A Formal Transparency Framework for Validation of Real-Time Discrete-Event Control Requirements Modeled by Timed Transition Graphs

Amrith Dhananjayan and Kiam Tian Seow

***Abstract*— In control of discrete-event systems, translating natural language control requirements into formal specifications in computable graphical form can be error prone, and system designers are often confronted with the longstanding problem of uncertainty in specification formalization, namely: how do we know that such a formalized specification is the one intended? This necessitates specification validation, i.e., manual inspection of the specification's graphical structure to clarify if it formalizes the intended requirement. The uncertainty is compounded in the specification formalization for timed discrete-event systems (TDESs) as timed transition graphs (TTGs), where real-time behavior also needs to be correctly specified. In the fundamental control framework of TDESs, a TTG prescribes a timed regulation of logical behavior restricting a TDES to some timed event-transition sequences. To help validate specification TTGs, we develop a new specification concept of TTG transparency. Our concept formulation embodies the essence of 'summarizing' a specification TTG's transition sequences for a TDES, to highlight intermittent transitions essential or relevant for comprehending the specification's non-trivial timed restrictions. The transparency concept governs the reconstruction of a specification TTG into a transparent one. We investigate the problem of maximizing the transparency of specification TTGs for TDESs and show that it is NP-hard. We then develop a polynomial time algorithm for computing a highly transparent TTG. Through two examples, we show that the transparent TTG computed may support specification validation.**

***Index Terms*— Timed discrete-event systems, human cognition, formal specification, transparency.**

I. Introduction

The control theory of timed discrete-event systems (TDESs) by Brandin and Wonham [1] and its extensions [2], [3], [4], [5] provide a control-theoretic framework for synthesizing real-time supervisors for event-driven systems to comply with specified control requirements. The elements of the theory, namely timed discrete-event systems, control requirements and supervisors are represented using finite automata in the form of timed transition graphs (TTGs). This TTG based framework is about the simplest known for modeling controlled TDESs. However, while computable, requirements formalized as specification TTGs would need to be manually validated in general. Manual validation refers to designer inspection of a specification's graphical structure, to ascertain and clarify if it indeed models the intended control requirement for a system. This paper is concerned with the problem of restructuring a specification TTG for clarity, in order to better comprehend or understand the real-time specification during validation.

In practice, control requirements for discrete-event systems (DESs), both untimed or logical [6], [7], [8], [9] and timed [1], are often first described in natural language (English) statements. Examples of such statements include: follow a first-come first-served policy, avoid deadlock, and stop a moving car within, say, 4 seconds after its brake is applied. System designers then prescribe specifications based on their understanding of these requirements for the system model. To harness the potential benefits ("guarantees to do the things right") of automata-based control synthesis from supervisory control theory, the specifications should first correctly formalize these statements [10] ("do the right things"). However, with these frameworks, the prescription task of formalizing a natural language description into a graphical specification is non-trivial and requires expertise in DES modeling [11]. Moreover, with DESs being designed, there are no invariant physical laws to constrain system configurations, often leading to complex system behavior [12]. These complicate the designer's specification formalization. Arbitrarily done, the mental translation from natural language requirements to formal graphical specifications can be tedious and error-prone [13], [14]; designers hardly get it correct on the first attempt [15]. In fact, many reported applications of the automata-based DES control theory [16], such as robotics [17], [18], automated manufacturing [19], [20], [21], [22], communication networks [23] and intelligent service transportation [24], have encountered such difficulties in prescribing specifications.

The specification prescription process should be carried out iteratively through manual formalization of natural language

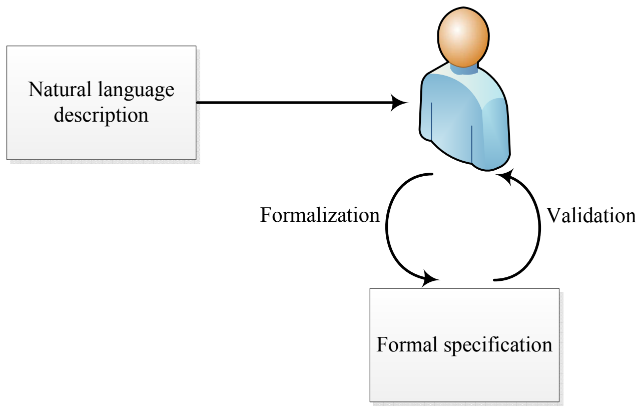


Fig. 1. Specification prescription process

This research work was supported by the Singapore Ministry of Education, under NTU-AcRF Tier 1 Grant No: RG65/07.

The work was done while the authors were with the School of Computer Engineering, Nanyang Technological University, Republic of Singapore. A. Dhananjayan is contactable at amri0005@e.ntu.edu.sg; and K.T. Seow is contactable at asktseow@ntu.edu.sg or ktseow@rit.kaist.ac.kr.

requirements and validation, until the system designer is confident that the prescribed specifications are as intended (Fig. 1). Manual validation of formal specifications is important, as detecting errors earlier in the system design process can save time and cost. However, without automated support tools to aid in human comprehension of formal specifications, manual validation can be a daunting task. System designers may be predisposed to presuming that formalized specifications are truthful and complete, and could find it difficult to ascertain the correctness of specifications, especially those that are incomplete [25]. The challenges of human comprehensibility may limit the use of automata-based supervisory control in solving industrial problems [26], [16].

To facilitate validation in the specification TTG formalization process, we present a framework with the purpose of enhancing specification comprehensibility. Theoretical frameworks to help resolve the uncertainty and enhance the comprehensibility of untimed specification automata for logical DESs have been developed in [27], [28] and [29], separately treating the event set and the state set as fundamental. In a parallel development for real-time specifications, we propose the new concept of transparency for specification TTGs, formulated by treating the timed state-event or transition space as fundamental since timed event evolution is explicitly modeled by transitions in TTGs [1]. Our concept formulation embodies the essence of ‘summarizing’ a specification TTG’s event-transition sequences for a TDES, to highlight intermittent transitions that may be relevant for supporting the comprehension of the specification’s non-trivial timed restrictions. Essentially, in jointly reachable specification TTG and system states, events that can occur next in the specification TTG, in synchrony with, but do not change the specification’s current restrictions on, the TDES, have their corresponding transitions deemed as specification irrelevant. Our concept embodiment ‘hides’ in self-loops the events of a specification TTG whose corresponding transitions are identified as specification irrelevant, and explicates all specification relevant transitions as diligent transitions (i.e., those connecting distinctly different states). In effect, the transparency concept governs the reconstruction of a specification TTG to reduce the number of states in it without removing any essential temporal or sequential prescription of the specified dynamics. The resulting specification TTG is said to be a transparent one that may be comprehensible or more readily so in general (in terms of its prescribed dynamics). Various automata-theoretic control studies [30], [31], [32], [33] have acknowledged, in one way or another, that by reducing the number of automaton states without removing any essential prescription of the control action, a supervisor automaton may be made more readily comprehensible in general (in terms of its control action). That the transparency concept supports specification comprehensibility is inferred from these related but different studies, in that, without removing any essential information of interest, state reduction may render an automaton more comprehensible in general.

The proposed specification transparency framework complements the one in [34] that translates a class of metric temporal logic (MTL) specifications to TTGs generating sublanguages of the (marked) language of a given TDES. While the latter framework gets a specification TTG right by construction, a designer may not always know if it is the right specification TTG for control synthesis without manually inspecting it. Transforming the translated specification TTG into a more transparent form with respect to the TDES may aid in comprehension of the TTG vis-à-vis the MTL formula, and facilitates complementing assurances of correctness from the graphical semantics and temporal syntactic views. Even for specification TTGs that do not have an MTL counterpart, the framework is still potentially useful, since a specification TTG hand-prescribed by one designer might not be readily understood by another designer.

The rest of the paper is organized as follows. Related work is discussed in detail in Section II. Section III reviews relevant concepts in TDES control theory. Section IV describes and formalizes the concept of timed specification transparency, states the formal problem of finding a maximally transparent specification TTG and proves its NP-hardness. As extended from [35], this paper presents a refined and more complete mathematical formulation of the transparency framework and mathematical proofs are given for the stated theorems. A provably correct polynomial time algorithm for computing transparent specification TTGs is proposed in Section V. Two illustrative examples, including one from a real-world application, are presented in Section VI to illustrate the concept of a transparent specification TTG. The discussion is presented in Section VII.

II. Related Work

Issues of specification comprehensibility are inherent in formal system design [36]. In the rudimentary automata-theoretic framework of discrete-event control, the concept of specification transparency is developed to mitigate the comprehensibility problem. A related work [29] for untimed DESs attempts to make specification automata more comprehensible from a state perspective, by showing the compliant execution of the DES through a minimum number of specification relevant states called specification epochs. In [27] and [28], attempts are made from an event perspective by highlighting the precedence ordering among a minimal set of events deemed relevant to the specification. This is done by projecting out [11] and ‘hiding’ in self-loops all events that are considered irrelevant to the specification but can occur in the DES. To simplify complex control specifications and reduce redundancy of information, [37] proposes that a part of the control requirements for a DES be embedded within the DES model in a process-theoretic framework. The resulting “partially-supervised” DES model would require only the remaining part of the requirements to be formalized, leading to simpler specifications.

This paper presents a transparency framework for TDES specifications modeled by TTGs [1]. Since timed event evolution in TTGs is explicitly modeled by interleaving transitions of events and the special *tick* event (denoting the passage of one unit of time), the timed transition transparency concept is necessarily formulated in the timed state-event or transition space.

Technically related are algorithms for state reduction of supervisors (e.g., [30], [31]) which may render their control action more readily comprehensible to designers. When applied to specifications, these algorithms may return specification TTGs that are easier to understand. However, whenever this happens, it is only coincidental as their main objective is to reduce the number of states required to express the same specification (by removing constraints already enforced by the TDES) for reasons of economy in implementation. While it may be more readily comprehensible as a supervisor automaton, the output of these state reduction algorithms is not always appealing to designers [26] as a specification automaton. The output may provide clarity as a by-product, but that invariably focuses only on the control action and not the prescribed dynamics that matters for understanding a specification.

III. Timed Discrete-Event Systems

In this paper, we use the TDES model proposed by Brandin and Wonham [1] as the formalism for modeling real-time systems. In the Brandin-Wonham framework, the base model of a TDES is a finite automaton $G_{act} = (A_{act}, \Sigma_{act}, \delta_{act}, a_0, A_m)$ called an *activity transition graph* (ATG). The untimed behavior of a TDES is represented by its ATG. In the ATG G_{act}, A_{act} is the finite set of activities, Σ_{act} is the finite set of events, $\delta_{act} : \Sigma_{act} \times A_{act}$ is the partial activity transition function, $a_0 \in A_{act}$ is the initial (starting) activity and $A_m \subseteq A_{act}$ is the set of marked activities.

Let **N** denote the set of natural numbers. Each event label $\sigma \in \Sigma_{act}$ is equipped with a *lower time bound* $l_\sigma \in \mathbf{N}$ and an *upper time bound* $u_\sigma \in \mathbf{N} \cup \{\infty\}$ such that $l_\sigma \leq u_\sigma$. Σ_{act} is partitioned into two subsets $\Sigma_{spe} = \{\sigma \in \Sigma_{act} \mid u_\sigma \in \mathbf{N}\}$ and $\Sigma_{rem} = \{\sigma \in \Sigma_{act} \mid u_\sigma = \infty\}$, denoting the sets of prospective and remote events, respectively. A prospective event has a finite upper time bound; a remote event has an infinite upper time bound. In modeling a system, l_σ would typically represent a delay and u_σ would represent a hard deadline.

For $\sigma \in \Sigma_{act}$ let

$$T_\sigma = \begin{cases} [0, u_\sigma], & \text{if } \sigma \in \Sigma_{spe} \\ [0, l_\sigma], & \text{if } \sigma \in \Sigma_{rem} \end{cases}.$$

T_σ is called the *timer interval* for σ.

A timed transition graph (TTG) $G = (Q, \Sigma, \delta, q_0, Q_m)$ is a finite automaton that incorporates the lower and upper time bounds of events into its transition structure. The timed behavior of a TDES is represented by its TTG. The state set Q is defined as $Q = A_{act} \times \prod\{T_\sigma \mid \sigma \in \Sigma_{act}\}$. Thus a state $q \in Q$ is an element of the form $q = (a, \{t_\sigma \mid \sigma \in \Sigma_{act}\})$, where $a \in A_{act}$ and $t_\sigma \in T_\sigma$. The set of events Σ is defined as $\Sigma = \Sigma_{act} \cup \{tick\}$, where the additional event $tick$ represents the advancement of one time unit. The initial state $q_0 \in Q$ is $q_0 := (a_0, \{t_{\sigma,0} \mid \sigma \in \Sigma_{act}\})$, where $t_{\sigma,0} = \begin{cases} u_\sigma, & \text{if } \sigma \in \Sigma_{spe} \\ l_\sigma, & \text{if } \sigma \in \Sigma_{rem} \end{cases}$.

The marked states, each of which represents the completion of some task or operation, constitute the state subset of the form $Q_m \subseteq A_m \times \prod\{T_\sigma \mid \sigma \in \Sigma_{act}\}$, i.e., a marked state is represented by a marked activity and a suitable assignment of the timers. The state transition function $\delta : \Sigma \times Q \to Q$ is defined as follows. $\delta(\sigma, q)$ is defined for any $q = (a, \{t_\sigma \mid \sigma \in \Sigma_{act}\}) \in Q$ and $\sigma \in \Sigma$, written $\delta(\sigma, q)!$, if and only if any of the following three conditions are satisfied.

1) $\sigma = tick$ and $(\forall \tau \in \Sigma_{spe}) t_\tau > 0$;
2) $\sigma \in \Sigma_{spe}$, $\delta_{act}(\sigma, a)!$, and $0 \leq t_\sigma \leq u_\sigma - l_\sigma$;
3) $\sigma \in \Sigma_{rem}$, $\delta_{act}(\sigma, a)!$, and $t_\sigma = 0$.

We write $\neg\delta(\sigma, q)!$ to denote that $\delta(\sigma, q)!$ is not defined. An entrance state $q' = \delta(\sigma, q) = (a', \{t'_\tau \mid \tau \in \Sigma_{act}\})$ is defined as follows whenever $\delta(\sigma, q)!$.

- when $\sigma = tick$, $a' = a$ and $\forall \tau \in \Sigma_{act}$,
$$t'_\tau = \begin{cases} t_\tau - 1, & \text{if } \delta_{act}(\tau, a)! \text{ and } t_\tau > 0 \\ t_\tau, & \text{otherwise} \end{cases};$$
- when $\sigma \in \Sigma_{act}$, $a' = \delta_{act}(\sigma, a)$, $t'_\sigma = t_{\sigma,0}$ and $\forall \tau \in \Sigma_{act}$ such that $\tau \neq \sigma$,
$$t'_\tau = \begin{cases} t_\tau, & \text{if } \delta_{act}(\tau, a')! \\ t_{\tau,0}, & \text{otherwise} \end{cases}.$$

In the Brandin-Wonham framework, the control design is carried out on the TTG of a TDES. Graphically, a TTG can be represented by an edge-labeled directed graph, with a node denoting a state and an edge denoting an event-labeled transition. In the graph, transitions do not have any time bounds associated with them and all timing behavior is described using transitions of the *tick* event.

The set of transitions in G, denoted by $TR(G)$ is defined as $TR(G) = \{(q, \sigma) \in Q \times \Sigma \mid \delta(\sigma, q)!\}$.

Let Σ^* be the set of all finite sequences called strings, of events from Σ, including the empty string ε (a sequence with no events). A string t' is a *prefix* of t, if there exists a string s such that $t's = t$. A *language* L over Σ is a subset of Σ^*. We say L_1 is a *sublanguage* of L_2 if $L_1 \subseteq L_2$. For a language L, its *prefix closure* $\bar{L}$ is the language consisting of all prefixes of its strings. As any string s in Σ^* is a prefix of itself, we have $L \subseteq \bar{L}$. A language L is considered *prefixed-closed* if $L = \bar{L}$.

The prefix-closed language $L(G)$ and the marked language $L_m(G)$ describe the behavior of G. $L(G)$ is the set of all strings that can be generated by G. The marked language is the set of all strings in $L(G)$ for which the terminal state is a marked state. Extending the transition function δ to Σ^*, we have as follows: $\delta(\varepsilon, q) = q$ and $(\forall \sigma \in \Sigma)(\forall s \in \Sigma^*)\delta(s\sigma, q) = \delta(\sigma, \delta(s, q))$, which is defined if $q' = \delta(s, q)$ and $\delta(\sigma, q')$ are both defined. Then formally,

$$L(G) = \{s \in \Sigma^* \mid \delta(s, q_0)!\}, \quad (1)$$
$$L_m(G) = \{s \in L(G) \mid \delta(s, q_0) \in Q_m\}. \quad (2)$$

A state $q \in Q$ is *reachable* if $(\exists s \in \Sigma^*)\ \delta(s, q_0) = q$, and *coreachable* if $(\exists s \in \Sigma^*)\ \delta(s, q) \in Q_m$. G is *reachable* if all its states are reachable, and is *coreachable* if all its states are coreachable, i.e., $\overline{L_m(G)} = L(G)$. If G is both reachable and coreachable, then it is said to be *trim*.

Finally, let $|P|$ denote the cardinality of a set P.

IV. Problem Concepts and Description

A. Concept and Problem for Specification Comprehensibility

We now formally develop our concept of TTG transparency which governs the reconstruction of a specification TTG into a transparent one in which the events of a specification TTG whose corresponding transitions are identified as specification irrelevant are 'hidden' as self-loops and specification relevant transitions are explicated as diligent transitions. A transparent specification TTG is hopefully more comprehensible as all the self-loop transitions at a state of the transparent TTG, considered irrelevant to the specification, always have at least one associated event defined and can therefore occur in a state of the TDES, entered upon every possible TDES-synchronized and diligent transition into the state of the transparent TTG. This means that, upon entering any such state of the transparent TTG, the next synchronized transition with the TDES need not immediately exit that state, implying that it is a distinct state of ongoing activity for the specification of interest. As a result, a transparent TTG is not only generally more compact, but also retains those a priori TDES constraints that help furnish a clearer structure of specification-distinct states that aids in specification comprehension.

The goal is finding the most or maximally transparent TTG for a full specification TTG H ('full' in the sense that the specification TTG generates a sublanguage of that for a TDES, and has all the a priori transitional constraints of the TDES embedded in it). Different transparent specification TTGs can be obtained for H based on the latter's relevant transition sets of different cardinality. Subject to human cognition limits, a maximally transparent TTG is one that models the original restrictiveness of H on the TDES, and is constructed from H based on a relevant transition set of minimal cardinality. Such a maximally transparent specification TTG could graphically display only what is needed to understand the specification. With a more tractable linguistic description, a TTG may be more readily interpreted by designers when deciding if the given specification captures the intended control requirement. We formulate the transparency maximization problem and prove that it is NP-hard. As a polynomial time algorithm cannot be expected for this problem, we propose a polynomial time algorithm that can achieve maximal transparency in individual cases but not in general.

B. Specification TTG and Transparency

A specification TTG A for TDES G models a (marked) sublanguage of G over the set Σ of events. The sublanguage $L_m(A) \cap L_m(G)$ is well modeled so that every common prefix string in $L(A) \cap L(G)$ can be extended to a marked string in $L_m(A) \cap L_m(G)$, thereby specifying an uninhibited sequence of executions that complete some task. Definition 1 of a specification TTG (adapted from [27]) follows naturally.

Definition 1: Given a TDES $G = (Q, \Sigma, \delta, q_0, Q_m)$, and a regular language L such that for TTG $A = (X, F, \xi, x_0, X_m)$ we have $L = L_m(A)$. If A is said to be a specification TTG (of L for TDES G), then 1) $F = \Sigma$, 2) $\overline{L_m(A) \cap L_m(G)} = L(A) \cap L(G)$, and 3) A is trim.

Given a specification TTG A for TDES G, a trim specification TTG H such that $L_m(H) = L_m(A) \cap L_m(G)$ also embodies the a priori transitional constraints of G. Such a specification TTG H, so that $L_m(H) \subseteq L_m(G)$, is said to be a full specification TTG representing the full nonblocking behavioral specification for G under $L_m(A)$, and can easily be computed from A and G using the composition operation in TDES theory [1].

For a given full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ for a TDES $G = (Q, \Sigma, \delta, q_0, Q_m)$, we define the following.

Let $E : Y \to 2^{\Sigma}$ be the set of events that are defined at $y \in Y$ such that $E(y) = \{\sigma \in \Sigma \mid \zeta(\sigma, y)!\}$.

Let $D : Y \to 2^{\Sigma}$ be the set of events that are not permitted at $y \in Y$ such that $D(y) = \{\sigma \in \Sigma \mid \neg\zeta(\sigma, y)!\ and\ (\exists s \in \Sigma^*)[\zeta(s, y_0) = y\ and\ \delta(s\sigma, q_0)!]\}$.

Let $S : Y \to \{true, false\}$ with $S(y) = true$ if $(\exists s \in \Sigma^*)[\zeta(s, y_0) = y\ and\ \delta(s, q_0) \in Q_m]$. $S(y)$ is true if y is reachable by some string in the marked language of G.

Let $M : Y \to \{true, false\}$ with $M(y) = true$ if $y \in Y_m$. $M(y)$ is true iff y is a marked state.

Using these definitions, we now define the specification compatibility of a pair of states of H.

Definition 2: For a given full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ on TDES G, let $C \subseteq Y \times Y$ be the set of specification compatible state pairs of H on G. Then for $y, y' \in Y$, we have $(y, y') \in C$ (i.e., the state pair $(y, y') \in Y \times Y$ is specification compatible) iff

1) $E(y) \cap D(y') = E(y') \cap D(y) = \emptyset$;
2) $S(y) = S(y') \Rightarrow M(y) = M(y')$.

By Condition 1 of Definition 2, for $(y, y') \in C$, an event that is permitted at y should not be denied permission at y' and vice versa. Condition 2 articulates that for $(y, y') \in C$, y and y' should be consistently marked $true$ or $false$ if both the states are reachable by some strings in the marked language of G (i.e., whenever $S(y) = S(y') = true$), or if neither is reachable by any string in the marked language of G (i.e., whenever $S(y) = S(y') = false$).

Procedure $ChkIfCompatible$ is developed to check if a state pair is specification compatible. Lemma 1 follows.

Procedure $ChkIfCompatible(y, y')$

```
Input: Two states $y, y' \in Y$ of a full specification TTG
       $H = (Y, \Sigma, \zeta, y_0, Y_m)$;
Output: $true$ if $(y, y') \in C$; $false$, otherwise;
begin
    if $(E(y) \cap D(y')) \cup (E(y') \cap D(y)) = \emptyset$ then
        if $S(y) = S(y')$ then
            if $M(y) = M(y')$ then
                return $true$;
            end
        end
        else
            return $true$;
        end
    end
    return $false$;
end
```

Lemma 1: Let C be the set of specification compatible state pairs of a full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ on TDES G. Then for a pair of input states $y, y' \in Y$, Procedure $ChkIfCompatible(y, y')$ returns $true$ iff $(y, y') \in C$.

A cover of a set Y is a family of nonempty subsets of Y whose union is Y. Each element of a cover is called a cell. A partition is a cover in which the cells are pairwise disjoint.

We now formally define what is called a specification-equivalent partition.

Definition 3: Let C be the set of specification compatible state pairs of a full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ on TDES G and X an index set. A partition $P = \{Y_x \subseteq Y \mid x \in X\}$ is a specification-equivalent partition of H if

1) $(\forall x \in X)(\forall y, y' \in Y_x)(y, y') \in C$;
2) $(\forall x \in X)(\forall \sigma \in \Sigma)(\exists x' \in X)[(\forall y \in Y_x)\zeta(\sigma, y)! \Rightarrow \zeta(\sigma, y) \in Y_{x'}]$.

For a specification-equivalent partition $P = \{Y_x \subseteq Y \mid x \in X\}$ of H, a cell containing a state $y \in Y$ is represented as $[y]$, i.e., for $y \in Y_x$, we have $[y] = Y_x$.

According to Condition 1 of Definition 3, all pairs of states within a cell of a specification-equivalent partition should be specification compatible. According to Condition 2 of Definition 3, the states reachable by a one-step transition of the same event from states of a cell Y_x of P should all belong to some cell $Y_{x'}$ of P.

We now present a procedure called $TTran$ to compute and return an induced TTG $A = (X, \Sigma, \xi, x_0, X_m)$ from a given specification-equivalent partition $P = \{Y_x \subseteq Y \mid x \in X\}$ defined on a full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$. Each cell Y_x of P induces a state x of A such that the initial state x_0 of A corresponds to the cell containing the initial state y_0 of H and the marked states of A correspond to cells containing marked states of H. A transition with event label $\sigma \in \Sigma$ is defined from state x to state x' if there is a transition having the same event label from some state in Y_x to some state in $Y_{x'}$. The procedure considers all states and events of H while computing A, resulting in a complexity of $O(|Y||\Sigma|)$.

Procedure $TTran(H, P)$

Input: A full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ and a specification-equivalent partition $P = \{Y_x \subseteq Y \mid x \in X\}$ of H;
Output: An induced specification TTG A, where each state of A represents a cell of P;

```
1 begin
2     $x_0 = x \in X$ such that $y_0 \in Y_x$;
3     $X_m = \{x \in X \mid Y_x \cap Y_m \neq \emptyset\}$;
4     $\xi : \Sigma \times X \to X$(pfn) with $\xi(\sigma, x) = x'$ for
      $(x, x' \in X)$ and $(\sigma \in \Sigma)$, such that
      $(\exists y \in Y_x)\zeta(\sigma, y) \in Y_{x'}$;
5     return $A = (X, \Sigma, \xi, x_0, X_m)$;
6 end
```

We now state our first theorem.

Theorem 1: For a given full specification TTG H on TDES G and a specification-equivalent partition P of H, $A = TTran(H, P)$ is a specification TTG modeling $L_m(H)$ on G, i.e., $L_m(A) \cap L_m(G) = L_m(H)$.

Proof: Let $H = (Y, \Sigma, \zeta, y_0, Y_m)$, $G = (Q, \Sigma, \delta, q_0, Q_m)$ and $A = (X, \Sigma, \xi, x_0, X_m)$.

A is a specification TTG for G if it satisfies the three conditions of Definition 1. By construction, the event set of A is Σ, the same as that of G, satisfying Condition 1. Since H is trim, we have $\overline{L_m(H)} = L(H)$. As a result, Condition 2, that $\overline{L_m(A) \cap L_m(G)} = L(A) \cap L(G)$, can be proved by showing that $L_m(H) = L_m(A) \cap L_m(G)$ and $L(H) = L(A) \cap L(G)$. Condition 3 requires A to be trim, i.e., both reachable and coreachable. Consider any state $x \in X$. Let $y \in Y$ be such that $y \in Y_x$. As H is trim, $(\exists s \in \Sigma^*)$ such that $\zeta(s, y_0) = y$ and $(\exists s' \in \Sigma^*)$ such that $\zeta(s', y) \in Y_m$. By construction, this implies $\xi(s, x_0) = x$ (i.e., reachable) and $\xi(s', x) \in X_m$ (i.e., coreachable), respectively. Repeating this argument for all $x \in X$, we have A is trim, satisfying Condition 3.

According to Definition 1, a specification TTG H on G models a sublanguage $L_m(H) \cap L_m(G)$ of G over Σ. Since H is a full specification TTG of G, $L_m(H) \subseteq L_m(G)$, implying $L_m(H) \cap L_m(G) = L_m(H)$. A specification TTG A models the same sublanguage of G if $L_m(A) \cap L_m(G) = L_m(H) \cap L_m(G) = L_m(H)$.

From our discussion so far, to prove that under G, A is a specification TTG modeling $L_m(H)$, we need to show that $L_m(H) = L_m(A) \cap L_m(G)$ and $L(H) = L(A) \cap L(G)$.

For proving $L_m(H) = L_m(A) \cap L_m(G)$, we show that $L_m(H) \subseteq L_m(A) \cap L_m(G)$ and $L_m(H) \supseteq L_m(A) \cap L_m(G)$. Similarly, for proving $L(H) = L(A) \cap L(G)$, we show that $L(H) \subseteq L(A) \cap L(G)$ and $L(H) \supseteq L(A) \cap L(G)$.

1) Proof of $L_m(H) \subseteq L_m(A) \cap L_m(G)$.
   To prove this, we show that $L_m(H) \subseteq L_m(G)$ and $L_m(H) \subseteq L_m(A)$.
   Since H is a full specification TTG on G, $L_m(H) \subseteq L_m(G)$.
   We now show that $L_m(H) \subseteq L_m(A)$, i.e., every string in $L_m(H)$ also belongs to $L_m(A)$. Consider a string $s \in L_m(H)$.
   Suppose $s = \varepsilon$, we have $y_0 \in Y_m$. By line 2 of Procedure $TTran$, we have $y_0 \in Y_{x_0}$. By line 3 of Procedure $TTran$, $Y_{x_0} \cap Y_m \neq \emptyset \Rightarrow x_0 \in X_m$. So $\varepsilon \in L_m(A)$.
   Suppose $s = \sigma_0$, we have $\zeta(\sigma_0, y_0)!$. Let $y' = \zeta(\sigma_0, y_0)$. By Condition 2 of Definition 3, $(\exists x, x' \in X)$ such that $y_0 \in Y_x$ and $y' \in Y_{x'}$. Since $y_0 \in Y_{x_0}$, we get $x = x_0$. By the definition of ξ (line 4 of Procedure $TTran$), we have $\xi(\sigma_0, x_0) = x'$. Also, $\sigma_0 \in L_m(H)$ implies $y' \in Y_m$. By line 3 of Procedure $TTran$, $y' \in Y_m$ and $y' \in Y_{x'} \Rightarrow x' \in X_m$. So $\sigma_0 \in L_m(A)$.
   Similarly for $s = \sigma_0\sigma_1$, $(\exists x, x' \in X)$ such that $\zeta(\sigma_0, y_0) \in Y_x$ and $\zeta(\sigma_0\sigma_1, y_0) \in Y_{x'}$, implying $\xi(\sigma_0, x_0) = x$ and $\xi(\sigma_1, x) = x'$. Also, $\zeta(\sigma_0\sigma_1, y_0) \in Y_m \Rightarrow x' \in X_m$. So $s = \sigma_0\sigma_1 \in L_m(A)$.
   Repeating this argument j-times for a string $s = \sigma_0\sigma_1 \cdots \sigma_j \in L_m(H)$, we get, $s \in L_m(H)$ implies $s \in L_m(A)$.
2) Proof of $L(H) \subseteq L(A) \cap L(G)$.
   We have $L_m(H) \subseteq L_m(A) \cap L_m(G)$. On taking

closures, we get $\overline{L_m(H)} \subseteq \overline{L_m(A) \cap L_m(G)}$. We have $L(H) = \overline{L_m(H)}$ (as H is trim) and $\overline{L_m(A) \cap L_m(G)} \subseteq L(A) \cap L(G)$, implying $L(H) \subseteq L(A) \cap L(G)$.

3) Proof of $L(H) \supseteq L(A) \cap L(G)$.
Assume that $s \in L(A) \cap L(G)$. We need to show that $s \in L(H)$, as follows.
Suppose $s = \varepsilon$. As $L(H) \neq \emptyset$, we have $s = \varepsilon \in L(H)$.
Suppose $s = \sigma$, we have $\sigma \in L(A) \Rightarrow \xi(\sigma, x_0)!$. Then $\exists y \in Y_{x_0}$ such that $\zeta(\sigma, y)!$, i.e., $\sigma \in E(y)$. By line 2 of Procedure $TTran$, $y_0 \in Y_{x_0}$. By Condition 1 of Definition 3, we have $\sigma \notin D(y_0)$. Then, either $\zeta(\sigma, y_0)!$ or $(\nexists s' \in \Sigma^*)[\zeta(s', y_0) = y_0 \textit{ and } \delta(s'\sigma, q_0)!]$. The second condition fails for $s' = \varepsilon$ as $\zeta(\varepsilon, y_0) = y_0$ and $s = \sigma \in L(G) \Rightarrow \delta(\sigma, q_0)!$. So $\zeta(\sigma, y_0)!$, implying $s \in L(H)$.
Repeating this argument j-times for a string $s = \sigma_0\sigma_1 \cdots \sigma_j \in L(A) \cap L(G)$, we get $\xi(s, x_0)!$ *and* $\delta(s, q_0)! \Rightarrow \zeta(s, y_0)!$. Therefore, $L(H) \supseteq L(A) \cap L(G)$.

4) Proof of $L_m(H) \supseteq L_m(A) \cap L_m(G)$.
Assume that $s \in L_m(A) \cap L_m(G)$. We need to show that $s \in L_m(H)$, as follows.
Since $L(H) \supseteq L(A) \cap L(G)$, we have $s \in L_m(A) \cap L_m(G) \Rightarrow s \in L(H)$, i.e., $\zeta(s, y_0)!$. Let $\zeta(s, y_0) = y$. Since $s \in L_m(G)$, $S(y) = true$.
Since $s \in L_m(A)$, we have $\xi(s, x_0)!$. Let $\xi(s, x_0) = x$. By line 3 of Procedure $TTran$, $\exists y' \in Y_x$ such that $y' \in Y_m$, i.e., $M(y') = true$.
Let s' be such that $\zeta(s', y_0) = y'$. Then as $L_m(H) \subseteq L_m(G)$, we have $s' \in L_m(H) \Rightarrow s' \in L_m(G)$, implying $S(y') = true$.
By Condition 1 of Definition 3, $y, y' \in Y_x \Rightarrow (y, y') \in C$. By Condition 2 of Definition 2, if $(y, y') \in C$, then $S(y) = S(y') \Rightarrow M(y) = M(y')$. So we have $M(y) = true$, i.e, $\zeta(s, y_0) = y \in Y_m$ and hence $s \in L_m(H)$.

Hence, under G, A is a specification TTG modeling $L_m(H)$. ■

Remark 1: The concept of a specification-equivalent partition (in Definition 3) for TDES specifications is mathematically equivalent to that of a control congruence defined for untimed or logical DES supervisors, if $L_m(H)$ is assumed to be controllable [30]. It follows that, for a control congruence P_c defined on such an automaton H, invoking Procedure $TTran(H, P_c)$ returns a state-reduced supervisor [30] that realizes $L_m(H)$ for G. Using this partition, our work shares the same mathematical basis as that on supervisor state reduction [30]. Beyond this point, however, our problem of interest requires a new and more specialized partition of specification-equivalence for H, as presented next.

Definition 4: Given a full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ on TDES G. A specification-equivalent partition $P = \{Y_x \subseteq Y \mid x \in X\}$ of H is said to be a T-transparent partition of H for $T \subseteq TR(H)$ if $(\forall (y, \sigma) \in T)[(\exists x \in X)(y \in Y_x) \Rightarrow (\zeta(\sigma, y) \in Y_x)$.

A T-transparent partition formalizes a specification transparent partition. The states connected by a transition in T belong to the same cell of the partition.

Definition 5: Given a full specification TTG H on TDES G. Under G, for $T \subseteq TR(H)$, a TTG A is said to be a T-transparent specification of H if $A = TTran(H, P)$, where P is a T-transparent specification-equivalent partition of H.

A T-transparent specification TTG formalizes a transparent specification, where T is said to define an irrelevant transition set of TTG H, and $TR(H) - T$ defines a relevant transition set. Intuitively, in a T-transparent specification TTG A of H, all transitions corresponding to transitions of H in T appear only as self-loops.

We postulate that the most (or maximally) transparent specification TTG A should hide in self-loops as many irrelevant transitions of H as possible, i.e., T should be of maximal cardinality.

C. Problem Statement

The transparency maximization problem can now be formally stated as follows.

Problem 1: Given a full specification TTG H for a TDES G. Construct a specification TTG A so that

1) Under G, A is a T-transparent specification TTG of H;
2) $(\forall T' \subseteq TR(H), |T'| > |T|)$, there is no T'-transparent specification TTG A' of H that models $L_m(H)$ on G.

We now state our second theorem.

Theorem 2: Problem 1 is NP-hard.

Proof: See Appendix. ■

V. PROCEDURES AND SOLUTION ALGORITHM

As the transparency maximization problem is NP-hard, we cannot expect a polynomial-time algorithm that can always return a specification TTG of maximal transparency. In this section, we propose a polynomial-time algorithm that can achieve maximal transparency in individual cases but not in general.

A. Transition Irrelevance Check

Procedure $ChkIrrelevance$ is developed to check the specification irrelevance of transitions of a full specification TTG H. To check the irrelevance of a transition, the procedure takes in as input the state pair connected by the transition and returns $true$ if the transition is irrelevant to the specification. For this, initially, the procedure checks if the input state pair is specification compatible by invoking Procedure $ChkIfCompatible$ and thereafter recursively checks if all state pairs reachable by identical strings from the input state pair are also specification compatible, thereby ensuring that the computed partition is a specification-equivalent partition. The procedure returns $true$ if all the state pairs considered are specification compatible; and $false$, otherwise. A list $waitlist$ (that is initialized to the empty set $\emptyset$ whenever Procedure $ChkIrrelevance$ is invoked by Procedure $CompTransPartition$) is updated with each state pair that is considered. In the worst case, the greedy procedure may have to check the specification compatibility of every possible pair of states of $H = (Y, \Sigma, \zeta, y_0, Y_m)$. In that case, the procedure could make $\frac{1}{2}|Y|(|Y| - 1)$ calls to itself, resulting in a complexity of $O(|Y|^2)$.

Procedure $ChkIrrelevance(y, y', waitlist)$

Input: Two states y, y' of a full specification TTG H and a list $waitlist$ of state pairs that are considered so far;
Output: $flag = true$, if the transition connecting y and y' is irrelevant to the specification; $flag = false$, otherwise;

```
begin
    Let $W(y) = \{y'' \mid \{(y, y''), (y'', y)\} \cap waitlist \neq \emptyset\}$;
    foreach $y_1 \in [y] \cup \bigcup_{y_1' \in W(y)} [y_1']$ do
        foreach $y_2 \in [y\prime] \cup \bigcup_{y_2' \in W(y\prime)} [y_2']$ do
            if $\{(y_1, y_2), (y_2, y_1)\} \cap waitlist = \emptyset$ and $[y_1] \neq [y_2]$ then
                if $ChkIfCompatible(y_1, y_2) = false$ then
                    return $false$;
                end
                $waitlist := waitlist \cup \{(y_1, y_2)\}$;
                foreach $\sigma \in \Sigma$ such that $\zeta(\sigma, y_1)!$ and $\zeta(\sigma, y_2)!$ do
                    $flag = ChkIrrelevance(\zeta(\sigma, y_1), \zeta(\sigma, y_2), waitlist)$;
                    if $flag = false$ then
                        return $false$;
                    end
                end
            end
        end
    end
    return $true$;
end
```

B. Computation of a Specification Transparent Partition

For an input full specification TTG H on TDES G, Procedure $CompTransPartition$ computes a specification transparent partition P of H. Initially, the procedure defines a $\emptyset$-transparent partition of H such that each state belongs to a distinct cell. The procedure then uses Procedure $ChkIrrelevance$ to check the specification irrelevance of each transition of H. Whenever a transition under consideration is irrelevant to the specification, Procedure $ChkIrrelevance$ returns $true$ and each state pair in $waitlist$ is placed in the same cell of P. The procedure terminates after it has considered all transitions of H. As the procedure has to consider all transitions of H (i.e., in worst case, $|Y||\Sigma|$ transitions), calling Procedure $ChkIrrelevance$ for each transition, the complexity is $O(|Y|^3|\Sigma|)$.

C. Solution Algorithm

For an input full specification TTG H on TDES G, Algorithm 1 computes a transparent specification TTG of H. The algorithm uses Procedure $CompTransPartition$ to compute a specification transparent partition P of H. The algorithm then computes and returns a specification TTG $A = TTran(H, P)$, which by Theorem 1 and Definition 5, is a transparent specification TTG of H such that $L_m(A) \cap L_m(G) = L_m(H)$, modeling $L_m(H)$ on G. The complexity of Algorithm 1 is the sum of complexities of procedures $CompTransPartition$ and $TTran$, i.e., $O(|Y|^3|\Sigma| + |Y||\Sigma|) \approx O(|Y|^3|\Sigma|)$.

Procedure $CompTransPartition(H, G)$

Input: A full specification TTG $H = (Y, \Sigma, \zeta, y_0, Y_m)$ on TDES G;
Output: A specification transparent partition P of H;

```
begin
    Let $W(y) = \{y'' \mid \{(y, y''), (y'', y)\} \cap waitlist \neq \emptyset\}$;
    $P = \{[y] \mid [y] = \{y\}$ for $y \in Y\}$;
    foreach $y \in Y$ and $\sigma \in \Sigma$ do
        if $\zeta(\sigma, y)!$ then
            Let $\zeta(\sigma, y) = y'$;
            $waitlist := \emptyset$;
            $flag = ChkIrrelevance(y, y', waitlist)$;
            if $flag = true$ then
                $P' = \{[y] \cup \bigcup_{y' \in W(y)} [y'] \mid [y], [y'] \in P\}$;
                $P = P'$;
            end
        end
    end
    return $P$;
end
```

Algorithm 1: Computation of a transparent specification TTG

Input: A full specification TTG H on TDES G;
Output: A transparent specification TTG A of H on G;

```
begin
    $P = CompTransPartition(H, G)$;
    $A = TTran(H, P)$;
    return $A$;
end
```

We now state our third theorem.

Theorem 3: Given a full specification TTG H on TDES G, Algorithm 1 returns a transparent specification TTG of H on G.

Proof: For proving that Algorithm 1 returns a transparent specification TTG, we need to show that for a full specification TTG H on TDES G, the output P of Procedure $CompTransPartition$ is a specification transparent partition of H, i.e., $(\exists T \subseteq TR(H))$ such that P is T-transparent.

Let $H = (Y, \Sigma, \zeta, y_0, Y_m)$. The for loop in line 4 of Procedure $CompTransPartition$ considers every state and event of H and the if condition in line 5 checks whether a transition labeled by the event under consideration is defined at the state that is considered. This ensures that all transitions of H are considered. Let (y_1, σ_1) be the first transition to be considered such that $\zeta(\sigma_1, y_1) = y_1', y_1' \in Y$. By Definition 4, a T-transparent partition is a specification-equivalent partition in which states connected by transitions in T belong to same cells. We now proceed to show that P is a specification-equivalent partition of H.

Initially $P = \{[y] \mid [y] = \{y\}$ for $y \in Y\}$ is a

$\emptyset$-transparent partition of H, satisfying Definition 4 (of a specification transparent partition). The list $waitlist$ is initialized to the empty set $\emptyset$ whenever a new transition is considered. Procedure $ChkIrrelevance(y_1, y_1', waitlist)$ is called to check if the transition (y_1, σ_1), connecting states y_1 and y_1', is irrelevant to the specification. The list $waitlist$ is updated with state pairs reachable by identical strings from y_1 and y_1'. If the procedure returns $false$, then P does not change and remains a specification-equivalent partition. If Procedure $ChkIrrelevance$ returns $true$, Procedure $CompTransPartition$ computes a partition P' by augmenting P such that each state pair in $waitlist$ is placed in the same cell of P'. By line 10 of Procedure $CompTransPartition$, there is no sharing of states between cells of P'. As the procedure then replaces P with P' (line 11), the cells of P are disjoint.

By lines 3-4 and 6-8 in Procedure $ChkIrrelevance$, and Lemma 1, we can derive for the output P of Procedure $CompTransPartition$ that $(\forall p \in P)(\forall y_1, y_2 \in p)[(y_1, y_2) \in C]$, satisfying Condition 1 of Definition 3 (of a specification-equivalent partition). By lines 3-4 and 10-15 of Procedure $ChkIrrelevance$, we can derive that $(\forall p \in P)(\forall \sigma \in \Sigma)(\exists p' \in P)[(\forall y_1 \in p)\zeta(\sigma, y_1)! \Rightarrow \zeta(\sigma, y_1) \in p']$, satisfying Condition 2 of Definition 3.

Applying the same argument for every transition of H, i.e., for each $(y, \sigma) \in TR(H)$, we can show that the resulting P is a specification transparent partition of H. Let T be the set of all transitions that are computed as irrelevant to the specification, i.e., those transitions for which Procedure $ChkIrrelevance$ returns $true$. Then, by construction, the output P of Procedure $CompTransPartition$ is a T-transparent partition of H.

Hence, by Theorem 1 and Definition 5, the output of Algorithm 1, $A = TTran(H, P)$, is a transparent specification TTG of H on G. ■

Remark 2: Note that, when applied to specification H, supervisor state reduction algorithms (such as those in [30]) would perform "state reduction whenever possible", unlike Algorithm 1 that does so only upon satisfying the non-trivial conditions of transition relevance (see Definition 4). Intuitively, therefore, Algorithm 1 constructs A by self-looping only as many of the specification irrelevant transitions of H as possible. As a result, Algorithm 1 returns a transparent specification TTG that hides in self-loops only those transitions it determines to be irrelevant to the specification, and exposes as diligent all relevant transitions it determines in the process to capture the essence of the specification. A specification TTG that is merely state-reduced is not guaranteed to be transparent (Definition 5), as Example 2 in Section VI will expose.

VI. ILLUSTRATIVE EXAMPLES

The concept of specification transparency and how it may be useful for human designer validation are now illustrated using two examples. Every TTG is depicted as a directed graph with each state represented by a node, the initial state by a node with an entering arrow and each marked state by a darkened node. Directed edges, representing state-to-state transitions, are labeled by events.

Example 1 (Railroad crossing system): The first example is a railroad system (inspired by an example from [38]) that consists of a gate and a train. The train is modeled to approach a gate (app), arrive at the gate (arr) and then depart from the gate (dep). The gate is modeled to detect the approach of the train (app), and can be lowered (low) and lifted up (up). The ATG models of the train and gate, along with time bounds of events are given in Fig. 2. The corresponding TTG models, $TRAIN$ and $GATE$ respectively, with timing constraints explicitly modeled using transitions of *tick*, are displayed in Fig. 3. The TDES G for the real-time system is formed by the composition [1] of $TRAIN$ and $GATE$.

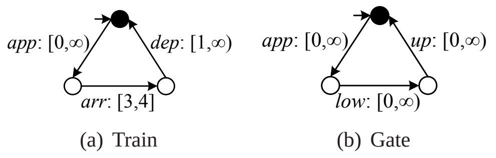


Fig. 2. Railroad crossing system: ATG models with time bounded events

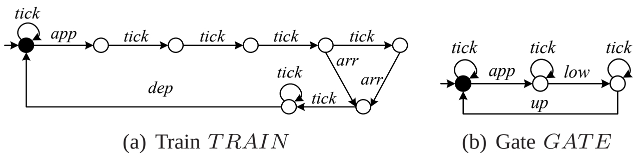


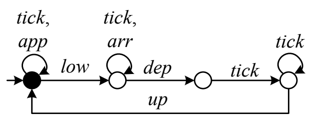
Fig. 3. Railroad crossing system: TTG models

A control requirement for G is to prevent collision between the train and vehicles passing through the railroad crossing. This can be achieved by making sure that the gate is lowered before the train arrives at the gate. As a precaution, the gate is required to lift up only after one unit of time has passed since the departure of the train.

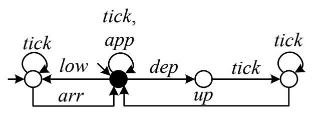
A full specification TTG H of this control requirement for TDES G is given in Fig. 4(c). It may be prescribed by a designer either directly as such, or initially as some specification TTG B (Figs. 4(a) and 4(b)), such that $L_m(B) \cap L_m(G) = L_m(H)$. Applying Algorithm 1 to H, we obtain a highly transparent specification TTG A [Fig. 4(d)] of H. It is incidental that A is in fact maximally transparent.

The TTG A shows that the gate must be lowered before the arrival of the train (low precedes arr) and that the gate must be lifted up only after one unit of time has lapsed following train departure (up succeeds dep and one $tick$). This safety-critical essence as captured by A may not be as readily evident in some other specification TTG prescribing the same requirement. Interpreting A, the designer may be able to more easily validate if the given specification is technically the intended requirement, as only those execution sequences that form the essence of the control requirement are highlighted in A.

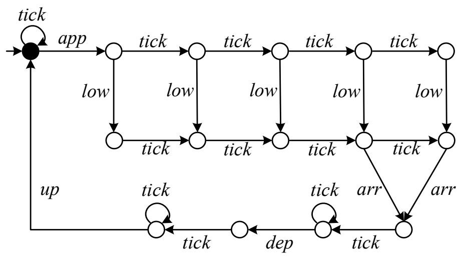
Example 2 (Preemptive scheduling of sporadic tasks): The second example is that of a processor executing two sporadic tasks and is adapted from [39]. In this example we also differentiate the task of making a specification transparent from that of minimizing the number of states of the specification.



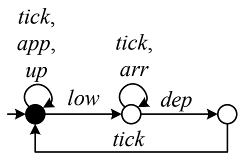
(a) A probable specification TTG B



(b) Another probable specification TTG B

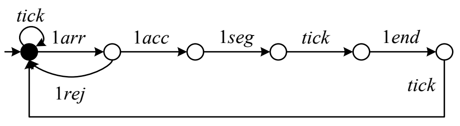


(c) Full specification TTG H

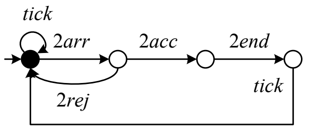


(d) Maximally transparent specification TTG A

Fig. 4. Specification models



(a) Task T_1



(b) Task T_2

Fig. 5. Task execution models

We consider two sporadic tasks T_1 [Fig. 5(a)] and T_2 [Fig. 5(b)] that can arrive at any arbitrary time. Each task is divided into segments that take one unit of time to execute. In the model, this is represented by a $tick$ transition following the execution of each segment. For $i \in \{1, 2\}$, let E_i and D_i denote the time taken for execution and the relative deadline for execution (i.e., the time within which an accepted task should finish its execution) of task T_i, respectively. In this example, we consider a case where $E_1 = 2$, $D_1 = 3$, $E_2 = 1$ and $D_2 = 1$. Accordingly, T_1 has two segments, $1seg$ and $1end$, while T_2 has only one segment, $2end$. Each task arriving for processing can either be accepted or rejected by the processor. Once a task is accepted, each segment of the task is executed. A newly arrived task can always preempt the currently executing task. Table I summarizes the description of events. The TDES G for the real-time system is formed by the composition [1] of T_1 and T_2.

TABLE I
DESCRIPTION OF EVENTS

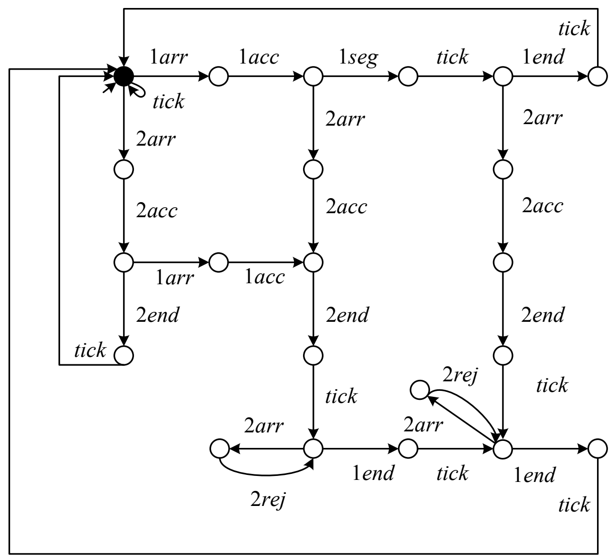
Event	Description
iarr	Arrival of task T_i
irej	Rejection of task T_i
iacc	Acceptance of task T_i
$1seg$	Execution of segment 1 of task T_1
iend	Execution of the last segment of task T_i
$tick$	Advancement of one unit of time

Given the execution time and deadline of each task, a specification for the system requires that a newly arrived task should be accepted only if it can be completed without missing the deadlines for previously accepted tasks. From the given execution times and deadlines for execution, it is clear that T_1 can be preempted only once by T_2 (as $D_1 - E_1 = E_2$) and that T_2 cannot be preempted (as $D_2 - E_2 = 0$).

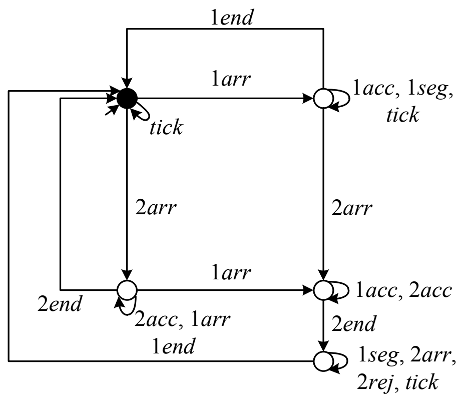
A full specification TTG H of this control requirement for TDES G is given in Fig. 6(a). In general, it is the result of initially prescribing the requirement as some specification TTG B, such that $L_m(B) \cap L_m(G) = L_m(H)$. Applying Algorithm 1 to H, we obtain a specification TTG A [Fig. 6(b)] of H that, incidentally, is maximally transparent. A minimal-state specification TTG $H_{minstate}$ of the same control requirement is shown in Fig. 6(c). As explained below, the control requirement of A is easy to comprehend, while that of $H_{minstate}$ is not.

When T_2 arrives while T_1 is being executed, the processor preempts T_1 and switches to T_2. T_1 resumes only after the last segment of T_2 has finished execution. Subsequent arrival of T_2 before the execution of the last segment of T_1 is rejected. In terms of event execution sequences, whenever $2arr$ succeeds $1arr$ and precedes $1end$, $2end$ precedes $1end$. Any subsequent occurrence of $2arr$ before $1end$ will result in the occurrence of $2rej$. Observe that this execution trajectory is clearly highlighted in A. But for a designer tracking this execution sequence in $H_{minstate}$, the fact that $1end$ is permitted immediately following the execution of the event sequence $1arr \to 1acc \to 2arr$ may cause confusion. Only a close study of G would reveal that, even though $1end$ is permitted in the specification, its occurrence is prevented by the a priori constraints enforced by G.

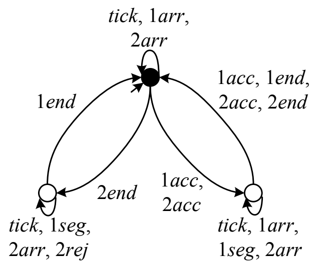
Also, when T_1 arrives while T_2 is being executed, the processor accepts T_1, but starts executing T_1 only after the last segment of T_2 is executed. In terms of execution sequences, whenever $2arr$ precedes $1arr$, $2end$ precedes $1end$. In this case also, we can observe that the transparent specification TTG A highlights the essence of the specification. It may not be so in some other specification TTGs prescribed for the same requirement by a system designer. For example in $H_{minstate}$, even at some instances where $2arr$ precedes $1arr$, both $1end$ and $2end$ are immediately permitted by the specification, causing confusion, even though the occurrence of $1end$ is prevented by the a priori constraints of G.



(a) Full specification TTG H



(b) Maximally transparent specification TTG A



(c) State-minimized specification TTG $H_{minstate}$

Fig. 6. Specification models

VII. CONCLUSION

To support comprehensibility of specification TTGs and facilitate their validation, the concept of specification TTG transparency is developed, and the problem of maximizing TTG transparency for TDESs is formulated and shown to be NP-hard. A polynomial time algorithm for transforming a given specification TTG into a highly transparent one is proposed. Through examples, it is demonstrated that the transparent TTG computed can provide a structure that may support comprehensibility. Together with the transparency concept for untimed specification automata for logical DESs [27], [28], [29], a unified automata-theoretic transparency framework is now in place for both logical and timed specification automata.

Human comprehensible specification TTGs will always exhibit some form of transparency as formalized in this paper. While the illustrative examples show that the transparent specification TTGs computed are human comprehensible, future work will need to experimentally investigate how often a transparent specification TTG computed by our proposed framework is human comprehensible. The experimental insights gained could motivate additional new concepts for specification TTG comprehensibility.

APPENDIX

In this appendix, we show that Problem 1 is NP-hard.

Before presenting the proof, we present some basics in graph theory [40]. An undirected graph Gr is a 2-tuple $(Vertex, Edge)$, where $Vertex$ is the set of vertices and $Edge \subseteq Vertex \times Vertex$ is the set of edges.

For a full specification TTG H on TDES G, the transparency maximization problem (Problem 1) requires computing the largest set $Tr \subseteq TR(H)$ such that there is a Tr-transparent partition of H.

We reduce in polynomial time a known NP-hard problem, the *Maximum Edge Clique Partition* problem [41], to Problem 1. Given an undirected graph $Gr = (Vertex, Edge)$, a clique is a subset of its vertices such that each pair of vertices in the subset are connected by an edge. The $Maximum\ Edge\ Clique\ Partition$ problem is to partition $Vertex$ into cliques such that the total number of edges within the cliques is maximized.

We shall now map a TDES $GrTDES$ and a full specification TTG $GrSpec$ for $GrTDES$ onto an undirected graph $Gr = (Vertex, Edge)$, in such a way that the problem of computing the largest set $Tr \subseteq TR(H)$ such that there is a Tr-transparent partition of H corresponds to the problem of partitioning Gr into cliques with maximum edges within the cliques. The transformation process is as follows.

1) Let $Vertex = \{u_0 \cdots u_n\}$. Define symbols $Vertex' := \{u_{ij} \mid 0 \le i < j \le n\}$. Let

$$Y' := Vertex \cup Vertex'$$

$$Q' := Y'$$

$$\Sigma = \emptyset$$

2) Define (partial) transition function ζ and δ as follows.
   a) Create events $e_i, 0 \le i < n$ and add them to Σ. Let

   $$\zeta(e_i, u_i) = u_{i+1}$$

   $$\delta(e_i, u_i) = u_{i+1}$$

   b) $\forall i, j, 0 \le i < j \le n$ such that $(u_i, u_j) \notin Edge$, create events e_{ij} and add them to Σ. Define

   $$\zeta(e_{ij}, u_j) = u_{ij} \tag{3}$$

   $$\delta(e_{ij}, u_i) = \delta(e_{ij}, u_j) = u_{ij} \tag{4}$$

3) Removing unreachable states and considering all states

as marked, we define

$$Y := \{u \in Y' \mid (\exists s \in \Sigma^*)\zeta(s, u_0) = u\} \quad (5)$$

$$y_0 := u_0 \quad (6)$$

$$Y_m := Y \quad (7)$$

$$Q := \{u \in Q' \mid (\exists s \in \Sigma^*)\delta(s, u_0) = u\} \quad (8)$$

$$q_0 := u_0 \quad (9)$$

$$Q_m := Q \quad (10)$$

$$GrSpec = (Y, \Sigma, \zeta, y_0, Y_m) \quad (11)$$

$$GrTDES = (Q, \Sigma, \delta, q_0, Q_m) \quad (12)$$

We now have a lemma that relates a specification transparent partition of $GrSpec$ (for $GrTDES$) to a clique partition on Gr.

Lemma 2: Gr can be partitioned into cliques such that there are $K, K \leq |Edge|$ edges within cliques iff there is a Tr-transparent partition of $GrSpec$ such that $Tr \subseteq TR(GrSpec)$ and $|Tr| = K$.

We are now ready to prove Theorem 2, as follows: Problem 1 involves computing the largest set $Tr \subseteq TR(H)$ such that there is a Tr-transparent partition of H. By Lemma 2, the *Maximum Edge Clique Partition* problem, which is a known NP-hard problem, is polynomially reducible to the problem of computing the largest set $Tr \subseteq TR(H)$ such that there is a Tr-transparent partition of H. As a result, finding a maximally transparent specification TTG is at least as hard finding a maximum edge clique partition. Hence Problem 1 is NP-hard.

References

[1] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[2] F. Lin and W. M. Wonham, "Supervisory control of timed discrete-event systems under partial observation," *IEEE Transactions on Automatic Control*, vol. 40, no. 3, pp. 558 – 562, 1995.

[3] K. C. Wong and W. M. Wonham, "Hierarchical control of timed discrete-event systems," *Discrete Event Dynamic Systems*, vol. 6, pp. 275–306, 1996.

[4] L. Oudraogo, A. Khoumsi, and M. Nourelfath, "A new method for centralised and modular supervisory control of real-time discrete event systems," *International Journal of Control*, vol. 83, no. 1, pp. 1–39, 2010.

[5] M. Nomura and S. Takai, "Decentralized supervisory control of timed discrete event systems using a partition of the forcible event set," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E95-A, no. 5, pp. 952–960, May 2012.

[6] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206–230, January 1987.

[7] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, "Diagnosis of a class of distributed discrete-event systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 30, no. 6, pp. 731 –752, November 2000.

[8] W. Qiu and R. Kumar, "Decentralized failure diagnosis of discrete event systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 36, no. 2, pp. 384 – 395, March 2006.

[9] S. Takai and R. Kumar, "Decentralized diagnosis for nonfailures of discrete event systems using inference-based ambiguity management," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 40, no. 2, pp. 406 –412, March 2010.

[10] Y.-C. Ou and J. Hu, "A modified method for supervisor specification and synthesis of a class of discrete event systems," *Asian Journal of Control*, vol. 2, no. 4, pp. 263–273, December 2000.

[11] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer, 2008.

[12] Y.-C. Ho and X.-R. Cao, "Introduction to discrete event dynamic systems," in *Perturbation Analysis of Discrete Event Dynamic Systems*, ser. The Springer International Series in Engineering and Computer Science. Springer US, 1991, vol. 145, pp. 1–13.

[13] E. Tronci, "Automatic synthesis of controllers from formal specifications," in *Proceedings of the 2nd International Conference on Formal Engineering Methods*, Brisbane, Australia, December 1998, pp. 134 – 143.

[14] S. Jiang and R. Kumar, "Supervisory control of discrete event systems with CTL* temporal logic specifications," in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 5, Orlando, Florida, USA, December 2001, pp. 4122–4127.

[15] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Knighofer, M. Roveri, V. Schuppan, and R. Seeber, "RATSY - A New Requirements Analysis Tool with Synthesis," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, 2010, vol. 6174, pp. 425–429.

[16] L. Grigorov, B. Butler, J. Cury, and K. Rudie, "Conceptual design of discrete-event systems using templates," *Discrete Event Dynamic Systems*, vol. 21, pp. 257–303, 2011.

[17] J. Košecká and R. Bajcsy, "Discrete event systems for autonomous mobile agents," *Robotics and Autonomous Systems*, vol. 12, no. 3-4, pp. 187–198, April 1994.

[18] S. L. Ricker, N. Sarkar, and K. Rudie, "A discrete event systems approach to modeling dextrous manipulation," *Robotica*, vol. 14, no. 5, pp. 515–525, 1996.

[19] B. A. Brandin, "The real-time supervisory control of an experimental manufacturing cell," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 1–14, February 1996.

[20] S. C. Lauzon, A. K. L. Ma, J. K. Mills, and B. Benhabib, "Application of discrete event system theory to flexible manufacturing," *IEEE Control Systems Magazine*, vol. 16, no. 1, pp. 41–48, February 1996.

[21] J.-K. Lee and T.-E. Lee, "Automata-based supervisory control logic design for a multi-robot assembly cell," *International Journal of Computer Integrated Manufacturing*, vol. 15, no. 4, pp. 319 – 334, 2002.

[22] F. Chriaux, L. Picci, J. Provost, and J.-M. Faure, "Conformance test of logic controllers of critical systems from industrial specifications," in *Proceedings of the 2010 European Conference on Safety and Reliability*, Rhodes, Greece, 2010.

[23] K. Rudie and W. M. Wonham, "Supervisory control of communicating processes," in *Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification X*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1990, pp. 243–257.

[24] K. T. Seow and M. Pasquier, "Supervising passenger land-transport systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 3, pp. 165–176, September 2004.

[25] G. Smith, "Representational effects on the solving of an unstructured decision problem," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 1083–1090, 1989.

[26] W. M. Wonham, "Supervisory control theory: Models and methods," in *Workshop on Discrete Event Systems Control, 24th International Conference on Application and Theory of Petri Nets (ATPN 2003)*, Eindhoven, The Netherlands, June 2003, pp. 1–14.

[27] M. T. Pham, A. Dhananjayan, and K. T. Seow, "On the transparency of automata as discrete-event control specifications," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, May 2010, pp. 1474–1479.

[28] M. T. Pham, A. Dhananjayan, and K. T. Seow, "On specification transparency: Towards a formal framework for designer comprehensibility of discrete-event control specifications in finite automata," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 1, pp. 139 – 148, 2013.

[29] A. Dhananjayan and K. T. Seow, "On specification informatics in discrete-event systems: State-transparency for clarity of finite automata as control specifications," in *Proceedings of the 2012 International Conference on Informatics in Control, Automation and Robotics*, Rome, Italy, July 2012, pp. 357 – 367.

[30] R. Su and W. M. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dynamic Systems : Theory and Applications*, vol. 14, no. 1, pp. 31–53, 2004.

[31] A. Saadatpoor and W. M. Wonham, "Supervisor state size reduction for timed discrete-event systems," in *Proceedings of the 2007 American Control Conference*, New York, USA, July 2007, pp. 4280–4284.

[32] G. Faraut, L. Piétrac, and E. Niel, "Equivalence of behaviors between centralized and multi-model approaches," in *IEEE Conference on Automation Science and Engineering (CASE)*. IEEE, 2011, pp. 32–38.

[33] ——, “Process tracking by equivalent states in modal supervisory control,” in *IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2011, pp. 1–8.
[34] A. Dhananjayan and K. T. Seow, “A metric temporal logic specification interface for real-time discrete-event control,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2014.
[35] A. Dhananjayan and K. T. Seow, “Automating timed specification transparency for human designer validation of real-time discrete-event control requirements,” in *Proceedings of the IEEE International Conference on Automation Science and Engineering*, Seoul, South Korea, August 2012, pp. 908 – 913.
[36] B. Liskov and S. Zilles, “Specification techniques for data abstractions,” *IEEE Transactions on Software Engineering*, vol. 1, no. 1, pp. 7–19, 1975.
[37] J. Markovski, D. A. van Beek, and J. Baeten, “Partially-supervised plants: Embedding control requirements in plant components,” in *Integrated Formal Methods*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7321, pp. 253–267.
[38] J. S. Ostroff and W. M. Wonham, “A framework for real-time discrete event control,” *IEEE Transactions on Automatic Control*, vol. 35, no. 4, pp. 386 – 397, 1990.
[39] Seong-Jin Park and Kwang-Hyun Cho, “Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems,” *Information Sciences*, vol. 178, pp. 3393–3401, 2008.
[40] R. F. Mihalcea and D. R. Radev, *Graph-based Natural Language Processing and Information Retrieval*. Cambridge University Press, 2011.
[41] A. Dessmark, J. Jansson, A. Lingas, E.-M. Lundell, and M. Persson, “On the approximability of maximum and minimum edge clique partition problems,” in *Proceedings of the 12th Computing: The Australasian Theory Symposium*, vol. 51, Hobart, Australia, 2006, pp. 101–105.