

Supervisory control based on multi-face modelling of discrete event systems

Gábor Kovács* Laurent Piétrac** Eric Niel**

* *Department of Control Systems and Information Technology,
Budapest University of Technology and Economics, Budapest, Hungary*

** *Laboratoire Ampère, INSA Lyon, France*

Abstract: This paper reports a supervisory control design methodology based on the multi-face modelling of discrete-event systems in order to allow rapid prototyping and flexible implementation of controllers for reactive systems. Although Supervisory Control Theory assures that the closed loop system meets the prescribed requirements, it uses ordinary finite state machines as process models, which results in complicated and large-scale controllers. A new modelling methodology simplifies modelling by introducing functional models based on tasks, which allow the reduction of component models. The paper presents a multi-level supervisory control architecture for systems modelled in the framework and derives the properties of the overall control system. Propositions for the implementation of such supervisory architectures are also made.

Keywords: Discrete Event Systems, Supervisory Control, Rapid Control Prototyping

1. INTRODUCTION

In the last decades, with the evolution of electronics, informatics and mass production, the systems to be controlled have become more and more complex, facing control engineers a hard task to assure desired behaviour of such processes. Ad hoc and intuitive methods, which are used in case of small-scale systems, are no longer applicable.

Formalization of discrete event controller design has been started and methods for modelling discrete event systems and their control has been proposed, see Cassandras and Lafortune (1999). Besides using finite state automata themselves, other approaches based on them like Petri nets, Grafcet, Statecharts or the concept of Function Blocks have been adopted by academic and/or industrial practice. However, these advanced models provide only a tool for modelling, and not for controller synthesis. Controllers are still designed commonly in an ad hoc and intuitive manner, based on human expertise and creativity.

Another issue, which has arisen along with the complexity, is the safety and security of systems. Control engineers dealing with complicated systems are not capable of assuring the safe operation of such systems in each and every state. By applying verification and failure analysis, the risks can be minimized, however, these procedures need time-consuming iterative design of controllers to finally meet the requirements.

The need for formal methods has been fulfilled by Supervisory Control Theory (SCT), providing a theoretically well-based framework for the design of discrete-event controller structures. Supervisors synthesized according to the principles of SCT can assure that the closed loop

system respects the prescribed requirements, and therefore makes verification unnecessary. However, SCT uses ordinary finite state machines as modelling tool, so controller design for large-scale systems is cumbersome due to the enormous number of states in complex models. Moreover, some operations of the supervisor synthesis require the combination of process and specification models, resulting in automata with state spaces really hard to handle. To overcome these problems, modular and hierarchic supervisory control have been introduced (see Wong and Wonham (1996) and Wonham and Ramadge (1988)), but modelling of complex systems has remained cumbersome.

No matter which tool is used for modelling, supervisory controllers are mainly designed in a tailored manner, i.e. new process and specification models are created for each application. However control engineers save their previously designed models and try to apply them for new systems, there exist no unified model libraries, commonly used in other fields of control engineering. This phenomenon makes the design of supervisory controllers a time-consuming procedure. A novel modelling framework has been proposed in Kovács and Piétrac (2009), which helps control engineers to work with smaller-scale models and also allows reusability of previously designed models while keeping the whole procedure within the framework of SCT. The main idea of the approach is borrowed from rapid control prototyping, where the process model (and sometimes also the controller model) is built up from predefined, off-the-shelf components. These predefined component models are then connected and their global behaviour is specified. Two main representations are defined for each component. From the technological representation, which follows from the physical properties of the component, a functional representation is obtained, which describes only the most important functional properties by the means of tasks carried out by the given component.

* This research was partially funded by the Hungarian Scientific Research Fund under grant OTKA K 71762.

Therefore, when assembling the model of the system of building blocks of components, the control engineer has to focus only on important functionalities, and technological details are left hidden. Nevertheless, the size of functional models is smaller than the technological ones, and so the model of the whole system can be kept in a reasonable size.

The approach also introduces a novel concept for supervisory control design. Two kinds of controllers, namely task controllers and functional controllers are introduced. Task controllers, which are parts of the component representations and therefore are stored in the model library, are defined for the tasks and are responsible for respecting the safety, security and liveness requirements. The functional controller, defined for the ensemble of the components used, is responsible for the coordination of their tasks.

The remaining part of the paper is organized as follows. Section 2 summarizes some of the most important notations and principles of Supervisory Control Theory. In Section 3 an overview on the proposed framework for controller design and the models used is given, while Section 4 presents the procedure of supervisor synthesis. Section 5 proposes control architectures for the framework and Section 6 concludes the paper.

2. PRELIMINARIES

Here only some fundamental principles and notations of SCT and automata theory are presented in order to keep the paper as self-contained as possible. For further details the reader may refer to Wonham (2002) and Hopcroft and Ulmann (1979).

The discrete-event system G is described by the 5-tuple $G = \{Q^G, \Sigma^G, \rho^G, q_0^G, Q_m^G\}$ with Q^G as its state set, Σ^G as its event set, $\rho^G : Q^G \times \Sigma^G \rightarrow Q^G$ as its partial transition function, q_0^G as its initial state and Q_m^G as the set of its marking states. The event set Σ^G can be divided into the distinct sets of controllable and uncontrollable events so that $\Sigma^G = \Sigma_C^G \cup \Sigma_U^G$, where $\Sigma_C^G \cap \Sigma_U^G = \emptyset$. The notation $\rho^G(q, \sigma)!$ means that there exists a transition associated with the event $\sigma \in \Sigma^G$ leaving the state $q \in Q^G$. Partial transition functions can be extended to the set of strings instead of symbols of an alphabet.

The language generated by G is denoted by $L(G)$. The notation \bar{L} stands for the prefix closure of a language L . The natural projection of a language $L \in \Sigma^*$ to an alphabet Σ' is denoted by $P_{\Sigma'}(L)$ while the inverse projection is denoted by $P_{\Sigma}^{-1}(L')$.

The synchronous product (or parallel composition) of two DESs G_1 and G_2 describes the operation of a system in which the G_1 and G_2 operates synchronously. The synchronous product operation of two DESs is defined by $L(G_1 \parallel G_2) = P_{\Sigma_{G_1} \cup \Sigma_{G_2}}^{-1}(L(G_1)) \cap P_{\Sigma_{G_1} \cup \Sigma_{G_2}}^{-1}(L(G_2))$.

The goal of supervisor synthesis is to define a supervisor which can restrict the operation of the system to meet the constraints of the specifications defined by the language E , often given in the form its generator. The supervisor S is a function $S : L(G) \rightarrow \Gamma$ defined by $\Gamma = \{\gamma = PWR(\Sigma) \mid \gamma \supseteq \Sigma_U\}$, where γ represents the set of events authorized by S and $PWR(\Sigma)$ is the set of

all subsets (the power set) of Σ . If the specifications are controllable, the automaton S/G describing the supervised system is the product of G and E . In other cases the supremal controllable sublanguage (or maximal permissive sublanguage) of E can be found, which allows the greatest possible set of controllable events. This sublanguage allows only those operations described by $L(G)$ which respects the constraints given by E . The supremal controllable sublanguage of a specification E with respect to the language L will be denoted by $E^{\uparrow L}$.

3. OVERVIEW OF THE FRAMEWORK

Results presented in this paper are based on the framework presented in Kovács and Piétrac (2009). However, for the sake of self-dependence, most important features of the framework and component models used will be summarized in the followings, using the example of a two-degrees-of-freedom pick and place machine.

3.1 Framework for rapid prototyping

The framework is based on the use of components, which are such parts of a system, which can be distinguished by the mean of their functionality and IOs. Such components are collected to the component library, which contains reusable models. Components can be modeled fundamentally by two points of views. One is the technological aspect, which uses large-scale, detailed models with events corresponding to the evolution of IOs. On the other hand, the functional point of view focuses on the functionality of the component, with events corresponding to complex tasks (e.g. movement of the machine from one position to another), and with a moderate-sized state space. When components are composed to form new components, or to create the model of the whole system, the functional models are used for the composition, and therefore the problem of state explosion is less significant.

The technological and functional models are connected by the tasks. If one observes the common behaviour of DESs, he or she can note that there are operations which these systems execute time to time. Often these operations can be grouped together and referred to as *tasks*. For example, a task of a linear axis can be a movement from the negative extremity to the positive one, which comprises movement and stop of the motor in the right order. However, these operations correspond not just output patterns but can also depend on sensor inputs, e.g. movement of the axis should be stopped when the signal of an end switch rises.

The concept of tasks is not unfamiliar to the control engineers working with DESs (see, for example, Vyatkin and Hanish (2006)). However, task concept presented here is compatible with SCT, and therefore brings the benefits of formal methods close to industrial practice.

The procedure of modelling a system using the framework is as follows. The system is at first decomposed to components, which are, if needed, further decomposed to subcomponents. For example, the pick and place machine can be decomposed to the components of the two linear axis. If the axis model can be found in the component library, it is selected, otherwise its models presented in the followings have to be defined. The model of the machine is

composed of the functional models of the axis, and if tasks and a functional model is defined, it can be treated as a component, from which more complex systems (e.g. a cell containing two machines) can be assembled.

Supervisors are designed for each task of each component, and finally for the global system. Controllers corresponding to components are also stored in the library.

3.2 Models of a component

In the followings, the models corresponding to different faces of a component will be presented. To help the reader, most important models will be illustrated on the component of a linear axis, driven by an electric motor and equipped with two end switches at the positive and negative extremities.

Technological model In case of atomic components, the Technological model represents all possible evolution of inputs and outputs of the component which are allowed by its physical manifestation. At first the possible evolution of each IO is modeled, and then these models are restricted by physical constraints (e.g. in case of the linear axis, the two end switches can not be active at the same time), represented by the language of the evolution of IOs allowed by the constraints, L^{PhC} . Formally, the Technological model is given by the 5-tuple $G^{Tech} = \{Q^{Tech}, \Sigma^{Tech}, \rho^{Tech}, q_0^{Tech}, Q_m^{Tech}\}$. In case of atomic components, G^{Tech} is the generator of the language $L(G^{IO}) \cap L^{PhC}$.

In case of composed components, there is no need to define physical constraints, so $G^{Tech} = \parallel_j G_j^F$, where G_j^F is the functional model of the j^{th} subcomponent.

Specifications for Safety, Security and Liveliness In practice there are requirements of safety, security and liveliness which shall be respected no matter what functionality the given component should realize. They represent the avoidance of such operations which are physically possible, and therefore allowed by the technological model, but which cause instable or dangerous behaviour of the component. For example, the linear axis should not be started towards the negative direction if it is at the negative extremity. These specifications are referred to as Specifications of Safety, Security and Liveliness (S3L) which is given by the language E^{S3L} .

The Nominal model Obviously the aforementioned specifications are defined for the process model and it has to be guaranteed that they are respected before obtaining the functional model, which will be therefore based on a system assuring the specifications. An adequate DES can be defined using the supremal controllable sublanguage of the Technological model with respect to the Specifications of Safety, Security and Liveliness. Therefore the Nominal model is defined as the minimal trim automaton which marks the supremal controllable sublanguage of E^{S3L} with respect to the language of the technological model, i.e. $L(G^{Tech})$. The Nominal model is given by the 5-tuple $G^N = \{Q^N, \Sigma^N, \rho^N, q_0^N, Q_m^N\}$ and the language it generates will be referenced as L^N in the followings.

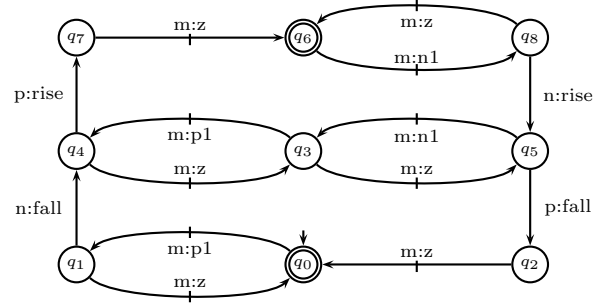


Fig. 1. Nominal model of the linear axis

The Nominal model of a linear axis is shown by Fig. 1. Events with prefixes p : and n : correspond to the signals of the end switches at the positive and negative extremities while events $m : z$, $m : n1$ and $m : p1$ correspond to the stop, negative and positive direction movements of the motor, respectively.

Tasks The resulting Nominal model depicts only such operation of the component which is allowed by the specifications of safety, security and liveliness. However, it describes the operation of the component in details, by low-level events, so the Nominal model is an ideal candidate to build a functional model on. As presented afore, the way from technological to functional representations is paved by the tasks.

At first, let us define the part of the IO model to be included in the task by the task core model, which is given by the 5-tuple $G_j^{Tc} = \{Q_j^{Tc}, \Sigma_j^{Tc}, \rho_j^{Tc}, q_{0,j}^{Tc}, Q_{m,j}^{Tc}\}$ for the j^{th} task. The state and event set of the task core model are subsets of the state and event set of the Nominal model, i.e. $Q_j^{Tc} \subseteq Q^N$ and $\Sigma_j^{Tc} \subseteq \Sigma^N$. The transition function is defined as follows:

$$\rho_j^{Tc}(q_j^{Tc}, \sigma) = \begin{cases} \rho^N(q_j^{Tc}, \sigma) & \text{if } \rho^N(q_j^{Tc}, \sigma) \in Q_j^{Tc} \\ \text{undefined} & \text{otherwise} \end{cases}$$

The initial state of the task core model is a state $q_{0,j}^{Tc} \in Q_j^{Tc}$ from where all the other states of the task core model are accessible:

$$\forall q \in Q_j^{Tc}, \exists s \in \Sigma_j^{Tc*} : \rho(q_{0,j}^{Tc}, s) = q$$

The set of the marked states of the task core model is $Q_{m,j}^{Tc} \subseteq Q_j^{Tc}$ such that

$$Q_{m,j}^{Tc} = \{q \in Q_j^{Tc} \mid \exists \sigma \in \Sigma_j^{Tc} \text{ such that } \rho(q, \sigma)!\}$$

Before giving the definition of the task, two new events shall be introduced. The first one corresponds to the start of the task which will be referred to as the *start event* and will be denoted by σ_j^{start} in the followings. The other, reporting the completion of the task will be referred to as the *confirmation event* and will be denoted by σ_j^{conf} . Since tasks are started by the controller, the start event is defined to be controllable. Confirmation events, enabled by the tasks (more accurately the task supervisors) themselves, are also controllable. These events are collected to the alphabet of task events $\Sigma^T = \bigcup_j \{\sigma_j^{start}, \sigma_j^{conf}\}$.

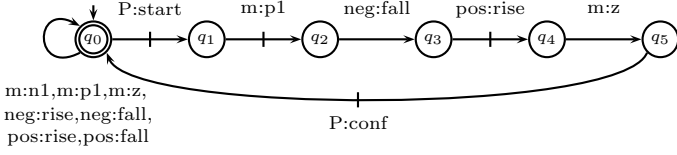


Fig. 2. Model of the task of positive motion

Using the task events and the task core model the task model can be defined by giving the 5-tuple $G_j^T = \{Q_j^T, \Sigma_j^T, \rho_j^T, q_0^T, Q_m^T\}$. The state set of the task model is $Q_j^T = q_0^T \cup Q_j^{Tc}$ while its event set is the union of the event set of the corresponding task core model and the task events: $\Sigma_j^T = \Sigma_j^{Tc} \cup \{\sigma_j^{start}, \sigma_j^{conf}\}$. The initial state of the task model is defined to be the newly added state, i.e. $q_0^T = q_0'$. The transition function is defined as follows:

$$\begin{aligned} \rho_j^T(q_0^T, \sigma_j^{start}) &= q_0^{Tc} \\ \rho_j^T(q_0^T, \sigma) &= q_0^T, \forall \sigma \in \Sigma_j^{Tc} \\ \rho_j^T(q, \sigma_j^{conf}) &= q_0^T, q \in Q_m^T \end{aligned}$$

For other states:

$$\rho_j^T(q, \sigma) = \rho_j^{Tc}(q, \sigma), \forall \rho_j^{Tc}(q, \sigma)!$$

The only marking state is the initial state i.e. $Q_m^T = \{q_0^T\}$, while the language generated by the task model is $L(G_j^T) = \overline{\sigma_j^{start} \cdot L_{m,j}^{Tc} \cdot \sigma_j^{conf}}$

Fig. 2 illustrates the task model corresponding to the movement of the linear axis towards the positive direction. The task start and confirmation events are $P : start$ and $P : conf$, respectively.

The definition of the tasks, i.e. the selection of task core models is left to the control engineer.

The Integral model Now a common model can be defined, which comprises also the functional and the technological representations. An intermediate step is to create a model by the parallel composition of the task models and the Technological model and check its controllability with respect to the so-called Task Alternance Specification.

The Task Alternance Specification (TAS) captures the property that technological events can only happen inside tasks (i.e. preceded by a task start event and succeeded by a task confirmation event), which is true if the model is well covered. It also allows the activity of only one task of a component at the same time. The TAS model is defined by $E^{TAS} = \{Q^{TAS}, \Sigma^{TAS}, \rho^{TAS}, q_0^{TAS}, Q_m^{TAS}\}$, where $Q^{TAS} = \{q_0^{TAS}, q_1^{TAS}\}$ and the former is defined to be the initial state, $\Sigma^{TAS} = \Sigma^{Tech} \cup \Sigma^T$, $Q_m^{TAS} = \{q_0^{TAS}\}$. The partial transition function is defined by

$$\begin{aligned} \rho^{TAS}(q_0^{TAS}, \sigma) &= q_1^{TAS}, \forall \sigma \in \bigcup_i \sigma_i^{start} \\ \rho^{TAS}(q_1^{TAS}, \sigma) &= \begin{cases} q_0^{TAS}, \forall \sigma \in \bigcup_i \sigma_i^{conf} \\ q_1^{TAS}, \forall \sigma \in \Sigma^{IO} \end{cases} \end{aligned}$$

If E^{TAS} is controllable with respect to $G^{Tech} \parallel_j G_j^T$, then the model is said to be well covered by the tasks.

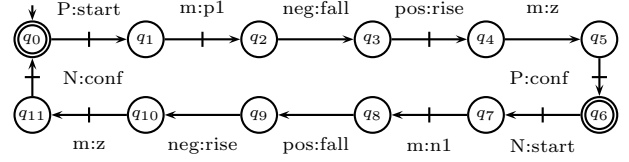


Fig. 3. Integral model of the linear axis

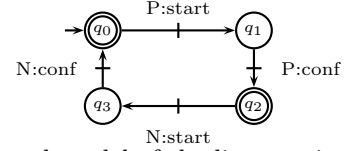


Fig. 4. Functional model of the linear axis

Otherwise, task models have to be redefined or new tasks have to be included.

The Integral model can then defined as the composition $G^I = G^{Tech} \parallel_j G_j^T \parallel G^{TAS}$ and is given by the 5-tuple $G^I = \{Q^I, \Sigma^I, \rho^I, q_0^I, Q_m^I\}$.

Since the functional model is derived from the integral one, it is important to show that, if we omit the events corresponding to the start and completion of the states, the integral model respects the specifications S3L. This property is justified by the following theorem.

Theorem 1. If the model is well covered by the tasks, then the projection of the language generated by the Integral model to the alphabet of technological events rests within the language of the Nominal model: $P_{\Sigma^{Tech}}(L^I) \subseteq L^N$.

The functional model The functional behaviour of the model can be described by the tasks, which depict the operation by collecting a series of low-level events to one single object. Therefore, the operation of the process (respecting the specifications of safety, security and liveness) can be considered as starting tasks and waiting for their completion. So the functional model can be obtained as the projection of the integral model to the task events. The functional model is the generator of the language $L^F = P_{\Sigma^T}(L^I)$ and is defined by the 5-tuple $G^F = \{Q^F, \Sigma^F, \rho^F, q_0^F, Q_m^F\}$ where $\Sigma^F = \Sigma^T$. The functional model of the linear axis component is shown by Fig. 4.

The following theorem states that the integral model of a component can be reconstructed from its functional model and its task models.

Theorem 2. The language generated by the parallel composition of the functional model and the task models equals the language generated by the integral model:

$$P_{\Sigma^{Tech} \cup \Sigma^T}^{-1}(L^F) \cap (\bigcap_j P_{\Sigma^{Tech} \cup \Sigma^T}^{-1}(L(G_j^T))) = L^I$$

Proof of the theorems are based on the proofs presented in Kovács and Piétrac (2009).

3.3 Assembling the model of the system

The model of the system can be obtained as the parallel composition of the functional models of the subcomponents, i.e. $G^F = \parallel_j G_j^F$. Note that the use of functional models causes a significant decrease of the size of the state space.

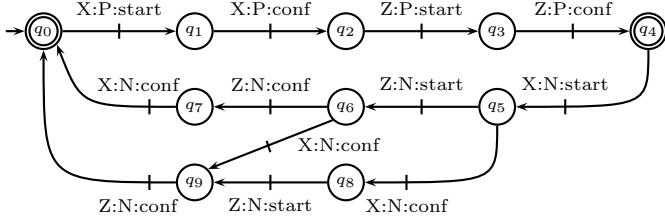


Fig. 5. Operation of the pick and place machine in closed loop

Based on the functional model of the system, appropriate functional specifications can be defined, and the model of the closed loop can be obtained. The model depicted by Fig. 5 represents the closed loop operation of the machine respecting the specifications that movements should be carried between the negative and positive extremitys of the two axis, and that the simultaneous movement of the two axis to the positive direction is forbidden. The prefixes X : and Z : denote the tasks corresponding to the X and Z axis, respectively.

4. SUPERVISOR DESIGN PROCEDURE

Following the component-based principles of modelling, the supervisory control architecture is also built up component by component. For each task, a task supervisor is defined, which deals with physical IOs in case of atomic components and the task events of the subcomponents in case of composed components. The role of the task controller is to ensure that the specifications of safety, security and liveness are respected when the given task is active.

Task controllers are running only if the given task is active. Activation of a tasks is carried out by the corresponding task start event, controlled by the functional supervisor. Therefore, the functional supervisor deals only with task events: it activates tasks by the appropriate start events and is notified about their successful termination by the corresponding confirmation events.

However, Supervisory Control Theory does not allow the generation of events for the supervisors, so in order to deal with the functional events, the process model has to be extended by a simple task event generator, which generates the start and confirmation events. The functional supervisor influences the occurrence of the controllable start events by enabling/disabling them, while the occurrence of the confirmation events is controlled by the task supervisors.

Definition 1. The task generator is a discrete event system $G^{TEG} = \{Q^{TEG}, \Sigma^{TEG}, \rho^{TEG}, q_0^{TEG}, Q_m^{TEG}\}$ where $Q^{TEG} = \{q\}$, $\Sigma^{TEG} = \Sigma^F$, $\rho^{TEG}(q, \sigma) = q, \forall \sigma \in \Sigma^{TEG}$, $q_0^{TEG} = q$, $Q_m^{TEG} = \{q\}$. The language generated by the system $G^{TEG} \parallel G^{Tech}$ will be noted by $L^{P'}$.

4.1 Task supervisors

Task supervisors are such supervisors which act only if the given task is activated, i.e. after the occurrence of the corresponding task start event.

Definition 2. The task supervisor corresponding to a task T_i is a mapping $S_i^T : L_i^T \mapsto \Gamma_i^T \in \text{Pwr}(\Sigma_i^T)$, where the control map is given by

$$\Gamma_i^T(s) = \Sigma_{i,UC}^T \cup \{\sigma \in \Sigma_{i,C}^T \mid \rho_i^T(q_{0,i}^T, s.\sigma)!\}$$

Roughly speaking, the task supervisor ensures that technological events occur only according to the task core models, and it allows the generation of the confirmation event if and only if the task has been completed, i.e. the task core language has been generated by the process.

4.2 Functional supervisor

The role of the functional supervisor is to coordinate the execution of the tasks. It is designed on the basis of the functional model of the system by the control engineer, in order to meet the requirements of the actual problem.

Definition 3. The functional supervisor is the mapping $S^F : L(F) \mapsto \Gamma^F \subseteq \text{Pwr}(\Sigma^T)$ where the control map is defined by

$$\Gamma^F(s) = \Sigma_{UC}^T \cup \{\sigma \in \Sigma_C^T \mid \rho^F(q_0^F, s.\sigma)!\}$$

4.3 Cooperation of the supervisors

The defined supervisory control architecture needs the functional supervisor and a particular subset of task supervisors to be active at each time instance. Therefore at first the composition of two supervisors should be ensured. The inverse projection of a supervisor to a language, i.e. the extension of its control map to deal with all events of the given language, is defined as follows.

Definition 4. Consider a language $L' \subseteq \Sigma'^*$ and a supervisor $S' : L' \mapsto \Gamma'$, where $\Gamma' \subseteq \text{Pwr}(\Sigma')$. The inverse projection of the supervisor to a language $L \subseteq \Sigma^*$, where $\Sigma' \subseteq \Sigma$ is $P_L^{-1}(S) : L \mapsto \Gamma$, where the control map is defined by

$$\Gamma(s) = (\Sigma \setminus \Sigma') \cup \Gamma'(P_{\Sigma'}(s))$$

Using the operation of inverse projection, composition of supervisors can be defined similarly to the parallel composition of languages.

Definition 5. The composition of two supervisors, $S_1 : L_1 \mapsto \Gamma_1$ and $S_2 : L_2 \mapsto \Gamma_2$ on the language L is denoted by $S = S_1 \wedge_L S_2$ and is defined by $S : L \mapsto \Gamma$. The control map is $\Gamma(s) = \Gamma_1'(s) \cap \Gamma_2'(s)$, where Γ_1' and Γ_2' are the control maps of $P_L^{-1}(S_1)$ and $P_L^{-1}(S_2)$.

Theorem 3. The language of the closed loop under the supervision of an arbitrary functional supervisor and the task supervisors does not violate the specifications of safety, security and liveness, i.e.

$$P_{\Sigma^{Tech}}(L(S_1^T \wedge_{L^P} \dots \wedge_{L^P} S_n^T \wedge_{L^P} S^F)) / (G^P \parallel G^{TEG}) \subseteq E^{S3L}.$$

Outline of the proof Let us assume that a simple functional supervisor is defined, which restricts the functional behaviour of the closed loop to $\sigma_1^{start}.\sigma_1^{conf}$. Therefore, the language of the closed loop is

$$S^F / (G^P \parallel G^{TEG}) = \bigcup_{u.v.w \in L^P} u.\sigma_1^{start}.v.\sigma_1^{conf}.w$$

According to Definition 2, the task supervisor enables events according the language of the task core model to be included between the start and the confirmation

event. Also, due to the definition of the task core model, it is controllable with respect to the specification E^{S3L} . Therefore,

$$\begin{aligned} & (S_1^T \wedge_{L^P} S^F) / (G^P \parallel G^{TEG}) = \\ & = \bigcup_{u.L_1^{Tc}.w \in L^P} u.\sigma_1^{start}.L_1^{Tc}.\sigma^{conf}.w \end{aligned}$$

However, if the model is well covered by the tasks, we have $u = \epsilon$ and $w = \epsilon$, which means that

$$(S_1^T \wedge_{L^P} S^F) / (G^P \parallel G^{TEG}) = L_i^T \subseteq L^I.$$

According to Theorem 2, $P_{\Sigma^{Tech}}(L^I) \subseteq L^N \subseteq E^{S3L}$, therefore the operation of the closed loop meets the specifications.

The procedure above can be generalized to any arbitrary functional supervisor inside the functional model.

5. SUPERVISORY CONTROL ARCHITECTURES

Supervisors defined in the previous sections can be implemented in various ways, depending on the system to be controlled and on the available resources.

5.1 Monolithic architecture

The classical way for supervisor implementation is to design one single monolithic controller running on one single device. However flattening the supervisor architecture to one single supervisor is possible, or the supervisor can be synthesized directly based on the integral model, such a controller does not benefit from the properties of moderate state and event-space. On the other hand, especially for small-scale systems, a monolithic architecture might be suitable as it needs one single, low-performance controller device and therefore cuts down expenses compared to more sophisticated architectures.

5.2 Distributed architecture

A more suitable method for implementation follows naturally from the architecture of the supervisors. By combining modular and hierarchical approaches, the controller can be implemented as follows.

Task controllers are implemented for each task or each component. Since task supervisors are relatively simple and only one task of a component can be active at a time, it is straightforward to implement all supervisors corresponding to tasks of a given component in one device. The functional controller is implemented independently from the tasks supervisors. Since there is no need for communication between the task controllers themselves, only between each task controller and the functional controller, a simple communication architecture can be used. Nevertheless, bus systems may also be suitable.

The only drawback of this architecture versus the monolithic one is that it needs several physical devices to implement the controllers on. On the other hand, it also provides flexibility for the system. If a given component is replaced by another one providing the same functionality, only its controller has to be adjusted and there is no need to reimplement the whole control system.

The distributed control architecture of the pick and place machine is shown by Fig. 6.

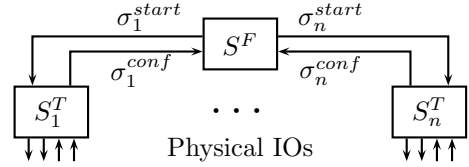


Fig. 6. Distributed control architecture

5.3 Single-device distributed architecture

By combining the two previous approaches, a distributed architecture can be implemented on one single device. Clearly, on single-core devices, only one controller (functional or one of the task controllers) can run at a time, therefore controller execution has to be based on time-sharing. In applications requiring hard real-time control, this option is not applicable. However, considering that the majority of the industrial processes are controlled by PLCs which can not guarantee arbitrarily small response times due to their cyclic behaviour, in case of moderate number of components such an architecture would be a suitable solution. A proposition for such distributed control software architecture for PLCs is given in Kovács (2009).

6. CONCLUSION

The supervisory control design procedure, based on the multi-face modelling of discrete-event systems, plays an important role in a new framework for the rapid prototyping of discrete event controller structures. It has been shown that by using the proposed supervisor architecture, the closed loop behaviour does not violate the prescribed specifications.

Future work includes the definition of the model library, integration with IEC61499-based control design and methods for automatic code generation for various platforms.

REFERENCES

- Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston.
- Hopcroft, J. and Ulmann, J. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison Wesley.
- Kovács, G. (2009). On the implementation of task-based supervisory controllers. *Proc. 13th IFAC Symposium on Information Control Problems in Manufacturing. Moscow, Russia*, 1, 430–435.
- Kovács, G. and Piétrac, L. (2009). Multi-faced modelling for rapid prototyping of discrete event control systems. *Proc. European Control Conference. Budapest, Hungary*, 1, 1463–1468.
- Vyatkin, V. and Hanish, H. (2006). Design of controllers for plug-and-play composition of automated systems from smart mechatronic components. *Proc. ANIPLA International Congress*.
- Wong, K. and Wonham, W. (1996). Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6, 241–273.
- Wonham, W. (2002). *Notes on Control of Discrete Event Systems*. University of Toronto.
- Wonham, W. and Ramadge, P. (1988). Modular supervisory control of discrete event systems.