Multi-Face Modeling for Rapid Prototyping of Discrete Event Control Systems

Gábor Kovács
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
Email: gkovacs@iit.bme.hu

Laurent Piétrac
Laboratoire Ampère
INSA de Lyon
Villeurbanne, France
Email: laurent.pietrac@insa-lyon.fr

Abstract— **This paper reports a methodology for the multi-face modeling of discrete-event systems to be used in a framework for the rapid prototyping of supervisory controllers. However Supervisory Control Theory provides a possibility for the synthesis of supervisors proven to respect the specifications, it uses ordinary finite state machines and operations resulting in state explosion, so therefore hardly applicable for large-scale systems. The aim of the presented methodology is to simplify and accelerate controller design procedure by model reduction. By the introduction of task philosophy and component-based design, a methodology for obtaining moderate-size functional models from technological representations is presented. The paper gives definitions for models serving as different representations of components and for the conversion between them.**

I. Introduction

With the evolution of technology, more and more complex systems are to be controlled, facing engineers a hard task of controller design. Ad hoc and intuitive methodologies, used in the industrial practice, are no more sufficient. Formalization of discrete event controller design has been started and several methods for modeling discrete event systems and their control has been proposed [1]. Automata-based approaches, used in industrial practice, has been adopted by many tools.

Besides using finite state automata themselves, many achievements have been made. The introduction of Petri nets, especially colored ones has allowed the use of moderate-size models. The Grafcet method, popular in industrial practice, allows the use of some well-known principles of programming in the field of automata-based modeling. One of the most advanced tools is Harel's Statecharts, providing hierarchic and concurrent modeling possibilities. Statecharts has been adopted by information technology, and became coherent part of the Unified Modeling Language (UML). However, these advanced models provide only a tool for modeling, and not for controller synthesis. Controllers are still designed commonly in an ad hoc and intuitive manner, and however the tools mentioned afore facilitate the task of control engineers by keeping the size of models more comprehensible, controller synthesis is mostly based on human experience and creativity.

The need for formal methods has been fulfilled by the Supervisory Control Theory (SCT), providing a theoretically based framework for the synthesis of discrete-event controller structures. Supervisors synthesized according to the principles of SCT can assure that the closed loop system respects the prescribed requirements. However, SCT uses ordinary finite state machines as modeling tool, so controller design for large-scales systems is cumbersome due to the enormous number of states in more complex models. Moreover, some operations of the supervisor synthesis require the combination of process and specification models, resulting in automatons with a size really hard to handle. To overcome these problems, modular and hierarchic supervisory control has been introduced (see [2] and [3]), but modeling of complex systems has remained cumbersome.

No matter which tool is used for modeling, supervisory controllers are mainly designed in a tailored manner, i.e. new 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.

This paper presents a novel methodology, which helps the control engineer to overcome both on the problem of large-scale models and reusability of previously defined models while keeping the procedure within the framework of SCT. Borrowing the idea from rapid control prototyping, the approach uses component-based modeling of systems. Components are modeled from two approach. 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. Therefore, when assembling the model of the system of building blocks of components, the control engineer has to focus only on important functionalities, and technical details left hidden. Nevertheless, the size of functional models is smaller then the technological ones, and so the model of the whole system can be kept in a reasonable size. Modeling is also supported by a component library allowing the acceleration of control design procedure.

This paper gives formal definitions of the models serving as the technological and functional representations of components, and gives a method for obtaining the functional

ISBN 978-963-311-369-1

model from the technological one. The remaining part of the paper is organized as follows. Section II summarizes some of the most important notations and principles of Supervisory Control Theory. In Section III an overview on the proposed framework for controller design is given. Section IV presents the notations of the technological representation of a component while Section V describes how a functional model can be obtained from it. Section VI concludes the paper.

II. 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 [4] and [5].

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, $\rho^G : Q^G \times \Sigma^{G*} \to Q^G$ as its extended 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 $\exists\rho(q,\sigma)$ means that there exists a transition associated with the event $\sigma \in \Sigma^G$ leaving the state $q \in Q^G$.

The language generated by G is denoted by $L(G)$. The prefix closure of a language L is denoted by $\overline{L}$. An important operation on languages is the natural projection to a given alphabet Σ, defined by the followings:

$$\begin{aligned} P_\Sigma(\epsilon) &= \epsilon \\ P_\Sigma(\sigma) &= \begin{cases} \sigma & \text{if } \sigma \in \Sigma \\ \epsilon & \text{otherwise} \end{cases} \\ P_\Sigma(\sigma.t) &= P_\Sigma(\sigma).P_\Sigma(t) \end{aligned}$$

The extension of the definition above for languages is straightforward. The inverse projection is defined by $P_\Sigma^{-1}(s) = \{t \in \Sigma | P_\Sigma(t) = s\}$ and can be also extended to languages.

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 $L(G_1 \parallel G_2) = P^{-1}_{\Sigma^{G_1} \cup \Sigma^{G_2}}(L(G_1)) \cap P^{-1}_{\Sigma^{G_1} \cup \Sigma^{G_2}}(L(G_2))$.

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, modeled by a DES E. The supervisor S is a function $S : L(G) \to \Gamma$ 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 $L(G)$ can be found, which allows the greatest possible set of controllable events, see [6] and [7]. This sublanguage allows only those operations described by $L(G)$ which respects the constraints given by E. The supremal controllable sublanguage of a language L regarding the specification given by E will be denoted by $L^{\uparrow E}$.

The controller model is also described by a 5-tuple $C = \{Q, \Sigma, \rho, q_0, Q_M\}$, possibly extended by a control map $\Theta : Q \times \Sigma_C \to \{0,1\}$. The controller C is constructed based on the automaton representing the supervised system and the supervisor itself, and gives the automaton model of the controller with the events to be enabled or disabled in each of its states defined by the control map.

Controller design is based on the models of the process and the specification, upon which the supervisor is synthesized. Then a controller model, which is a representation of the supervised system, is derived and is implemented on a suitable platform.

III. Overview of the framework

The results summarized in this paper are the theoretical basis of a framework for the rapid prototyping of supervisory controllers for discrete event systems. This framework comprises multiple modules for the support of controller synthesis and realization.

A. Component models

The framework borrows the idea of components from the methodology of system modeling and reflects the phenomenon that different systems are built up from the same components, e.g. various consumer products of different manufacturers use the same parts for a given task. Components are systems, devices or subsystems, which can be defined and which can act independently from other components. The most simple devices, such as valves or binary sensors, are represented by atomic components. Atomic means here that there is no use further decomposing them, or they can not be decomposed. On the other hand, more complex components can be built up using smaller components, i.e. a model of a cylinder can contain several valve components.

There are two fundamental points of view during the development. One is the technological point of view, considering technological details and thinking of low-level operations and physical signals. From the functional point of view details of the operations remain hidden and the component is considered as a device executing more complex tasks.

B. Component library

One of the basic ideas of the framework is the use of a component library. Commonly used components, like different sensors, actuators or more complex devices, are stored in a library, and during the modeling of the system only the interconnection of such components has to be defined.

The aforementioned method of modeling systems has an important benefit considering the conception of a component library. It is a common situations that there exist various components which realize the same function, e.g. a hydraulic or pneumatic bistable cylinder. Although their technological models are different, their functional representations are the same, which makes it possible to use one single model for the high-level description of different components.

C. Controller design

Controller design is supported by the aforementioned model library, therefore the procedure of system modeling can be significantly accelerated. The user can select the components the given physical process is built up from, and pick their discrete-event models from the library. These models also comprise low-level controllers in order to assure that the most important safety and security requirements are respected. Therefore, by using the component library, elementary safety properties are guaranteed, and a good percentage of errors resulting from inadequate modeling of system components can be eliminated.

Obtaining the component models from the library, the user has to define their interactions, i.e. give a specification how the given components should co-operate in order to fulfill the global requirements against the supervised system. Even though the models are already defined, in case of large-scale systems this task is cumbersome. The proposed framework supports this procedure by the introduction of functional models, which depict only the important functional details of the operation of a component, and therefore are much simpler and smaller then ordinary technological models. Moreover, the control engineer needs to keep in mind only the most important functional properties of components and does not obliged to deal with technological details.

The user gives the specifications of co-operation of components based on their functional models and generates a supervisor which assures that the functionality of the closed loop system meets the requirements stated in the specification. If the system is very complex, it can be first decomposed to a few independent subsystems, and the procedure can be carried out for them. Finally, from these subsystems the model of the whole system can be assembled and the final supervisor can be synthesized.

D. The need for adequate modeling methodology

The former sections depicted the general overview of the framework, using the notations of technological and functional representations of component models. It is clear that since the basis of the framework is the use of such models, they have to be well-defined and their properties have to be verified theoretically. The following sections give formal definitions of the aforementioned models and some of their most important properties will be presented.

IV. The technological representation

A. The process model

The model on which all other models are based on gives a purely technological representation of the component. It is defined by the control engineer and depicts all possible operations the given component can carry out. It contains only physical constraints but not the limitations which should be respected by using adequate control. Therefore, borrowing the term from the terminology of classic control theory, this model should be referred to as the *process model* of the component and defined by the following.

Definition 1: The process model of a component is a DES containing the discrete event model of the physical representation of the component and is given by the 5-tuple $G^P = \{Q^P, \Sigma^P, \rho^P, q_0^P, Q_m^P\}$.

Remark There is no aid by the framework for obtaining an adequate process model, so it needs the experience and knowledge of the control engineer.

B. Specifications for Safety, Security and Liveliness

In practice there are requirements of safety, security and liveliness which shall be considered 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 process model, but which cause instable or dangerous behavior of the component. An example is the activation of two counter-effect actuators, which shall not be active the same time. These specifications are referred to as *Specifications of Safety, Security and Liveliness (S3L)* and can be modeled by a DES.

Definition 2: The Specifications for Safety, Security and Liveliness (S3L) are given by the following 5-tuple: $E^{S3L} = \{Q^{S3L}, \Sigma^{S3L}, \rho^{S3L}, q_0^{S3L}, Q_m^{S3L}\}$.

C. The IO 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 process model with respect to the specifications of safety, security and liveliness. It is clear that the operation of a DES generating the language $L(G^P)^{\uparrow E^{S3L}}$ (or a sublanguage of it) will respect the limitations represented by E^{S3L}.

Definition 3: The IO model of a component is the minimal generator of the language $L(G^P)^{\uparrow E^{S3L}}$ and is represented by the 5-tuple $G^{IO} = \{Q^{IO}, \Sigma^{IO}, \rho^{IO}, q_0^{IO}, Q_m^{IO}\}$. The language generated by the IO model will be referenced as L^{IO} in the followings.

The resulting IO 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 IO model is an ideal candidate to build a functional model on.

V. From technological to functional representation

A. Tasks

The way from technological to functional representations is paved by the tasks. The main idea of the model reduction strategy presented in this paper is the use of tasks. At first, it will be informally explained what these tasks are, afterwards formal definitions will be given.

If one observes the common behavior 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 *task*s. For example, a task of a robotic manipulator can be a movement from position A to position B, which comprises movements of particular joints in a given order. However, these operations correspond not just output patterns but can also depend on sensor inputs, e.g. movement of a joint should be stopped when a given angular position measured by an encoder is reached.

Formally we can define a task as follows. At first, let us define the part of the IO model to be included in the task.

Definition 4: The j^{th} task core model is defined by a 5-tuple $G_j^{Tc} = \{Q_j^{Tc}, \Sigma_j^{Tc}, \rho_j^{Tc}, q_{0,j}^{Tc}, Q_{m,j}^{Tc}\}$. The state and event set of the task core model are subsets of the state and event set of the IO model, i.e. $Q_j^{Tc} \subseteq Q_j^{IO}$ and $\Sigma_j^{Tc} \subseteq \Sigma_j^{IO}$. The transition function is defined as follows:

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

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

$$Q_{m,j}^{Tc} = \{q \in Q_j^{Tc} | \nexists \rho(q, \sigma)\}$$

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. On the other hand, the confirmation event is defined to be uncontrollable. Then the task, containing the task core model can be defined as follows.

Definition 5: The task alphabet is the set of all task events: $\Sigma^T = \bigcup_j \{\sigma_j^{start}, \sigma_j^{conf}\}$.

Using the task events and the task core model the task model can be defined as follows.

Definition 6: The model of the j^{th} task is the 5-tuple $G_j^T = \{Q_j^T, \Sigma_j^T, \rho_j^T, q_{0,j}^T, Q_{m,j}^T\}$. The state set of the task model is $Q_j^T = q_0' \cup Q_j^{Tc}$ 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. $q_{0,j}^T = q_0'$. The transition function is defined as follows:

$$\begin{aligned} \rho_j^T(q_{0,j}^T, \sigma^{start_j}) &= q_{0,j}^{Tc} \\ \rho_j^T(q_{0,j}^T, \sigma) &= q_0^{Tc_j} \forall \sigma \in \Sigma^{Tc_j} \\ \rho_j^T(q, \sigma^{conf_j}) &= q_{0,j}^T, q \in Q_m^{Tc_j} \end{aligned}$$

For other states:

$$\rho_j^T(q, \sigma) = \rho^{Tc_j}(q, \sigma), q \in Q^{Tc_j}, \sigma \in \Sigma^{Tc_j}$$

The only marking state of the task model is its initial state i.e. $Q_{m,j}^T = q_{0,j}^T$. The language generated by the task model is $L(G_j^T) = \sigma_j^{start}.L_{m,j}^{Tc}.\sigma_j^{conf}$

The definition of the tasks, i.e. the selection of task core models is left to the control engineer. However, it is important that the tasks cover well the IO model.

Definition 7: The IO model of a component is said to be well covered by the tasks if $\bigcap_j P_{\Sigma_{IO}}^{-1}(L(G_j^{Tc})) \supseteq L^{IO}$ and $L_j^{Tc} \nsubseteq L_k^{Tc}$ for $j \neq k$.

B. The bridge between technological and functional representations

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 IO model.

Definition 8: The model B' represented by the 5-tuple: $G^{B'} = \{Q^{B'}, \Sigma^{B'}, \rho^{B'}, q_0^{B'}, Q_m^{B'}\}$ is obtained by the parallel composition $G^{B'} = G^{IO} \|_j G_j^T$. The language generated by $G^{B'}$ is $L^{B'}$.

The model defined afore is however not suitable. It allows launching multiple tasks at the same time and therefore can result in blocking. So a suitable specification has to be defined in order to launch only one task a time. This property can be assured by the following specification.

Definition 9: The Task alternation Specification (TAS) is a 5-tuple $E^T = \{Q^{E^T}, \Sigma^{E^T}, \rho^{E^T}, q_0^{E^T}, Q_m^{E^T}\}$. The state set contains two states: $Q^{E^T} = \{q_0^{E^T}, q_1^{E^T}\}$, and the former one is defined to be the initial state. The event set of the specification is $\Sigma^{E^T} = \Sigma^T \cup \Sigma^{IO}$. The partial transition function ρ^{E^T} is defined as follows:

$$\rho^{E^T}(q_0^{E^T}, \sigma) = \begin{cases} q_1^{E^T} & \forall \sigma \in \bigcup_j \sigma_j^{start} \\ q_0^{E^T} & \forall \sigma \in \Sigma^{IO} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\rho^{E^T}(q_1^{E^T}, \sigma) = \begin{cases} q_0^{E^T} & \forall \sigma \in \bigcup_j \sigma_j^{conf} \\ q_1^{E^T} & \forall \sigma \in \Sigma^{IO} \\ \text{undefined} & \text{otherwise} \end{cases}$$

The only marking state is the initial state, i.e. $Q_m^{E^T} = q_0^{E^T}$.

The specification above assures that only one task can be active at the same time, while the model B' describes a behavior in accordance with the IO model. Therefore a suitable integral model can be obtained as the system generating the supremal controllable sublanguage respecting the task alternation specification. In order to avoid confusion of the integral model and the IO model, we shall denote the former one by the index B.

Definition 10: The integral model is the generator of the language $L^B = L(B')^{\uparrow E^T}$ and is defined by the a 5-tuple $G^B = \{Q^B, \Sigma^B, \rho^B, q_0^B, Q_m^B\}$, where $\Sigma^B = \Sigma^{IO} \cup \Sigma^{IO}$.

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 realizes the operation of the IO model. 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 IO events is the language of the IO model: $P_{\Sigma_{IO}}(L^B) = L^{IO}$.

Proof: At first it will be shown that $P_{\Sigma_{IO}}(L^B) \supseteq L^{IO}$. Due to the definition of the tasks, $L(G_j^{Tc}) \subseteq L^{IO}$. Then $P^{-1}_{\Sigma_{IO}}(L(G_j^{Tc})) \supseteq L^{IO}$, and therefore it is also true that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^{Tc})) \supseteq L^{IO} \tag{1}$$

On the other hand, it also follows from Definition 6 that

$$L(G_j^T) = \overline{\sigma_j^{start}.L(G_{j,m}^{Tc}).\sigma_j^{conf}}. \tag{2}$$

Therefore,

$$\begin{aligned}&P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T)) = \\ &= \overline{P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\sigma_j^{start}).P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_{j,m}^{Tc})).P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\sigma_j^{conf})}\end{aligned} \tag{3}$$

Since $\sigma_j^{start} \notin \Sigma_{IO}$ it follows that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\sigma_j^{start}) \supset \Sigma_{IO}^* \tag{4}$$

Likely,

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\sigma_j^{conf}) \supset \Sigma_{IO}^*. \tag{5}$$

Also,

$$L^{IO} = \overline{L_1^{IO}.L(G_j^{Tc}).L_2^{IO}}, \tag{6}$$

where $L_1^{IO} \subseteq \Sigma_{IO}^*$ and $L_2^{IO} \subseteq \Sigma_{IO}^*$.

Therefore it follows that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T)) \supset L^{IO}. \tag{7}$$

so

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T)) = L^{IO} \cup L_j^c. \tag{8}$$

for some $L_j^c \subseteq (\Sigma_{IO} \cup \Sigma_T)^*$. Therefore,

$$\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T)) = \bigcap_j (L^{IO} \cup L_j^c) \supset L^{IO}, \tag{9}$$

so since $P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^{IO}) \supset L^{IO}$ it follows that

$$L^{B'} = \bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T)) \cap P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^{IO}) \supset L^{IO} \tag{10}$$

and therefore

$$L^{B'} = L^{IO} \cup L^C \tag{11}$$

where $L^C \subseteq (\Sigma_{IO} \cup \Sigma_T)^*$. Since $(K_1 \cup K_2)^{\uparrow} \supseteq K_1^{\uparrow} \cup K_2^{\uparrow}$ and due to Definition 9 $L^{IO\uparrow} = L^{IO}$, it follows that

$$L^B \supseteq L^{IO} \tag{12}$$

and therefore

$$P_{\Sigma_{IO}}(L^B) \supseteq L^{IO}. \tag{13}$$

Now it will be shown that $P_{\Sigma_{IO}}(L^B) \subseteq L^{IO}$. It follows from Definition 8 that

$$L^{B'} = P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\bigcap_j L(G_j^T)) \cap P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^{IO}). \tag{14}$$

Therefore,

$$\begin{aligned}&P_{\Sigma_{IO}}(L^{B'}) = \\ &= P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\textstyle\bigcap_j L(G_j^T)) \cap P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^{IO})),\end{aligned} \tag{15}$$

so

$$\begin{aligned}&P_{\Sigma_{IO}}(L^{B'}) \subseteq \\ &\subseteq P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\bigcap_j L(G_j^T)))\cap P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^{IO})).\end{aligned} \tag{16}$$

Considering the first part of the right side of (16), one can assume the followings.

$$P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(\bigcap_j L(G_j^T)) \subseteq \bigcap_j P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T))). \tag{17}$$

Also,

$$P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^T))) \subseteq P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(P_{\Sigma_{IO}}(L(G_j^T))). \tag{18}$$

It follows from (2) that

$$\begin{aligned}&P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(P_{\Sigma_{IO}}(L(G_j^T))) = \\ &= \overline{P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(P_{\Sigma_{IO}}(\sigma_j^{start}).P_{\Sigma_{IO}}(L_{m,j}^{Tc}).P_{\Sigma_{IO}}(\sigma_j^{conf}))}.\end{aligned} \tag{19}$$

Since $\sigma_j^{start} \notin \Sigma_{IO}$,$\sigma_j^{conf} \notin \Sigma_{IO}$ and $L(G_j^{Tc}) \subseteq \Sigma_{IO}^*$,

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(P_{\Sigma_{IO}}(L(G_j^T))) = P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^{Tc})). \tag{20}$$

and therefore

$$\begin{aligned}&P_{\Sigma_{IO}}(L^{B'}) \subseteq \\ &\subseteq \textstyle\bigcap_j P^{-1}_{\Sigma_{IO}\cap\Sigma T}(L(G_j^{Tc})) \cap P_{\Sigma_{IO}}(P_{\Sigma_{IO}\cup\Sigma_T}(L^{IO})).\end{aligned} \tag{21}$$

By considering the properties of $L(G_{m,j}^{Tc}) = \overline{L(G_{m,j}^{Tc})}$, it follows that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^{Tc})) \supset P^{-1}_{\Sigma_{IO}}(L(G_j^{Tc})) \supseteq L^{IO}. \tag{22}$$

Since

$$\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G_j^{Tc})) \supseteq L^{IO} \tag{23}$$

and

$$P_{\Sigma_{IO}}(P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G^{IO}))) = L^{IO} \tag{24}$$

it follows that

$$P_{\Sigma_{IO}}(L^{B'}) \subseteq L^{IO}. \tag{25}$$

Therefore, since $L^B \subseteq L^{B'}$ it follows that

$$P_{\Sigma_{IO}}(L^B) \subseteq L^{IO}. \tag{26}$$

(13) and (26) imply together that $P_{\Sigma_{IO}}(L^B) \subseteq L^{IO}$

■

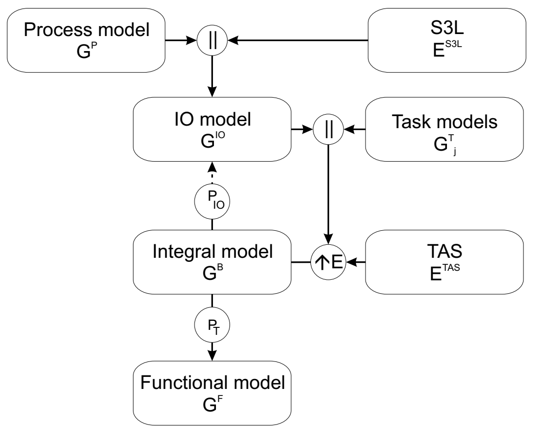

Fig. 1. Overview on the different models of a component

C. The functional model

The functional behavior 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 liveliness) 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.

Definition 11: The functional model is the generator of the language $L^F = P_{\Sigma^T}(L^B)$ and is defined by the 5-tuple $G^F = \{Q^F, \Sigma^F, \rho^F, q_0^F, Q_m^F\}$ where $\Sigma^F = \Sigma^T$.

Remark Note that, according to Definition 6, Definition 10 and properties of projection, all states of the functional model which are entry states of transitions corresponding the confirmation events are marking ones. It depicts the fact that all tasks correspond to the completion of a given operation.

The following theorem plays an important role.

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:
$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \cap (\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G^T_j))) = L^B$

Proof: The proof is based on three properties. At first, it follows from the definition of the task alternation specification that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \subset E^T \tag{27}$$

It can be also shown that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \cap (\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G^T_j))) \subseteq L^{B'} \tag{28}$$

Note that it follows from (27) and (28) that $P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \cap (\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G^T_j)))$ is a controllable sublanguage of $L^{B'}$ regarding the specification E^T.

Since $P^{-1}_{\Sigma_1\cup\Sigma_2}(K) \supseteq P^{-1}_{\Sigma_1}(K)$ and $P^{-1}_{\Sigma}(P_\Sigma(K)) \supseteq K$, by considering Definition 11 it follows that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \supseteq L^B \tag{29}$$

However, Definition 10 states that L^B is the supremal controllable sublanguage of $L^{B'}$, so

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \cap (\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L(G^T_j))) \subseteq L^B \tag{30}$$

It follows from (29) and (30) that

$$P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^F) \cap (\bigcap_j P^{-1}_{\Sigma_{IO}\cup\Sigma_T}(L^T_j)) = L^B \tag{31}$$

■

The following situation underlines the importance of the theorem. One can consider the control structure as a distributed and hierarchical one, where the plant is controlled by several low-level controllers, each realizing one of the task models. These low-level controllers have two interfaces: one for the process with events $\sigma_L \in \Sigma_{IO}$ and one for the high-level controller with $\sigma_H \in \Sigma_T$.

The operation of the controller structure can be depicted as follows. The plant is influenced directly by the low-level controllers, realizing G^T_j, and therefore assuring that the closed loop meets safety, security and liveliness specifications. The high-level supervisor, influencing the low-level ones, can be synthesized by using G^F as a model. As it has been shown, the operation of the plant will be the same as using the conventional way for synthesizing a monolithic controller.

VI. Conclusion

Multi-face modeling approach presented in this paper can serve as the basis of a new framework for the rapid prototyping of discrete event controller structures. Definition of functional models besides technological ones allows easier controller design supported by a component library. Presented theorems can guarantee that the supervisor synthesized for the functional model will fulfil the same requirements that the supervisor synthesized for the original system, i.e. the process model. Based on these models a suitable supervisory control design methodology can be defined.

Future work include the definition of the model library and the description of the algorithms for automatic code

ACKNOWLEDGEMENTS

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

References

[1] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Boston: Kluwer Academic Publishers, 1999.

[2] K. Wong and W. Wonham, “Hierarchical control of discrete-event systems,” *Discrete Event Dynamic Systems: Theory and Applications*.

[3] W. Wonham and P. Ramadge, “Modular supervisory control of discrete event systems,” pp. 13–30, 1988.

[4] W. Wonham, *Notes on Control of Discrete Event Systems*. University of Toronto, 2002.

[5] J. Hopcroft and J. Ulmann, *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[6] R. Kumar, V. Garg, and S. Marcus, “On controllability and normality of discrete event systems,” *Systems & Control Letters*, vol. 17, pp. 157–168, 1991.

[7] R. Brandt, V. Garg, R. Kumar, F. Lin, S. Marcus, and W. Wonham, “Formulas for calculating supremal controllable and normal sublanguages,” *System & Control Letters*, vol. 15, pp. 157–168, 1990.