An Educational Toolbox on Supervisory Control Theory using MATLAB Simulink Stateflow

From Theory to Practice in one week

Claudius Jordan, Canlong Ma, Julien Provost
Technical University of Munich
Safe Embedded Systems
Garching bei München, Germany
ma@ses.mw.tum.de, provost@ses.mw.tum.de

***Abstract*— Since the Supervisory Control Theory (SCT) was introduced by Ramadge and Wonham in 1987, many researchers contributed to its theoretical basis and to the development of algorithms. In academia, several self-contained tools are used to model systems and analyze their behavior. Despite that, there are still too few application tools of SCT in industry. Therefore, we developed the *MATLAB Supervisory Controller Implementation* toolbox (MSCI) for applying supervisory control theory, in order to have a first step in bridging the gap. It is based on MATLAB Simulink and Stateflow since MATLAB is a widespread and accepted environment in the industry. Results from a student practical lab prove that it is possible to teach SCT in a short time by using this toolbox.**

Keywords*— Discrete Event System (DES), Supervisory Control Theory (SCT), MATLAB, Simulink, Stateflow

I. Introduction

Supervisory Control Theory (SCT) is a method in the field of Discrete Event Systems (DES) for automatically synthesizing supervisors that restrict the behavior of a plant [1]. The greatest feature is that SCT bases itself on the safety point of view, i.e. *what should the plant not do*, and it differs from the traditional liveness perspective, i.e. *what should the plant do*.

Since first introduced by Ramadge and Wonham in 1987 [1], many researchers have contributed to the theoretical development of SCT. Based on this pioneering work, an *input/output perspective* was introduced in [2]. According to it, "a plant receives *commands* and reacts to these commands with *responses*." As the main obstacle to the broad application of SCT in industry, the discrepancy between the abstract supervisor and its physical implementation has been identified in [3]. Especially, the gap between the event-based asynchronous automata world and the synchronous signal-based systems was put into focus. We refer readers to [4] for a recommended outline of the state of the art in SCT.

Despite a significant number of contributions to the theoretical field, "the number of applications of SCT in the industry [is] still far from what most of the discrete event community expected fifteen years ago" [5]. That was in 2005. This statement is still valid now. Moreover, three key features that have been identified to be a major source of difficulty in implementing a controller were listed in [5]. Recent research shows that the computational complexity can be reduced and the so-called *supervisor* can be calculated more efficiently [6]. But still, the problem of modeling is critical and has not yet been overcome.

Various research groups are working on different approaches to bridge the gap between theory and implementation of discrete-event controllers and SCT [7, 8]. A summary of recent results is also presented in [9]. Most of these works consider the implementation on Programmable Logic Controllers (PLCs).

There exist also several stand-alone tools that provide graphical interfaces to model systems as automata and analyze the resulting *supervisor*, which can be generated automatically. Some go further and enable automatic PLC code generation. A non-exhaustive selection of such tools is listed here: DESUMA [10], IDES [11], Grail for Supervisory Control [12], Supremica [13], TCT [14], Nadzoru [15], and DEStool [16].

All those tools address the improvement of user-acceptance to help SCT spread outside the academic field in order to make it easier to model systems and analyze their behavior. For teaching purposes, as an example, Supremica [13, 17] is well suited for students to experience the straightforwardness of modeling systems.

However, if a user wants to apply SCT on a real example, there is currently, to the best of our knowledge, no tool to cover the modeling using an industrial standard language and control of an existing system altogether. This paper presents *MATLAB Supervisory Controller Implementation* (MSCI), a toolbox based on MATLAB Simulink and MATLAB Stateflow. The aim of this toolbox is to diminish the obstacle in applying SCT in industry.

The paper is structured as follows: Section II provides background knowledge. The functionality and illustration from users' perspective of the toolbox are presented in Section III and IV, respectively. In Section V and VI, a demonstration case study of the toolbox and the result of a student practical lab are presented. Finally, a discussion of this work is given in the last section.

II. Background

A. Supervisory Control Theory

In contrast to continuous state systems, DES are systems characterized by discrete states. A DES evolves at discrete time steps through the occurrence of asynchronous discrete events and thereby, changes its state. The behavior of the physical system is denominated *plant*. Generally speaking, a plant model contains possible physical actions and behavior of the system.

In order to control a plant, *specifications*, which are a set of formalized requirements, are used to limit the set of the possible plant's actions. To meet the constraints defined in the specifications, a control unit is used, which contains a so-called *supervisor* constructed automatically from both the plant model and the specifications. The interpretation of the system, *plant*, and control unit, *supervisor*, as a closed-loop system is called "*input/output perspective*" [2], displayed in Fig. 1.

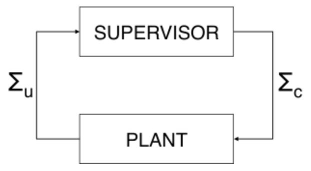

Fig. 1. Closed-loop system containing supervisor and plant

Systems, such as production systems, contain sensors and actuators. Sensor values are inputs for the supervisor; they are imposed by the plant and represented as *uncontrollable events*, i.e. Σ_u. On the other hand, actuator signals are outputs for the supervisor; they are imposed by the supervisor and represented as *controllable events*, i.e. Σ_c. Only events that relate to the input or output should be used to define the model of the plant [5].

In our case, binary sensors are used, whose statuses are represented as either *True* or *False*. Actuators are also treated as binary components that are either *On* or *Off*. This information can be modeled as Boolean values representing the electrical signals. In each cycle, the input signals are read and the output signals are set. However, SCT is based on events that occur asynchronously at discrete time instances [1]. Thus, events are derived by detecting rising and falling edges of those electrical signals (Fig. 2). On the other hand, actuator signals are created depending on events that correspond to those signals.

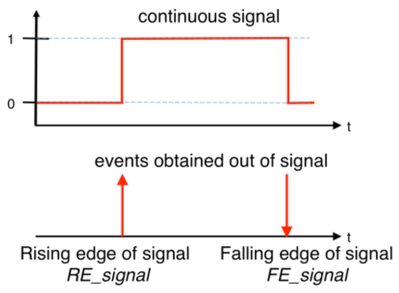

Fig. 2. Conversion of signals to events

B. *Representation of discrete-event systems as automata*

Deterministic Finite Automata (DFA) can be used to describe DES formally. An automaton is a device that is capable of representing a language according to well-defined rules [18].

A DFA is a five-tuple G = $\langle Q, \Sigma, \delta, q_0, Q_m \rangle$ where:

- Q is a finite set of states
- Σ is a set of event-labels, the *alphabet*
- $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function
- $q_0 \in Q$ is the initial state
- $Q_m \subseteq Q$ is a set of marked states

The simplest way to visualize an automaton is a directed graph. In this paper, MATLAB Stateflow is used to create such graphs that contain states and transitions with distinguishing labels. Afterwards, they are translated to formal DFA. Then, the *Discrete Event Control Kit* (DECK) [19] is used (more details about DECK are presented in Sec. III-G). For detailed information about MATLAB Stateflow, please refer to the official MathWorks® documentations.

III. *MATLAB Supervisory Controller Implementation* Toolbox

In order to provide a deeper understanding and more practical experience for students, we started the development of the *MATLAB Supervisory Controller Implementation* toolbox (MSCI). We chose the development environment of MATLAB because it is a well-known and widely used tool both in academia and industry. The long-term goal is to provide a toolbox for educational purposes to help augment the acceptance of SCT, as well as to provide executable use-cases for industry partners.

MSCI facilitates the configuration for lecturers so that only a few parts need to be customized for their specific requirements, i.e. communication with the physical system to be controlled. MSCI provides a quick learning curve, reducing the time to understand the concept of SCT-based control. The 'only' need for students is to model the system as automata in Stateflow. Everything else, i.e. conversion to formal language and SCT algorithms, is taken care of by MSCI.

A. *Overview*

An overview of the workflow with MSCI is given in Fig. 3. The physical system is taken as the input and modeled in Stateflow while a database needs to be created in order to obtain the executable Simulink model. The creation process takes the Stateflow models as a basis and compares them with the information specified in the database to verify syntax correctness. The executable model needs the information from the database for the communication with the system because the events handled within the control unit are mapped to electrical signals of the system.

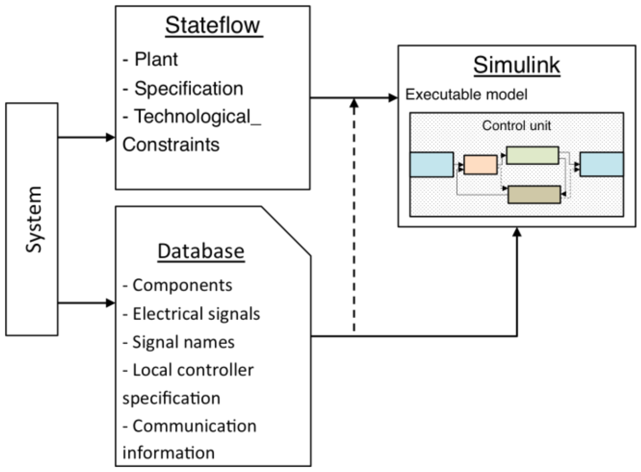

Fig. 3. Workflow with MSCI

B. *Closed-loop system*

The executable Simulink model is displayed in Fig. 4 with its components. The inputs are the system's sensor signals that are converted into events by the 'Signal-to-Event' block and then handed over to the 'Event-queuer' block, which queues the events in case there are more than one event in one cycle and buffers them such that the occurrence information does not get lost. Consequently, every event is processed separately afterward. The supervisor is the centerpiece of the control unit that manages the behavior of the system by outputting events that are converted to actuator signals by the 'Event-to-Signal' block. Depending on the setup, some events are directly passed to the 'Technological constraints' block that contains the virtual sensors and local controllers, which might directly command actuators without looping back to the supervisor.

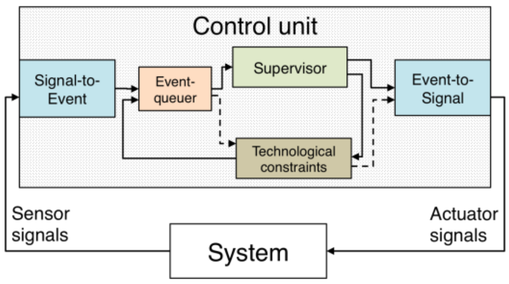

Fig. 4. Executable model on control unit in closed-loop

C. *System specific database*

The toolbox requires an underlying database that contains all the information about the available electrical signals in the plant, whose names need to be specified. Only events based on those names can be used for modeling. An exception is the events for virtual sensors that do not need to be specified in advance. Additionally to electrical signals, events for local controllers need to be predefined in the database.

The communication between the supervisor and the plant must be taken into account to decide what is needed for the toolbox to read and write data. As described in Sec. V-C, our platform uses Modbus TCP [20], but other protocols can also be implemented.

D. *Graphical user interface*

The modeling takes place within the environment of MATLAB Stateflow. Three different superstates are predefined, namely 'Plant', 'Specification', and 'Technological_Constraints'. They represent different parts of the model indicated by their names. Within the 'Plant' superstate, all automata representing the possible behavior of the system should be collected. The 'Specification' superstate is the trunk for all specifications that restrict the behavior of the plant. Everything else, such as *virtual sensors* or *local controllers*, should be located in the 'Technological_Constraints' superstate. The superstates with some exemplary automata are displayed in Fig. 5.

E. *Further modeling concepts: Technological_Constraints*

In addition to the plant model and the specifications, further modeling concepts are available providing more freedom and flexibility in modeling such as *virtual sensors*, *local controllers*, and *timing*.

1) Virtual Sensor: The plant's sensors perceive actual changes in the system, their signals are transformed into events,

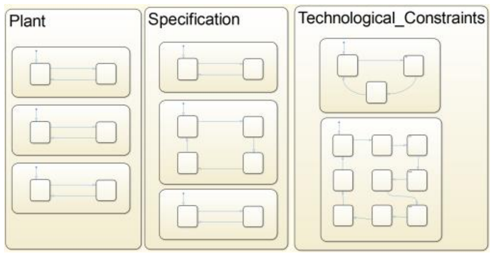

Fig. 5. Top-level superstates of the modeling environment

and finally the supervisor processes that information. The same holds for virtual sensors. They are used to inform the supervisor about events that are specified internally, which means that it not necessarily has a physical sensor but can be calculated based on other sensors value and timing. For example, in Sec. V-D a virtual sensor based on timing is implemented; it 'detects' a workpiece at a position, where there is no sensor located.

2) Timing in automata: Although the supervisor created using SCT is purely reactive and only prevents controllable events from occurring; it is possible to allow timing in certain parts of the model. This is useful as it might be necessary to let an action last for a specific period. Note that only in the models for virtual sensors and local controllers timing is allowed, whereas in the plant model and the specifications it is explicitly forbidden.

The time information is kept in so-called *timing transitions* that use the label *after (T, sec)*, where *T* is a positive value indicating the time in seconds to wait until the transition is fired.

Allowing the use of timing only in the 'Technological_Constraints'-superstate permits to handle many timed systems (where time is limited to a subpart of the whole systems) without extra cost to introduce the concept of time in SCT framework, which would lead to *state-space explosion*.

3) Local Controller: Controllable events allowed to occur by the supervisor represent physical actions. Some subsystems can do more than just one specific action; for them, the notion of local controllers is introduced. The supervisor calls them with a 'start' event, and when they have finished their process, they respond with a 'done' event. By doing so, hierarchical decomposition is possible. For example, in Sec. V-D a local controller is implemented to command a machine and another to control the movement of a pusher.

F. *Conversion to formal representation*

To successfully create a supervisor, the Stateflow model is checked for syntax and interpreted according to the guidelines given in Sec. IV-B. MSCI compares the labels of the Stateflow objects and verifies if they are used correctly. Additionally, a user-defined database is used to check whether every event is mapped to its signal counterpart. Afterward, the graphical model is transformed into its formal representation, and the supervisor is obtained by the *Discrete Event Control Kit* (DECK) that is introduced in the following section.

G. *Discrete Event Control Kit*

The MATLAB implemented *Discrete Event Control Kit* (DECK) was developed for analysis and internal processing of DES models [19]. Therefore, it is suitable for the MSCI toolbox.

The models are transformed from their graphical representation in Stateflow to the formal representation introduced in Sec. II-B. Then, DECK creates the supervisor based on the user's models, which is finally implemented into the Simulink model that embodies the control unit.

IV. User's perspective

In this section, the toolbox is presented in terms of a step-by-step description of processes and interactions with users. In this context, *processes* are the internal MSCI *procedures* as described in Sec. III-F and Sec. IV-C, and *interactions* are the actual operations of users. We suppose that the students/users should only be concerned about the modeling process and therefore be aware of the guidelines presented in Sec. IV-B. The preparation of the teaching platform, which is outlined in Sec. IV-A, is taken care of by the lecturer or experienced operator. We discuss the assumptions and constraints of the system for which we developed the toolbox in Sec. V, where further details are given on the connection of the Simulink model to the physical system. Also, we briefly present what needs to be customized in order to use the toolbox for other systems in Sec. IV-D.

A. *Preparation*

Before we can start to model and control any system, interfaces need to be defined and a way to communicate set up. Furthermore, a database of all the signals and their names must be prepared to serve as the basis for MSCI processes to ensure concordance of the user's model; the latter is compulsory for converting into formal language. Events for local controllers should be specified at the appropriate place before applying the conversion routine.

B. *Modeling guidelines*

Modeling is the key process, and this is the one step the student user should put all his/her effort in. MSCI provides the *CreateChart* function that creates a Simulink model with a Stateflow chart in it, which contains the three predefined superstates described in Sec. III-D. In order to convert the Stateflow chart properly into formal language, the user should respect some rules regarding syntax.

1) States: There are three different levels of states as mentioned in Sec. III-D. Within those top-level superstates, automata are modeled: each represents a part of the plant, or a specification, or a technological constraint. Those automata are Stateflow states containing states, and therefore, are also denominated superstates. We will refer to them simply as automata to distinguish between the two types of superstates and thus highlight the function of the latter. Each of them contains only one automaton and the states within one automaton are called substates. It is important to have unique names for each automaton. However, the substates' names can be arbitrary but must be unique within a single automaton. All the automata aggregated within the 'Plant' superstate model the behavior of the plant, whereas the 'Specification' superstate contains all the specifications. The automata in the 'Technological_Constraints' superstate need to have a prefix, either 'VS_' or 'LC_', to distinguish between each other.

2) Transitions: The rules for transition labels are derived from the syntax of transition action types in Stateflow. Two types of events, i.e. controllable and uncontrollable events, are distinguished. Transitions that represent uncontrollable events are labeled with the events' names only. In Stateflow, this notation is used for 'event triggers', which means the transition is fired when the event occurs. Transitions for controllable events need curly brackets around the events' names that represent 'condition_actions' in Stateflow. So when a transition is fired, its event is commanded. The difference can be seen in Fig. 6.

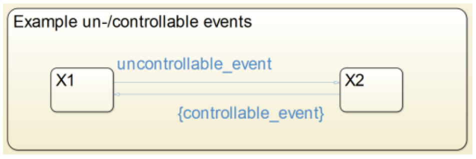

Fig. 6. Difference between uncontrollable and controllable events

In the database, all the physically represented signals need to be defined. As we utilize event-based algorithms, rising and falling edges of signals are distinguished. In order to do so, prefixes are added to the signal names. To indicate a rising edge, the prefix 'RE_' is added, while 'FE_' stands for falling edges. Only names that are defined in the database should be used. There is only one exception to this rule: events that belong to virtual sensors follow a specific syntax. Thereby, they are identified automatically without defining them in advance. Events of virtual sensors are not linked to signals and thus do not need to be specified in advance: the prefix 'vs_' identifies this type of events.

Timing transitions: It is possible to use *after (T, sec)* as transition label in models for virtual sensors and local controllers. An example is given in Fig. 7. This type of transition is only allowed within the 'Technological_Constraints' part and forbidden in the other parts as outlined in Sec. III-E2.

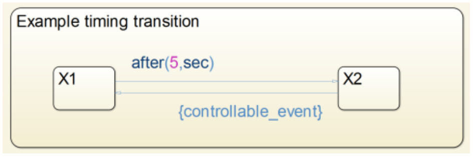

Fig. 7. Example for timing transition

3) General remarks: It is mandatory to abide the syntax that is presented here. The conversion process and its verification subroutines expect those specific prefixes. An error is reported if some unexpected label is found.

All above is the basis for users to start modeling. The advantages of those rules are the guidance for the user and the freedom that remains to model a system. The compulsory predefinition of signals supports the structured modeling process and ensures that any event used has its signal counterpart and vice versa.

C. *Executable Model*

Once the modeling process is finished, the user calls the *CreateModel* function, which creates a Simulink model that embodies the controller. A supervisor is obtained on the basis of the

Stateflow model as discussed in Sec. III-F, which is then integrated into the Simulink model that manages the input and output of signals, conversion from signals to events and vice versa, the queuing of events that occur in the same cycle, and most importantly the supervisory control. Together with the technological constraints, the control is applied using SCT.

D. *Customization*

Besides setting up a system that is capable of communicating with the control unit in a closed-loop way, the lecturer needs to set up a database mapping signals and their names to their counterpart. In addition, the information for the control unit how to read and write signals needs to be provided. The chosen communication protocol needs to be implemented in MATLAB, and the path to the database, which can be a Comma-Separated Values (CSV) file, should be adjusted. A more compliant way to customize the communication and database is under development.

V. Case Study

The MSCI toolbox has been applied to a didactic platform as a case study.

A. *Didactic platform*

The hardware platform presented in Fig. 8 is supplied by Fischertechnik®. The biggest advantage of this type of hardware is to enable students to observe the consequences of their models at once without worrying about any physical damage they may cause to the plant or to themselves. The system is quite complex with regard to number of inputs, outputs, and subsystems that can be executed in parallel (ca. 300 IOs and 12 subsystems).

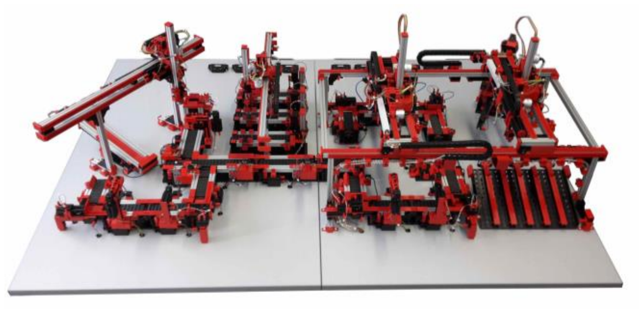

Fig. 8. Overview over the didactic platform at the SES department

B. *Hypotheses*

We propose the following hypotheses for the closed-loop system:

H1: The plant is much slower than the controller, i.e. the controller is fast enough to ensure a correct reaction time for the plant under control.

H2: An event can only occur once during a sampling time.

H3: The order of events that occur within one cycle is not important.

H4: The control unit processes events before their next occurrence.

H5: The supervisor on the controller evolves synchronously with the plant.

C. *Communication*

For any system, signal interfaces need to be defined, and the communication between plant and controller needs to be established. Various possibilities are available to fulfill this task. Here, Modbus TCP (using Java Modbus [20] as an implementation) is used to communicate between the user's application and the plant's control module. A drawback of this approach is that it cannot satisfy hard real-time constraints. Moreover, exact synchronization is not achievable due to time delays. For the designated purpose and with the assumptions declared above, the cycle time is small enough for the characteristics of our system.

Unique IP addresses are assigned to the *Remote Input Output Modules* (RIOM), and the coils on the modules can be unambiguously identified. As long as there are no contradictory assignments of outputs imposed by different control units, multiple users can have access to them.

D. *Illustrative example*

The model size grows exponentially with the number of inputs, outputs, and subsystems, which makes it inconvenient to fully present a large scale system like the didactic platform shown in Fig. 8. In the following, one part of a subsystem, *Indexedline1* (IL1), is displayed as an example in Fig. 9. The input and output signals for the control unit are displayed in Fig. 10. They are specified in the underlying database.

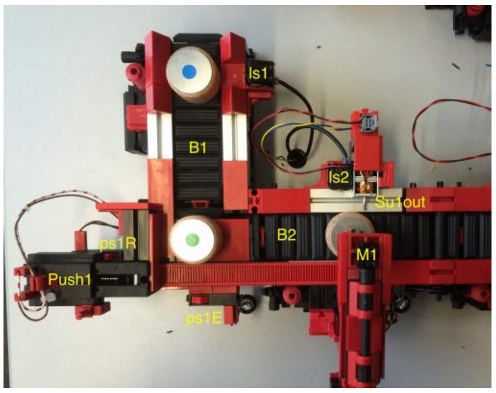

Fig. 9. Indexedline1

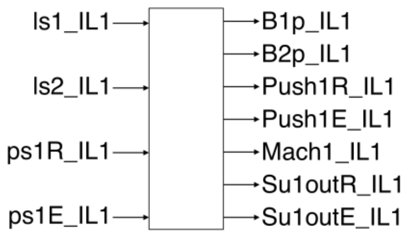

Fig. 10. Input and output signals of the control unit

The input and output events that are taken into account by the supervisor are displayed in Fig. 11. The external events are obtained from the signal names given in Fig. 10 by adding the prefixes 'RE_' and 'FE_' as outlined in Sec. IV-B. The internal events are defined by the user and need to be specified in the database as discussed in Sec. III-C. The events used by local controllers are obtained in the same way as external events in the supervisor.

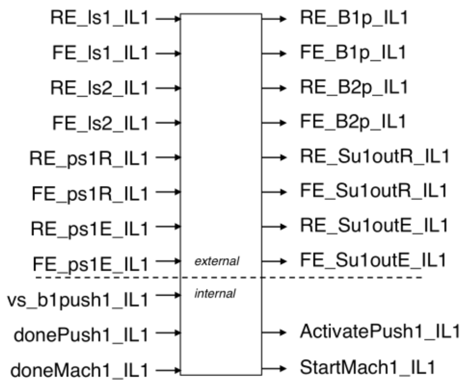

Fig. 11. Input and output events of the supervisor

1) Plant: The subsystem presented here is the first half of the so-called ‘Indexedline1’. It contains two conveyor belts (B1 and B2), one machine (Mach1), two light switch sensors (ls1 and ls2), one pusher subsystem (Push1) and one bi-stable separating unit (Su1out).

The pusher subsystem contains a two-way motor with the two signals *Push1R_IL1* and *Push1E_IL1* and two end position detectors (ps1R and ps1E). The separating unit is also referred to as a stopper and provides two signals, *Su1outR_IL1* for its retraction and *Su1outE_IL1* for its extension. For the sake of simplicity, the conveyor belts can only turn in their positive directions, i.e. they transport the workpiece forward. The corresponding signals are *B1p_IL1* and *B2p_IL1*. The machine is controlled by the signal *StartMach1_IL1*. The light switch sensors provide the signals *ls1_IL1* and *ls2_IL1*. The pusher can move forward and backward and the subsystem contains two end position sensors with signals *ps1R_IL1* for the retracted and *ps1E_IL1* for the extended position. The supervisor does only care if the pusher is currently ‘idle’. Thus, it can be activated by the event *ActivatePush1_IL1*, whereas the actual movement procedure is defined in a local controller. The models for B1 and Push1 are given in Fig. 12 and Fig. 13, respectively.

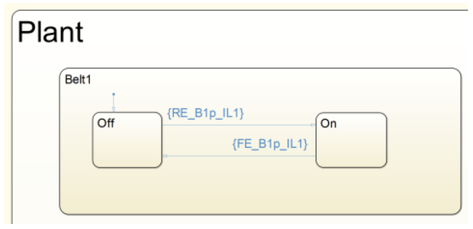

Fig. 12. Plant model of B1

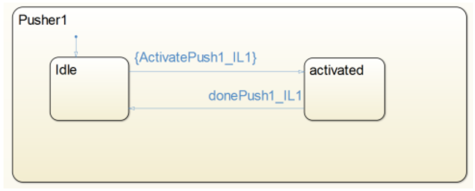

Fig. 13. Plant model of Push1

2) Specification:

S1: The conveyor belt 1 should start after a workpiece is detected by light switch sensor 1.

S2: The conveyor belt 1 should stop when the virtual sensor detects a workpiece.

S3: The pusher should be activated when the virtual sensor detects a workpiece.

S4: The conveyor belt 2 should start when the pusher pushes a workpiece to it.

S5: The conveyor belt 2 should stop when light switch sensor 2 detects a workpiece.

S6: The machine 1 should start when conveyor belt 2 is switched off and the stopper is extended. A virtual sensor can detect if that condition is satisfied.

S7: The separating unit should retract after the machine has finished.

S8: The belt 2 should start after the machine has finished.

S9: The separating unit should extend when the pusher is activated.

S10: The conveyor belt 2 should stop after the workpiece has left the subsystem. A virtual sensor can detect that.

More specifications can be used to enable the transportation of workpieces through the system. The ones listed here indicate the straightforwardness of the SCT approach and the Specification S1 is shown as an example in Fig. 14.

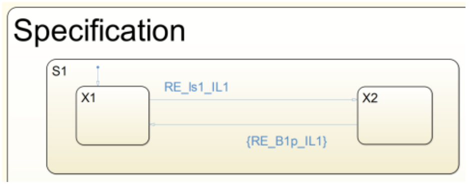

Fig. 14. Model of S1

S2, S3, and S9 are all triggered by the activation of the pusher represented by the virtual sensor event *vs_b1push1*, which thereby can be modeled together or separately. Here, S2 and S3 are combined in ‘S2_S3’, whereas S9 is kept separate as depicted in Fig. 15.

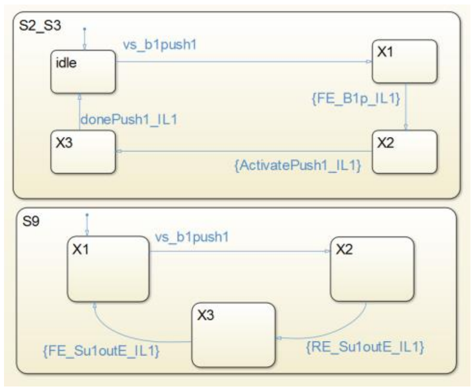

Fig. 15. Model of S2 and S3 combined, and S9 separately

S5 and S10 both specify situations when B2 should stop. Therefore, they are modeled together as ‘S5_S10’, which is shown in Fig. 16. Similarly, S4 and S8 are merged together because otherwise the supervisor would prevent the controllable event until both conditions have been satisfied, which is not desired.

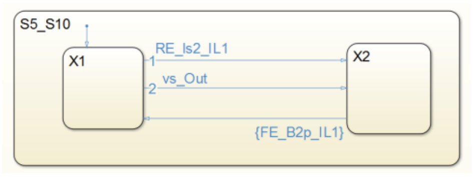

Fig. 16. Model of S5 together with S10

3) Technological Constraints: For the pusher, there is no sensor that detects if the workpiece, which should be transported to it via the conveyor belt 1, has already arrived. In such a case, a virtual sensor can be used. For example, it can be assumed that after 1.5 seconds the workpiece should have reached the desired position and after that period the event vs_b1push1 can occur. The model for this virtual sensor is given in Fig. 17.

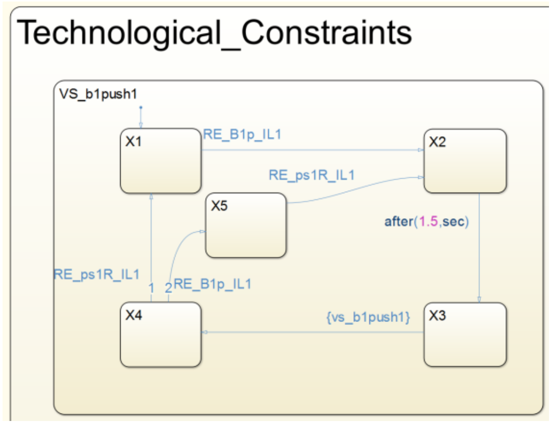

Fig. 17. Virtual sensor model

Mach1 is a subsystem that is only started by the supervisor commanding the event *StartMach1_IL1*. The local controller of Mach1 then starts a desired sequence as depicted in Fig. 18. When it is finished, it produces the event *doneMach1_IL1*.

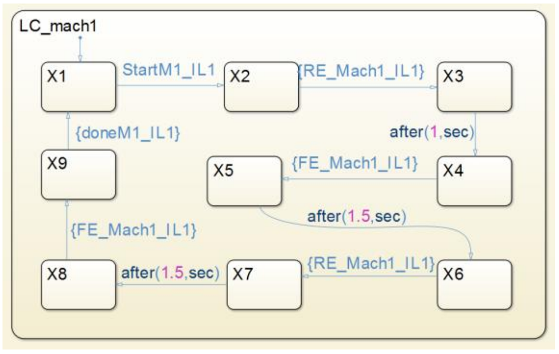

Fig. 18. Local controller model of Mach1

As mentioned above, the pusher is also modeled as a local controller and its model is given in Fig. 19.

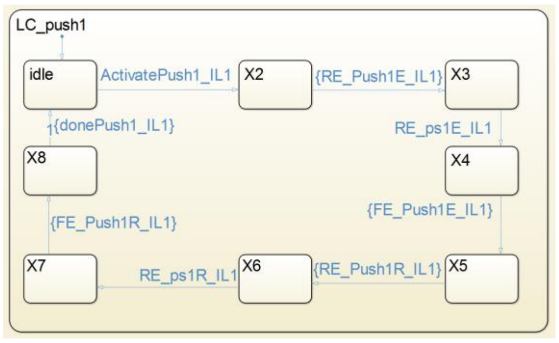

Fig. 19. Local controller model of Push1

VI. Practical Lab Results

In October 2016, a one-week practical lab using MSCI and a didactic platform was carried out as an elective subject for master students in the department of mechanical engineering.

Students from various study programs (e.g. mechanical engineering, mechatronics, aerospace and power engineering) took part in this lab, two-thirds of which had no previous knowledge of SCT.

The students formed groups of 2 persons. The task of each group was to model and control a subsystem of the platform in Fig. 8. They were expected to understand the basic idea of SCT and use it to control a real platform.

A. Global schedule

The lab takes 40 hours (plus an optional introduction lecture with 4 hours).

On the first day, the students learn the basic theory knowledge of SCT, in which a few simple examples of plant model and specifications were given.

Then, the students spent four days designing their own specifications, formalizing them into Stateflow models, checking and modifying the models connecting to the platform.

B. Performance evaluation

In the end, the result of each group was evaluated according to a demonstration on the platform, the Stateflow models, a written report, and an oral exam.

The evaluation shows that all students successfully obtained one or several valid functional models and they grasped the fundamental idea of SCT through only one week of practice.

Therefore, all students passed the lab assessment, which conforms to lecturer’s expectation.

C. Student feedback

All students were requested to give their feedback about the lab. Following are some excerpts:

- *“We had an intense and productive week, and we achieved the main goals of this lab while enjoying.”*
- *“The fact of having a real model to try our system has motivated us to try to get the best results.”*

- *“We furnished a quite good work with good state flows because we use virtual sensors and local controllers in a good way.”*
- *“Implementing a counter of the current position or the number of released workpieces in an outflow would have been very helpful. However, the event-based approach makes this implementation rather complicated.”*
- *“A disadvantage of SCT is: a centralized approach to create a supervisor (by brute force) cannot overcome a state space explosion scenario in a complex process.”*
- *“SCT is suitable for controlling equipment with a large number of freedom, i.e. different tasks are done by different actuators. But if several tasks have to be done by only one actuator, a lot specification is needed.”*

VII. Conclusion

This paper presents the MSCI toolbox and its current capabilities. The lecturer needs to set up the communication with the system and prepare a database that allows the input signal from sensors and control signals to actuators. Students can then start to model the system based on that predefined database. The actual control of the plant is then performed by the toolbox.

The central benefit is the incorporation into the MATLAB environment, which makes it easy for students to start modeling a system in an environment they already know. Almost immediately the students get feedback on their models by looking at the plant’s behavior and are therefore granted deeper knowledge of the impact of changes and the way they model their system. In that way, students can rapidly be given a broaden experience with SCT in an almost playful manner, applying their knowledge not only to a pure software simulator interface.

The first application of this toolbox in a student practical lab shows the success of teaching SCT in a short time.

Outlook

The development of this toolbox is still in progress and functions will be added to the toolbox in order to improve its usability and further increase the value for users.

The modeling process often requires many iterations driven by trial and error because the user is sometimes not aware of the full impact of his specifications, which sometimes lead to too restricted behavior. A ‘simulator’ can be implemented that imposes events, which are triggered by the user, on the Stateflow model and displays its evolution.

At the moment, a cycle time of 80 ms is achieved that can be reduced by optimization of the implementation. Additionally, the modeling boundaries, e.g. maximal number of states or maximal size of the model, will be explored.

References

[1] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *Analysis and Optimization of Systems,* vol. 63, pp. 475--498, 1984.

[2] S. Balemi, "Control of Discrete Event Systems: Theory and Application," 1992.

[3] M. Fabian and A. Hellgren, "PLC-based Implementation of Supervisory Control for Discrete Event Systems," *37th IEEE Conference on Decision and Control (Cat. No.98CH36171),* vol. 3, pp. 3305--3310, 1998.

[4] W. M. Wonham, "Supervisory Control of Discrete-Event Systems," 2016.

[5] J.-M. Roussel and G. Alessandro, "Designing Dependable Logic Controllers Using the Supervisory Control Theory," *Proceedings of the 16th IFAC World Congress,* 2005.

[6] S. Miremadi and B. Lennartson, "Symbolic On-the-Fly Synthesis in Supervisory Control Theory," *IEEE Transactions on Control Systems Technology,* vol. PP, 2016.

[7] F. Basile, P. Chiacchio, and D. Gerbasio, "On the Implementation of Industrial Automation Systems Based on PLC," *IEEE Transactions on Automation Science and Engineering,* vol. 10, pp. 990--1003, 2013.

[8] M. V. Moreira and J. C. Basilio, "Bridging the gap between design and implementation of discrete-event controllers," *IEEE Transactions on Automation Science and Engineering,* vol. 11, pp. 48--65, 2014.

[9] R. J. Leduc, Y. Wang, and F. Ahmed, "Sampled-data supervisory control," *Discrete Event Dynamic Systems: Theory and Applications,* vol. 24, pp. 541--579, 2014.

[10] L. Ricker, S. Lafortune, and S. Gene, "DESUMA: A Tool Integrating GIDDES and UMDES," *8th International Workshop on Discrete Event Systems,* pp. 392--393, 2006.

[11] K. Rudie, "The Integrated Discrete-Event Systems Tool," *Discrete Event Systems, 2006 8th International Workshop on,* pp. 394--395, 2006.

[12] C. Reiser, A. E. C. da Cunha, and J. E. R. Cury, "The Environment Grail for Supervisory Control of Discrete Event Systems," pp. 390--391, 2006.

[13] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems," *8th International Workshop on Discrete Event Systems,* pp. 384--385, 2006.

[14] L. Feng and W. M. Wonham, "TCT: A Computation Tool for Supervisory Control Synthesis," *8th International Workshop on Discrete Event Systems,* pp. 388--389, 2006.

[15] L. P. Pinheiro, Y. K. Lopes, and A. B. a. Leal, "Nadzoru: A Software Tool for Supervisory Control of Discrete Event Systems," *IFAC-PapersOnLine,* vol. 48, pp. 182--187, 2015.

[16] T. Moor, C. Baier, and X. Bai. "DEStool" [Online]. Available: http://www.rt.techfak.fau.de/FGdes/destool/index.html

[17] K. Akesson, M. Fabian, and H. Flordal, "Supremica in a Nutshell – Draft," *October,* 2007.

[18] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*: Springer-Verlag New York Inc., 2006.

[19] S. H. Zad, "Discrete Event Control Kit," 2013.

[20] D. Wimberger and J. Charlton. Java Modbus Library (jamod) [Online]. Available: http://jamod.sourceforge.net