Modular Synthesis of Supervisors Based on a Petri Net Approach

Giovanni C. Barroso†, Antonio M. N. Lima‡ e Angelo Perkusich†
†Departamento de Física
Universidade Federal do Ceará
‡Departamento de Engenharia Elétrica
Universidade Federal da Paraíba
Av. Aprígio Veloso, 882 - 58109-970 - Campina Grande - PB
e-mail: (gcb,marcus,perkusic)@dee.ufpb.br

Abstract

This paper presents a modular synthesis approach based on a class of modular Petri nets named G-Nets, conceived to be employed with the Supervisory Control Theory (SCT). Two algorithms are presented, and the supervisor synthesis is obtained by processing both the system and specification models through these algorithms. These algorithms make possible to obtain the controller of a discrete event system based on a given specification. Moreover its simplicity and efficiency are demonstrated using a typical manufacturing cell problem.
1 Introduction

Over the last years the use of the Supervisory Control Theory (SCT) in the design of the controllers for Discrete Event Systems (DES), has rapidly increased. As a result, the need for new tools and methods for modeling and implementing such systems, is also increasing. Among other formalisms, Petri nets (PN) [Mur89] have been widely used as a powerful tool in the analysis and design of discrete event systems [GD94, Sre93]. In general, the controller synthesis is achieved by changing the model structure. In most cases the structure modification does not follow any standard procedure.

This paper presents an approach to the supervisors synthesis, using the SCT, and an extension of a class of Petri nets, called Petri Nets with Transition Enabling Functions (PNTEF) [PC92]. It is introduced a modular formalism to construct the controller based on the G-Net systems [Per94]. Colored Petri Nets [Jen92], are used in the internal structure of each module. Two algorithms based on the well known reachability tree [Mur89] are presented in order to synthesize the controller of a DES.

The present paper is organized as follows: In Section 2 the notational conventions and the basic definitions of supervisory control theory, G-Net systems, and PNTEFs are presented. In Section 3 the proposed algorithms to synthesize the discrete event controller are presented. In Section 4 an example based on a manufacturing cell is presented. And finally in Section 5 the main conclusion of the paper are presented.

2 Background Issues

2.1 Supervisory Control Theory

In the SCT [RW89], the system behavior is represented by a 5-tuple \( G = (\Sigma, Q, \delta, q_0, Q_m) \), called generator. The symbol \( \Sigma \) represent the set of event labels, or event alphabet; \( Q \) is a set of states; \( \delta : \Sigma^* \times Q \rightarrow Q \) is the transition function. The set of all strings formed by any number of symbols from \( \Sigma \), including the empty symbol \( \epsilon \), is denoted by \( \Sigma^* \).

A state \( q \) is said to be accessible iff \( \exists s \in \Sigma^* | \delta(s, q_0) = q \). A state \( q \) is said to be coaccessible iff \( \exists s \in \Sigma^* | \delta(s, q) \in Q_m \). \( G \) is trim iff it is accessible and coaccessible.

Each generator \( G \) has two associated languages: \( L(G) \) is the language generated by \( G \) and \( L_m(G) \) is the language marked by \( G \). The language \( L(G) \) represents the physically possible behavior of the system, while \( L_m(G) \) represents the tasks it is able to complete.
To control a DES it is necessary to admit that some events may be disabled when desired. To model such control action, the $\Sigma$ set is partitioned into: i) $\Sigma_c$ the controllable event set and ii) $\Sigma_u$ the uncontrollable event set, such that $\Sigma = \Sigma_c \cup \Sigma_u$ and $\Sigma_c \cap \Sigma_u = \emptyset$. All the events in $\Sigma_c$ may be disabled at any time.

A control input for $G$ consists of a subset $\gamma \subseteq \Sigma$, satisfying $\Sigma_u \subseteq \gamma$. If $\sigma \in \gamma$, then $\sigma$ is enabled by $\gamma$, otherwise, $\sigma$ is disabled. The condition $\Sigma_u \subseteq \gamma$ means that the events in $\Sigma_u$ are always enabled.

Let $\Gamma \subseteq 2^\Sigma$ denote the set of control inputs. A DES represented by $G$, with a set of control inputs $\Gamma$ is called a controlled DES (CDES). For convenience, one refers to a CDES by its underlying generator $G$.

Controlling a CDES $G$, consists of generating a sequence of elements $\gamma, \gamma', \gamma'', \ldots$ in $\Gamma$, in response to the previously observed events generated by $G$. Such a controller will be called a supervisor.

A supervisor is a map $f : L \rightarrow \Gamma$, specifying for each possible string of generated events $w$, the control input $f(w)$ to be applied.

A supervisor is represented by an automaton and an output map, i.e.: $S = (\Sigma, X, \xi, x_0)$ is an automaton and $\psi : X \rightarrow \Gamma$ is the output map.

One says that the pair $(S, \Psi)$ realizes the supervisor $f$ if for each $w \in L(G/f)$, $\Psi(\xi(w, x_0)) = f(w)$, where $L(G/f)$ represents the closed behavior of the composed system $G$ supervised by $f$.

The basic supervisory control problem may be stated as follows: Given a DES $G$ with open-loop behavior given by $L$, what closed-loop behavior $K \subseteq L$ can be achieved by supervision? To solve this problem, it is necessary to define the concept of controllability.

Given two arbitrary languages $L$, $K \subseteq \Sigma^*$ and an alphabet $\Sigma = \Sigma_c \cup \Sigma_u$, one says that $K$ is $L$-closed iff $K = \bar{K} \cap L$, and $K$ is $L$-controllable iff: $\bar{K} \Sigma_u \cap L \subseteq \bar{K}$, where $K$ is a prefix of $\overline{K}$, i.e., the set of all prefixes of the strings in $K$.

Given a generator $G$, the language $K \subseteq L_m(G)$ models the desired specification. The language $K$ represents the tasks to be executed under supervision. Then, the goal is to find a proper supervisor $S$ to $G$ so that the closed-loop system satisfies the condition $L(S/G) = K$.

There exists a proper supervisor so that $L(S/G) = K$ iff $K$ is $L_m(G)$-closed and $L(G)$-controllable [RW89]. When these conditions are not satisfied, it is always possible to meet a supremal controllable sublanguage $K^\uparrow$ such that $K^\uparrow \subseteq K$. In the case of finite state generators, $K^\uparrow$ is always computable [RW89].
2.2 G-Nets and G-Net Systems

Deng and Perkusich [DCdFP93, PdFM93] introduced the concept of G-Nets and G-Net systems. G-Nets are a Petri net based framework for the modular design and specification of distributed information systems. The framework is an integration of Petri net theory with the object oriented software engineering approach for system design. The motivation of this integration is to bridge the gap between the formal treatment of Petri nets and a modular, object-oriented approach for the specification and prototyping of complex software systems. The G-Net notation incorporates the notions of module and system structure into Petri nets, and promotes abstraction, encapsulation and loose coupling among the modules.

A specification or design based on G-Nets consists of a set of independent and loosely-coupled modules (G-Nets) organized in terms of various system structures. A G-Net is encapsulated in such a way that a module can only access another module through a well defined mechanism called G-Net abstraction, avoiding interference in the internal structure of another module.

A G-Net GN, is composed of two parts: a special place called Generic Switching Place (GSP) and an Internal Structure (IS). The GSP provides the abstraction of the module, and serves as an interface between the G-Net and other modules. The internal structure is a modified Petri net, and represents the detailed internal realization of the modeled application. The notation for G-Nets is very close to the Petri net notation [Mur89]. Among other features this notation allows the user to indicate communication among G-Nets and termination. The notation for G-Nets is shown in Figure 1. The IS of the net is enclosed by a rounded corner rectangle, defining the internal structure boundary. The GSP is indicated by the ellipse in the left upper corner of the rectangle defining the IS boundary. The inscription GSP(net_name) defines the name of the net to be referred by other G-Nets. The rounded corner rectangle in the upper right corner of the IS boundary is used to identify the methods and attributes for the net, where: \(\text{attribute.name} = \{\text{type}\}\) defines the attribute for the net where: \(\text{attribute.name}\) is the name of the attributes, and \(\text{type}\) is a type for the attribute; \(\text{method.name}\) is the name for a method; \(\text{description}\) is a description for the method, \((p1 : \text{description}, \cdots, pn : \text{description})\) is a list of arguments for the method. Finally, \(\text{sp}\) is the name of the initial place for the method. A circle represents a normal place. An ellipse in the internal structure represents an instantiated switching place (isp). The isp is used to provide inter-G-Net communication. The inscription isp(GN'.mi) indicates the invocation of the net GN' with method mi. A rectangle represents a transition, that may
have an inscription associated with it. This inscriptions may be either an attribution or a firing restriction. We will use the standard Language C notation for both attributions and firing restrictions. A double circle represents the termination place or goal place. Places and transitions are connected through arcs that may carry an expression.

The GSP of a G-Net GN, denoted by GSP(net.name) in the ellipse of Figure 1, uniquely identifies the module. The rounded-corner rectangle in the GSP side contains a description of one or more methods, which specify the functions, operations or services defined by the net, and a set of attributes specifying the passive properties of the module (if any). The detailed structures and information flows of each method are defined by a modified high-level net in the internal structure. More specifically, a method defines the input parameters, the initial marking of the corresponding internal high-level net (the initial state of the execution), that in this paper will be a Colored Petri Net. The collection of the methods and the attributes (if any) provides the abstraction or the external view of the module. For details about the formal definition an other details can be found in [Per94, DCdFP93, PdFM93]

2.3 PNTEF

A Petri net with transition enabling functions is a 4-tuple PNTEF = (N, I, M₀, Φ), where: N = (P, T, I, O) is a structure of a place/transition Petri net, where: P = {p₁, ..., pₙ} is a finite set of places, represented by circles; T = {t₁, ..., tₘ} is a finite set of transitions represented by bars or boxes; P ∩ T = ∅ and P ∪ T ≠ ∅. I : P × T → ℕ is the input function that specifies the arcs directed from places to transitions (ℕ = {0, 1, 2, ...}); O : T × P → ℕ is the output function that specifies the arcs directed from transitions to places; l : T → Σ is a function that labels the transitions with symbols of the alphabet Σ; M₀ is the initial marking;
$\Phi = (\varphi_1, ..., \varphi_m)$ is a set of logical expressions associated with the transitions.

A transition $t_j$ is enabled in a $\text{PNTEF}$, given a marking $M'$, if: $M'(p_i) \geq I(p_i, t_j), \forall p_i \in P$; the logical expression $\varphi_j$ associated to $t_j$ is truth; the firing of $t_j$, enabled by $M'$, generates a new marking $M''$, so defined: $M''(p_i) = M'(p_i) + O(t_j, p_i) - I(t_j, p_i)$

3 Supervisor Synthesis Algorithms

In this paper the synthesis of the supervisor is obtained executing the two algorithms presented in this section and introduced in [BLP95]. The first one is a modified version of the reachability tree algorithm presented in [Mur89]. The second algorithm is derived from the one proposed by Ziller [ZC94].

3.1 Modified Reachability Tree

For modeling physical systems, it is natural to consider an upper bound for the number of tokens that each place can hold. A PN with this constraint is called finite capacity Petri net, and not all the markings are present in its reachability tree [Mur89]. Some information is lost when one uses the symbol $w$ to replace the markings where the number of tokens tends to grow infinitely.

Algorithm 1 Modified Reachability Tree Algorithm for finite capacity PNs.

Begin

1. Label the initial marking $M_0$ as the root and tag it new;

2. While new markings exist do:

   (a) Select a new marking $M$;

   (b) If $M$ is identical to a marking that already exists, tag it old;

   (c) If no transition is enabled at $M$, tag it as dead;

   (d) While enabled transitions exist at $M$, do the following for each enabled transition:

       i. Obtain the marking $M'$ that results from firing $t$ at $M$;

       ii. If there exists a marking $M''$ such that the capacity of a place is exceeded, then replace $M'(p)$ by $w$ for each $p$ such that its capacity is exceeded;

       iii. Introduce $M'$ as a node, draw an arc with label $t$ from $M$ to $M'$ and tag $M'$ not-permitted if the capacity of a place is exceeded, otherwise tag it new.

End
3.2 Algorithm to Construct a Generator for $supC(L)$

Different algorithms have been proposed to compute $supC(L)$. Generally these algorithms require the computation of the $trim$ components of $G$. If the system and the specification are modeled by finite automata, then the $supC(L)$ exists [RW89]. A new algorithm to find the $trim$ component of a system being modeled by a finite capacity PN and then compute the $supC(L)$, is introduced. To compute $supC(L)$ (item 5 of the algorithm) it is applied the algorithm presented in [ZC94].

**Algorithm 2 Algorithm to construct a generator for $supC(L)$ from a PN generator:**

*Begin*

1. Create a dynamic list of states, $block.list$, and initialize it with the blocking states (blocking markings);

2. Add to the list the states that the only output event (transition) is input of a blocking state;

3. Add to the list the states that have, at least one output transition, labeled by an uncontrollable event, that is an input transition of a blocking state;

4. Create a dynamic list of states and events, $danger.list$, and initialize it with the $block.list$ ancestors (states) together with the events that link $block.list$ states to $danger.list$ states, since the ancestor does not belong to the $block.list$;

5. Given the desired system specification, compute the supremal controllable sublanguage $supC(L)$;

6. Add to the $danger.list$ the states and their respective output events that will be disabled to execute the $supC(L)$, since these states do not belong to the $danger.list$.

*End*

4 Application to a Manufacturing Cell

Considering a manufacturing cell, as shown in Figure 2(a), it is possible to identify two raw-material deposits RM1 and RM2; two processing centers MP1 and MP2; and an assembling center A. A robot R moves raw-material from deposits RM1 and RM2 to the processing centers MP1 e MP2, respectively, and from these centers to the assembling center A. Figure 2(b) shows the $G$-Net system model for this cell. We use a colored Petri net [Jen92] to implement the internal structure of each $G$-Net. For this example, each two assembled parts
Figure 2: (a) Example of a Manufacturing Cell; (b) G-Net system model of the cell

constitute an item of this cell. Items are then transported to another cell by an external agent, and this action is not modeled.

The maximum number of parts that the assembling center A supports is two. These parts are enough to assemble an item, i.e., these are the upper bound in the number of tokens that place \( I_a \) of \( GN(A) \) may hold in any marking of the net.

Let \( \Sigma = \Sigma_p \cup \Sigma_r \cup \Sigma_a \), where \( \Sigma_p = \{ \alpha_1, \alpha_2, \alpha_3, \alpha_4 \} \), \( \Sigma_r = \{ \beta_1, \beta_2, \beta_3 \} \), \( \Sigma_a = \{ \gamma_1 \} \), and \( \Sigma_e = \{ \alpha_1, \alpha_3 \} \) and \( \Sigma_u = \{ \alpha_2, \alpha_4, \beta_1, \beta_2, \beta_3, \gamma_1 \} \). For the meaning of the \( \alpha_i \), \( \beta_i \) and \( \gamma_i \) events, the reader should check Figure 2(b).

Consider the following specifications: All parts must be processed and assembled; the assembling center A may hold only two parts at a time; an item is assembled with a part from MP1 and other from MP2; a part from MP1 must be sent first to the assembling center A.

Considering that only the events \( \alpha_1 \) and \( \alpha_3 \) are controllable in this cell, it is necessary to execute the Algorithms 1 and 2 only to the \( G-Net \) \( GN(\text{MAQ}) \). After executing Algorithm 1,
Figure 3: Table of markings and their respective events to be disabled

the modified reachability tree for the $GN(MAQ)$ model of the manufacturing cell, is obtained. This tree has 147 states or markings. Here it is presented only that markings that are found after running Algorithm 2.

When executing the $G$-Net system and a token of type $rm_i$ ($i = 1, 2$) reaches the place $isp(st)$, the net $GN(R)$ is invoked with method $st$ and a token is put in the place $I_r$ of $GN(R)$. When the transition $tr_2$ fires, a token is put in the place $isp(A.it)$, then the net $GN(A)$ is invoked and a token is put in the place $I_a$ of $GN(A)$. This means that a part was transported from a processing center to the assembling center $A$ by the robot.

Note that transition $t_4$ of $GN(MAQ)$ can not fire unless an acknowledge is received when the goal place $GP_2$ of the net $GN(R)$ is reached. Moreover, the transition $tr_3$ in $GN(R)$ only can fire when the goal place $GP_a$ in the net $GN(A)$ is reached. This happens when an item is assembled. Once a while the place $isp(st)$, in $GN(MAQ)$, contains a copy of each token (part) that was sent to $GN(A)$.

Thus, to execute the specifications, it is necessary that: the goal place $GP$ in $GN(MAQ)$ be reached for all tokens put in the place $p_2$ when $GN(MAQ)$ is initialized (this guarantees that all parts were sent to the assembling center); to avoid the place $isp(st)$ of holding more than one part from MP1 (tokens of type $rm_1$); and that a part from MP2 (tokens of type $rm_2$) be only put in the $isp(st)$ when there already exists a token of type $rm_1$ there. Finally, it is necessary to avoid more than two tokens in the $isp(st)$.

We then execute the Algorithm 2 to find out if the specifications above are possible or not. Executing Algorithm 2, the danger list, as it is shown in Figure 3, is constructed. Thus, all the states and their respective transitions to be disabled when the system is in these states,
are presented. On basis of the danger_list, we may infer that the transition $t_3$ (event $\alpha_3$) in $G(N/MAQ)$ have to fire with token of type $mp_1$ only if there is no token in the place $isp(R.st)$. In the same way, event $\alpha_3$ may occur with token of type $mp_2$ only if the place $isp(R.st)$ holds only one token of type $rm_1$. In any other situation, this transition will be disabled, as shown by the respective expressions associated to the transition $t_3$ ($\alpha_3$) of the $G-Net$ supervisor, as shown in Figure 4.

5 Conclusions

This paper presented an extended Petri net model with transition enabling functions. This extension was used in modeling and synthesis of supervisors. A modified reachability tree algorithm was also presented. This algorithm allows obtaining all the markings in a not modified finite capacity PN model. Finally, an algorithm to the construction of a $supC(L)$ generator was presented. This algorithm is executed based on the modified reachability
tree of the system PN model, and enumerates all the states and respective output events to be disabled. Using the PNTEF model and the enumerated algorithms, it was proposed a systematic approach to solve the supervisory synthesis problem, with no modification in the structure of the net that models the system.

On the other hand, this approach do not solve the classic state explosion problem of DESs, once the modified reachability tree has been employed. To minimize this problem we employed a modular approach (G-Net systems) for the modeling, analysis and synthesis of the system and the controller. Using this approach, we simplify the construction of the supervisor because it is possible to restrict the analysis to that module or G-Net of interesting. Thus, instead of constructing the reachability tree of all the system to be controlled, it is necessary to construct only the reachability trees of the modules we are interested, then reducing drastically the problem of state explosion.

References


