MODELING AND CONTROL OF DEDS USING STATEFLOW

M. Blaho, Ľ. Farkas***

[*]Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic

[**] SYPRIN s.r.o., Žehrianska 10, 851 07 Bratislava, Slovak Republic

Abstract

Most of the manufacturing systems can be described with the states and the events. These systems are also known as discrete event dynamic systems (DEDS). Matlab provides a tool for modeling and control of these systems - Stateflow. Our goal was to model a manufacturing process and design an optimal control with a genetic algorithm.

1 Introduction

Systems can be described as the continuous systems or the discrete systems and the combination of both as the hybrid systems. In continuous systems the variables are functions of time. In discrete systems variables are sampled in specific periods. Discrete systems can be activated by time or by events [1]. In this paper we only focus on the discrete systems which are activated by events. Event-driven systems transition from one operating mode to another in response to events and conditions. Event-driven systems can be modeled as finite-state machines. Finite-state machines represent operating modes as states [2]. Finite-state machines can be constructed in Matlab tool called Stateflow. Stateflow also extends the capabilities of traditional state charts by adding some useful properties (hierarchy, parallel states, truth tables etc.).

The rest of the paper is organized as follows. In section 2 we describe the manufacturing process and its properties. In section 3 we describe the model of the manufacturing process as a state chart in Stateflow. In Section 4 we show the rest of the simulation schema. Section 5 describes the design of the control strategy for manufacturing process. In Section 6 we shortly show how to write a control program on industrial device which uses Ethernet for real-time communication. Section 7 summarizes the paper.

2 Process

As a control object we chose a manufacturing process on the figure 1 [1]. The system consists of one input and one output conveyor (C1, C3). Other two conveyors (C2, C4) are for transport. For simplicity we assumed that the capacity of the conveyors is one. There are also three robots (R1, R2, R3) of which two have a part of their workspace same. We had to be careful that these two robots didn't crash. The rest of the system are four machines (M1, M2, M3, M4). The system can make various products depending of the order of operations at the machines.

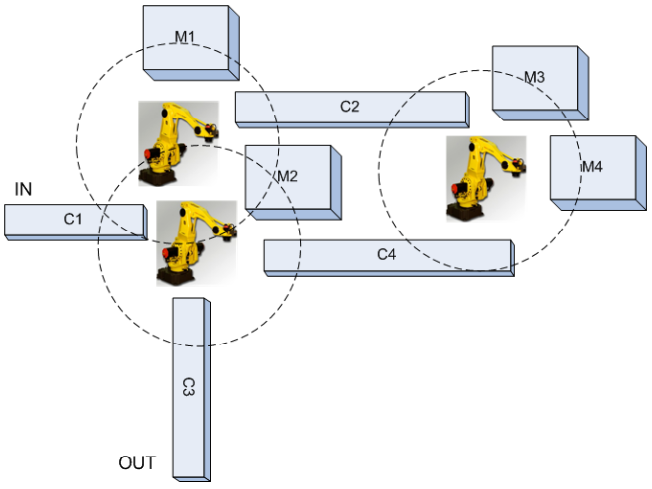

Figure 1: Manufacturing process

In the table 1 we can see the necessary operations for making three products. Each machine can make only one operation at the same time. Therefore other products must wait for ending of operation on the machine. Some operations can be made on two machines for a specified product.

TABLE 1: LIST OF OPERATIONS

	Product		
Operation	A	B	C
1	M1	M2	M1
2	M2 or M3	M4	M3
3	M4	M3 or M4	M3 or M4
4			M2

In the table 2, there are the times for operations on the specific machine. Depending on the product, same operation on various machines can take various or same time (for simplicity we chose same time). These times can be used to design the control algorithm as a criterion.

TABLE 2: TIMES FOR OPERATIONS

Product											
	A				B			C			
Op.	Machine				Machine			Machine			
	M1	M2	M3	M4	M2	M3	M4	M1	M2	M3	M4
1	t1				t5			t9			
2		t2	t3				t6			t10	
3				t4		t7	t8			t11	t12
4									t13		

3 Stateflow model

The model of the system was made in Simulink. The Simulink model consists of three main parts: Stateflow model, control algorithm and visualization. The simulation was set with fixed-step and no continuous states. In this Section we describe the Stateflow model.

As you can see on the figure 1 the whole manufacturing process consists of eleven subjects. These subjects work independently, so we made eleven substates with parallel (AND) decomposition. Inputs for the Stateflow model were events either from the visualization or from the control. The exception was the first event which changes trigger with every sample time (*impulz*). Outputs from the Stateflow model were data for the visualization.

Substates one to four represents the behavior of the conveyors. Figure 2 represents the input conveyor C1. At the start we entered the state *Daj*. This state indicates that someone put a product on the input conveyor (we are not interested who it was, just assume that someone put it there). In the next sample time we entered state *Ide* which signalizes that the conveyor is moving. Event *koniecC1* shows that the product is at the end of conveyor and the substate comes to state *Caka*. After next event *koniecC1* conveyor entered the state *Prazdny* which indicates that conveyor is empty. For visualization

we used two variables. The variable *dielnaC1* represents that product is on the conveyor and variable *ideC1* represents that conveyor is moving.

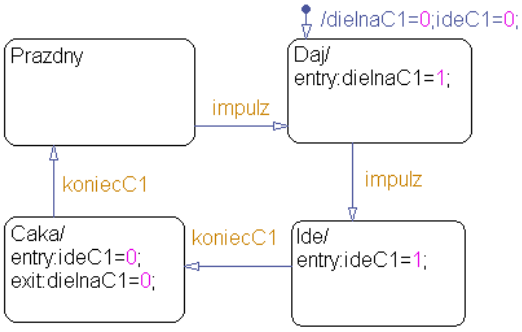

Figure 2: State representation of the conveyor C1

Figure 3 represents conveyor C2 (C4 is similar to C2). At the start we entered state *Prazdny* which indicates that conveyor is stopped. Now we are waiting either for the event *L_C2* or *P_C2*. This events represents that the product came on the left side or the right side of the conveyor. Next we assumed that it came on the right side. The next state signalizes the product on the right side. After a sample period we entered state *Vlavo* which indicates that conveyor is moving to the left. Event *L_C2* tells us that the product is at the end of the conveyor. Now we are in state *Stop* which means that we are waiting for someone to pick up the product. After next event *L_C2* conveyor comes to the state *Prazdny* again. The variable *dielnaC2* represents that the product is on the conveyor and variable *ideC2* represents that machine is moving. These variables are also used in visualization.

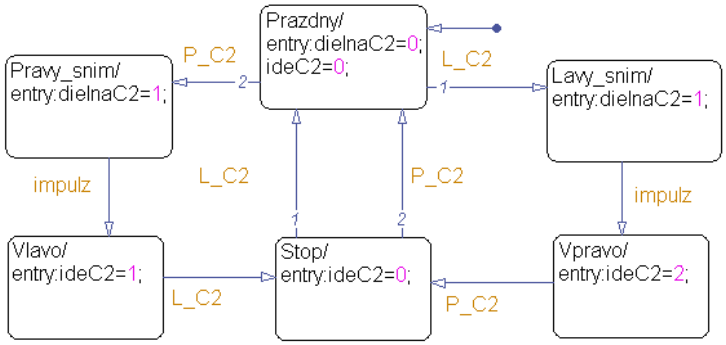

Figure 3: State representation of the conveyor C2

Figure 4 represents the output conveyor C3. At the start we entered state *Caka*. This state indicates that the conveyor waits for product and is stopped. Event *koniecC3* indicates that someone puts a product on the conveyor and we entered state *Daj*. After a sample period we come to the state *Ide* which indicates that the conveyor is moving. The event *koniecC3* indicates that someone gets the product from the conveyor and we entered state *Caka*. For visualization we used two variables. The variable *dielnaC3* represents that the product is on the conveyor and variable *ideC3* represents that the conveyor is moving.

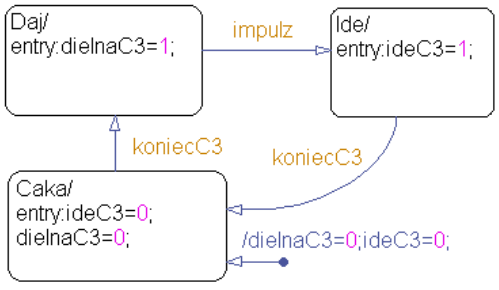

Figure 4: State representation of the conveyor C3

The substates five to eight represent the behavior of the machines. One of these substates for machine M1 you can see on the figure 5. At the start we entered the state *Prazdny*. This state indicates that the machine is empty. After event *snimM1* we entered the state *Vstup* what means that the product is in the machine. In the next sample time the machine starts to make the required operation in state *Obraba*. Then we wait for end of the operation which is indicated by event *koniecM1*. Next event *snimM1* indicates that the product was taken from the machine. The variable *dielnaM1* represents that the product is in the machine and variable *ideM1* represents that the machine is working. These variables are used in visualization.

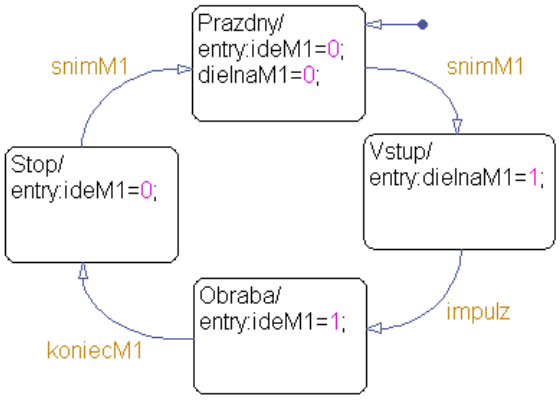

Figure 5: State representation of the machine M1

Last three substates represent the behavior of the robots. We describe the behavior of the most complicated R3, others are working similar. On the figure 6 you can see the substate for R3. At the start we entered state *Stop* which indicates that the robot isn't moving. Other states represent that the robot is moving to the desired location. On our figure this states are *doC2* (to conveyor C2), *doM3* (to machine M3), *doM4* (to machine M4) and *doC4* (to conveyor C4). To these states we can enter after events *R3XYp* or *R3XYn* where *XY* is the target object (conveyor or machine), *p* indicates that robot is only moving and *n* that robot is caring product to the target object. The event *koniecR3* means that the robot finished moving. The variable *pozR3* represents the desired location and variable *nosR3* represents that robot is carrying a product. These variables are also used in visualization.

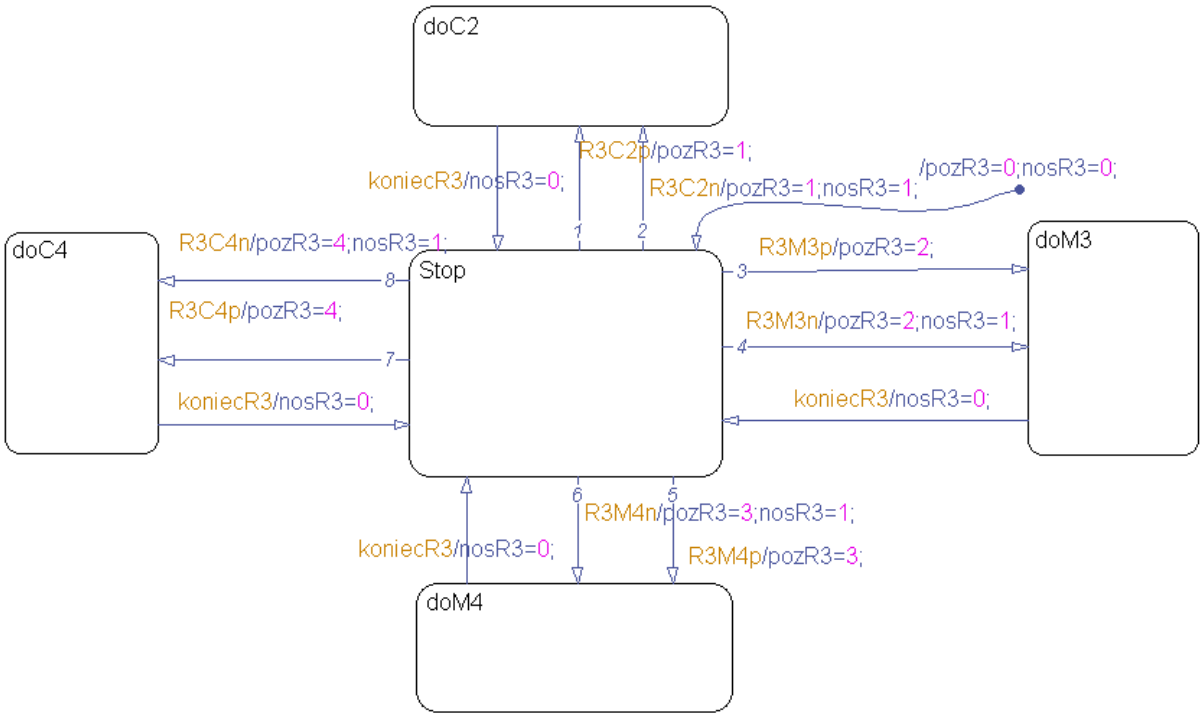

Figure 6: State representation of the robot R3

4 Visualization and Control

Another two main parts of the simulation program are visualization and control. The visualization has two functions. One is to visualize what is happening in the process and the second is to activate the events. Visualization activates events at the end of each part in process, for example the product comes at the end of the conveyor or when robot comes to desired location. The function of control is to activate the optimal events for machines and robots. For this we used a genetic algorithm.

Genetic algorithms are universal stochastic searching or optimizing approach which is in bounded space of the solutions able to find or come near global optimum. Mechanism of the genetic algorithm is on the figure 7 and is explained for example in [3], [4], [5] and [6] or in literature dealing with evolutionary techniques.

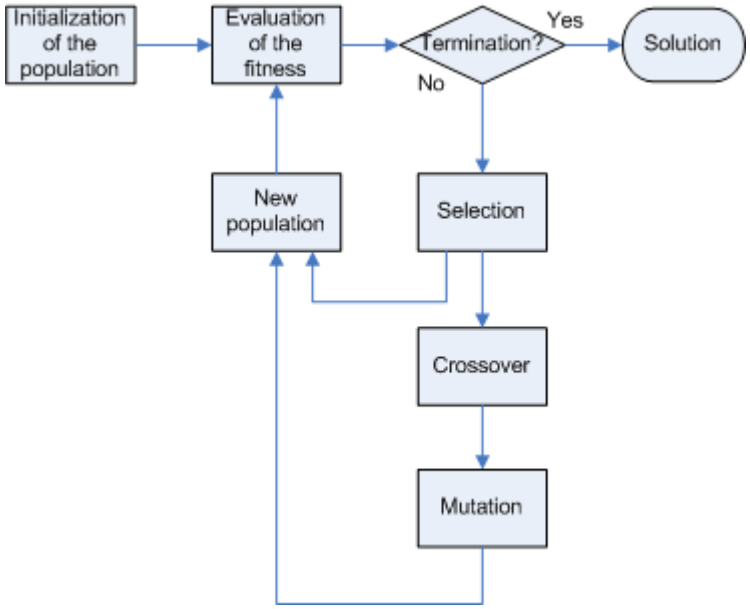

Figure 7: Block scheme of the genetic algorithm

In our genetic algorithm we optimized two problems. In first we must decide order of the products for the process. Wrong order could cause overflow in system. Another problem is to generate optimal commands for the machines and robots. Moving robots to wrong positions could cause time leaks. The criterion for genetic algorithm was the minimal time for the manufacturing process with given amount of the products. We must also check the availability for the given solution, if the solution is reachable in the system. In the table 3 you can see one of the solutions for system with our specified times.

TABLE 3: ORDER OF OPERATIONS FOR SPECIFIED ORDER OF PRODUCTS

Order of products	Order of operations			
C	M1	M3	M3	M2
B	M2	M4	M4	
A	M1	M2	M4	
C	M1	M3	M4	M2
A	M1	M3	M4	
C	M1	M3	M4	M2
A	M1	M3	M4	
B	M2	M4	M4	

5 Practical realization by the EtherCAT fieldbus

It is convenient to control a similar production line by a fieldbus in practical realisation. In general, fieldbuses are convenient for controlling of the production process due to unified form of communication between the sensors and the actuators and because of easy way to collect data. If we want to controll the process in real time, we need a fieldbus with high data bandwith and with apparatus for assuring data exchange with minimal latency. As a suitable fieldbus for our purposes appears the EtherCAT fielbus. EtherCAT is a new and open real-time ethernet technology, based on master-slave communication. On the present it is an open standard controlled by the EtherCAT technology group.

The EtherCAT fieldbus uses a different way of processing the ethernet frame as the ethernet. The ethernet frame is processed by EtherCAT „on the fly“. The newly developed FMMU (fieldbus memory management unit) in each slave node reads the data addressed to it, while the telegram is forwarded to the next device. Similarly, input data is inserted while the telegram passes through. The telegrams are only delayed by a few nanoseconds [6]. In ethernet, the CSMA/CD media access is used, but collisions may appear thus variable delay of the data may appear. Common property of both technologies is, that at both can the same inexpensive, commercially available standard network interface cards and ethernet cables used. The usage of these components, and the high data bandwith are the advantages of EtherCAT technology against other fieldbus technologies. It allows easy and inexpensive installation and the possibility of further expansion of the network. Example of connection of the remote input/output devices is on the fig. 8.

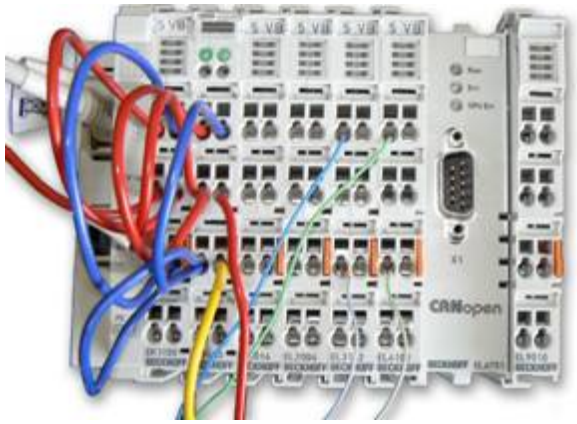

Figure 8: Connection of the slave devices on the strip, there is also a CANOpen extension module present

The master device on the network is an industrial PC with a software based PLC with hard real-time handling capability installed. In our case, it is the TwinCAT software from the Beckhoff Automation company. This software is a real-time runtime system in the Windows XP operating system.

EtherCAT supports almost any known topologies. We can use the bus, star or the branch topology. It is useful to use the combination of the bus and the star topology mainly in the system connection. The required interfaces are integrated in the couplers, therefore no additional network switches are needed. Though the whole EtherCAT network can be interconnected by classic switches known from the conventional ethernet networks [7].

TwinCAT PLC is programmable according to IEC 61131-3 standard. Supported are all programming languages according to the standard: Ladder logic (graphical), Function block diagram (graphical), Structured text (text), Instruction list (text). The combination of cyclical running of the controller program and any of the mentioned programming languages is suitable for control of discrete event systems similar to ours. In our case, the controller program could be programmed exactly according the algorithm optimized by the Genetic algorithm. We could divide the program into several function blocks due to limpidity. We are able to adjust also the period of running of the individual function blocks in the TwinCAT environment.

6 Summary

Our goal in this work was to model and control a manufacturing process with 4 conveyors, 3 robots and 4 machines. The plant made 3 types of products. For modeling and visualization of the process we used Stateflow and Matlab graphics. We explained how to model each element of the process in Stateflow. Then we designed the control of the plant by a genetic algorithm. The control had to be optimal in term of minimal time leaks. After the design of control, we explained the EtherCAT fieldbus and its capabilities. The optimized control algorithm can be programmed into the software PLC provided by the TwinCAT system.

7 Acknowledgments

This work has received support from the Slovak Research and Development Agency. Project reference number is APVV-99-045805. We also want to thanks for support Scientific Grant Agency VEGA under ref. No. 1/3841/06.

References

[1] B. Hrúz, L. Mrafko. *Modelovanie a riadenie diskrétnych udalostných dynamických systémov s využitím Petriho sietí a iných nástrojov*, Slovenská technická univerzita v Bratislave 2003.
[2] The MathWorks, *Getting started with Stateflow 7*, 2005
[3] I. Sekaj. *Evolučné výpočty a ich vyyužitie v praxi. 0. vyd*. : Vydavateľstvo IRIS, 2005.
[4] I. Sekaj, M. Foltin. *Matbal Toolbox – genetické algoritmy*, Konference Matlab 2003, Prague 2003
[5] M. Foltin, S. Bartoš, M. Kollárčik, I. Sekaj. *Budovanie virtuálneho laboratória a návrh regulátorov cez sieť,* Konference Matlab 2003, Prague 2003
[6] EtherCAT — *Ethernet Control Automation Technology* [online] http://www.ethercat.org/ , EtherCAT Technology Group, 2007
[7] T. Murgaš, P. Fodrek, Ľ. Farkas. *Priemyselné zbernice s pripojením na Ethernet*, Elosys 2007, Trenčín 2007

Michal Blaho
michal.blaho@stuba.sk

Ľudovít Farkas
ludovit.farkas@syprin.sk