Matlab Simulink Stateflow for Supervisory Control Simulation and Automated Code Generation

Anderson S. Carvalho∗ Felipe Alves Pereira∗ Marcelo Götz∗ Ivan Müller∗∗

∗ Departamento de Automação e Energia, Universidade Federal do Rio Grande do Sul, RS (e-mails: andercarvalho04@gmail.com, felipepereira82@gmail.com, mgoetz@ufrgs.br).

∗∗ Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, RS (e-mail: ivan.muller@ufrgs.br)

Abstract: In manufacturing systems, a class of Discrete Event Systems (DES), plant behavior can be modeled through automata which serve as the basis for synthesizing controllers using Supervisory Control Theory (SCT). The present paper proposes the usage of Matlab Simulink Stateflow for modeling automata and for automated code generation. The usage of Matlab Simulink is encouraged by its widespread acceptance in industry providing, also, an intrinsic possibility of simulation and flexible automated code generation capability. The feasibility of the proposal is shown by preparing a testbed based on an embedded system target used to set up a hardware-in-the-loop simulation architecture.

Keywords: Discrete Event Systems; Supervisory Control Theory; Hardware-in-the-loop simulation; Matlab Simulink Stateflow; Embedded System.

1. INTRODUCTION

Discrete Event Systems (DES) refers to a system class that differs essentially from Continuous Variable Systems (CVS). DES is event-driven and deals with discrete states (Cassandras and Lafortune (2008)). Several systems are DES examples: software tasks in a computation system that need to be processed by the microprocessor; traffic lights for controlling vehicles in streets intersections; people in a queue standing to be served in a bus stop; manufacturing plants with machines, conveyors, belts, etc. Manufacturing systems, especially, has received more attention in the past years since this area seeks new formalism and methods to enable high performance and flexible plants design (Fabian and Hellgren (1998); Hasdemir et al. (2008)).

Different modeling techniques, such as Petri Nets, Markov Chains, queueing theory, automata form the most basic class of DES models, as mentioned in Cassandras and Lafortune (2008). Ramadge and Wonham (1987) proposes a formalism and a methodology to derive a DES controller using automata, the Supervisory Control Theory (SCT).

Even though SCT provides a powerful framework for designing DES controller using a systematic and formal methodology, especially to manufacturing systems, it has not gained acceptance in the industry yet (Wonham et al. (2018)). This is mostly since SCT implementation on Programmable Logic Controllers (PLCs), which is the still most used technology in the manufacturing industry, has some obstacles to be overcome. PLCs are basically synchronous systems, in counterpart to the essentially asynchronous modeling bases of SCT theory (Fabian and Hellgren (1998)). Moreover, PLCs have been used widely in industry for more than 25 years, and it forms the bases technology for the most current and new applications (Hasdemir et al. (2008)).

Academic society, however, has demonstrated a great interest for DES and SCT theory. Several attempts to overcome the gap between scientific achievements and industrial use have been promoted, like Fabian and Hellgren (1998); Hasdemir et al. (2008); Afzalian et al. (2008); Jean-Marc ROUSSEL (2005). Additionally, several academic software tools have being developed, like for instance Ricker et al. (2006); Moor et al. (2008); Rudie (2006); Preischadt et al. (2015); Thomas Moor, Klaus Schmidt et al. (2010); Jordan et al. (2017), enabling the design and simulation of plants and controllers for manufacturing systems.

In this direction, it is proposed the usage of Matlab Simulink Stateflow for modeling supervisor and plant automata, a widespread development tool well accepted in the industry and academia. In addition, the controller is deployed using the automated code generator, also supported in Matlab Simulink. In this paper, Embedded Coder was used. Nevertheless, Matlab Simulink provides automated code generation for other languages and environments, like Hardware Description Language (HDL) suitable for Field Programmable Gate Array (FPGA).
The remaining of the paper is divided as follows. The next section will introduce fundamental concepts and tools along with related work. The proposal is explained in Section 3 and experimental results and analysis are given in Section 4 using, therefore, a case study. Finally, Section 5 draws some conclusions about this work and point some future work.

2. PRELIMINARIES AND RELATED WORK

The behavior of manufacturing systems elements can be modeled using languages. Each word (finite sequence of events), part of a language, represents a possible behavior of an element. For instance, using $a_1$ to represent the "switch on" event and $b_1$ to represent the "switch off" event, of a specific element, some possible words are: $a_1b_1$ (representing an on-off switching cycle), $a_1b_1a_1$ (representing that the element is switched on-off and on again), and so forth. However, a more compact and feasible way to represent such languages is automata. For instance, the above language can be modeled using the automaton represented in Fig. 1.

![Figure 1. An automaton model example.](image)

Having the model of each element, one can get the complete plant behavior by calculating the synchronization Cassandras and Lafortune (2008) of all automata. This final automaton, however, represents all behaviors that the plant can present, including not desirable ones. For instance, a robot cannot pike a workpiece being processed by a machine until the machine’s job is ready.

Based on an automaton model of a plant and its behavior restrictions, Ramadge and Wonham (1987) have proposed a powerful methodology, called Supervisory Control Theory (SCT), which enables the synthesis of a controller, called supervisory, for this plant. When this supervisory is connected in a closed-loop way with the plant, only desirable behaviors of this plant can happen.

To achieve this, the SCT methodology is based on: (i) model the plant using automata ($G$); (ii) model the requirements and specifications using automata as well ($H_s$); (iii) combine these models by calculating the synchronous parallel product ($H = G || H_s$); (iv) calculate the supremal controllable language ($supC_R$), $R = supC(G, H)$, where $R$ is the supervisor automaton that when in closed-loop with the plant, guarantees the least restrictive behavior possible and deadlock and livelock free.

Following SCT strategy, the supervisory achieved guarantees a deadlock and livelock free plant behaviors. Moreover, it derives an optimal controller, in the sense that it restricts plant behavior just the necessary to achieve the specified requirements.

The supervisory, which is also an automaton, avoid undesirable behavior of a plant by disabling certain events to occur depending on which state the plant is. To do so, the current plant state is followed by the supervisory observing events generated by this plant. The complexity of supervisory synthesis, solved by SCT, is that not all events of a plant are able to be disabled. Following SCT modeling definitions, controllable events are those that can be disabled, while uncontrollable events are those that cannot be disabled.

This modeling mechanism using automata, as well as SCT methodology, is a general strategy valid for a wide range of DES systems. Nevertheless, it is very suitable when applied for manufacturing systems, in the way it proposes a clear and systematic way for modeling and controller synthesis. However, some obstacles using this strategy must be overcome in order to make it usable for manufacturing industries, which is strongly based on Programmable Logic Controllers (PLCs). The reason for this can be resumed in three main points: (i) differences between asynchronous automaton modeling compared to PLC synchronous behavior; (ii) state-space explosion, intrinsic to automata modeling and synchronization; and (iii) the manufacturing plant do not spontaneously generate events (assumed by SCT theory), but respond with events as a reaction to events generated on its input.

The first point implies that, in practice, more than one event might need to be considered at the same time (due to PLC input sampling time), which is an intrinsic characteristic of synchronous systems. This point can be addressed as in Fabian and Hellgren (1998), by verifying the interleave insensitivity of supervisors.

For the second point, there are some adaptations and extensions to the basic SCT methodology. Also, instead of having just one supervisor for the whole plant (called, in this case, a monolithic supervisor), several supervisors can be calculated, each one responsible to impose a specific requirement on the plant behavior. This is called a modular approach (Wonham and Ramadge (1988)). However, this approach still needs $G$ model to be an automaton for the whole plant. An approach called local modular (Queiroz and Cury (2000)), overcome this limitation, by allowing several supervisors and several local sub-plants, called local plants. In this approach, a local supervisor just needs to set up in closed-loop with part of the plant (a local plant), interacting with a smaller number of events, compared to the total number of plant events.

Despite all these former proposals, and other successful results by applying SCT on practical case studies (Fabian and Hellgren (1998); Hasdemir et al. (2008); Afzalian et al. (2008); Jean-Marc ROUSSEL (2005)), still the usage of SCT on the industry has a way to go (Wonham et al. (2018)). On this direction, the work presented here aims to help promote the application of SCT theory using tools well accepted in the industry, like Matlab/Simulink. By using Stateflow, a package from Matlab/Simulink, it is possible to develop the supervisors, test it in Simulink environment, and finally deploy it on an execution environment, supported by the automated code generation, directly from Simulink.

3. PROPOSED FRAMEWORK

In the following subsections, the set of tools used for the proposal here presented are described. These tools

2220

DOI: 10.17648/sbai-2019-111467
facilitate the modeling and calculation of supervisors and provide an environment that emulates a manufacturing system, enabling to set up a Hardware-in-the-loop (HIL) for a proper case study.

3.1 FlexFact

FlexFact (Moor (2019b)) is a free licensed tool developed by the Discrete Event Systems Group of Friedrich-Alexander Universität Erlangen-Nürnberg that mimics the dynamical behavior of multiple components commonly used in flexible factories layouts. The software enables the user to assemble these elements in several different positions in any number and in a manner that many singular factory layouts can be mounted. Several elements are available for simulation, like for instance conveyor belt, processing machine, rotary table, rail transport system, stack feeder, exit slide, among others.

Each element simulated using FlexFact has a set of variables that must be handled by the user in order to work properly. In other words, the simulator demands the same amount of effort in terms of connections and controllable signals from the user than the real system would. In addition, FlexFact provides a visual interface for simulation as well as an interface through a Modbus TCP/IP open protocol (Modbus Organization (2019)) to access all signals. That is, one can use a real PLC, for instance, to control a plant emulated by FlexFact, a HIL test case.

3.2 DESTool

DESTool (Moor (2019a)) is also a free licensed software developed by the Discrete Event Systems Group of Friedrich-Alexander Universität Erlangen-Nürnberg. This tool allows the user to graphically design automata, calculate supervisors and simulate these automata acting altogether. Technically, DESTool is a front-end to a set of functionalities provided in a library, called libFAUDES (Moor et al. (2008)). For the simulation, also, the system can communicate using Modbus/TCP-IP protocol, which enables the usage of a real plant, or a plant within FlexFact.

3.3 Controller Implementation

Since the goal of the work here presented is to deploy a supervisor controller on a real execution environment, like an embedded system, in this case, those points presented in Section 2 need to be tackled. An interesting approach that can be used in this case is the work presented by Queiroz and Cury (2002). It proposes an interface between the real system and the supervisors, based on a three-layer architecture (Fig. 2).

At the top layer, Modular Supervisors, are the supervisors calculated exactly as in SCT theory, that is, it can assume that events are generated spontaneously by the plant. So, the plant automata (suited at Product System layer), along with its evolution, send the events to the supervisors, which then send back the event set that is disabled on the plant.

Product System layer contains all plant automata models. These automata execute concurrently, mimicking in this way, a synchronous product. In this layer, controllable events, that are not disabled by the supervisors, are triggered. Triggering an event here means that it needs to be informed to the supervisors (upper level), and to the lower level (Operational Procedures), by sending commands. Also, the signals received by the plant, and translated by the Operational Procedures, are mapped to events (uncontrollable ones), that are evaluated in this level and sent do the upper level (supervisors).

The Operational Procedures layer is an interface between plant automata and the real system, translating the signals sent/received to/from the plant to commands/responses. This layer is closed related to the technology used to interface with the real plant. For instance, commands can be seen as sending a message to the plant through a communication bus system or using direct electrical signals connected to sensors/actuators.

The Real System block shown in the figure represents the real plant that is being controlled (like for instance a manufacturing system). For the work here presented, the Real System is set up using FlexFact and interacting with the Control System using Modbus/TCP/IP communication protocol.

3.4 Matlab/Simulink Stateflow design

Using the Stateflow package from Matlab/Simulink one can design state machines. Each transition between two states inside a state machine can have events, conditions as guards and actions. An event is a detection of a rising edge, falling edge, or both, of an input signal. A condition in a transition must be satisfied in order to fire this transaction. Upon firing, the associated action is performed. Furthermore, each state can take actions upon entering the state, while on the state, or by exiting the state. For this work, each state machine of the system resides in one single chart.

Since the goal here also follows the three-layer architecture previously presented, for the two upper layers, a set of state machines need to be designed: for supervisors and for sub-plants (called plants hereafter). Each plant state machine needs to check if a possible event is disabled by one or more related supervisors. So, during design, the user needs to impose a guard for the event, represented, by comparison, to check if one of the related supervisors do not disable this event. So, there must be a way to count how many
supervisors do not disable the related event. This can be achieved by using a Sum function block from Simulink. Upon entering the state, a variable is set to one, signaling that the event is not disabled.

3.5 Controller Deployment

To deploy the controller on a real execution environment, in this case, an Embedded System, the Embedded Coder Target was used. In Matlab/Simulink this package enables the automated code generation for a chosen target runtime execution environment. This can be done from a certain number of modeling languages and, in addition, it needs to be in accordance with certain guidelines. Stateflow is one of the modeling languages supported by Embedded Coder Target, nevertheless, the model needs to be set up as a discrete system, with fixed step time. These are the requirements for the Embedded Coder Target package.

The code generated of the system is organized in one function (which receives the name <system_name>_step()), and some extra functions for initialization, setup, and finalization. If no operating system is defined, the user is responsible to design an extra code in order to call this function in a periodic way, calling also previously the initialization and setup functions. The time interval between each call needs to be equal to the step time fixed in advance for the model.

4. EXPERIMENTAL RESULTS AND ANALYSIS

In order to evaluate the proposal environment, a case study was conducted. Fig. 3 shows the overall testbed setup. The microcontroller-based embedded system used was an Arduino Uno Platform (Arduino (2019)) along with an Ethernet Shield to enable the Modbus TCP/IP communication protocol.

4.1 Case Study Description

The case study consists of one Stack Feeder, two Conveyor Belt and one Exit Slide (Fig. 4). The Stack Feeder has the main function of feeding the production line with workpieces, previously arranged in a stack. It is composed basically by three components: motor, workpiece sensor, and feeder sensor. The motor is responsible for pushing the feeder which is in direct contact with the workpiece. Once the workpiece leaves the Stack Feeder and goes to the next assembling step, the sensor, that was active because of the workpiece presence, turns off.

The Conveyor Belt, in the sequence, is responsible to move a workpiece through the manufacturing assembling line. It has the function of transporting the workpiece from one machine of the production line to another. The Conveyor Belt is constituted by two basic components: motor (coupled to the belt) and a sensor that indicates when a workpiece passes through the machine. For this case study, there are two conveyor belts.

The signals and events related to the Stack Feeder can be listed as: SF_FDM signal that controls when the motor is on or off; sf_fdon controllable event that switches the motor on; sf_fdooff controllable event that switches the motor off; SF_PDS sensor signal that is on when the pusher is on home position; sf_fhhome uncontrollable event that indicates when the SF_PDS sensor turns on (pusher was set in home position).

The signals and events related to the first Conveyor Belt can be listed as: CB1_BM signal that controls when the motor is on or off; cb1_bm controllable event that switches the motor on; cb1_boff uncontrollable event that switches the motor off; CB1_WPS sensor signal that indicates when there is a workpiece present or not on the belt (capable to sensitize the sensor); cb1_wpar uncontrollable event that indicates when a workpiece arrives on the belt; cb1_wplv uncontrollable event that indicates when a workpiece leaves the belt. For the second Conveyor Belt, the signals and events are essentially the same, and changes only by the index number, in order to distinguish between two conveyors. For instance, CB2_BM, cb2_bm and cb2_off are signal and events, related to the second conveyor belt motor, and CB2_WPS, cb2_wpar and cb2_wplv are signal and events associated with the second conveyor belt sensor.

Finally, the Exit Slide is a simple machine that serves only for accumulating workpieces exiting the production line. It is composed of one sensor that indicates when a workpiece arrives at the machine and slides to the bottom.

The signals and events related to the Exit Slide can be listed as: XS_WPS sensor signal positioned at the beginning of the Exit Slide, indicating when there is a workpiece on the sensor; xs_wpar uncontrollable event indicating that a workpiece arrives in the Exit Slide; xs_wplv uncontrollable event that indicates when a workpiece completely passes through the sensor.
Automata models for each of these components (plants) are shown in Fig. 5. A ticked arc of an automaton indicates that the related event is controllable.

Figure 5. Automaton Model for: (a) Exit Slide, (b) Stack Feeder, (c) Conveyor Belt 2 and (d) Conveyor Belt 1.

4.2 Requirements and specifications

In order to work in a desired way, the following four specifications (working requirements) have been defined:

1. Conveyor Belt 1 must switch on the belt when a workpiece leaves the Stack Feeder;
2. Conveyor Belt 2 must switch on the belt when a workpiece leaves Conveyor Belt 1;
3. Conveyor Belt 1 must switch off the belt when a workpiece arrives Conveyor Belt 2;
4. Conveyor Belt 2 must switch off the belt when a workpiece arrives Exit Slide;

4.3 Local Modular Supervisors

Using the specifications and plants automata, supervisors are calculated following the Local Modular approach, as described in Section 2. So, four supervisors are obtained, shown in Fig. 6.

4.4 Matlab/Simulink Stateflow models

Following the framework proposed in Section 3, each supervisor obtained in the previous subsection is modeled using state machines using Stateflow. As an example, Fig. 7 shows how Supervisor 1 was modeled. Following the proposed guidelines, variables inside a state indicate when the event is disabled or not. In this case, while in state 2, cb1_bm event is enabled and sf_fdon not. If cb1_bm event is triggered by the plant, it is also executed here, represented by the guarded transition from state 2 to state 1. Supervisor 2, which also might disable this event, also must have this kind of variable setting.
Therefore, Plant 2 (Conveyor Belt 1) must check if \( cb1\_bm \) event remains enabled by both supervisors. This is represented in the transition guard of the transition from state1 to state2 (Fig. 8), where \( H\_cb1\_bm \) will have value 2 only if both supervisors are not disabling this event (\( cb1\_bm \)). If this transition occurs, then an action is performed: \( A\_cb1\_bm = 1 \). That is, the variable \( A\_cb1\_bm \) is set to one, which is then read by the Operational Procedures layer in order to perform the effective action (switch on the motor from Conveyor Belt 1) and to inform to the supervisor that this event happened. The Operational Procedures layer undergo the switching action by sending a Modbus message to the manufacturing system (set on FlexFact).

The whole system is modeled in Stateflow by following this strategy (described in Section 3). Fig. 9 shows the complete model of the system, which is seen as an interaction between charts (state machine of each automaton). The transition entries on the top of each chart represent the uncontrollable events that state machines can recognize (by the rising edge of these entries). The lines for these entries are skipped in the figure for readability reasons.

Before going to the deployment step, since the complete controller are modeled in Stateflow, one can make simulations, including an extra part in the model for mimicking the reactive behavior of the plant. By doing so, it enables the user to perform a complete system simulation and some wrong behaviors or possible mistakes made during the translation of automata to state machines in Statellow can be identified and corrected in this step.

### 4.5 System Deployment

The system is deployed in a runtime execution platform described at the beginning of the current section. Therefore, the fixed step size was set to 10ms and \texttt{arduino\_ec.tlc} (a script to generate Arduino embedded code from a Simulink model) was used for generating the embedded C++ code for the target environment. This parameter can be changed depending on which target execution platform is chosen.

Bringing the system to execution, interacting with the plant (residing in the FlexFact tool) via Modbus messages, the controller and the plant presented the same behavior compared to the case when tested using Destool and Flexfact. Thus, for this case study, it could be shown that the concept here presented, achieved the proposed goals.

### 5. CONCLUSION

This paper presented a Matlab/Simulink Stateflow based concept to model the local supervisors and plant, following the three-layer approach from Queiroz and Cury (2002), which was successfully achieved. It includes the possibility to simulate in Simulink the supervisory controllers with also the possibility to deploy these controllers by using the automated code generation from Simulink. All this together provides a system for fast prototyping, avoiding the error-prone phase of manual coding. In addition, the target execution platform is easy to be changed by simply setting it properly at Simulink parameters.
The step of translating the supervisors and automata to the Stateflow charts are manual and still error-prone. This problem can be minimized by using the possibility of doing simulations on Simulink before deploying the system. However, a proper solution would be the development of a translation tool, from a specific modeling format to Stateflow format, observing with it the guidelines defined in this work. Another solution would be to develop a proper Matlab add-on package, to include the SCT methodology to be executed inside Matlab/Simulink.

REFERENCES


