Supervisory control synthesis for exception handling in printers

Esmée L.G. Bertens (TU/e), Ronald Fabel (Océ)

M. Petreczky (TU/e), D.A. van Beek (TU/e), J.E. Rooda (TU/e)

Abstract: *The purpose of this paper is to demonstrate the viability of supervisory control synthesis by presenting the formulation and a preliminary solution of a real-life case in Océ printers. Supervisory control theory (SCT) provides a formal approach to (automatic) supervisory control synthesis.*

Introduction

This paper proposes a new approach, namely, supervisory control theory (SCT) [1], for design of high-level control software for high-volume printers. SCT has already been researched extensively and applied to some industrial cases. However, application of this theory in printers is new. The viability of the proposed approach will be demonstrated by applying it to exception handling in printers.

Productive printing systems have evolved to be innovative and complex multi-disciplinary products. The amount of software control is increasing because of the many offered functionalities. This increase is possible because the computing power available for control software has become relatively powerful. The latter allows for more control performance, inviting the use of more complex control strategies in order to get more out of the system. As a consequence, the real costs are shifted towards development of the software, and handling its complexity.

In order to give a flavor of the complexity, we will briefly sketch the encountered challenges. High productive printing systems can transport sheets at speed up to 4 m/s. As many as 40 paper sheets can be transported concurrently. Each sheet is controlled by a separate process. The task of the control process is to guide the sheet through all the paper handling functionalities. This is quite similar to road traffic in which each car is given an individual driver to control the car and to navigate towards the goal. As with road traffic, the complexity of control increases significantly once an accident happens. In this case, one needs a supervisor to redirect the traffic. In a similar fashion, in case of a paper jam in a printer, supervised redirection is needed to keep the productivity as high as possible, to prevent damage of the printer system, and to limit the annoyance for the operator. The complexity is high because many sheets are involved with different sizes and positions in the paper path topology. Though very rare, we need to be able to control a paper handling exception in any situation.

We propose to use supervisory control theory for development of control software in order to decrease development time and costs and to improve reliability and evolvability of the control software. In a nutshell, supervisory control theory, initiated by Ramadge and Wonham, allows for automatic generation of correct-by-design control software. As inputs, it requires a formal model of the uncontrolled system (plant) and the formal specification of the desired behavior of the controlled system (control requirements). Major advantages of this approach are correctness of the supervisory controller (supervisor) by construction and automatic supervisor synthesis. As a consequence, the design process of a supervisory controller changes from implementing and debugging controller code to designing and debugging control requirements. The latter allows for a faster incorporation of requirement modifications into the control design. In turn, this leads to a reduction in the number of design-test-redesign loops.

Supervisory control is one of the many activities within the model based engineering (MBE) methodology. Like other activities, such as *simulation, design*, and *verification*, the *supervisory synthesis procedure* extracts information from the information model. The information model is the collection of all the knowledge on the various multidisciplinary aspects of the system. The success of the supervisory control synthesis will hugely depend on how well the method integrates into an existing modeling environment. Moreover, the accessibility of the theory to engineers and the maturity of the tools are influential factors in this new approach.

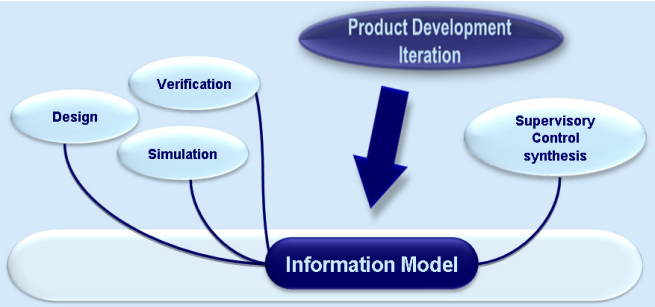


Figure 1: Model based engineering framework

This work has been carried out as a part of the ITEA project Twins 05004.

Supervisory Control

We start with describing the role of supervisors in the global system architecture. For the purposes of supervisory control, we choose a specific decomposition of the system into different levels, see Figure 2. The physical (uncontrolled) hardware represents the lowest level. The next level is formed by physical actuators and sensors which enable us to change, respectively detect, the behavior of the hardware. The subsequent level is formed by the low-level control processes (drivers) which are responsible for translating the high-level commands to actuator signals and for monitoring the states of the sensors. This level may already include some low-level control loops, for instance, the low-level speed and position control of motors. Note that, in advanced machine architectures, several low-level subsystems can be active simultaneously. Consequently, a high-level controller is required to ensure correct functionality. In particular, the high-level controller is responsible for (optimal) coordination of low-level tasks, for interpretation of user input, and for processing of high-level sensor data. In this paper we will call the high-level controller the supervisor. The task of supervisory control theory is to automatically generate such a supervisor *offline*.

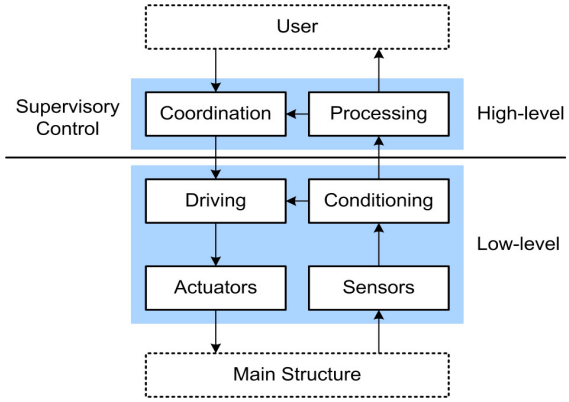


Figure 2: System architecture

Supervisory Control Theory

Intuitively, the solution represented by SCT based supervisors is very similar to solving the game of chess by building a look-up table. From any chess position, the game can be solved by looking-up the next move without calculation time. The availability of low cost embedded memory space enables this approach to be used for solving complex control problems for exception handling. The challenge is in the offline synthesis of this large look-up table. Furthermore, the exception handling game has an extra challenge that the playing field (printer topology) and also the rules of the game change during the development phase and differ from printer to printer.

More formally, in SCT a supervisor is computed that restricts the behavior of an uncontrolled system, called the plant, in such a way that the behavior of the restricted plant meets the control requirements [1]. The plant is assumed to spontaneously generate events. Events are divided into two classes. Controllable events are those which a supervisor can directly forbid. Uncontrollable events are events of which occurrence cannot be directly forbidden, such as error conditions. The supervisor observes the sequence of events generated by the plant and, based on the current state, disables controllable events in order to stay within the desired behavior state set. Note that a supervisor can prevent uncontrollable events only by disabling some controllable events which lead to the occurrence of an uncontrollable one. For synthesis, first, the plant and its control requirements are formally specified in terms of finite-state automata. Subsequently, from these formal models the supervisor is synthesized, which is correct by construction (w.r.t. accuracy of the plant model and the control requirements). The resulting supervisor ensures proper behavior and a deadlock and livelock free controlled system.

In the original Ramadge-Wonham (RW) framework, a supervisor is implemented by a state transition diagram (Figure 3). The first task of the supervisor is to keep track of the state of the plant; the second is to make control decisions, i.e. enable events, based on the current plant state. However, for complex systems, which have many components, the state-space of the plant might become too large for the current tools to handle.

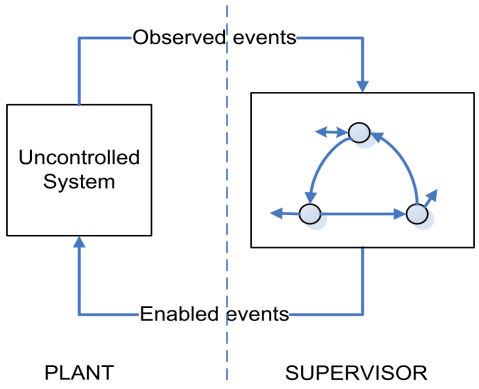


Figure 3: Control Diagram in the RW Framework

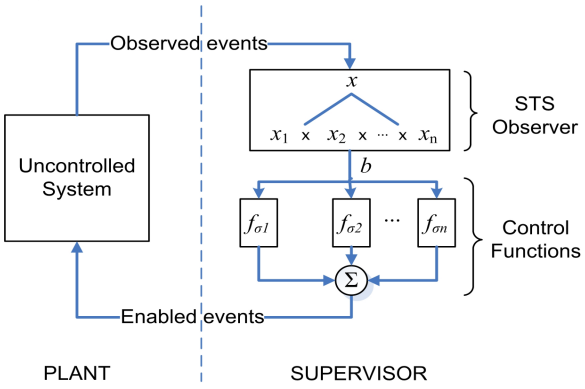


Figure 4: STS Control Diagram

State Tree Structures
A way to improve the efficiency of the synthesis is by structuring the system component models using state tree structures (STS) [2]. STS are structured and compact representations of discrete-event systems that include hierarchy and concurrency. The system is modeled as the combination of a set of component models called *holons*, and a state tree defining the structure of and relation between these holons. Due to these state tree structures, it is no longer necessary to generate one large flat state transition diagram describing the system behavior as needed for classical SCT. The STS control requirements are state based and defined as logical expressions. Two types of logical expressions are used in STS; (i) *state exclusion*, requiring that the system is never in certain states, (ii) *state-transition exclusion*, excluding controllable transitions in a certain state.

The control diagram of an STS supervisor is presented in Figure 4. Here, the supervisor is split into the STS observer that tracks the actual plant state and the control functions (in parallel for each controllable event) that determine which events are enabled, based on the actual plant state, in order to stay within the correct state set.

Case study: Exception handling in printers

A printer fetches sheets of paper from the paper tray, prints the required image on it and puts it into the so-called finisher. The route of the paper is referred to as the *paper path*. The transportation of the sheets is realized by *pinches*, driven by *motors.* A schematic representation of a simple paper path is illustrated in Figure 5.

Currently, the transportation of each sheet is controlled by a separate process, which controls motors, switches, and sensors. Although in the current design error handling has successfully been solved, the proposed solution is not as flexible as desired. In particular, modifications of exception handling requirements quickly interfere with the controller software design and redesign takes a lot of effort and introduces errors. Therefore, supervisory control is applied to exception handling in printers.

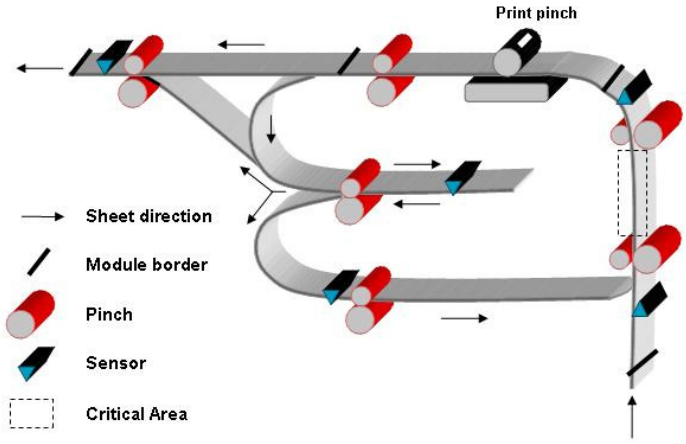


Figure 5: Schematic representation of a simple paper path

Error detection is provided by sensors checking the leading edge (LE) and the trailing edge (TE) of sheets in a certain time interval. When no edge of the sheet is detected in the expected time interval, an error is reported. Two different error situations are considered; (i) sensor is covered (e.g. previous sheet did not leave the sensor in time), (ii) sensor is not covered (e.g. scheduled sheet is too late at sensor for some reason). In case of an error the printer system needs to be supervised in order to minimize the number of service calls, and to maximize the convenience for the operator. Therefore, the error handling must ensure that the system ends up in a state that satisfies the conditions listed below.

1) No sheets are in the print process.
2) No sheets are on module borders (a border between two modules which can physically be decoupled).
3) No jamming and/or overlap of sheets in critical areas.
4) As many sheets as possible should be transported to areas for convenient operator recovery, e.g. finisher.

Note that these conditions do not hold for error sheets (only for sheets that are still controllable).

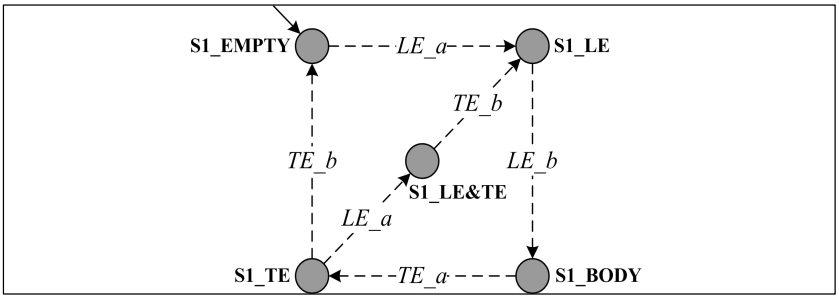
Interface
In order to implement a supervisor in the existing simulation environment of Océ, an interface between the control software of Océ, called the *controller*, and the supervisor was defined. Control of *nominal behavior* (i.e. when no errors occur) and error detection is already covered by the existing controller of Océ. In order to make error handling decisions when an error has occurred, the supervisor must know the actual system state. Therefore, in nominal behavior, the supervisor monitors the system by receiving events from the controller, see the table below. As soon as the controller reports an error to the supervisor, the supervisor becomes responsible for bringing the system into a state which satisfies the four conditions above. This implies that the controller continues with its control actions and that the supervisor governs the controller by giving stop commands to stop sheets and switch commands to change the direction of sheets in order to reach another destination.

	Controller → Supervisor	Supervisor → Controller
NOMINAL	- LE and TE detection of sheets at some positions. - Errors - Direction of switches	*No communication*
ERROR HANDLING	- LE and TE detection of sheets at some positions. - Errors - Direction of switches	- Stop command: Stop sheet(s) in a segment instantaneously. - Switch command: Change the direction of a switch.

Model Example

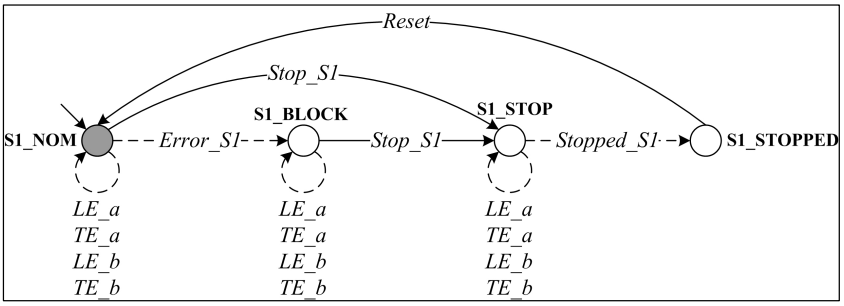
Having defined the interface functionality, the plant component models can be modeled as in Figure 6. Of course, for this paper, the models represent a simplification of the real case. In the models, states are denoted by vertices, initial states are indicated by an unconnected incoming arrow, and marked states (i.e. states in which the system is allowed to end) are denoted by filled vertices. Controllable and uncontrollable events are drawn with solid and dashed arrows, respectively. Multiple events on an arrow represent an arrow for each event.

We have chosen the plant model to represent the printer hardware and the Océ controller together. The supervisor must be able to make control decisions based on the current plant state. Therefore, the plant model must contain an overview of the position of the error, and the positions and number of sheets in the system. We have chosen to monitor the paper path, by dividing the paper path into segments and monitoring the presence of sheets in each segment. More precisely, we assume that each segment is shorter than the length of a paper sheet. By leading and trailing edge detection at the entrance and exit of the segment, the state of the segment can be monitored. Segments are consecutive, i.e. the exit of a segment is the entrance of the next segment. In Figure 6a, the automaton for monitoring segment 1, from position *a* to position *b*, is presented. The transitions represent detection of edges (LE or TE) at a certain position (*a* or *b*). Figure 6b represents the control actions a segment can execute. We define an error signal for each segment. An error signal brings a segment from the nominal state (**NOM**) into the block state (**BLOCK**), from which it can be stopped via the controllable *Stop* event. The uncontrollable *Stopped* event occurs when the segment is stopped. This implies that no leading or trailing edge detection for that segment can occur anymore. When all error handling actions have been completed, the segment returns to the nominal state via the controllable *Reset* event. Note that, a segment can also be stopped from the initial state.



EMPTY: no sheet in segment
LE: a leading edge in segment
BODY: sheet fully covers segment
TE: a trailing edge in segment
LE&TE: a leading and a trailing edge (both of different sheets) in segment

Figure 6a: Monitoring segment 1 (S1)



NOM: segment is in nominal behavior
BLOCK: segment is in error
STOP: segment is stopping
STOPPED: segment is stopped

Figure 6b: Control actions segment 1 (S1)

For a total plant model with, for instance, five segments (segment 1 to segment 5), both components of Figure 6a and 6b must be instantiated five times with unique events and states. Then, the resulting model considers the positions *a*, *b, c, d, e,* and *f* for tracking the state of segments.

Control Requirements

The control requirements are defined as constraints on the states the plant is not allowed to enter (i.e. state exclusion), and as constraints on the events the plant is not allowed to generate in certain states (i.e. state-transition exclusion). We will not present all the requirements, rather we give two examples. These examples are typical for the control requirements which can occur in practice:

- Example 1: State-transition exclusion
  For the model of Figure 6, plant component (b) defines that a segment can always be stopped from the initial state. However, we require the supervisor to only enable control actions when the system is in error mode. Then, the state-transition exclusion, disabling controllable *Stop* events if all segments are in the nominal state, is as follows:

  (**S1_NOM** $\wedge$ **S2_NOM** $\wedge$ **S3_NOM** $\wedge$ **S4_NOM** $\wedge$ **S5_NOM**) $\not\rightarrow$ *Stop_S1,Stop_S2,Stop_S3,Stop_S4,Stop_S5*

  ,where $\wedge$ denotes a logical and.
- Example 2: State exclusion
  According to condition 2 (page 3), module borders must be free of sheets. Suppose an error occurs in the system. Subsequently, that segment will enter the **BLOCK** state from which it will be stopped. We require the supervisor to free the module border from the paper tray to segment 1, which corresponds to position *a.* This implies that the supervisor must not ($\neg$) enter the following states:

  $\neg$(**S1_STOPPED** $\wedge$ **S1_LE**)
  $\neg$(**S1_STOPPED** $\wedge$ **S1_BODY**)
  $\neg$(**S1_STOPPED** $\wedge$ **S1_LE&TE**)

  The set of control requirements above excludes the states in which segment 1 is stopped and a sheet is still covering the module border, i.e. segment 1 is in state **S1_LE**, **S1_BODY**, or **S1_LE&TE**. As mentioned before, a supervisor can only disable controllable events. Therefore, the uncontrollable *Stopped_S1* event can only be prevented by disabling the controllable *Stop_S1* event which leads to the occurrence of the uncontrollable *Stopped_S1* event (Figure 6). This example shows the strength of SCT, since the theory is able to automatically remove sequences that lead to forbidden states.

Preliminary Results

- **Method integration in existing simulation environment**
  First, a proof of concept has been delivered by implementing of a simple supervisor in the existing simulation environment. The uncontrolled plant has been represented by a small subset of the paper path. Together with a few control requirements, a supervisor has been synthesized. The supervisor state-transition diagram has been translated to a state diagram suitable for implementation in the control software of the simulation environment. The correctness of the implementation has been validated by means of simulations.
- **Experience in applying method**
  Both the benefits (e.g. automatic generation, formal approach, offline synthesis) and the pitfalls (e.g. state explosion problem, limitations of the existing tooling) have been experienced when applying this method. Experience in modeling the plant and the control requirements as well as in the available supervisory tooling has been gained.
- **Design of interface and supervisor**
  Innovative design of the interface and supervisor on a high level of abstraction has been proposed.

Future work

- Refine and implement the interface.
- Complete control requirements for improvement of the supervisor functionality.
- Complete method integration into the existing simulation environment.
- In product testing.
- Explore new application domains.

Acknowledgements

We would like to thank R.J.M. Theunissen for his contribution to the models.

References

[1] C. Cassandras, S. Lafortune (2004). *Introduction to discrete event systems*, Kluwer Academic Publishers.
[2] C. Ma, W. Wonham (2005). *Nonblocking supervisory control of state tree structures*, Springer.