Tetra Pak becomes more and more complex with growing costumer demand, a centralized approach becomes more and more inadequate. Agent-based technology provides a way to implement a desirable, robust and decentralized manufacturing environment. Agents can be viewed as generalizations of objects in object oriented programming, which takes decisions based on own intelligence as well as others. A community of agents, efficiently cooperating in order to reach a higher level or global goal, is called a MAS (Multi Agent System). This thesis deals with applying an agent based solution into the Tetra Pak A3 packaging line. A thorough study of decentralized systems in manufacturing environments is made which finally leads to the choosing of CBR (Case Based Reasoning) as reasoning paradigm for the agents. Agents adapt solutions to new problems by initializing voting procedures among all agents in the community. FIPA (the Foundation for Intelligent Physical Agents) provides specifications regarding infrastructure of the architecture surrounding the agents with focus on communication and organization. Studies of seven different cases is carried out to verify the strength of the control policies developed. Agent behaviour and line configuration is simulated in the Line Simulator implemented in Matlab.			
<i>Keywords</i>			
<i>Classification system and/or index terms (if any)</i>			
<i>Supplementary bibliographical information</i>			
<i>ISSN and key title</i> 0280-5316			<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 71	<i>Recipient's notes</i>	
<i>Security classification</i>			



# Contents

0.1	Abstract . . . . .	3
<b>1</b>	<b>Introduction</b> . . . . .	<b>4</b>
1.1	Background . . . . .	4
1.2	Objective . . . . .	4
1.3	Problem Formulation . . . . .	5
<b>2</b>	<b>The Tetra Pak Packaging Line</b> . . . . .	<b>6</b>
2.1	Introduction . . . . .	6
2.1.1	The monitoring of the packaging line . . . . .	6
2.2	Packaging Line Machines . . . . .	7
2.2.1	Tetra Pak A3/flex . . . . .	7
2.2.2	Accumulator Helix . . . . .	8
2.2.3	Distribution Equipment . . . . .	9
2.2.4	Divider . . . . .	9
2.2.5	Merger . . . . .	10
2.3	Machine States . . . . .	10
2.4	Emergency Situations . . . . .	10
2.5	Different Packaging Line Configurations . . . . .	11
<b>3</b>	<b>Agent-Based Technology</b> . . . . .	<b>12</b>
3.1	Introduction . . . . .	12
3.1.1	Background . . . . .	13
3.2	Agent architecture . . . . .	13
3.2.1	Data table . . . . .	13
3.2.2	Diagnostics Element . . . . .	14
3.2.3	Planner Element . . . . .	15
3.2.4	Flow Model . . . . .	17
3.2.5	Execution control . . . . .	17
3.3	Conclusion . . . . .	17
<b>4</b>	<b>Multi-Agent Systems</b> . . . . .	<b>18</b>
4.1	Main Purpose . . . . .	18
4.2	Autonomy of Agents . . . . .	18
4.3	Holonic Manufacturing Systems . . . . .	19
4.4	Multi-Agent System for Real-Time Control . . . . .	19
4.5	Agent Communication . . . . .	19
4.6	Architectural Overview . . . . .	21
4.6.1	Agents and Services . . . . .	21

4.6.2	the Multi-Agent System platform . . . . .	21
4.7	Areas of Application . . . . .	22
4.8	Benefits and Drawbacks . . . . .	23
4.8.1	Advantages . . . . .	23
4.8.2	Disadvantages . . . . .	23
<b>5</b>	<b>Application of Agent-Based Technology</b>	<b>25</b>
5.1	The Agent . . . . .	26
5.1.1	Data table . . . . .	26
5.1.2	Diagnostics Element . . . . .	26
5.1.3	Planner Element . . . . .	27
5.1.4	work flow . . . . .	32
5.1.5	Flow Model . . . . .	37
5.2	Design of the Multi-Agent System . . . . .	37
5.2.1	Multi-Agent System Platform . . . . .	37
5.2.2	Directory Facilitator . . . . .	38
5.2.3	Interaction Rules . . . . .	39
5.2.4	Voting Rules . . . . .	39
<b>6</b>	<b>Production Case Studies</b>	<b>40</b>
6.1	Production Line Cases . . . . .	40
6.1.1	Case 1 . . . . .	40
6.1.2	Case 2 . . . . .	40
6.1.3	Case 3 . . . . .	41
6.1.4	Case 4 . . . . .	41
6.1.5	Case 5 . . . . .	41
6.1.6	Case 6 . . . . .	41
6.1.7	Case 7 . . . . .	41
6.2	Case Verification . . . . .	41
6.2.1	Case 1 . . . . .	41
6.2.2	Case 2 . . . . .	43
6.2.3	Case 3 . . . . .	44
6.2.4	Case 4 . . . . .	44
6.2.5	Case 5 . . . . .	45
6.2.6	Case 6 . . . . .	45
6.2.7	Case 7 . . . . .	46
<b>7</b>	<b>Simulation</b>	<b>49</b>
7.1	Introduction . . . . .	49
7.2	Design Description . . . . .	49
7.2.1	Improvements from older version . . . . .	49
7.2.2	Agent software structure . . . . .	50
7.3	Simulation Cases . . . . .	53
7.4	Simulation Results . . . . .	54
7.4.1	Case Verification - Case 1 . . . . .	54
7.4.2	Case Verification - Case 7 . . . . .	55
7.4.3	Case Verification - Case 3 . . . . .	56
7.5	conclusion . . . . .	57

**0.1 Abstract**

# Chapter 1

## Introduction

### 1.1 Background

Tetra Pak is a multinational food packaging company which was founded in 1951 in Lund, Sweden, by Ruben Rausing and Erik Wallenberg. Tetra Pak has since been involved in the food packaging industry. Most notably it has been one of the companies who have been pushing the food packaging technology forward. Tetra Pak has been doing this by introducing a wide range of new packaging methods and concepts.

When Tetra Pak in 1951 emerged on the market they first revolutionized the food packaging industry with a new form of aseptic liquid food packaging method. The method is still today one of the mainstays of the Tetra Pak packaging concept. This way to package liquid food makes it possible to store beverages for up to a year. The reason for the big impact of this new packaging technology is that it made it a lot easier for the food producers to transport their products over great distances without a cooling chain, and thus being able to reach a large number of consumers relatively cost efficient.

Tetra Pak have in recent years had much success with their self developed packaging lines. One of the principal ideas has been to give the customers as much freedom of choice as possible regarding types of packages, caps etc. Tetra Pak is also able to give the customer (for example a dairy) the opportunity of alternating between different types of package attributes, such as caps, within the same packaging line, which means more freedom for the customer. Due to this fact the packaging lines are growing more and more complex. The problem of controlling the package flow is no longer trivial. There is always a need for better process control in order to reduce package waste, working hours and production standstill. It is thus important for a company like Tetra Pak to always be updated with the latest control technologies. In this thesis we will investigate how agent-based technology can be used to control one of the companies packaging lines.

### 1.2 Objective

Tetra Pak want to explore the concepts of agent-based technology in order to get a better understanding of the technology, and how it can be applied to their



type of production. The main area in which Tetra Pak wants to improve is the steadiness of the production flow, also they want to know if the agent-based technology can handle the branching of the packaging line in a satisfactory way. We say a packaging line is branched when there are more than one path available for the packages to flow. Our objective is to investigate how agent-based technology can be applied to the packaging line. Further we shall design an agent-based control system for the line and show how it functions by applying it to a simulation of the packaging line.

### 1.3 Problem Formulation

The growing demand on the actors of the food packaging market of today is forcing the food producers to make changes in their production more often. The changes can for example be between different types of cap applicators, or alternation between a handle applicator and a straw applicator. With the packaging lines traditionally being used, these changes can be both expensive and time consuming. Thus a market for packaging lines where these types of changes can be made swiftly is emerging. It is not only changes in the packaging line configuration, but also of the packaging line itself that must be made easily and cheaply. It is crucial that the food producer is able to add or remove components of the line inexpensively. It is also of big importance that the number of persons required to operate the production line is kept at a minimum. Further it is also important that the waste generated by the packaging line isn't unnecessarily high.

One stage in achieving the desired flexibility mentioned above is to move away from the standard method of a centrally controlled line. A large centrally controlled system often means inflexibility and a high susceptibility of failure, and when failure is due they are often severe to the extent of paralyzing the entire line. This is mainly due to the fact that it's virtually impossible for a central control system to anticipate all of the possible control situations which can arise in the system. Naturally there is a desire from Tetra Pak to make its production line better in a way that prevents these types of problems. Therefore they are interested in investigating the possibilities of applying agent-based technology when they designing the line controllers to their packaging lines. For now we can explain agent-based technology as an aspiration to distribute the control and decision making in the plant. Rather than having a centrally controlled system you instead strive to have the controller near the actual place where the real component is. One way to put it is that you want the intelligence close to the place where the decision must be taken.

The objective with the thesis is to design an agent-based system to control the packaging line in a satisfactory way according to a number of predefined case scenarios (see section 6). The performance of the control system should be demonstrated on a simulation of the real packaging line.

## Chapter 2

# The Tetra Pak Packaging Line

### 2.1 Introduction

Tetra Pak provides a wide range of packaging line solutions. Tetra Pak are today striving to make their line faster, more fault tolerant and more flexible in that they should be easier to reconfigure. Tetra Pak's complete solution includes machines creating the packages, applying caps and straws on the packets and there are also machines able to package the packets in cardboard packages or wrap them in elastic film.

There are mainly three types of machine behavior, they can create, process or transfer a package. Most of the available machines use a combination of the those.

#### 2.1.1 The monitoring of the packaging line

An example of a Tetra Pak packaging line is shown in figure 2.2, as this figure suggests the line provides everything from components which applies straws to the packages to components which packages the packets in cardboard packages. The packaging lines provides the ability to monitor the operational performance of the production thru feedback in various forms. You have, for example packaging counters throughout the whole line. Also you have online monitoring of the machines different states. Further you can also monitor the accumulation level of the helix. Throughout the packaging line there are a number of photocells. Generally there are two placed before each machine. The task of the photocells is to monitor the package flow and notify the controller if there is a queue situation on the packaging line.

The photocells are denominated the speed photocell and the overflow photocell. The photocells are similar in that they notifies the controller when a queuing situation has occurred, the difference between them being severity of the queue. The overflow situation is more severe than the speed situation.

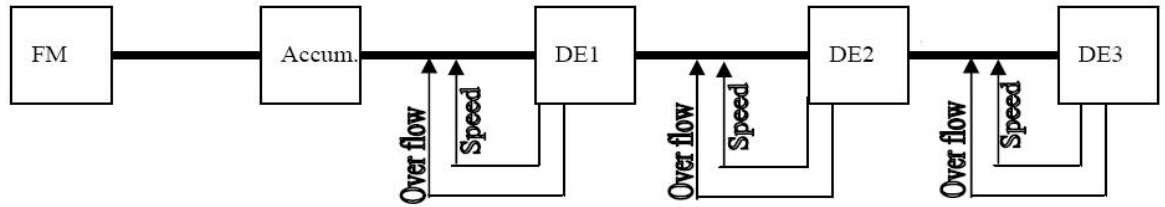


Figure 2.1: Figure showing the location of the speed- and overflow photocells. FM - Filler, Accum. - Helix Accumulator, DE - Distribution Equipment

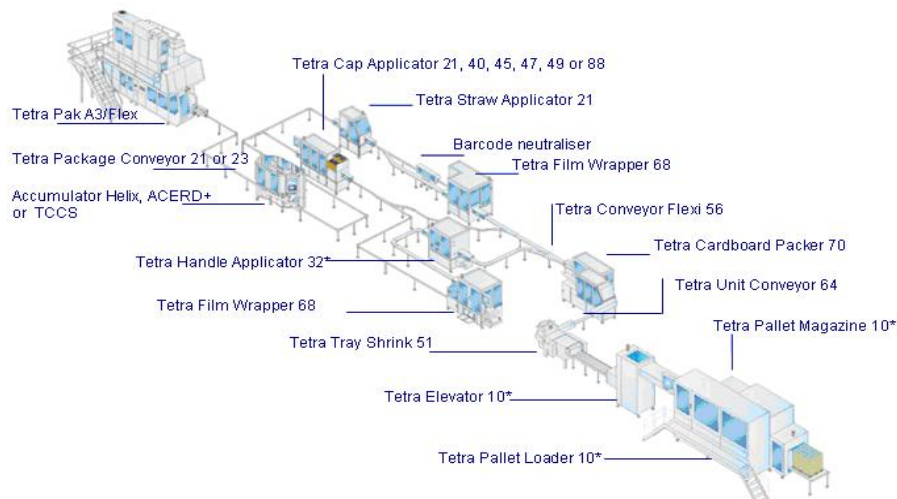


Figure 2.2: Example of a Tetra Pak packaging line

## 2.2 Packaging Line Machines

### 2.2.1 Tetra Pak A3/flex

This is always the first machine in the line configuration. As the source unit of the packaging line it creates the package while it fills it with a liquid of some sort, the Tetra Pak A3/flex is often called the filler machine. The A3/flex can deliver somewhere between 10000 to 25000 packages per hour, depending on package volume. The machine is very accurate in that you can control the package flow very precisely. The actual outflow do not vary from the given input signal. The only problem with the filler rises when you are trying to change the delivery speed. You do not want to end up in a situation where the delivery speed of the filler machine must be changed. Instead problems like this should be handled by the Helix accumulator downstream. You also always try to avoid, except in emergencies, to stop the filler machine completely of reasons other than a

scheduled stop. This is because much waste is generated and it takes a long time to restart the Filler after an unscheduled stop. The machine is shown in Figure 5.19



Figure 2.3: the Tetra Pak A3/flex, often called the Filler machine.

### 2.2.2 Accumulator Helix

The Accumulator Helix is, as the name implies, a distribution-equipment that has the ability to accumulate packages. It consists of two parallel conveyors, which operate at the same speed, yet are independent of one another. One conveyor for infeed and one for outfeed, there is also a floating gear which regulates the accumulation level of the machine, this is shown in figure 2.4. For example, when a stop occurs on one of the two conveyors will slow down or stop moving (the infeed conveyor if the stop occurs upstream or the outfeed conveyor if the stop occurs downstream). In this situation the floating gear will immediately register the stoppage and start moving up/downwards in an expanding spiral causing the unaffected conveyor to accumulate packages. The machine is shown in Figure 2.5

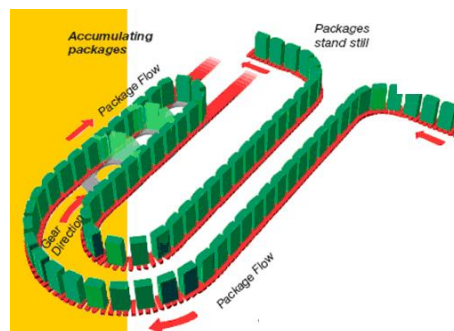


Figure 2.4: The accumulation mechanism of the Helix Accumulator



Figure 2.5: the Helix Accumulator.

### 2.2.3 Distribution Equipment

A wide range of machines goes under the distribution equipment epithet. Those which are interesting in our case are equipment with attributes similar to the Tetra Straw Applicator or the Tetra Cap applicator. The features specific for the distribution machines is that they have two conveyors, one infeed and one outfeed. In between the conveyor is a process unit, this is where the package is being changed.

The only attributes we are concerned with is the speed of the infeed and the outfeed conveyor and the time it takes for the machine to process the package i.e. the delay of the machine. Other types of distribution equipments are the Film Wrapper and the Tetra Cardboard Packer. They group a number of packages and pack them in cardboard or wrap them in clear film respectively. These machines have a behavior that could be likened to an accumulator of small capacity since the machines each time must accumulate enough packages to fill the cardboard package or the film wrapping.

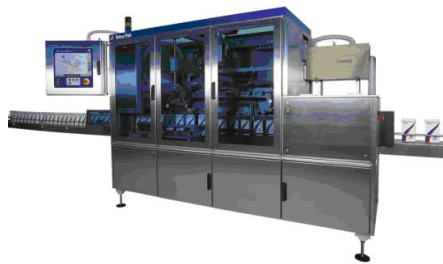


Figure 2.6: example of a Distribution Equipment, here a Cardboard Packer

### 2.2.4 Divider

The Divider is only present in the production line when the production flows in different branches. It is a very simple type of equipment, its role is to divide the package flow from the one line upstream to two lines downstream. The Divider

do not have any restrictions regarding infeed speed or outfeed speed, it can work as fast as needed.

### 2.2.5 Merger

The Merger works in the opposite way from the Divider. It is only present in the production line when the production flows in different branches. The Mergers objective is to converge the package flow of two production lines into one single production line. The Merger doesn't have any restrictions regarding the speed of the package flow, it can, just like the divider, work as fast as needed.

## 2.3 Machine States

There are a few different states available to describe the condition of the machine. The different states are `BLOCKED`, `READY_FOR_PRODUCTION`, `RECIEVE`, `RECEIVE_AND_DELIVER`, `DELIVER` and `PREPARATION`. The states are concerned with the actual machine and not the agent.

- `BLOCKED` - This state occurs when an internal error prevents the machine from functioning properly. This could, for example, occur due to a packet gone sidelong inside the machine, blocking the package flow. This undesirable situation can only be resolved thru external action by an operator.
- `READY_FOR_PRODUCTION`- This state announces that the machine is ready to start production, it only needs a command telling the machine to start producing, a so called *step up command*.
- `RECIEVE` - The machine has started to produce, though there are only packages on the infeed. No packages are delivered by the machine in this state.
- `RECIEVE_AND_DELIVER` - In this state the machine are both receiving and delivering packages, this is the normal state when the production line is running. The machine is both recieving and delivering packages.
- `DELIVER` - This is the state when no more packages are arriving at the infeed but, there are still packages in the machine to process and deliver i.e. the machine is still delivering packages.
- `PREPARATION` - The machine is preparing to start production, there could for example be a need to apply lubrication on some device.

## 2.4 Emergency Situations

-

- `Speed` - This is the situation when the speed photocell (see section 2.1.1) registers a package queue at the infeed of a packaging machine. The failure is not considered to be severe, it basically serves as a warning saying that a more serious situation might occur if nothing is done to solve the problem.

- Overflow - Just like the speed situation a queue situation has occurred on the infeed conveyor of the machine (see section 2.1.1). The difference from the speed photocell is that the overflow situation is considered to be more severe than the speed situation and action must be taken. In all cases when the overflow photocell is turned on, the speed photocell will also be turned on.
- Helix accumulation - In this situation the maximum accumulation level of the Helix is reached. It is considered to be a very severe situation since there are only two ways to solve this problem. Either you start to unload the Helix, which might create problems further down the production line or you can slow down the Filler machine which in most cases is no option. In reality when the Helix have accumulated packages, the priority of unloading them is very high to avoid the situation described above.
- Machine Blocked - This situation occurs when a machine cannot proceed with production due to some internal failure. The case could for example be a packet gone sidelong inside the machine, blocking the package flow. This is generally a failure that must be solved by a production line operator. This is also a machine state.

## 2.5 Different Packaging Line Configurations

There are two different line configurations that have been considered during our work with the thesis. Both configurations can be seen in figure 6.1.

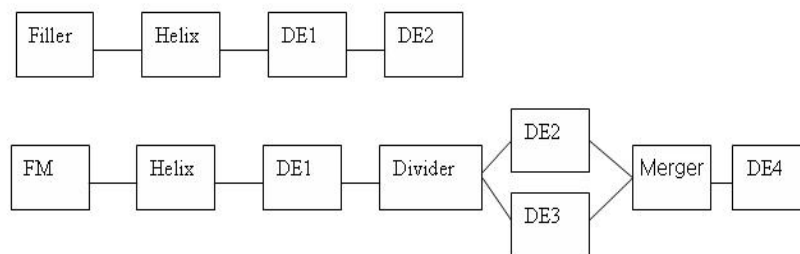


Figure 2.7: the two different line configurations used in the thesis

- FM=Filler Machine
- Helix = Helix accumulator
- DE= Distribution Equipment

The distribution equipments in these configurations is considered to be Straw applicators, cap applicators or similar machines. The lines between the machines should be considered as simple conveyors. As shown in the figure the main difference between the two configurations is the branching of the line.

## Chapter 3

# Agent-Based Technology

### 3.1 Introduction

What exactly is an agent? Is there even a definition? Which are the traditional agent research areas? Agents are everywhere. People encounter intelligent agents, information agents, mobile agents, personal assistant agents. One might wonder if this apparent anarchy makes any sense. What is it that makes an agent? Is there something agents have in common? Is it possible to organize the agents to carry out a task?

First and foremost, agents should be considered as entities within an environment, with attributes useful in a certain domain and that they can sense and act on changes in the environment. This means that agents are not isolated entities, and that they are able to communicate and collaborate with the environment. Agents which aren't able to communicate with their environment would be quite useless, considered as agents.

Once the agents are ready to collaborate with their environment, they need to find the agents with whom they want to collaborate. This would be an easy task if the system of agents (Multi-Agent System) knew exactly which agents to contact and where to find them. This is very seldom the case though. Multi-Agent System tend to be dynamic systems with a population changing over time. The agents need support to find other agents.

The dynamic nature of the Multi-Agent Systems and the complexity of the communication between the agents leads to that much of the research in this area is focused on the standardization of Multi-Agent System architectures. The research on this area first started in the early 1970's when scientists from the Distributed Artificial Intelligence field formulated some of the basic theories.

Expanding manufacturing companies all over the world are trying to make their production more cost-efficient and more profitable. The problem that's arising today is that many control systems are centralized and therefore have difficulties in coping with high complexity and change. Agent-based technology provides a way to implement a desirable, robust and decentralized manufacturing environment. This open control environment provides new dimensions to decision making due to interaction not only between human and machine but also between machines and their control systems.



### 3.1.1 Background

Recently markets have changed from stock driven to customer demand driven. The innovation cycles are shorter, greater customized production specifications and shorter delivery times. Implementing and constructing a scheduling system to take care of this will call for high robustness to changing requirements of the manufacturing process.

Most of the control systems used by manufacturing companies today are centralized. In such a system, the controller is concentrated to one location. As companies strive to be more efficient and productive as well as tolerant to changes in production, a centralized approach becomes more and more inadequate. Of course there are areas where distributed solutions do not apply at all; where variety in production is low, disruptions are well known, or where distribution isn't clear and change is slow. These environments are unlikely to benefit from agent-based technology. Let's focus on areas with the right pre-requisites for distributed solutions.

Real-time production control is a good example of when a distributed solution could be beneficial [13]. Typical environments are assembly lines with a large number of machines, where tasks have to be assigned and then executed. When facing this kind of control problem you handle low-level task assignment and decision-making with critical time constraints. The complexity of the control problem is relatively low but the results will be satisfactory; an easier system to reconfigure, better management of changes of production demands resource failures and technology updates.

## 3.2 Agent architecture

It is important to have a well defined architecture of the agents when you are designing a Multi-Agent System. At the moment there are a number of different theories regarding the architecture of the agent.

We have found that the architecture of an agent consisting of four different elements is most suitable for our situation [1]. Study figure 3.1 to get an understanding of the elements building the agent. As shown in figure 3.1, the agents consist of four main elements, a Diagnostics Element, Planner Element, Flow Model and an Execution Control Element. A more thorough explanation of the elements will follow in chapter 5. It should also be said that the planner element is chosen to act according to the Case-Based reasoning paradigm, also described in chapter 5. It is important to have that in mind when studying the data table and the diagnostics element.

### 3.2.1 Data table

Depending on which agentification paradigm used when the agent system is developed the agent will be responsible for only one, or a number stand-alone equipment. It is important to accentuate that the agents are not a part of the actual machines, but should only be viewed as closely coupled to them. The machines themselves are operating according to some rules decided by the controller of that machine. The sensors and actuators associated with the actual machine are connected to the agent using a network link for input and output signals. The different state variables of the machine are then stored in the data

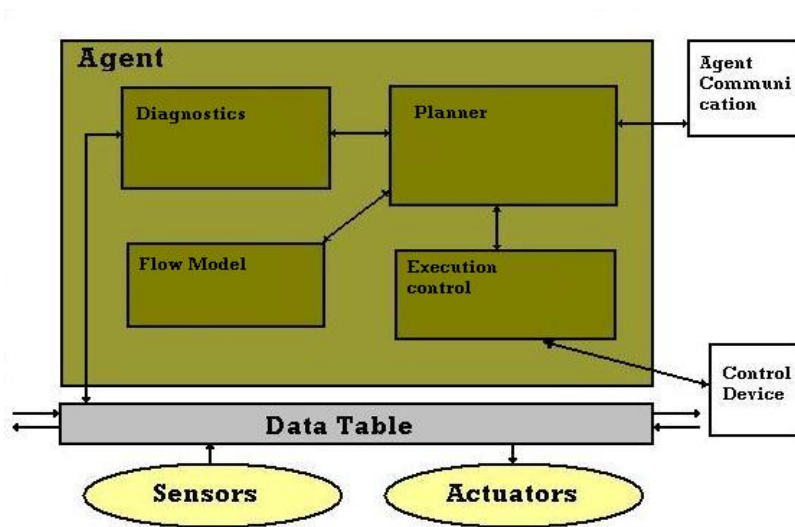


Figure 3.1: the architecture of the agent

table, which basically is a memory space for variables and attributes associated with the machine. In some cases variables of other machines are also stored in the Data Table. The data table's role is to provide a way for the agent to monitor the signal map of the machine, to help the agent in the reasoning about the process. The data table is, even though it's not an actual part of the agent, a crucial element for the agent, since the ability to monitor the equipment's health is all-important.

The actions and organization of the data table is taken care of by a low-level agent, which only duty is to make sure that the information stored in the data table is up-to-date. In some situations agents are depending on information from neighboring agents (in the start- and stop-policy for example), this information is stored in the data table by the neighboring agents. The Data Table is best described as an interface between the machine and the agent.

### 3.2.2 Diagnostics Element

The first step in the decision making process of the agent is when the Diagnostics Element diagnose the health of the machine under its scope.

First of all it is important to consider the entire picture when you want to understand the diagnostics element. As mentioned above the physical device is connected to the agent via the Data Table. This is where the pertinent information regarding the machine is stored. The information is stored in the form of variables and states. For example, the state of the machine and the accumulation level of the Helix are two types of information that typically can be deduced from the Data Table. Simply put, the role of the Diagnostics Element is to observe the information stored in the Data Table and when a change is due it must notify the Planner Element. It is then up to the Planner Element

to decide if action is needed. It is also up to the planner to decide what type of action it should be. The message that is sent from the Diagnostics Element to the Planner must include enough information to fully describe the new situation. The reason for this is to simplify the Planners work in correcting the potential error.

### 3.2.3 Planner Element

The Planner Element of the agent is best described as the reasoning engine of the agent. The Planner does work all by it self, in a vacuum, towards a solution to the problem defined by the diagnostics component. It will here consult a library of pre-specified plans and a problem solving mechanism. There are several problem-solving paradigms that may be used at this stage in the architecture. Among them are Case-Based Reasoning, Rule-Based Reasoning and Model-Based Reasoning just to name a few. We have chosen to use Case-Based Reasoning (CBR) since it holds a number of advantages over the other paradigms. CBR is, for example, able to propose solutions to problems that are not well defined without deriving these solutions from scratch, saving important time and thus being able to more easily meet the systems real-time constraints. A simple explanation of cased-based reasoning is as a process of solving new problems based on the solution of old problems. It is not only a powerful method for solving computer problems but also a very common behavior in everyday human problem solving. The scheme for CBR is showed in figure 5.4. The strength with CBR is that it involves elements from problem solving, understanding and learning and integrates it all. It is possible for CBR to solve new problems by adapting previously successful solutions to new problems but with similar characteristics. The steps of CBR are the following [10].

- **Indexing:** The purpose of putting an index of the incoming diagnose is to make it easy for the planner to compare diagnoses with eachother. The case indices are a very important feature for characterizing an event to get the possibility to retrieve it later. As in figure 3.1 the new situation is detected by the diagnostics element, which assigns the right index for the discovered problem, and then send this information to the planner. The indexing should work in the simple way that similar problems have similar indexes. The rules regulating this are up to the programmer and the implementation and they are described further in chapter 5 .
- **Retrieval:** The indexing of the problem received from the diagnostics element are used to investigate if the same problem have been solved earlier. If the same problem have been solved before all the planner have to do is to retrieve the solution to the problem from the case memory and send the solution to the execution-control element. This leads to a simpler and quicker solution to the problem than if the solution would have to be implemented from scratch. This will not always be the case though. The situation when there isn't a matching solution in the case memory will arise. What the planner does in this case is to retrieve the solution from the case memory that matches the new problem best. The next step will be to adapt the retrieved case according to some adaptation rules until we have a solution ready to solve the new problem. The new solution will then be tested on the equipment model, if the result from the testing

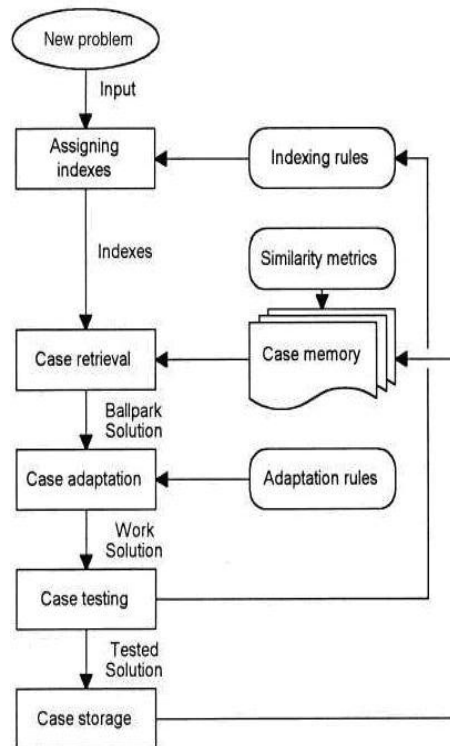


Figure 3.2: description of the Case-Based reasoning paradigm

is positive, the solution is implemented on the real process and a case is created and stored in the case memory.

- **Adaptation:** The adaptation step of the CBR paradigm only comes into play when the solution retrieved from the case memory isn't a perfect match for the new problem. In other words, a solution to the new problem doesn't exist in the case memory. The same goes for the situation when even a similar case isn't possible to retrieve. In these situations the retrieved case will have to be adapted to the new problem according to some adaptation rules implemented by the programmer. It is important that the adaptation rules are strong enough to derive a solution from scratch, The agent must work properly even if there are no cases stored in the case base.
- **Test:** In this step the proposed solution will be tested. That is, the solution developed by the adaptation step of the CBR. An important feature of this step is the equipment model the testing will be done on. Of course it is here imperative to have a model with characteristics likening the physical equipment. If the test result isn't satisfactory the problem solving process will start over again at the case retrieval step, with the new information that the last solution proposal was inadequate.
- **Storing:** If the above mentioned testing went well, and the results were

satisfactory the solution will be stored in the case memory. It will be stored under the index which was assigned to it in the initial step.

As we will describe in chapter 5, we have in our application of agent-based technology modified the Case-Based reasoning paradigm to suit the need of the packaging line better.

### 3.2.4 Flow Model

It is on this model the testing of the solution derived by the CBR-paradigm is done. It will be a model of only the physical machine in the agents responsibility and not the entire production line. This element is a decision making support system. Models of the equipment are put here, in order to help the Planner in its work.

### 3.2.5 Execution control

Once process of the Case-Based Reasoning is completed and the agent have come up with a solution. The next step will then be for the agent to send the right signals to the physical equipment in order to solve the problem. The execution control translates the committed plans into signals understandable for the machine controller. In other words, it works as an interface between the agent and the machine.

## 3.3 Conclusion

The problem of controlling the packaging line, involves low-level task assignment and execution constraints involving considerable time constraints. The range of decisions for the control system is generally narrow, the machine at hand only executes a few operations, and the complexity of the operations is low. On the other hand can a missed time constraint or an inappropriate outcome lead to lost production time or product waste. In this case, a good motivation for a distributed solution is to simplify and enhance the production environment's reconfigurability to better deal with changing product demand, resource failures and technology updates. In solutions for distributed real-time control, the software agents often correspond one-to-one to the machine resources in the real environment (i.e. one agent per machine). To generate the solution the different agents interact and act upon the available information to determine the best possible solution. Due to the time constraints of the solutions generated by the agents is more often based on reactive behavior than a deliberative behavior based on complex models and proactive strategies. Their duty is, with other words, to react on local changes in the production environment.

## Chapter 4

# Multi-Agent Systems

### 4.1 Main Purpose

A Multi-Agent System is a system consisting of more than one agent. The main characteristics of a Multi-Agent System is that all agents in the community should communicate asynchronously and apply deliberative proactive reasoning to choose the most beneficial outcome based on the agents apprehension of its environment and the other agents states. The communication is very important to agents in terms of improving their ability to understand the purposes, intentions, states and capabilities of other agents. The communication between agents is also very important in the Case-Based reasoning paradigms adaptation phase. In addition they should also understand and respect rules and constraints of the other entities.

### 4.2 Autonomy of Agents

An agent which is based solely on "built-in knowledge" would pay no attention to other units in the process and would act completely on its own. This kind of agent is not desirable since it's incapable of learning from its own mistakes. Think of an alarm clock for example. For example, if the manufacturer of the clock you just bought for some reason knew that you were going on a trip to China at some particular date, then a mechanism could be built in to adjust the time by six hours at just the right time. The behavior would be successful, but the intelligence would seem to belong to the clock manufacturer and not the clock.

Autonomy within agents is achieved through communication among agents and agent memory. This provides a behavior based on own experience, making it possible for agents to evaluate a certain situation and make decisions according to what's best for them and all other units in the process [11]. Of course it's crucial to provide agents with a little bit of built-in knowledge as well, so that they know how to act in the beginning of the process when they lack experience. Otherwise they would operate randomly, depending on the operator to interact and provide assistance. Look at it as a way to provide initial knowledge in order to keep the agents operating long enough to finally learn by them selves. An autonomous intelligent agent should be able to adapt to many different

environments, given sufficient time to adapt.

### 4.3 Holonic Manufacturing Systems

In the case where agents are used to improve real-time production control each agent is linked to a physical manufacturing unit e.g. a robot, conveyor or packaging machine. Such an entity is called a Holon, and is defined as a complete, autonomous, cooperative manufacturing unit. Holons have the ability not only to undertake tasks but also to accept, plan, control and schedule them [7]. Intelligent Scheduling is one of the main research areas connected to Holonic Manufacturing Systems. Trying to achieve hard decision robustness is essential to these systems. The idea of Intelligent Scheduling is to examine disruptions in the manufacturing system, both internal and external, as well as coping with the computational complexity of decisions. It is often essential to decrease the amount of machine breakdowns in a system. To do so, a sort of human-machine interface with the ability to adapt to different user profiles is needed. By storing trouble history (i.e. reasoning in case of error) in each agent, it might be possible to achieve optimal trouble-shooting.

### 4.4 Multi-Agent System for Real-Time Control

There is a quite narrow set of tasks that actually can improve from agent solutions when coming to real-time control systems. It's mostly about reconfiguration, updates and local changes "on the fly", that's important for the system to handle. Researchers have thus come up with a more general solution to the structure of holons. It consists of a low-level real-time-control subsystem combined with an advisory intelligent software agent [13]. The sub-system is implemented with function blocks and/or ladder logics. Communication takes place between sub-systems and agents (Intraholon Communication), among agents themselves (Interholon Communication) and between sub-systems belonging to different holons (Direct Communication). A description of the agent communication can be viewed in Figure 4.1.

### 4.5 Agent Communication

The main purpose of agents is often to help a machine communicate with the user and other agents, in order to finally reach the best over-all decision. A process module and a decision module are often already present in cases with controllable machines i.e. robot arms, packaging machines. By adding a third module, HMI (Human-Machine-Interface), it's possible for the user to interoperate with the machine in a smooth way. The HMI consists of a GUI (Graphical User Interface) and an agent. The GUI provides the platform in which interaction between user and machine take place. An agent handles the communication between user and machine and also communication within the structure. It's convenient to divide the communication into three main [2].

At first the user observes the actual process of the machine by looking at the current state of the process. The HMI provides all information concerning state to the user, with aid from the agent. It could be all kind of different relevant

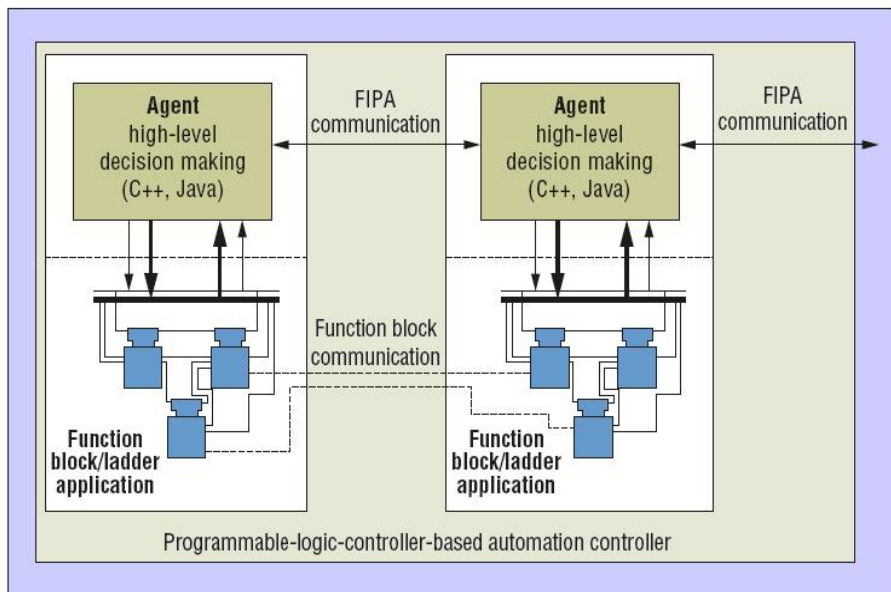


Figure 4.1: Description of intra-holon and inter-holon communication

parameters, values and updates. Secondly the user modifies the flow of the process in some way, for example by changing some parameter value. Therefore the HMI must provide an easy and straightforward process editor. Finally the HMI, actually the agent, has to make a model check of the user intention and send the result back. This is the case each time a user enters new data into the process. Thus, the agent should be able to handle different kinds of user profiles to increase usability for all kinds of users.

The problem of developing an agent system is not only concerned with the creation of the intelligent agent itself. It is of equal importance to create a good platform for the agents, which simplifies the management of, and the communication between the agents. One way to create a Multi-Agent System is to develop the system according to the specifications provided by the organization named FIPA [16]. The Foundation for Intelligent Physical Agents (FIPA) is a non-profit association registered in Geneva, Switzerland, founded in 1995. The main goal for FIPA is to maximize interoperability across agent-based applications, services and equipment. This is done through the FIPA specifications. They provide specifications of agent-based technology that can be integrated by agent-system developers, to construct complex systems with a high degree of interoperability. FIPA also specifies a set of interfaces which the agent uses for interaction with various components in the agents environment, for example humans, other agents or non-agent software. The specifications presented by FIPA is not so much concerned with the actual agents but with the infrastructure of the architecture surrounding the agents with focus on communication and organization. The abstract architecture presented by FIPA can not be directly implemented, it should be viewed as a basis or specification framework for the development of particular architectural specifications. The FIPA abstract architecture specifications cover three important areas, namely.



- agent communication
- agent management
- agent message transport

We will here at an abstract level describe the architecture of a Multi-agent System and how the communication between agents is managed. But in order to do this a set of architectural elements and their relationships must first be explained.

## 4.6 Architectural Overview

### 4.6.1 Agents and Services

An agent can in most cases be described as a computational process which implements the autonomous and communicating behavior of an application. In a concrete realization of a FIPA abstract architecture an agent may be implemented in a number of ways. It can for example be realized as a Java component, it may also execute as a native process on a physical computer under an operating system, or be supported by an interpreter like a Java Virtual Machine. Agents communicate with each other by sending messages which represents speech acts. The messages are encoded in an Agent Communication Language (ACL) [14], which best is described as any language in which communicative acts can be represented, and therefore also messages constructed. The complete specification of an ACL is provided by FIPA.

A service is a coherent set of mechanisms that support the operation of agents and other services. The main objective for all services is to provide support-services for the agents. The organization of the services depends on the specific implementation, though there are a few services that must be available for the Multi-Agent System to function. These services are the Agent-Directory-Service, the Service-Directory-Service and the Message-Transport-Service. The mandatory services will be described in more detail later. The way services are implemented is up to the programmer, they may very well be implemented as agents or as software accessed thru method invocation using some interface provided java or C++. It must be said that if a service is implemented as an agent this agent do not have the autonomy usually attributed to the agents. The reason is that is not desirable to have a service which arbitrarily can refuse to execute the service.

### 4.6.2 the Multi-Agent System platform

#### Agent Management System

The Agent Management System is a mandatory component of every concrete instantiation of the FIPA abstract architecture. The main purpose of the management system is to provide a platform which exerts the global control of the Multi-Agent System. There will only exist one agent management system per agent platform. The AMS is responsible for managing the operation of an agent platform. The agent management system is closely linked to the Agent-Service-Directory, or another way to put it is to say that the Agent-Service-Directory is a partial set of the Agent-Management System.

The AMS provides all operations necessary for complete control of the agents life cycle, which will be better understood if you consider figure 4.2. A good way

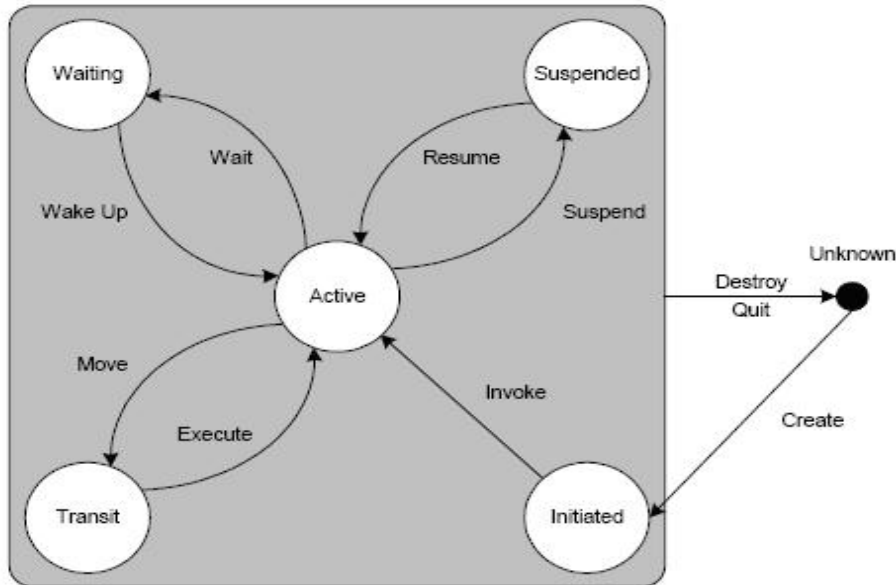


Figure 4.2: description of the agents life cycle

to describe the agent management system is to say it provides a white-pages service for the agents. This is so because the management system contains all the agent-names and their transport descriptions to every agent registered on an agent platform. A transport description is a description of how to transport a message to a certain agent.

### Directory Facilitator

The best way describe the directory facilitator is to say it provides a yellow pages service for the agents. This is where the agents go when they need help in finding a suitable service for a problem. This is a reification of the service-directory-service mentioned earlier. If a directory facilitator is implemented it is important that it is up to date, that it maintains an accurate, complete and timely list of the agents on the platform. There must at least be one directory facilitator present at each agent platform.

## 4.7 Areas of Application

*Real-time control:* As mentioned before the most common usage of agents is in real-time control where high variety and volume, in a discrete environment, describes the world in which an agent operates. Line production is a good example of such an area where the goal is to mass produce individually customized products. Negotiation between agents aren't that complex here as the actual scheduling and planning.

*Control of physically highly distributed systems:* Utilities as water and electricity need to be distributed in a city in an efficient way. In order to get a working system a lot of information must be shared within the system. Quick and local decisions are to be made continuously. Information sharing and joint decisions between autonomous agents are made when timing allows it.

*Transportation control and material-handling:* This area is often represented by conveyor lines in a production system or other transportation systems. Here the frequent need of configuration changes and disruption-handling is taken care of. To avoid congestion among the units in a production line it is important to the agent to have some predefined information as well as reasoning history. Which path is most cost efficient and what sensors and switches are connected to it are other important questions.

*Reintegration of equipment and frequent upgrading:* A system that constantly needs to be upgraded and integrated with different equipment has to have an efficient way of coping with management change. By structuring the software (or hardware) in an agent-like manner, we will support system integration in an effective and desirable way.

## 4.8 Benefits and Drawbacks

### 4.8.1 Advantages

Agent-based solutions to different manufacturing systems may result in a variety of benefits to the company.

*Practicability* If the only possible way of reaching an automated solution is by distributed decision making, agents is the perfect solution.

*Robustness and flexibility* This is often called the "key justification" for agent-based solutions because of the extensive profits it brings to the system performance. Robustness against breakdowns is generated because of the absence of central elements and centralized decision making. The ability to reorganize production on the fly without having to reprogram the whole software system is another. External disruptions such as scheduling and change in production are handled effectively without stopping the process. At the same time the system can perform tasks related to equipment failure and plan changing.

*Reconfigurability* The ability to change, add or remove both hardware and software modules on the fly. This clearly stimulates upgrading from old to new technology and it makes system maintenance considerable cheaper.

*Reorganizability* Applying the agent-based philosophy to more than one level e.g. hard real-time control, soft real-time control, decision making for control tasks and supply chain management, to name a few, will make the production more efficient. By using the same communication language on all different levels, automatic communication and negotiation can take place between units on different levels and subsystems.

### 4.8.2 Disadvantages

Actual industrial applications of agent-based solutions are today still very rare. Not only are they few, they are often not working in the way implied due to restrictions in functionality.

- *Cost* There are huge investments needed in implementing an agent-based solution if you compare with the more common centralized solution. If the process intended to gain from agent-based technology in fact is not in need of all the dimensions of flexibility that the solution provides, the cost will be unnecessary large.
- *Control Systems* A majority of the control systems used by companies today support centralized control but no distributed alternative. Agent-based solutions, including possibility to communicate asynchronously, must be offered by the vendors of the systems in order to make the vision of agent-based solutions possible for most companies.
- *Education* Most control- and system engineers have studied how to design centralized control systems, not so many know anything about distributed agent-based solutions. When the lack of knowledge is that widely spread it causes a big barriers. The companies will have to educate their employees and recruit educated people before they can be prepared to deal with the new technology.

## Chapter 5

# Application of Agent-Based Technology

When the decision has been taken, that agent based technology should be used to control the production line, then the first step in designing the architecture is to consider where in the system the agents will be needed, and how big their scope of responsibility should be. This is often referred to as the agentification process. Here you decide what the responsibility of each agent should be. There are a few different guidelines you can follow at this stage. We have here used one-to-one mapping, which as the name implies, means that each equipment component is the responsibility of one agent. The architecture of the Multi-Agent System will look like shown in figure 5.17.

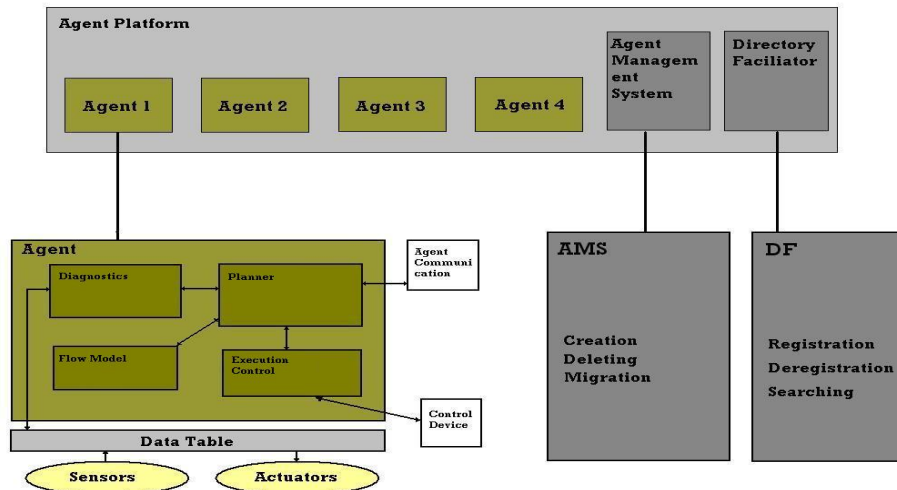
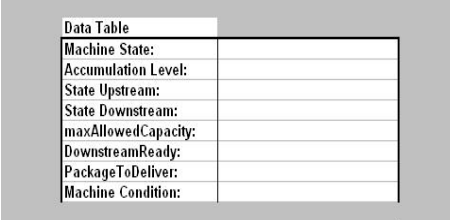


Figure 5.1: the architecture of the Multi-Agent System.

## 5.1 The Agent

### 5.1.1 Data table

Depending on which agentification paradigm used when the agent system is developed, the agent will be responsible for only one, or a number stand-alone equipments. It is important to accentuate that the agents are not a part of the actual machines, but should only be viewed as closely coupled to them. The machines movements are controlled according to control rules regulating the machines movements. The machines themselves are operating according to some rules decided by the controller of that machine. The sensors and actuators associated with the machine are connected to the agent using a network link for input- and output-signals. The different state variables of the machine are then stored in the data table, which basically is a memory space for the variables. The data table's role is to provide a way for the agent to monitor the signal map of the machine, in order to help the agent in the reasoning about the process. The data table is, even though it's not an actual part of the agent, a crucial element for the agent, since the ability to monitor the equipments health is all-important. In the Multi-Agent System designed for the Tetra Pak packaging line the data table will look like follows.



Data Table	
Machine State:	
Accumulation Level:	
State Upstream:	
State Downstream:	
maxAllowedCapacity:	
DownstreamReady:	
PackageToDeliver:	
Machine Condition:	

Figure 5.2: the information stored in the data table.

This means that there must be a connection between the data-tables of the neighboring agents. The reason is that there exist information of an agents neighbors in the data table.

### 5.1.2 Diagnostics Element

The first step in the decision making process of the agent is when the diagnostics element diagnose the health of the machine.

First of all it is important to consider the entire picture when studying the diagnostics element. As mentioned above the physical machine is connected to the agent via the data table. This is where the pertinent information regarding the machine is stored. The information is stored in the form of variables and states. For example, the state of the machine and the accumulation level of the helix are two types of information that typically can be deduced from the data table. Simply put, the role of the diagnostics element is to observe the information stored in the data table (i.e. the condition of the machine) and notify the planner when a change is due. The message that will be sent from the diagnostics element to the planer must include information regarding the type of change that have taken place. The reason for this is to simplify the

planner's work in correcting the error. The diagnostics element will provide the information shown in figure 5.10.

Diagnose	
<b>Machine State:</b>	<i>recieve and deliver</i>
<b>Machine Condition:</b>	<i>overflow</i>
<b>Production Level:</b>	<i>normal</i>
<b>State Upstream</b>	<i>recieve and deliver</i>
<b>State Downstream</b>	<i>receive and deliver</i>
<b>Packages to deliver</b>	<i>yes</i>
<b>Downstream Ready</b>	<i>yes</i>
<b>Accumulation Level:</b>	<i>NA</i>
<b>productionFlowWeight:</b>	<i>maxAllowedCapacity</i>

Figure 5.3: the information stored in the diagnose.

This will help the planner to identify the problem, even if the planner hasn't come upon it before. The diagnostics element will continuously monitor the health of the machine with the help of the information stored in the data table. It is entirely up to the diagnostics element to recognize the situations which need to be attended. The different attributes which will be monitored are the following.

The diagnose element should make a diagnose and send a diagnose message to the planner whenever it notices a change in the data table. It will then use the information stored in the data table and add the information necessary a message according to the above. In other words, a diagnose should be set whenever an attribute of the data table changes, then it is up to the planner to analyze the situation in order to find out if the situation need action or not. The way this will be done is by simply sample the data table continuously and always storing the last sampled version of the data table. Whenever the diagnostics element notices a difference between the stored version of the data table and the current version it will set a diagnose and send it to the planner element. When setting the diagnose the element must of course include the newest value of the parameters that should be included in the message. The parameters which can be deduced from the data table is simply copied from the data table to the diagnose message.

### 5.1.3 Planner Element

The planner is the reasoning enginge of the agent. This is where the decision making takes place. First the paradigm which is used when deriving the solutions will be described.

### Case-Based Reasoning

The scheme for the case-based reasoning paradigm is shown in figure 5.4 [10]. The first event will be to receive a diagnose message from the diagnostics element. The message will be of the type shown in figure 5.10. The first action in the paradigm is to put an index on the incoming diagnose messages. This is done according to a set of indexing rules.

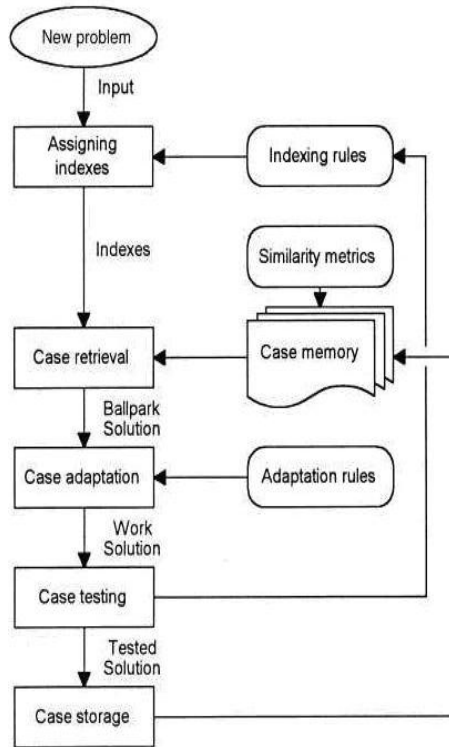


Figure 5.4: description of the Case-Based Reasoning paradigm

**Indexing Rules** The indexing rules must be valid both for new problems which needs to be solved and to the cases stored in the case memory. This is due to the fact that the case-retrieval process must be able to match the new problem with the old cases stored in the memory.

Case indexing involves assigning indices to facilitate their retrieval. The indexes should:

- Be predictive.
- Address the purpose the case will be used for.
- Be abstract enough to allow for widening the future use of the case base.
- Be concrete enough to be recognized in the future.

In the history of Case-Based reasoning both manual and automated methods have been used to select indexes. Naturally, in the current application, it is not



an option to assign the indexes manually. Among the most commonly used automated indexing methods are:

- *Indexing cases by features and dimensions*, that tend to be predictive across the entire domain i.e. descriptors of the case which are responsible for solving it or which influence its outcome. In this method the domain is analysed and the dimensions that tend to be important are computed. These are put in a checklist and all cases are indexed by their values along these dimensions.
- *Difference based indexing*, selects indices that differentiate a case from other cases. During this process the system discovers which features of a case differentiate it from other similar cases, choosing as indices those features that differentiate cases best.
- *Similarity and explanation-based generalization methods*, which produce an appropriate set of indices for abstract cases created from cases that share some common set of features, whilst the unshared features are used as indices to the original cases.
- *Inductive learning methods*, which identify predictive features that are then used as indices.

We have used a variation of difference based indexing. We have focused on how to best differentiate the cases from each other. The best way to do this is to define the cases by the problem situation, which is specified in the diagnose sent from the Diagnostics element. The different kinds of situations that can occur in the production line is fully covered by the diagnose message and its different possible parameter values. So we have used the parameters of the diagnose as indexes. This way the indexes cover every possible situation in the packaging line.

**Case Memory Implementation** The case-solutions will be stored in the case memory as key-value pairs according to figure 5.11. Case storage is an important aspect in designing efficient CBR-systems in that it should reflect the conceptual view of what is represented in the case and take into account the indices that characterize the case. The case-base should be organized into a manageable structure that supports efficient search and retrieval methods. A balance must be found between storing methods that preserve the richness of information of the cases and their indices and methods that simplify the retrieval of relevant cases without compromising the amount of information stored in the solutions.

The case memory will in this implementation be represented by a key-value tuple, as in figure 5.7. The pairs in the tuple will consist of the key, i.e. the case-index, and a value which will be represented by the case solution. An example of a case solution is shown in figure 5.5. This organization makes it easy to search the directory for cases.

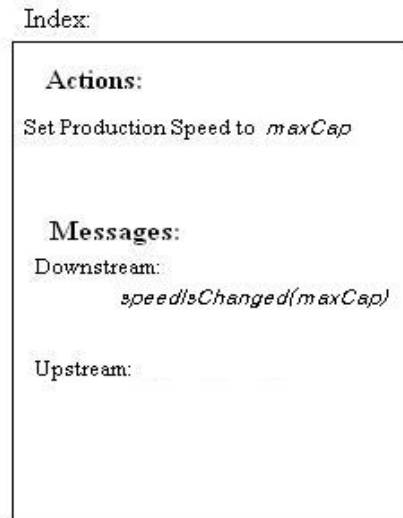


Figure 5.5: example of a solution stored in the Case-Memory

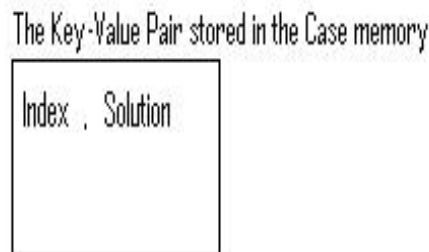


Figure 5.6: example of a case-memory entry

**Case Retrieval** Given a description of a problem, a retrieval algorithm, using the indices, should retrieve the most similar case to the current problem or situation. The retrieval algorithm relies on the indices and the organization of the memory to direct the search to potentially useful cases. Case-based reasoning will be ready for large scale problems only when retrieval algorithms are efficient enough at handling thousands of cases. The issue of choosing the best matching case have been addressed by many researchers. Among the well-known methods for case-retrieval are the nearest-neighbor method.

In our Multi-Agent design we have not used a particularly complex retrieval method. We simply search the case memory in order to investigate if a solution with a matching index is available. If there is a matching index in the memory the solution will directly be sent to the execution control element. If there is no matching case in the memory, then the Case-Based Reasoning process will continue with the next step, which is the adaptation rules.

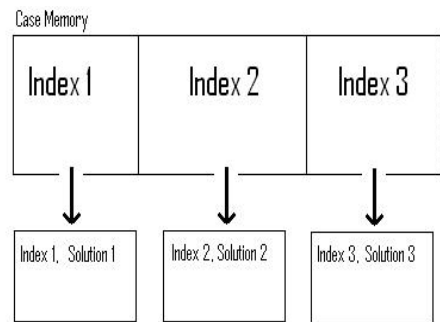


Figure 5.7: description of the case-memory

**Adaptation Rules** Once it is known that a matching case doesn't exist in the case base, the CBR system must construct a new case in order to solve the problem situation. An ideal set of adaptation rules must be strong enough to be able to generate complete solution from scratch. Because of the simple nature of our solutions, we have chosen to use the method of weighted votings.

The first thing that happens in the situation where no previous solution is available is that the planner sends a message to the directory facilitator with the information shown in figure 5.13. This message is called a vote request. The directory facilitator will then stand as the organizer of the vote. The main idea with the whole voting procedure is that every agent in the packaging line will cast a vote based on what action they want the agent with the problem to take. The agent with the problem will not cast a vote itself. This way the result of the vote will be the best possible action for the whole of the line. An additional feature in the voting procedure is that the votes of the agents have different weights depending on where they are located relatively the agent with the problem. The voting rules of the different agents are designed according to which decision by the problem-agent that would be best for them.

After receiving the vote request, the directory facilitator will send out the vote message with the information shown in figure 5.14. When this message is received by the other agents in the packaging line. The information is processed by the individual agents voting rules and a vote is sent to the directory facilitator with the information shown in figure figure 5.9. The distance is an integer telling how many machines away the machine, the vote is about, is located. The utility is the weight associated with the vote. The directory facilitator register all votes, makes sure that all agents (except the problem agent) have voted, then returns the result of the vote to the agent with the problem, who applies the solution according to the execution control element.

**Case Testing** The main objective with the case testing procedure in the case-based reasoning paradigm is to verify that it is safe to apply the proposed solution. In our Multi-Agent System the testing will be done against the flow model element.

**Case Storing** The most important thing to remember in this part of the case-based reasoning process is to remember to not store solutions unless they

Vote Message from DF	
<b>Condition:</b>	<i>overflow</i>
<b>State:</b>	<i>recieve&amp;deliver</i>
<b>DownstreamReady:</b>	<i>yes</i>
<b>Packages to Deliver:</b>	<i>true</i>
<b>Orientation:</b>	<i>Downstream</i>
<b>Distance:</b>	<i>n</i>
<b>Machine Type:</b>	<i>Distribution Equipment</i>

Figure 5.8: the information in the message sent by the Directory Facilitator

Vote Message from DF		Own condition:	Normal
<b>Condition:</b>	<i>lowAcc</i>		
<b>State:</b>	<i>recieve&amp;deliver</i>		
<b>DownstreamReady:</b>	<i>yes</i>		
<b>Packages to Deliver:</b>	<i>true</i>		
<b>Orientation:</b>	<i>Downstream</i>		
<b>Distance:</b>	<i>n</i>		
<b>Machine Type:</b>	<i>Helix</i>		
		Vote	
		<b>Preferred production speed:</b>	<i>MaxCap</i>
		<b>Machine Command</b>	<i>StartProduction</i>
		<b>Utility:</b>	<i>1</i>

Figure 5.9: Description of how Filler machines will vote in a given situation.

are proven to be successful. In order to achive this the cases aren't stored until the problem situation of the agent is resolved. If it isn't resolved then a new diagnose will be made by the diagnostics element and a new voting process. Another situation that could cause problems is that the problem is never resolved, but it isn't getting any worse. Then, of course, new action needs to be taken in order to resolve the problem. This problem is solved by simply using a timer, which will cause a new voting when the data table is unchanged after a specified time period (now without the previous solution as an alternative), and a new solution proposal. An additional feature is to define which situations in the data table that are undesirable and which situations that are desirable. Whenever you are in an undesirable state new votes should be conducted continually.

### 5.1.4 work flow

There are only two ways to trigger the planner element to take action. The first way is a message from the diagnostics element, telling the planner that a change in the data table is due, and also information about what has changed. The second way is a message from another agent. It could for example be a notification message saying that the machine upstream has raised it's speed or a message starting a voting process. In order to get a good understanding for

the behavior of the planner the different possible scenarios will be explained.

**Diagnose message** In the situation with a diagnose message from the diagnose element, the first event that occurs in the planner is that a diagnose message is sent from the diagnostics element to the planner element. This message will be sent whenever there is a change in the parameters of the data table, the planner must then decide whether the new situation is desirable or not and if action is needed. Always when this situation arises, the planner will use the theory of case-based reasoning when solving the problem. The message from the Diagnostics element will be of the type shown in figure 5.10

Diagnose	
<b>Machine State:</b>	<i>recieve and deliver</i>
<b>Machine Condition:</b>	<i>overflow</i>
<b>Production Level:</b>	<i>normal</i>
<b>State Upstream</b>	<i>recieve and deliver</i>
<b>State Downstream</b>	<i>receive and deliver</i>
<b>Packages to deliver</b>	<i>yes</i>
<b>Downstream Ready</b>	<i>yes</i>
<b>Accumulation Level:</b>	<i>NA</i>
<b>productionFlowWeight:</b>	<i>maxAllowedCapacity</i>

Figure 5.10: the type of message sent by the Diagnostics element.

First a description of the way a message from the diagnostics is handled will be provided. The planner will receive a message from the diagnostics, the message will be of the type presented in figure 5.10. When the planner first gets a message with a diagnose from the diagnostics element, two scenarios are possible. In the first case the situation has occurred before and there is already a solution to the problem stored in the case base. In the second situation there is no stored solution to the occurred situation/problem, a new solution must be derived. We will start by considering the first case. We have a diagnose of the type shown in figure 5.10.

In this situation the planner will use the diagnose in order to create a matching index for the current problem situation. This is a relatively simple process. We have used a type of difference based indexing. In the case indexing we have focused on how to best differentiate the cases from each other. We found that the best way was to define the cases by the problem situation. The indexing rules of the Multi-Agent System will simply be based on the parameters of the diagnose. The indexes will in this way cover all different problem situations. The step following the indexing will be to search the Case Base. This is where the solutions to the previous problems are stored in the form of a key-value tuple according to figure figure 5.11. The index is the key and the solution is the value. In the case where there is a case stored in the case-base with a matching index, then the solution belonging to that entry is retrieved. Where the solu-

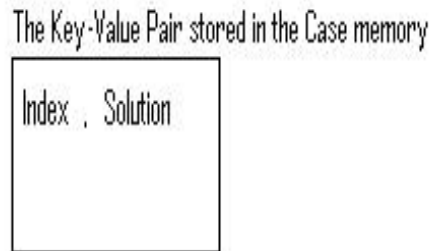


Figure 5.11: the Case-Memory entry.

tion will include the successful values of the parameters needed to be changed in order to solve the problem situation. The solution would for example look something like shown in figure 5.12, for a common distribution equipment in a speed-situation. In this case the problem is overflow in the situation where the

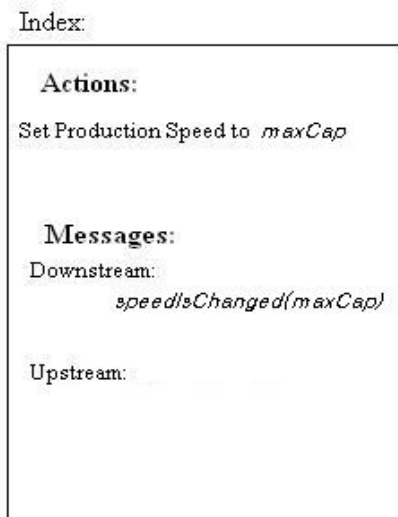


Figure 5.12: example of a solution stored in the Case-Memory

machine where the overflow has occurred runs at nominal capacity, and therefore has the ability to raise its capacity. The solution to this problem is to raise the capacity to `maxAllowedCapacity` and send messages to the machine downstream and inform it that a production raise is due, and also send a message to the (nearest) accumulator upstream and request a slowdown of the package flow. The retrieved case will then be sent to the execution control element which will transform the information in the case to the language understandable for the machine interface.

The most critical situation arises when the planner doesn't find a stored case matching the case diagnose from the diagnostics element. In this situation the agent will take help from the other agents in the process of deciding the best

solution to the problem. This will done with a weighted voting protocol. The main advantage with this protocol is that the agents will have a way to tell if they are in urgent need for a certain solution. If that is the case then the agents vote will be weighted heavier than the other agents votes. A thorough explanation is given in 5.2.2.

The first step the planner will take when it gets the negative answer from the case base search is to send a message to the Director Facillitator agent notifying it that a vote is necessary. This message will include the information shown in figure 5.13. Every agent in the production line will receive a message of the type shown in figure 5.14. As the figure indicates it is important for every agent to know whether the decision to be taken is for a machine upstream or downstream. Also it is important to know what type of machine it is and what type of emergency situation it is. The distance is an integer indicating how many machines away the situation is. It is important to have this information in order to calculate the weights associated with each vote. The voting rules for all the agents are defined in the *voting rules* map.

Vote Request	
Condition:	overflow
State:	recieve and Deliver
packagesToDeliver:	yes
DownstreamReady:	true
Machine Type:	Distribution Equipment

Figure 5.13: the information stored in a voting request.

Vote Message from DF	
Condition:	overflow
State:	recieve&deliver
DownstreamReady:	yes
Packages to Deliver:	true
Orientation:	Downstream
Distance:	n
Machine Type:	Distribution Equipment

Figure 5.14: the information stored in the vote message from the Director Facillitator.

When the other agents receive the vote message from the DF they decide which action they want the agent to take, and then send this information in a message to the directory facilitator, the information in the message is shown in figure 5.15.

**The voting process** The most important thing in the voting process is the rules controlling the way agents vote in a given situation. The agent are supposed to have certain rules deciding how the agent should vote in a given

Vote	
Preferred production speed:	MacCap
Machine Command:	StartProduction
Utility:	(robCMachines-n)/2

Figure 5.15: Example of an agents vote.

situation. This is where the real intelligence of the agent is situated. It is the voting process which make the agents adaptable to changing environments.

First and foremost it is important that the agent will function even if there are no cases stored in the case-base. This means that there must be a new voting-process each new diagnose. The goal with the agents voting rules is that the result of the planners work in this case is the same, though probably a bit slower. Naturally it is imperative that the voting rules for the agents is implemented with care. The voting rules for the agents is shown in the voting rules files.

**Message from another agent** This is the only other way to make the planner take action. In this case the planner receives a message from another agent directly. The information in this message will control the behavior of the planner. The different messages possible for the planner to receive is shown in the agent messages file.

The other reason for starting the decision making process in the planner element is by a message from another agent. The message can typically be of the type shown in figure 5.16. The different message types can be deduced from the message protocol excel file.

Production Speed Notification	
Distance:	1
Orientation:	upstream
Outfeed Speed:	

Figure 5.16: Example of a notification message.

When a speed notification message is recieved from another agent, the production speed is automatically modified so that machine *A* has a production speed 2% than machine *B* if *A* is directly downstream from *B*. The production speed is with other words raised with 2% per machine in the downstream direction.



### 5.1.5 Flow Model

Always when a solution has been derived with the help of case-based reasoning, the new solution will be tested on the Flow Model in order to verify that the solution is safe to apply on the real machine. Or more important, that it will give the desired result in terms of the infeed- and outfeed-flow of the machine. The testing rule of the Flow Model will be to simply check if the machine next down stream is able to handle the package flow delivered by the machine. The information about how much packages the machine downstream can handle is stored in the Data Table. The Flow Model of the Divider agent will be different from the other agents. The objective of the Divider agent is to always divide the flow from one infeed conveyor to two outfeed conveyors in a desirable way. The way this is done is decided in the Flow Model. The Case-Based Reasoner of the Divider agent will take the decisions regarding the total infeed and total outfeed flow. The decision about how much of the outfeed flow that should go to each agent is taken in the Flow Model.

## 5.2 Design of the Multi-Agent System

### 5.2.1 Multi-Agent System Platform

If we start to consider the agent platform it should be said that the platform consists of everything regarding the Multi-agent system including the physical machines and the physical communication network. A schematic picture of the Multi-Agent System is shown in figure 5.17. The architecture and design of the agents are thoroughly discussed in section 3.2, here we will discuss the remaining parts of the Multi-Agent System such as the directory facilitator, the voting- and the intraction-rules. We also describe the policies behind the voting rules and the interaction rules of the Multi-Agent System.

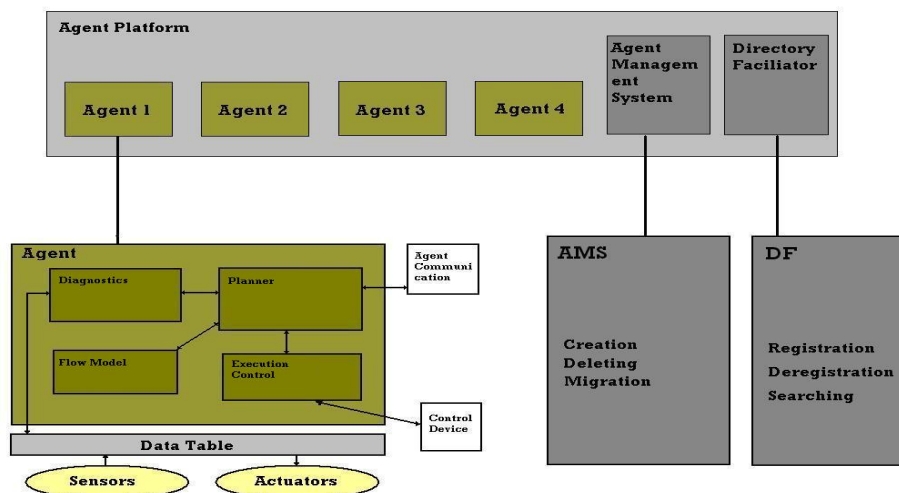


Figure 5.17: the multi-agent system architecture

### 5.2.2 Directory Facilitator

The director facilitator is a very important element in the MAS. It can be viewed as an organizational part of the Multi-Agent System [16]. Several important services are provided by the Director Facilitator agent, among them are, accumulation service, Voting Organizing Service, here is also a map of the packaging line, where the states of the machines is stored. This makes it possible for the other machines to easily get valuable information about the other machines in the packaging line. This operation provides the opportunity for the agents to make decisions based on more information.

**Accumulation Request Service** The Accumulation Request Service is activated thru a message from an agent with an overflow situation (for example). The directory facilitator receives a message of the type shown in figure 5.18. After this message is received the DF will check on the map of the packaging line which accumulator is closest upstream from the agent who sent the message. It will then redirect the message received from the agent to the closest accumulator upstream who most probably will start to accumulate packages.

Overflow Request	
Machine Type:	
DownstreamReady:	
Packages to Deliver:	

Figure 5.18: The information in the overflow request from an agent.

**Voting Service** The Voting Service is a service located in the Director Facilitator and it is used by agents who wants to conduct votings. The procedure and mechanism of the votings is further discussed in section 5.1.4

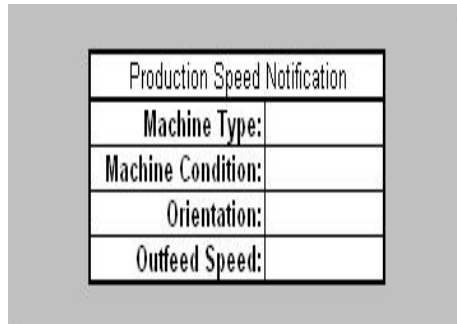


Figure 5.19: the Tetra Pak A3/flex, often called the Filler machine

### 5.2.3 Interaction Rules

There are two situations where predefined messages will be sent by the agents. The situations are when an agents changes its outfeed speed or when an agent has the overflow condition.

**Speed changes** The information these type of messages will be as shown in figure 5.20. The idea with there messages is that a machine downstream from a production change always should get the information about production speed changes upstream, so they can take these changes into consideration when deciding its own production speed. This message should also be sent when a machine lowers its outfeed speed. The rule is to always when the packaging speed when the outfeed speed is changed.



Production Speed Notification	
Machine Type:	
Machine Condition:	
Orientation:	
Outfeed Speed:	

Figure 5.20: The information in the speed notification messages.

**Overflow Messages** The other type of predefined message is sent when a machine is in an overflow situation. The message contains the information shown in figure 5.18. The agent with the overflow situation wants the accumulator closest upstream to start accumulating packages. The message is sent to the directory facilitator, which redirects the message to the accumulator closest upstream.

### 5.2.4 Voting Rules

The voting rules are what essentially decide will take in different situations, this is where the intelligence is located. A example of some voting rules are shown in figure 5.9.

# Chapter 6

## Production Case Studies

### 6.1 Production Line Cases

The different cases will be defined on one of the two line configurations shown in figure 6.1.

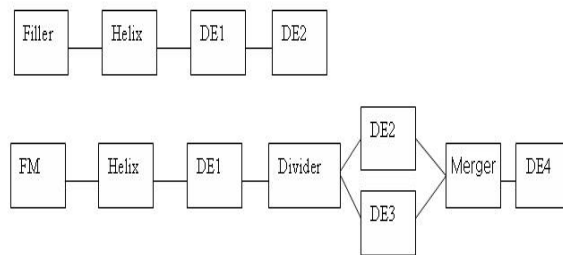


Figure 6.1: The different packaging line configurations

#### 6.1.1 Case 1

A simple Start- & Stop policy for the first line, (should also be verified for the second line configuration) in order to verify that the machines start and stop policies work satisfactory. This is done by introducing a blocked condition for the last DE, this should start a chain-reaction making every machine upstream stop, including the Helix (after full accumulation level).

#### 6.1.2 Case 2

Similar to the first case. The only difference is that when the Helix have a 50% accumulation level the blocked DE will be ready for production and the package flow will start again. This case will verify that the package line is able to recover from a blocked situation and also that the line is able to handle a situation where the Helix must unload a large number of packages.

### 6.1.3 Case 3

In the second line. One of the branched DE's becomes blocked. The Divider must now direct the flow to the other DE in order to maximize throughput. This case will show that the Divider is able to direct the flow, in order to maximize throughput, when one of the branched DE is blocked.

### 6.1.4 Case 4

The second line. There is an increase of package flow, due to the Helix unloading or a production increase from the Filler. This will verify that the production line is able to handle an increase of package flow.

### 6.1.5 Case 5

The second line. In this case there should be a redistribution of the flow thru the branch, due to a capacity change in one of the branched DE's. This case will show that the Divider is able to dynamically distribute the flow thru the branching, and also that it is able to react to unforeseen events.

### 6.1.6 Case 6

There is a situation in the first line where one DE becomes blocked which causes the Helix to accumulate. The next event is a blocked situation in the Filler machine. The next event in the case is that the DE resolves the blocked situation but the filler remains in blocked, the Helix will now have packages accumulated. This will make the Helix unload its packages and then make a normal stop. The normal stop will propagate downstream until all machines have made a normal stop.

### 6.1.7 Case 7

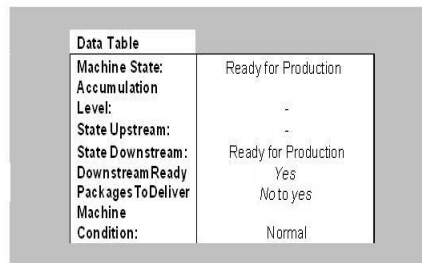
The second line. The first thing that happens is that a stop immediate makes the last machine in the line stop. This will make the stop propagate upstream. Then the DE before the divider will be in a situation that makes it do a stop immediate. After that the last DE will resolve its stop situation and be able to produce again. This will open the possibility for the machines in between to empty their packages and make a normal stop.

## 6.2 Case Verification

### 6.2.1 Case 1

After the startProduction message is sent to the packaging line. The PackagesToDeliver parameter in the DF is turned on (to yes). This means that the Filler will start to produce packages as soon as the downstreamReady parameter is true, the parameter is stored in the data table element of every agent. It is also stored in the DF, in case an agent wants to know the status of this parameter for another agent.

All the machines in the production line will start in preparation. When the machines preparation procedure is done they will automatically enter the Ready for Production state. After this the DE2 will by definition have downstreamReady true since it is the last machine in the line, this is true provided the end of the line is ready to handle the outgoing packages. This will change the downstreamReady parameter in the next machine upstream, this behavior will continue until all machines in the line have downstreamReady true. It is every data tables responsibility to make sure that the next machine upstream has the right value on the downstreamReady parameter. The Data Table of DE2 will have to make sure that the data table of DE1 has the right value on the downstreamReady parameter. Practically this is done telling the DE1 that downstream is ready only when DE2 is in ready for production and its endline is ready, in all other situations the value should be false. The Diagnostics element in the FM will then notice a change in its Data Table according to figure 6.2. As seen in the figure there is a change from no to yes on the downstreamReady



Data Table	
Machine State:	Ready for Production
Accumulation Level:	-
State Upstream:	-
State Downstream:	Ready for Production
DownstreamReady	Yes
PackagesToDeliver	No to yes
Machine Condition:	Normal

Figure 6.2: the information stored in the data table

parameter. This change has actually taken place in every agent along the line. The result has been that every agent, when this change has happened, has started a vote, asking the other agents in the production line what to do in this new situation. First it should be noted that every time a change is due in the Data Table of an agent, the diagnostics element of that agent will notice this and set a diagnose of the new state, in order to investigate if action is needed for this new situation. The voting rules of the different machines are presented in the voting rules documents. Due to the voting rules the result of the votings leading up to the current situation, have all reached the solution that nothing should be changed in the machines state. In the current voting, where the Filler has downstreamReady and is Read for Production with packages-to-deliver the other machines will vote for the Filler to start producing.

Every time a voting process is completed and a solution is reached, the solution may very well be to do nothing, the solution is stored in the Case-Base. More precisely, the case will be stored after it is noted that the solution solved the problem i.e. when the undesirable situation in the data table is gone. This is done according to the indexing rules of the Case-Based reasoner paradigm. The point with this is to make the problem solving process more effective. Instead of having a vote when this problem is encountered next time, the solution can be found in the Case-Base directly.

The next event that will happen in the production line is that the Helix will

change state from Ready for Production to Receive and eventually to receive and deliver. These changes will be noted by the neighboring machines who will, the first time this change is due, engage the other agents in a vote deciding which action the agent should take. In this situation the solution will be to not change its current state, i.e. the Filler should keep producing packages and the distribution equipment should remain in Ready for Production. The same thing will happen for every machine down the line, all the machines downstream will then be in Receive and Deliver.

The next event in the packaging line is that the last machine becomes blocked, this is a situation which can only be resolved by external actions i.e. an operator fixing the problem. Once DE2 becomes blocked two things will happen. The Data Table of both DE1 and DE2 will change its state due to the blocked situation. Both machines will start a voting process in order to reach a decision about which action to take. The voting process will then decide that DE2 should send the message startProduction to its machine, it should start producing as soon as the blocked situation is resolved. The vote result for DE1 will be to stopImmediate in order to avoid a large package queue at DE2. This decision will soon cause DE1 to change state from receive and deliver to receive, and when full change to ready-for-production. DownstreamReady gets false when the machine is full, and this will propagate upstream in the same way until eventually all machines are full including the Filler.

### 6.2.2 Case 2

This case is closely based on the previous case. The analysis will therefore start where the last case ended.

The situation is the following. The DE2 will be blocked. Due to this, the machines upstream have all made stopImmediate, including the Helix which have started to accumulate packages. According to the case description the blocked situation by DE2 will be resolved when the accumulation level is 50%. Before this though, a number of votes have taken place. Once the Helix recognizes that it has packages accumulated it will start a vote in order to find out what to do. Since a machine is blocked downstream the other machines will vote for the Helix to keep accumulating. This situation will change when the blocked situation is resolved, then downstreamReady will be true for the Helix and when a new vote takes place the other machines will vote for the Helix to not only startProduction but also to unload its accumulated packages. In reality this means that the Helix will unload packages at maximum capacity, it will of course also notify the machines downstream that its outfeed speed is raised so the machines can act accordingly. This is done by simply sending a message to the machine downstream in order to notify them that the production speed is raised. How the recipient treats this information is up to that agent, though it will in most cases also raise its production speed. Naturally the speed changes due to the raised output from the Helix will not be a problem. The Helix will then keep unloading packages until its condition changes from packages accumulated to normal, i.e. no more packages are accumulated. Then the Helix will call the DF for a vote. The result of the vote will be the five votes suggesting that the Helix should be in production at nominal capacity. With no other emergency situations the package flow will continue until packagesToDeliver is false for the Filler, a vote will then take place. The result of the vote will be six votes sug-

gesting the Filler should make a normal stop. When the Filler after the stop is in Ready for production, both the Filler and the Helix will notice this change and ask for votings. The result for the Filler will be the same as before the state change, this goes for the Helix as well. The Helix will after a while have no packagesToDeliver, naturally a vote will then take place. The result of the vote will be the same as for the Filler, a normal stop. The normal stops will continue downstream all machines are stopped and there are no more packages in the line.

### 6.2.3 Case 3

At the start of this case the Filler is producing at nominal capacity and there are not any speed or overflow situations in the line. The first event is that DE2 becomes blocked. This will cause a vote on what action DE2 should take. The result will be a suggestion for the DE2 to be in startProduction at nominal capacity, though this cannot be done until the blocked situation is resolved. There are two more votes that will take place, the Divider and the Merger will notice a change in the state of DE2. The result of their votes will be to keep producing at nominal capacity. Due to the Flow Model of the Divider agent it will send the entire flow to the DE3, this means that DE3 will get a flow larger than its maximum allowed capacity. This fact is detected by the divider which automatically sends a message to the DF with a question for the nearest (upstream) accumulator to start accumulating. This is done in order to prevent a queue at DE3. When this happens the Helix will start to accumulate in order to lower the package flow at the divider. After a while the Helix will start a vote asking what it should do with its accumulated packages. Every machine except the Divider will tell the Helix to unload, but since there is an emergency situation at the branching the utility for the dividers vote will be higher than usual, high enough for its proposition to win. The result is that the Helix will not unload until the blocked situation at DE2 is resolved. When DE2 is in production again, a new vote will take place, this time the result will be to still be in production with nominal capacity. The Helix will in a while request a new vote, this time will the Dividers vote be normal, which means that the Helix will start to unload. And the following machines will automatically adjust their speed to the preceding machine. This will continue until the packagesToDeliver is negative and the state of the machines subsequently becomes ready for production.

### 6.2.4 Case 4

In this case the dynamics of a flow increase due to a production increase from the filler or unloading from the helix will be showed. It doesn't matter which of the two situations it is that occurs, the impact on the rest of the line will be the same. Therefore it is assumed that the flow increase is due to the Helix unloading, since this is a more complex and more often occurring situation.

The first event in this scenario will be the Helix noticing that it has packages accumulated. A voting will then take place, in order to investigate what action the Helix should take. In this situation the result of the vote will be as shown in figure 6.3;

Every machine votes for the helix to unload, since there are no other emergency situations. The result of the unloading decision will be that the Helix



Voting Result for: Helix									
Agent	Filler	Helix	DE1	DE2	DE3	DE4	Divider	Merger	Result:
StartProduction (Nominal)									
StartProduction (MaxCapacity)									
StartProduction (underNominal)									
stopImmediate	1	-	7	5	5	3	6	4	31
normalStop									

Figure 6.3: the voting result

outfeed speed is increased. By definition, when a machines raises its outfeed speed it sends a message to the machine following downstream notifying it that more packages will arrive. When a message of this type is received by a machine, it is basically up to that machine to do what it wants with the information. In almost every case the machine will increase its own speed. If this is not the case, there will most definitely be a speed or overflow situation which will be taken care of by a new voting considering that situation. In this situation there are no other emergency situations though, so whenever a machine receives a message with information saying that the machine upstream has raised its capacity it will also raise its capacity. This is essentially the way any type of production flow will be handled.

### 6.2.5 Case 5

At the start of this case the machines are in Receive and Deliver and the flow is divided evenly thru the branching. Then an event occurs which means that the flow must be divided differently thru the branch. This can happen due to external events such as an operator sending this information thru the DF or an internal event like a failure in the machine causing a processing slowdown, but not a blocked situation. The information regarding what package flow the DE can accept is stored in the machine and can be updated by the DF or the DE itself. It is up to the two Distribution Equipments following the Divider to update their production flow weights in the data table of the divider. When no external or internal events have occurred causing a change in the capacity of the DE, the production flow weights will be the same as their maximum allowed capacity. The Divider will divide the flow according to the formula shown in figure 6.4. Where  $C$  is the flow coming into the divider,  $N$  is the production flow weight for the right machine and  $M$  is the production flow weight for the left machine. As long as the production flow weights are updated continually in the divider, the flow will always be divided in the right way.

### 6.2.6 Case 6

The first event in this case is that DE1 becomes blocked. A voting will then take place for the Helix, DE1 and DE2. The vote result for DE1 will be to keep producing, since that is the best scenario for all the machines. The vote result is shown in figure 6.1figure 6.5. The result for the Helix and DE2 is shown in figures 6.6 and 6.7;

The vote result for the helix is to stopImmediate since downstreamReady is

$$flow\_right = C * \frac{N}{N+M}$$

$$flow\_left = C * \frac{M}{N+M}$$

Figure 6.4: the Formulas deciding dividing of the flow

Voting Result for: DE2		Filler	Helix	DE1	DE2	Result:
Agent						
StartProduction (Nominal)		1	1	1	-	3
StartProduction (MaxCapacity)						
StartProduction (underNominal)						
stopImmediate						
normalStop						

Figure 6.5: the result of the voting

false, it will then start to accumulate packages. D2 will continue to receive and deliver packages as long as there are packages to deliver. When the parameter packagesToDeliver is false, the agents planner will set a new diagnose and a new vote which will cause the DE2 to make a normalStop. The situation in the packaging line is the following, the filler is producing just like before, the Helix is receiving packages but not delivering any, the DE1 is blocked and the DE2 has made a normalStop. Now the blocked situation in DE1 is resolved, which is noticed by DE1, DE2 and the Helix, three votings will therefore take place. Also the filler now becomes blocked. The result of the Helix voting is the following.

The Helix will unload its packages since downstreamReady is positive. DE 1 will also start to produce (Receive & Deliver) as well as DE2. When the Helix is empty it doesn't have any more packages to deliver and it will therefore make a normalStop due to a voting. The same behavior is true for the machines following downstream.

### 6.2.7 Case 7

The first event in this case is that DE4 becomes blocked and stops production immediately. When this happens two votings will immediately take place, one for DE4 and one for the Merger. The result for the first vote is shown in figure 6.8.

The second vote result will be the following shown in figure 6.9, due to the fact that downstreamReady is false for the merger.

Since downstreamReady is false for all machines upstream from DE4, these machines will have votings with the same result as above, they will stopImme-

Voting Result for: DE2							Result:
Agent	Filler	Helix	DE1	DE2			
StartProduction (Nominal)	1	1	1	-			3
StartProduction (MaxCapacity)							
StartProduction (underNominal)							
stopImmediate							
normalStop							

Figure 6.6: the result of the voting

Voting Result for: DE2							Result:
Agent	Filler	Helix	DE1	DE2			
StartProduction (Nominal)							
StartProduction (MaxCapacity)	1	2	3	-			6
StartProduction (underNominal)							
stopImmediate							
normalStop							

Figure 6.7: the result of the voting

diate, including DE1. The Helix will start accumulating. The next event in the case is that the blocked situation in DE4 is resolved. This will make downstreamReady positive for the machines following downstream. Because of this, a number of votings will take place. The result for the DE2 and DE3 will look like the following shown in figure 6.10.

The result for the merger will be the same. The effect will be that all the packages downstream from the DE1 will be emptied from the production line.

Voting Result for: <i>DE4</i>									
Agent	Filler	Helix	DE1	DE2	DE3	DE4	Divider	Merger	Result:
StartProduction (Nominal)	1	1	2	3	3	-	5	1	16
StartProduction (MaxCapacity)									
StartProduction (underNominal)									
stopImmediate									
normalStop									

Figure 6.8: the result of the voting

Voting Result for: <i>Merger</i>									
Agent	Filler	Helix	DE1	DE2	DE3	DE4	Divider	Merger	Result:
StartProduction (Nominal)									
StartProduction (MaxCapacity)									
StartProduction (underNominal)									
stopImmediate	1	2	1	1	1	7	6	-	19
normalStop									

Figure 6.9: the result of the voting

Voting Result for: <i>DE2 &amp; DE3</i>									
Agent	Filler	Helix	DE1	DE2	DE3	DE4	Divider	Merger	Result:
StartProduction (Nominal)	1	2	3	-	-	6	3	7	22
StartProduction (MaxCapacity)									
StartProduction (underNominal)									
stopImmediate									
normalStop									

Figure 6.10: the result of the voting

# Chapter 7

## Simulation

### 7.1 Introduction

The aim of creating this simulator was to have a possibility to try out the agent intelligence in a not to complex line configuration. The simulator is a reconstruction of an already existing simulator, provided by Tetra Pak [?]. In general the simulator works in the same way but the control structure is changed. A separate control script has been introduced as representation of a software agent. There is one agent and an agent database linked to each machine, and possibility to extract information from a data table.

Line Simulator is programmed in Matlab and is modeled with a single thread running at step time 0.01s. The agent interface executes in conjunction with the update of respective machine.

### 7.2 Design Description

The new version of the simulator has several changes making it differ from the old version, but the generic construction is still intact. It is therefore possible for the user to construct a customized line configuration simply by adding machines to a cell-matrix,  $l$ , representing the line (figure 7.1). Machines are represented as data structures i.e. collections of variables. These collections are representations of so called holons, i.e. machine and intelligent agent as one entity.

The line configuration script is the part where machines and agents are registered into the agent platform. It is in the line configuration script that the user is able to set parameters individual to each agent. For example an agent receives its identification number or maximum allowed capacity is defined. To extract information during runtime, agents access information from the agent data structure within the cell-matrix  $l$  (figure 7.2).

#### 7.2.1 Improvements from older version

To be able to simulate branching in the line it is necessary to make a different representation of the line in the simulator than the old cell array. By introducing a cell matrix as representation it is possible to create parallel lines. Separate machines can be reached by simple indexing i.e.  $l\{i\}\{j\}$ .

```

global percentage;
    percentage = 0.02;

%% FILLING MACHINE
filling_config;
l{1}{1} = s;
l{1}{1}.agent_database.id = length(l);
l{1}{1}.id(1)=1;
l{1}{1}.id(2)=1;
l{1}{1}.agent_database.position = 1;
l{1}{1}.nominal_capacity = 20000;
l{1}{1}.agent_database.nC_old= l{1}{1}.nominal_capacity;
l{1}{1}.nominal_conveyor_speed = l{1}{1}.nominal_capacity*(0.126)/3600;

%% HELIX MACHINE
helix_config;
l{2}{1} = s;
l{2}{1}.agent_database.id = length(l);
l{2}{1}.id(1)=2;
l{2}{1}.id(2)=1;
l{2}{1}.nominal_capacity = l{1}{1}.nominal_capacity + (percentage*l{1}{1}.nominal_capacity);
l{2}{1}.normal_capacity = l{2}{1}.nominal_capacity;
l{2}{1}.maximum_capacity = 31000;
l{2}{1}.agent_database.nC_old= l{2}{1}.nominal_capacity;
l{2}{1}.nominal_outfeed_speed = l{2}{1}.nominal_capacity*(0.126)/3600;
l{2}{1}.nominal_infeed_speed = l{2}{1}.normal_capacity*(0.126)/3600;

```

Figure 7.1: Line configuration is done manually simply by modifying parameters in the data structures representing machines and databases.

Three machines, not represented in the earlier version, have been implemented and added to the simulator.

- Divider - handles branching of line and separation of package flow
- Distribution Equipment - non accumulating equipment
- Merger - handles line assembling

A possibility to define the physical machine's capacity constraints has been introduced. The parameters are *nominal capacity* and *maximum capacity*, and are given in units of packages per hour. The old version only defined the flow of the filler machine and assumed that no complications would arise that made such constraints necessary. We know for a fact that a machine must have a limit to how much it may produce, and that this limit could create problems due to accumulators unloading or queues emptying.

## 7.2.2 Agent software structure

The software agents are designed as matlab-scripts, each divided into three main parts (figure 7.3). Each part is constructed as a matlab function, individually designed for each type of agent.

```

s = struct(...***** - HELIX PARAMETERS - *****
    'type',                ('HELIX'),...machine type
    'id',                  ([0,0]),...
    'nominal_capacity',    (0),...
    'normal_capacity',     (0),...
    'maximum_capacity',    (0),...
    'state',               (1),...[signal]
    'command',             (2),...[signal]
    'nominal_infeed_speed', (0.7),...infeed conveyor setting speed [m/s]
    'nominal_outfeed_speed', (0.7),...outfeed conveyor setting speed [m/s]
    'no_more_package_to_deliver_timer', (0),...photocell timer[s]
    'no_more_package_to_deliver_delay', (20),...photocell timer raising threshold
    'agent_database',struct(...
        'id',                (0),...
        'waiting',           (0),...
        'nC_old',            (0),...
        'downstream_ready_flag', (0),...
        'position',          (2),...
        'acc_request',       (0),...
        'parallel',          (0),...
        'old_prio',          (''),...

```

Figure 7.2: A holon is represented by a data structure containing all relevant information, including the database used by its agent.

- Diagnostics
- Case-Based Reasoning
- Flow-Model & Execution

Each agent has its own Agent Database where control variables are stored. Agents communicate by updating parameters in other agent databases. In other words there is no modeling of the actual communication in this simulator since the environment doesn't support modeling of networks.

### Diagnostics

In the beginning of each sequence an agent runs a diagnostic procedure where parameters are read from the data table and relevant information is gathered from other agents.

There is certain information, collected from other agents, that doesn't need direct actions by the specific agent. For example it could be information about a machine's maximum allowed capacity that has changed. This update will not need direct actions by most agents but in the case of the Divider it will automatically affect the dividing of packages.

In the end of the diagnosis there is a complete reading of the specific agent database. Any information here placed by other agents will both be forwarded

```

% DATABASE SCANNING %
% %
[requests,DE] = database_reading(DE,requests); %
% %
% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% =====CASE BASED REASONING=====%%
% %
if ~isempty(requests) %
% %
% ADAPTATION RULES %
% [decision,DE] = RBR(DE,requests,decision,ID); %
% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% =====FLOW-MODEL/EXECUTION=====%%
% %
[DE] = execute(DE,decision,ID); %
else %
DE.agent_database.normalCase = 1; %
end %
% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Figure 7.3: The three main sections in the agent.

to other agents that need the information and also stored for later use. This makes the agent take into account the needs of surrounding agents. If there is a parameter change the discovered information is stored in an array, *requests*, and passed on to be taken care of by the reasoning section. Typical diagnoses may be; *isOverflowed*, *startAccumulating* and *maxOutput*.

### Case Based Reasoning

The next section of the agent decides upon what is best for the local machine that it represents. The decision-making in the simulator software agents is based on a predefined priority list within each agent, which defines what information to handle first.

Initially the *requests* array is read. Depending on the name of the request, and the type of the agent receiving it, a decision is carried out for the request. Case-solutions provide complete schemes for what the agent should do in case of an emergency. For example, if the request from the diagnosis in the helix agent is *maxOutput*, a message is sent to the agent downstream in order to inform of the increase of speed. There will also be an action, *maxOutput* stored in the array *decisions*, sent to the Execution in the Helix agent.



### Flow Model and Execution

To ensure that the decision taken by agent so far is allowed, regarding machine limitations, the decision is run through the flow-model. This model is a set of algorithms that guarantees that no undefined and unwanted decisions for the specific agent are executed. In most cases this is to ensure that a machines infeed runs at the same speed as its outfeed.

Finally the decision is carried out inside the execution function of the agent.

## 7.3 Simulation Cases

To verify the start- and stop policy and the adaptation rules that we have worked out, it is convenient to run simulations in order to get a hint whether the policies are strong enough. The cases that are presented below are the ones that are more sensitive and thereby most important for the agents to handle. Due to the simple structure of most production lines, the most interesting part to study becomes the dividing of packages in order to create branching in the line. The simulation cases are run on two specific line configurations (figure 7.4); the simple line of four machines, and the little more complex line with a branching structure.

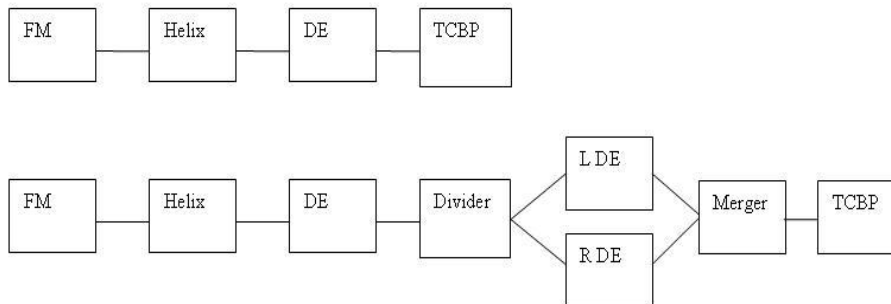


Figure 7.4: Simple line and line configuration with branching.

1) A simple Start- and Stop policy for the first line in order to verify that the machines start and stop policies work satisfactory. This is done by introducing a blocked condition for the last distribution equipment; this should start a chain-reaction making every machine upstream stop, including the Helix (after full accumulation level).

2) Similar to the first case. The only difference is that when the Helix has a 50% accumulation level the blocked distribution equipment will be ready for production and the package flow will start again. This case will verify that the package line is able to recover from a blocked situation and also that the line is able to handle a situation where the Helix must unload a large number of packages.

3) One of the branched distribution equipments becomes blocked. The divider must now direct the flow to the other distribution equipment in order to maximize throughput. This case will show that the divider is able to direct the flow, in order to maximize throughput, when one of the branching distribution equipments is blocked.

4) There is an increase of package flow, due to Helix unloading or production increase by the filler. This will verify that the production line is able to handle an increase of package flow.

5) A redistribution of the flow through the branch is generated, due to capacity change in one of the branched distribution equipments. This case will show that the divider is able to dynamically distribute the flow through the branching, and also that it is able to react to unforeseen events.

6) There is a situation in the first line where one distribution equipment becomes blocked which causes the Helix to accumulate. The next event is a blocked situation in the Filler machine. When the Helix has an accumulation level of 60% the distribution equipment resolves the blocked situation but the filler remains in blocked. This will make the Helix to unload its packages and after make a normal stop. The normal stop will propagate downstream until all machines have made a normal stop.

7) The first thing that happens is that a command *stopImmediate* makes the last machine in the line stop. This will make the stop propagate upstream. Then the distribution equipment, before the divider, will be in a situation that makes it do a *stopImmediate*. After that, the last distribution equipment will resolve its stop situation and be able to produce again. This will open the possibility for the machines in between to empty their packages and make a normal stop.

In the beginning of a simulation the user enters simulation duration and case number.

## 7.4 Simulation Results

The simulation results are plotted in a variety of windows with categories like; machine flow, package content, infeed speed, outfeed speed, state, and agent communication. All plots are based on different data stored, in a three-dimensional data matrix, each time step of the simulation. In order to be able to understand the results some definitions has to be posted.

Possible machine states

1. Preparation
2. Blocked
3. Ready for Production
4. Recieve and Deliver
5. Deliver
6. Recieve

### 7.4.1 Case Verification - Case 1

The first case listed above tests the start- and stop in the first line. At start up, as the machines are done preparing, each agent notifies upstream that downstream is ready and it is okay to start producing packages (figure 7.5). After 60 seconds the last machine in the line gets in a blocked state (figure 7.6). The TCBP agent immediately notifies the distribution equipment (DE) that downstream is not ready. The DE agent commands its machine to *stopImmediate*,

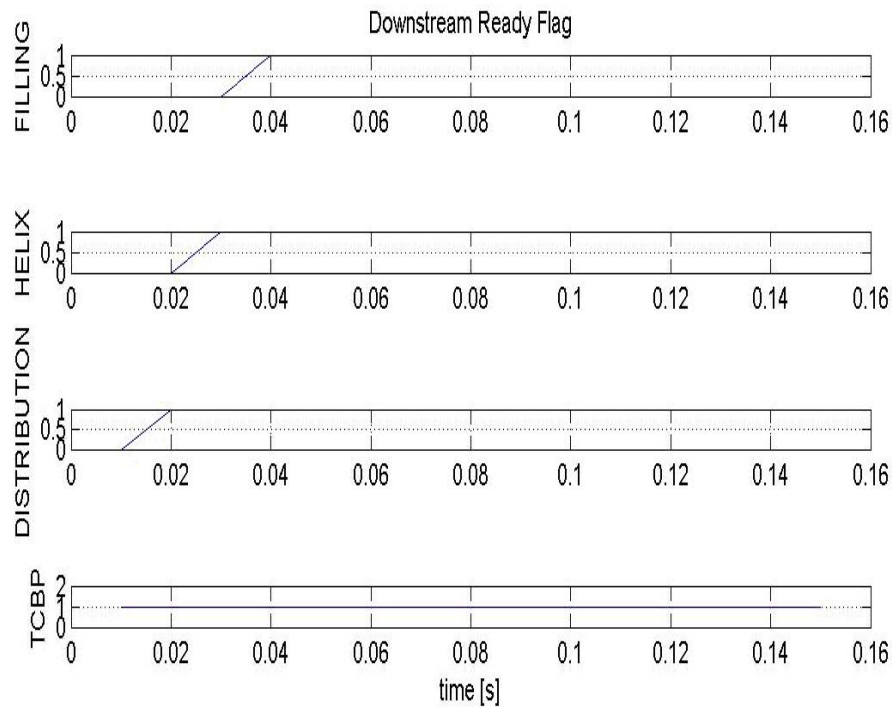


Figure 7.5: Downstream ready is notified, in a chain reaction, upstream as soon as a machine is done preparing. If downstream ready is false the machine will not start running.

why the machine goes into state *Receive*. At approximately 70 seconds the DE is full (figure 7.10) and therefore goes into *Ready for Production*. Now the Helix starts receiving until it is full and then stops. Finally the Filler produces packages until there is no more space in the machine and thereafter goes into *Ready for Production*, i.e. stops. The simulation sequence lasts for 300 seconds. About 200 seconds after the occurrence of the blocked situation in the last machine, the Filler gets full and stops, which make the entire line stand by doing nothing until the last machine resolves the situation (figures 7.7, 7.8, 7.9 7.11 and 7.12).

#### 7.4.2 Case Verification - Case 7

The last case of the ones listed above is a test of our start and stop policy in the second line configuration. At time 150s the last machine, the Tetra Pak Cardboard Packer (TCBP), gets blocked (figure 7.13). The TCBP-agent sends a message to the Merger-agent upstream that downstream is not ready. This cause the Merger to make a *Stop Immediate* which makes it go into state *Receive*. When the Merger is full it goes into state *Ready for Production* and sends a message to the closest machine (machines) upstream that downstream is not ready. This procedure continues until all machines are full or have no more packages to deliver. The reason why it takes longer time for the right

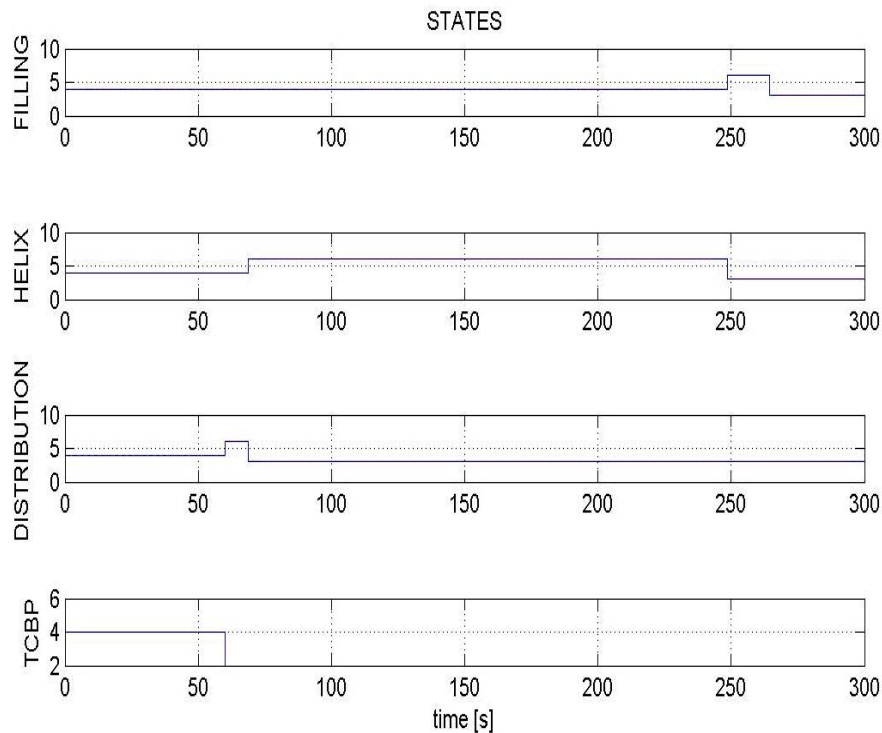


Figure 7.6: At 60 seconds the last machine in the line gets blocked. Upstream the distribution equipment starts filling up. When full they stop.

distribution equipment than the left, to get full, is that the divider is set to direct more packages to the left than to the right. It is important that this kind of situation does not create a problem.

Before the Helix reaches its full level the distribution equipment downstream of it becomes blocked. This occurs at 200s. For the moment this doesn't change the state of any machine downstream since they are all already in *Ready for Production*.

After 250s the TCBP resolves its blocked situation. The merger starts deliver packages until it's not full anymore and then goes into *Receive and Deliver*. Then, in a chain-reaction, upstream machines start the same procedure. After some time, starting with the divider, the machines get empty and stops.

When the machines start emptying their packages the agent in each separate machine will come to the conclusion that it is in a "full" situation and therefore start running at maximum capacity (figure 7.147.15).

### 7.4.3 Case Verification - Case 3

Case 3 is a test of our flow policy. It is run on the second line configuration like the previous case.

The Filler machine produces packages and delivers at 20 000 [p/h]. Initially the flow from the divider is divided equal; 10 000 [p/h] to the left and right

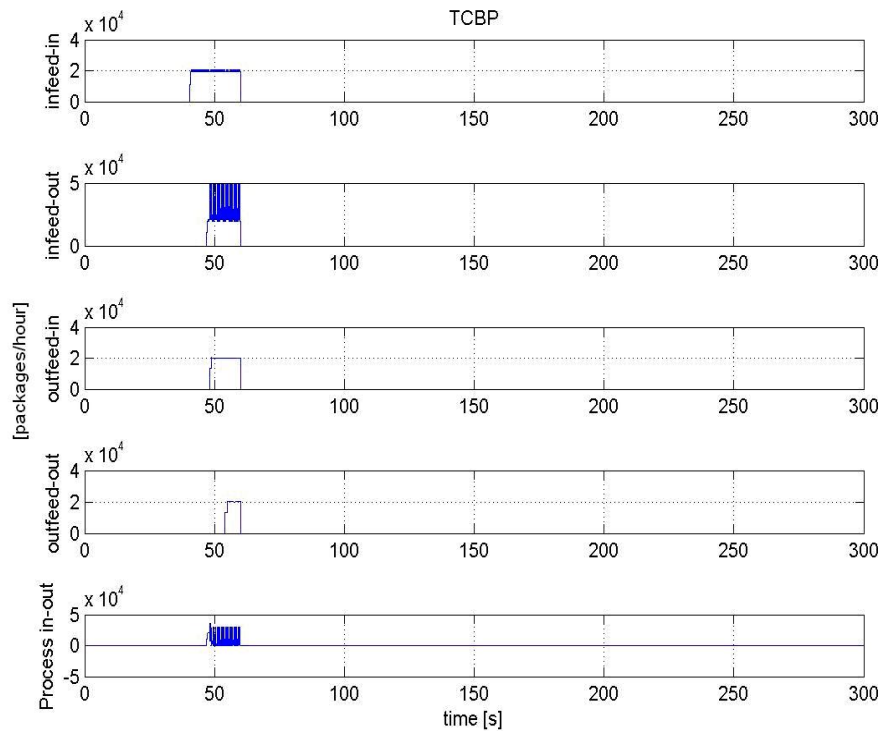


Figure 7.7: the TPCB becomes blocked one minute after start up

branch respectively (figure 7.16). As constraint on the branching distribution equipments a maximum allowed capacity of 15 000 [p/h] is introduced. After 80 seconds the right distribution equipment in the branch becomes blocked. The Divider agent redirects the flow so that the whole flow of 20 000 [p/h] goes left.

A request to the nearest accumulator upstream to decrease flow is now sent by the Divider agent since it detects that its inflow is greater than the sum of the branching machines maximum allowed capacities. Package content will start increasing on the left outfeed conveyor, in front of the left distribution equipment, until the decreased flow reaches the divider (figure 7.17). This is the explanation to the high flow on the left outfeed conveyor output during 50 seconds. The goal is to make the Helix accumulate just as much that the outflow of the helix becomes exactly 15 000 [p/h]. This information is included in the accumulation request by the Divider but is not that exact at this moment.

## 7.5 conclusion

Without implementing our theories it would have been hard to think about all possible situations that might happen when a line is running. This obviously helped us designing the voting rules for our policies.

Simulation results are satisfactory in all cases tested so far and it proves that the voting rules are strong enough to handle both line flow and start- and stop.

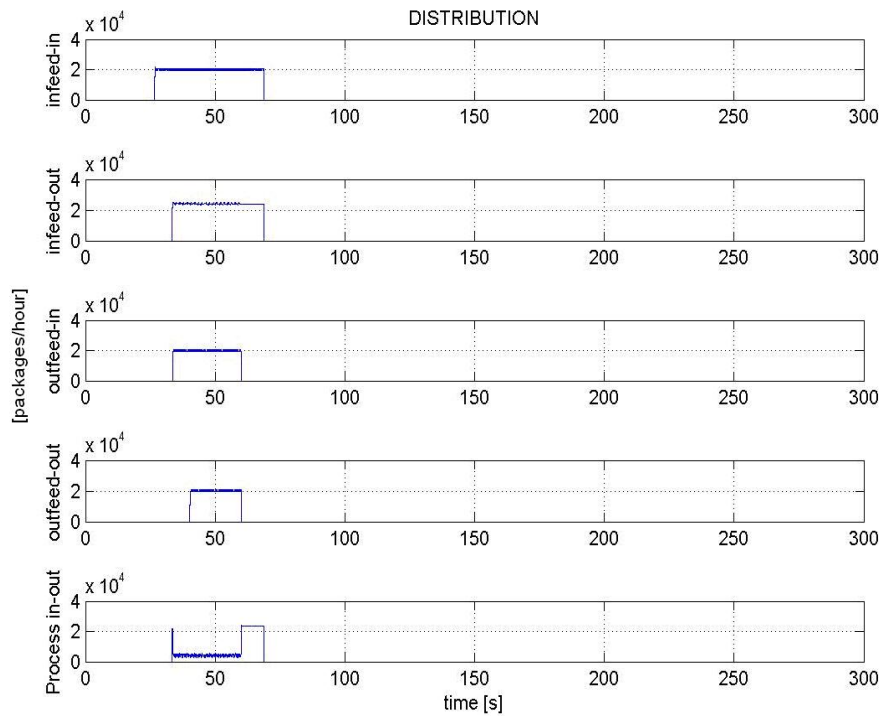


Figure 7.8: The distribution equipment receives packages until it gets full.

The dividing of packages and handling of capacity change when a production line branches out, has become one of the main parts of this project. It has therefore been very helpful to have a simulator to try out all theories on. The divider agent acts in a proactive manner when it informs nearest accumulator upstream of a problem that will happen in a while but has not yet occurred. This type of intelligent behavior is used in the implementation and works very well in the simulation.

Even though the results in the predefined cases came out positive there are still things to work on.

- All cases stored in the agents, as the simulator is implemented today, are solutions as result of our voting rules with no possibility of active voting.
- No concurrent execution of the agents is possible in our environment. During each time step the agents execute in a predefined order.
- The database reading sequence is cost expensive and makes the simulator run a bit slow.

All of the new machines implemented are designed based on assumptions and logical thinking, since as of today there are no machines of that exact kind. The implementation of these machines turned out to take more time than planned. This is one of the reasons why there wasn't time to try all ideas we had for the

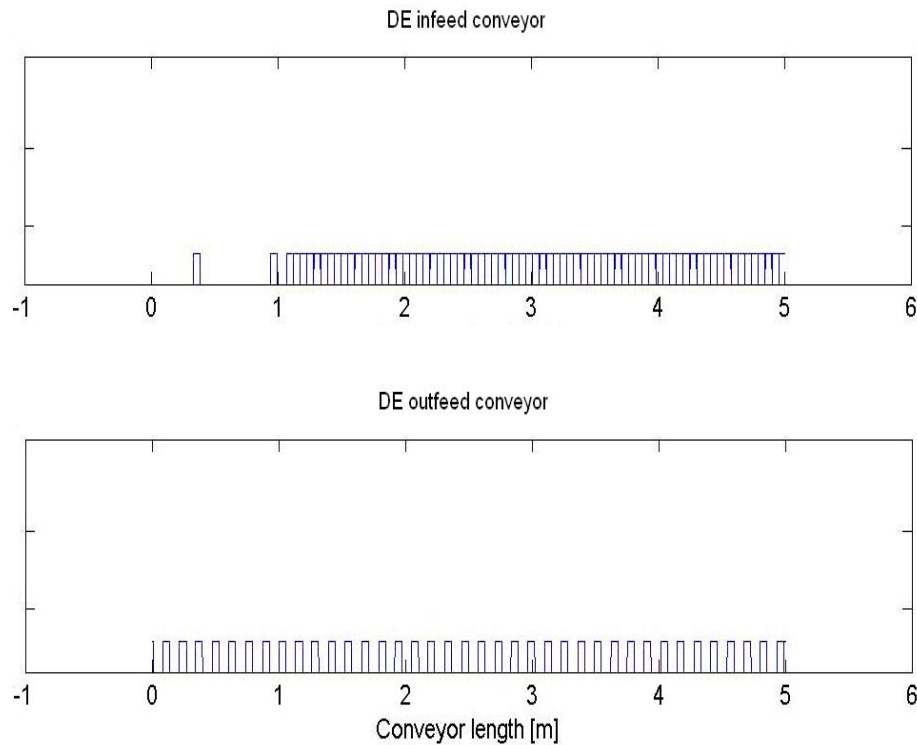


Figure 7.9: At time 70 seconds the infeed conveyor of the distribution equipment get full and the machine is declared full as well.

simulator. As it turned out it is still of much help when it comes to get an idea of whether our rules behind the control policies are strong or not.

It would be interesting to apply the agent solution and policies that we use in a simulator environment more compatible with the distributed agent technology. The ideal way to implement the simulator would be to use an environment that handles modeling of real-time kernels and networks for wireless radio communication, since that is what the real application would look like. In a not to distant future this could be the case. Today Tetra Pak uses software from Rockwell Automation in order to program their PLC's. In the lab at Rockwell Automation in Cleveland a way to simulate Multi-Agent Systems has been developed [13]. The ambition in the future is, if possible, to use the Rockwell tools in order to take the next step towards an agent based solution.

We think it is profitable for Tetra Pak to apply this technology in their packaging lines in the future, mainly because of three reasons;

The robustness of the system will improve with the ability to act swift and correct to unforeseen events.

There will be less waste on the floor due to the proactive behavior of the agents.

Due to the abstract architecture of the system it will be possible to integrate new machine types without reconfiguration.

The question is how distant in the future this lies and how much the company

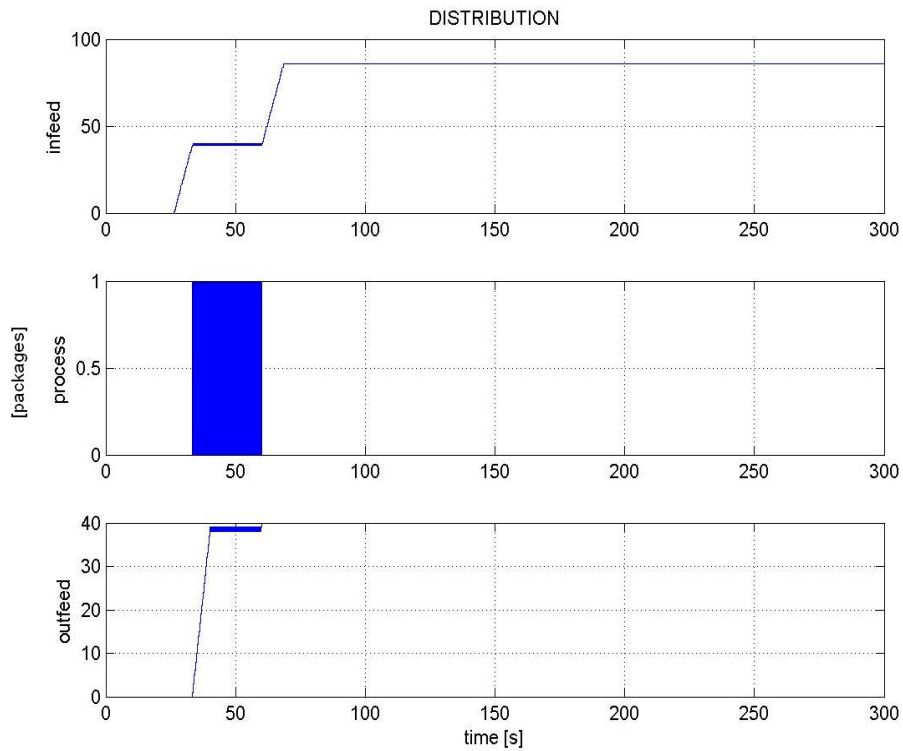


Figure 7.10: When the infeed conveyor of the distribution equipment is full, the machine is declared full. This happens at approximately 70 seconds.

is willing to pay to apply the agent based solution. The technology is still young so it is difficult to look at real applications because there are very few. There will probably have to be a lot more research done in this field before a complete solution feels realistic. Still, as far as we are concerned we see no reason why Tetra Pak shouldn't continue on this track.



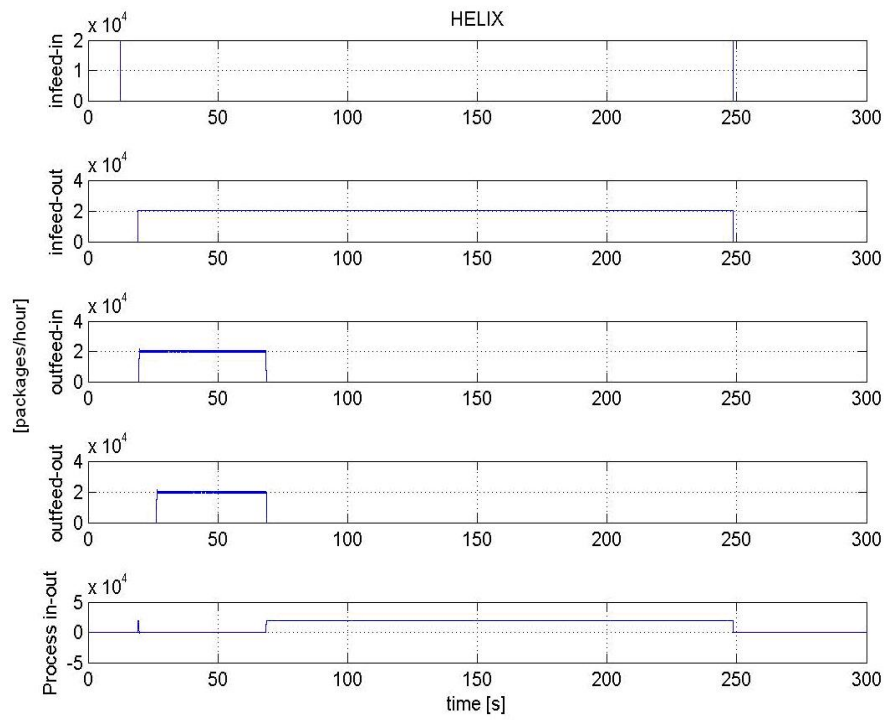


Figure 7.11: The helix start accumulating packages when downstream ready is false.

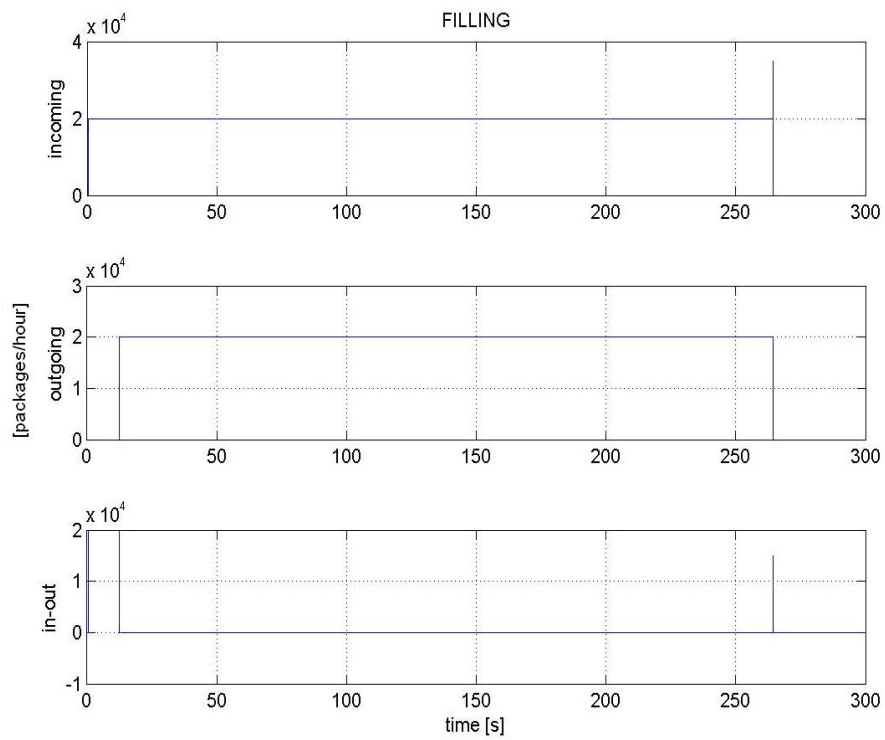


Figure 7.12: The filler machine start producing packages at 20 000 [p/h].

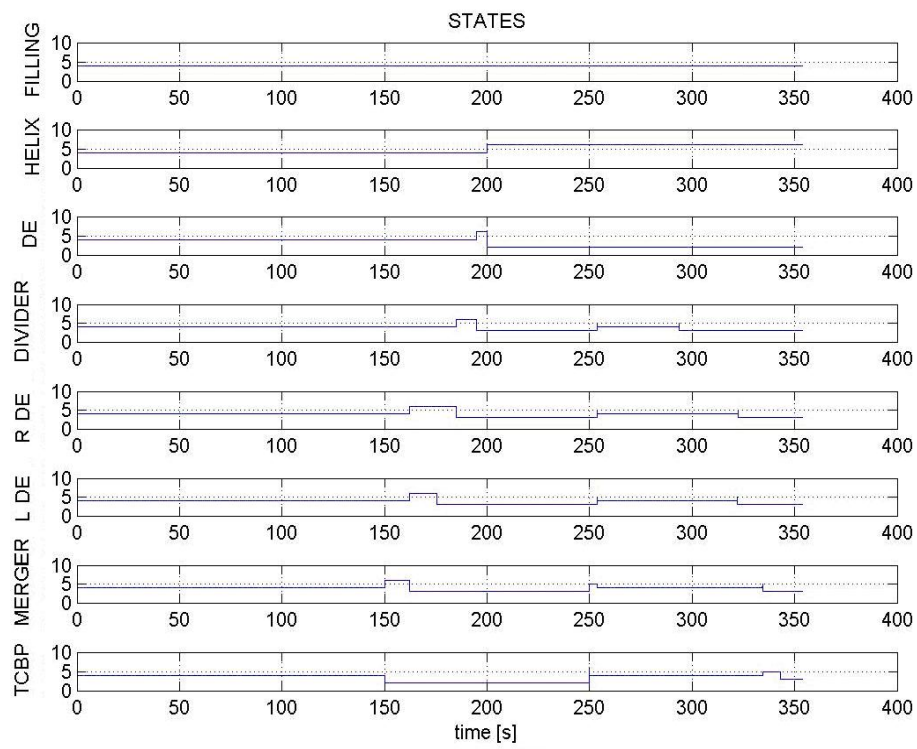


Figure 7.13: a plot showing the state changes in case 7

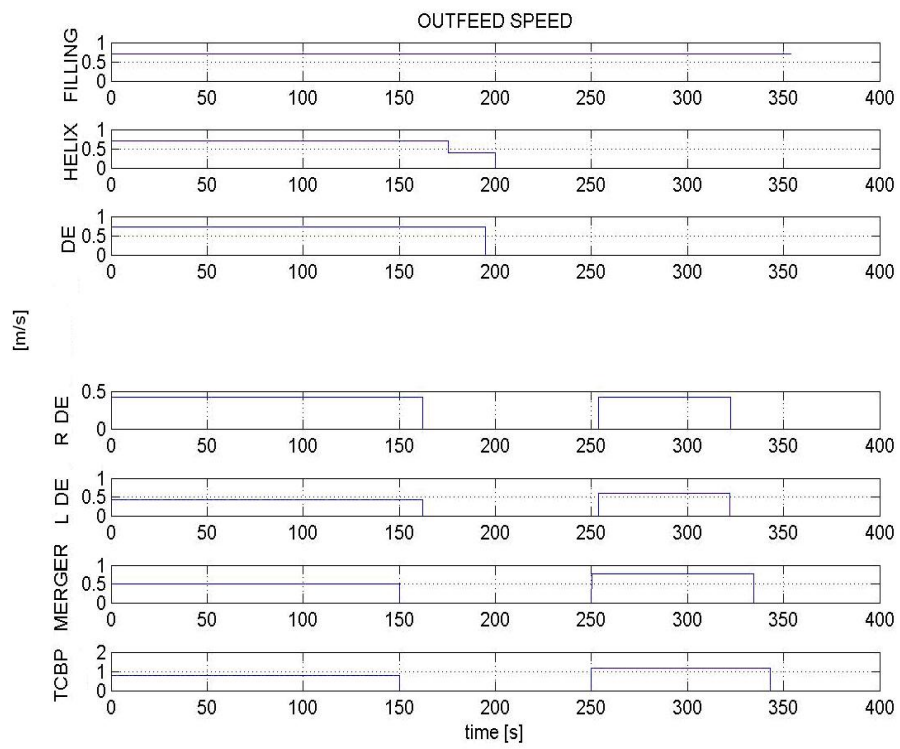


Figure 7.14: the outfeed speed of all machines that only have one outfeed conveyor, i.e. the divider is not included.

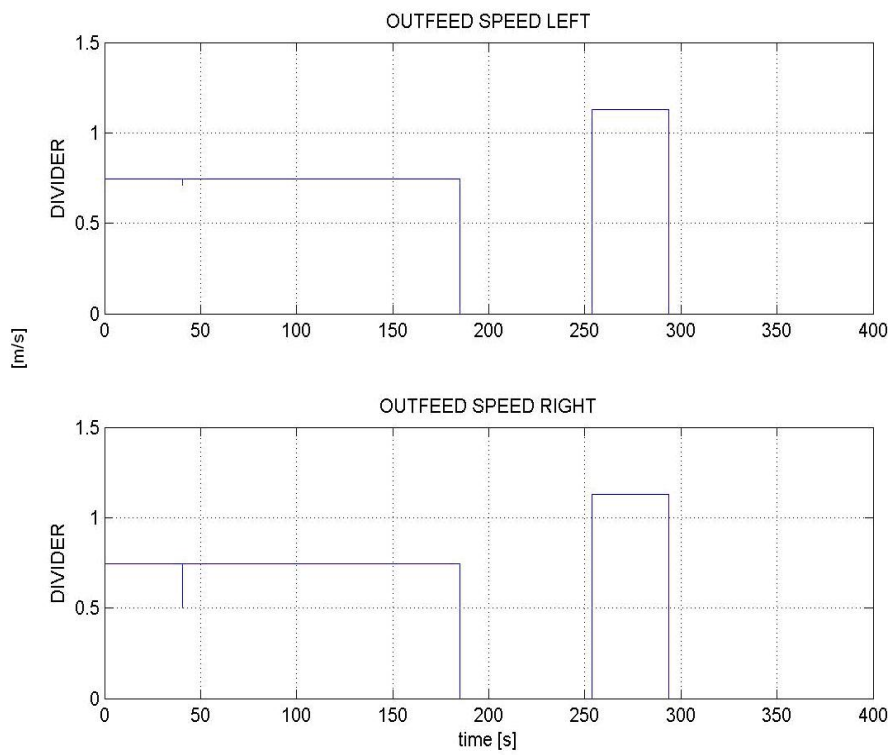


Figure 7.15: the outfeed speed of the dividers two outfeed conveyors

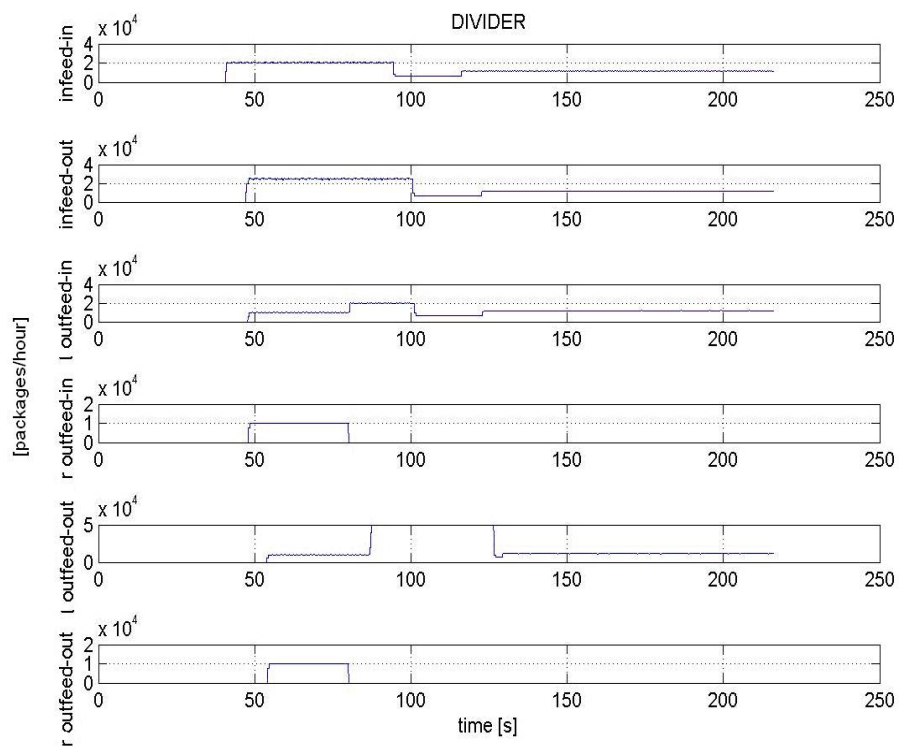


Figure 7.16: A redistribution of the flow through the divider is done after 80 seconds. The entire package flow is now directed to the left outfeed conveyor.

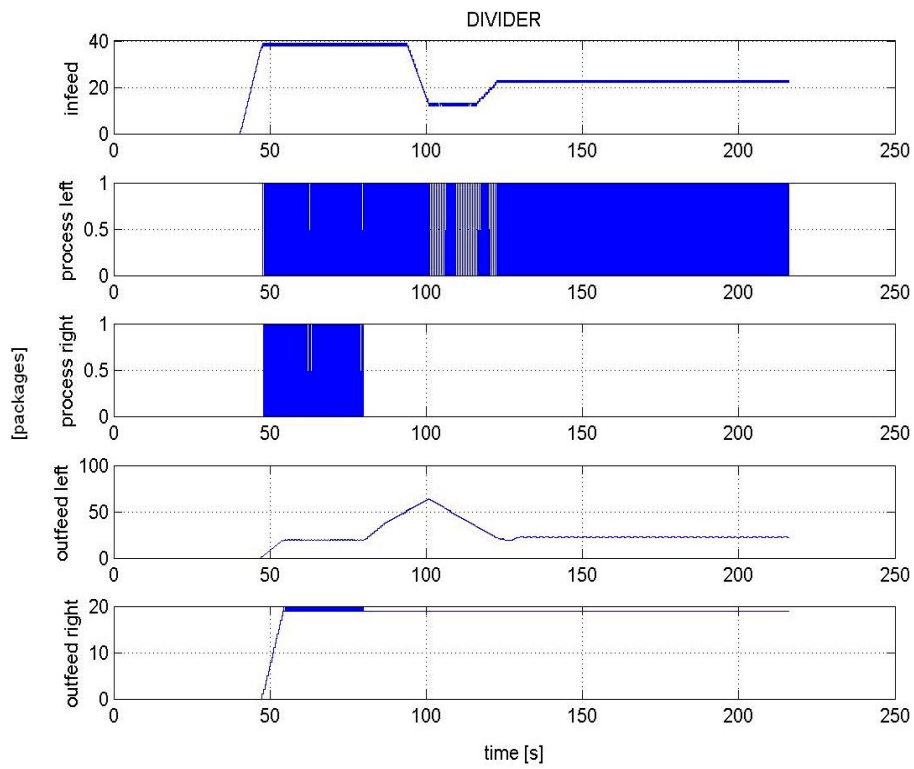


Figure 7.17: Immediately after that no more packages are sent to the right outfeed conveyor, packages starts stock on the left outfeed conveyor.

# Bibliography

- [1] F. Maturana, R. Kotina, P. Tichy, R. Staron, D. Carnahan: *Distributed Agent Control for Water Networks*, Rockwell Automation.
- [2] U. Berger, G. Vladimirova, R. Kretzschmann: *Performance Increase of Industrial Robot Plants through Agent-Based Technologies*, Brandenburg University of Technology, (2004)
- [3] P. Tichy, V. Marik, P. Vrba, F. Macurek, P. Slechta, R. Staron, F. Maturana, K. Hall: *Deployment of Agent Technology in Fault Critical and Scalable Industrial Applications*, Rockwell Automation
- [4] S. Kraus: *Automated Negotiation and Decision Making in Multiagent Environments*, Dept. of Mathematics and Computer Science, Bar-Ilan University
- [5] P. Tichy, V. Marik, P. Vrba, F. Macurek, P. Slechta, R. Staron, F. Maturana, K. Hall: *Intelligent Autonomous Control Architecture for Automated Ship Control, Damaged and Optimized Operations*, Rockwell Automation
- [6] L. Gianetti, P. Valigi, F. Maturana, F. Dscenzo: *Multi-Agent based Algorithm for a Municipal Water System*, Rockwell Automation, Dipartimentodi Ingegneria Elettronica e dell'Informazione, University of Perugia
- [7] P. Tichy, V. Marik, P. Vrba, F. Macurek, P. Slechta, R. Staron, F. Maturana, K. Hall: *Cost-Based Dynamic Reconfiguration System for Intelligent Agent Negotiation*, Rockwell Automation
- [8] E. Ephrati, J. Rosenschein: *The Clarke Tax as a Consensus Mechanism Among Automted Agents*, Computer Science Department, Hebrew University Givat Ram, Jerusalem , Israel
- [9] F. Discenzo, F. Maturana, D. Chung: *Managed Complexity in An Agent-Based Vent Fan Control System Based on Dynamic Re-Configuration*, Rockwell Automation
- [10] J. Taylor, A. Sayda: *An Intelligent Architecture for Integrated Control and Asset Management for Industrial Processes*, (2005)
- [11] S. Russel, P. Norvig: *Artificial Intelligence: A Modern Approach*, Prentice Hall (1995)
- [12] D. Vasko, F. Maturana, A. Bowles, S. Vandenberg: *Autonomous Cooperative Control*, (2000)



- [13] V. Marik, D. McFarlane: *Industrial Adoption of Agent-Based Technologies*, Rockwell Automation.
- [14] FIPA: *FIPA ACL Message Structure Specification* Foundation for Intelligent Physical Agents, (2000).
- [15] FIPA: *FIPA Communicative Act Library Specification* Foundation for Intelligent Physical Agents, (2000).
- [16] FIPA: *FIPA Abstract Architecture Specification* Foundation for Intelligent Physical Agents, (2000).
- [17] FIPA: *FIPA Content Language Library Specification* Foundation for Intelligent Physical Agents, (2000).
- [18] G. Goldoni, L. Tacconi: *Line Simulator 1.0, User Specification*

