CONTROL RECONFIGURATION OF DISCRETE EVENT SYSTEMS WITH DYNAMIC CONTROL SPECIFICATIONS

Rupa Sampath[1], Houshang Darabi[1], Ugo Buy[2], and Jing Liu[1]
[1] Department of Mechanical and Industrial Engineering, University of Illinois at Chicago
[2] Department of Computer Science, University of Illinois at Chicago
{rsampa3, hdarabi, buy, jliu4}@uic.edu

ABSTRACT

This paper defines a reconfiguration method for the class of Discrete Event Systems (DES) that is subject to linear constraints as their control specifications. Some existing methods for enforcing these constraints make use of Petri-net P-invariants for controller synthesis. These methods are quite appealing because their computational complexity is much more tractable than most other methods for controller synthesis. However, a common limitation of all existing P-invariant-based control architectures for DES plants is the assumption that the linear constraints defining the control specification of the plant do not change over time. Here, we relax this assumption and allow the control specifications to change during the controller run-time. Under certain assumptions on DES behavior, we automatically reconfigure the DES controller after the control specification is changed. In addition, if the current state of the controlled DES has become infeasible under the new control specification, we automatically generate a so-called *plant reconfiguration procedure* whose execution leads the system back to a feasible state. This reconfiguration procedure is optimal in that it seeks to minimize the cost of reconfiguration actions through an Integer Programming (IP) model. The objective function of the IP model can be used to generate reconfiguration solutions that meet some desired properties. Depending on the actual cost of each reconfiguration action, a minimum cost reconfiguration solution may use only actions contained in the current plant configuration (an internal response), or ask for a change in the plant configuration, for instance, by adding new resources (an external response), or a combination of both strategies. Finally, we illustrate our method by applying it to a hospital control system example.

I. INTRODUCTION

DES encompass a wide variety of systems such as manufacturing systems, transportation systems, supply chains networks, operating systems and communication systems [13]. These systems are dynamic and change their states with the occurrence of discrete events. DES controllers are used to restrict the behavior of DES to a desirable set of behaviors that do not violate the DES control specifications or

constraints. DES controller development consists of two phases, namely controller synthesis and system run-time. During controller synthesis, a DES and its control specifications are used to construct the DES controller. In the run-time phase, the constructed controller interacts with the DES in a closed loop structure. The interactions happen through the observation channels (by which the controller observes the controlled DES behavior) and control enforcement channels (by which the controller enforces the control commands to the DES). The behaviors of the DES controlled by this loop will comply with the given control specifications.

To date different methods have been used for specifying DES controllers. Examples include Petri nets, finite state automata, ladder logic, and sequential function charts [9, 10, 11]. All such methods are able to synthesize a controller using a suitable model of the DES and its control specifications. The controller specification can be either executed directly, such as a Programmable Logic Controller (PLC) executing a ladder logic program, or transformed into executable control code. Either way, the synthesized controller can guarantee the enforcement of control specifications, as long as all the assumptions and information used in the design and run-time phases hold true. However, in practice, this information can change during the run-time phase. In this case, if the plant execution is continued using the existing controller, the DES might generate incorrect or illegal behaviors. Generally, any change in the controlled system during run-time can fall under one of four different categories: Changes in the uncontrolled behavior of the DES, changes in the control specifications or constraints, changes in the observation channels, and changes in the control enforcement channels. A control reconfiguration consists of the necessary steps to modify a DES controller, in the run-time phase, when one or more of these changes happen. The modified controller must continue a proper control of the DES under the new conditions.

Different types of reconfiguration have been addressed by DES control researchers. Darabi et al. [5] and Liu and Darabi [21] developed a control switching formalism that revises the controller policy when the observation channels fail or are repaired. The DES and its control specifications are modeled by finite automata. A mega-controller monitors changes in the observation channels and selects the proper control

logic when changes occur. This work is an example of reconfiguration based on observation channel changes.

Lucas et al. [2] propose a reconfiguration approach for a modular plant. They assume one control module for each machine, and other control modules for coordination. The interfaces between the modules are well-defined, and all communication between modules occurs via these interfaces. The idea behind their methodology is that the definition of the operation sequence should remain in one control module, called the control plan. The other control modules are independent of the operation sequence. Thus, when the manufacturing scenario is altered, only the control plan needs to be reconfigured or changed. Also, if mechanical modules are added or removed from the system, the appropriate mechanical control modules must also be added or removed. This research is an example of reconfiguration based on changes in both DES behavior and control specifications for a special class of machining systems.

Oliver and Badouel [3] address another example of reconfiguration. They introduce a class of high level Petri nets, called reconfigurable nets that can dynamically modify their own structure by rewriting some of their components. The switch from one configuration to another due to a local change in DES behavior is carried out by introducing a new kind of place content which indicates whether a place does or does not exist in the current system state. The firing policy of transitions is similar to the case of the Petri net obtained by discarding the nonexistent places. This Petri net is called a configuration of the reconfigurable net. As long as no structure-modifying rule is fired, the reconfigurable net behaves exactly like this Petri net.

Hanisch and Vyatkin [4,8] defined a reconfiguration method within the framework of IEC 61499, a control system modeling standard defined by the International Electrotechnical Commission. They developed a software package for model-based simulation and verification united by a homogeneous graphical user interface. The controllers defined in IEC 61499 automatically generate the formal code of the controller from a graphical specification. The analysis of verification results through simulation and visualization along selected process trajectories facilitates an understanding of the reasons of failures. This work is one of few investigations that show the potential application of IEC 61499 in reconfiguring a

control system. However, their work assumes the availability of an IEC 61499 model of the DES and its specification such as a model based on function-block diagrams. Such models are not available for systems other than manufacturing automation, and therefore the proposed reconfiguration may not be applicable in systems other than manufacturing.

Lin [12] and Darabi et al. [20] discuss the class of finite automata based DES that are safely controlled based on a given controller. Those methods are additional examples of reconfiguration based on DES behavior changes. However, those methods are not applicable when the DES change makes the current controller state infeasible.

Lawley and Sulistyono [22] investigate the problem of control reconfiguration subject to unreliable resources. The objective of this work is to avoid possible deadlocks generated due to the failure of resources. When a resource fails, the reconfiguration approach revises the set of control policies to ensure that the enabled polices do not set the system to a blocking state.

In this paper, we present a reconfiguration method for a class of DES whose control specifications can change at run-time. Our approach is applicable to DES containing finite sets of tasks that can be executed in parallel. Different types of resources are needed by the tasks; for each resource type a finite number of resources are available in the DES. In addition, some tasks must precede other tasks. Tasks may be synchronized at predetermined points in their execution; tasks may also choose nondeterministically among multiple possible alternative behaviors. Tasks are subject to *resource* and *service constraints*. Resource constraints state that a task (or a task subset) requires a given set of resources in order to be executed; resources can be shared among tasks. Service constraints impose minimum levels of performance on the DES.

The goal of our method is to define a reconfiguration strategy when the resource constraints and service constraints for tasks contained in a DES are modified during system execution. We specifically assume that the following control specifications may change dynamically for some resources and/or service constraints: Resource availability, resource consumption by tasks, system service requirements, and the satisfaction ratio of different service requirements by different tasks. We define a *reconfiguration*

need to be a change in control specifications that occurs at run-time. We define a *reconfiguration response* to be the set of actions that our method defines in response to a reconfiguration need.

We model the DES subject to reconfiguration as an ordinary Petri net (PN). We choose this representation for several reasons. First, Petri nets can capture in a natural and compact way parallel execution, nondeterministic choice, and task synchronization, which characterize many classes of DES. Modeling this kind of system dynamics can be cumbersome using, for instance, pure integer linear programming models. Second, efficient methods for generating resource and service constraints exist for DES modeled as Petri nets. (See, e.g., [19].) Third, Petri nets support a linear-algebraic interpretation, which is the cornerstone our approach.

Our approach is subject to the following assumptions. We assume that the control specifications can be captured by a set of linear equalities and inequalities on the markings of the PN. The coefficients used in these constraints are finite integer numbers. The PN for the controlled system is constructed using the method introduced by Yamalidou et al. [19]. That method results in the addition of so-called *controller places* to the plant net that models the DES under consideration. In addition to the controller places generated by that method, we introduce so-called *controller transitions,* one transition for each controller place. In some cases, the firing of one such controller transition means increasing the number of resources of a given type by one. In other cases, firing a controller transition means decrementing the minimum level that satisfies a service constraint by one. The coefficients of the right-hand side and/or the left-hand side of the linear constraints can change during the run-time of the controlled system. Reconfiguration actions are limited to firing one or more plant or controller transition in a sequence.

We further assume that the controlled plant is frozen and no further reconfiguration need arises while a reconfiguration strategy is computed and executed. Under these assumptions, our method guarantees that the reconfiguration sequence is live (i.e., free of deadlock) and feasible. However, we assume also that all the transitions contained in the sequence can be *forced* to fire in their order in the sequence. This means that such transitions are controllable. In addition, the plant transitions linked to the controller places must

be controllable [19]. We assume that there is a cost associated with the firing of each transition (whether a plant or controller transition). Finally, we assume that all plant transitions are observable.

Our reconfiguration solution is general for problems that satisfy our assumptions in that it can handle both cases in which the current state of the controlled DES is feasible or becomes infeasible after changing the control specifications. When the current DES state remains feasible under the new controller specification, we define a reconfiguration of the controller structure to adapt to the new set of control specifications. If, however, the current DES state becomes infeasible, we model reconfiguration strategies for both the plant (DES) and the controller. All the proposed reconfigurations are formalized and expressed in terms of specific control actions such as firing a sequence of transitions.

Our approach to system reconfiguration can be fruitfully applied to a broad class of real-world DES. One example of application domain is hospital control systems. In these applications, tasks are different steps that a patient undergoes in the hospital (e.g., admission, test, and examination). Resources consist of medical personnel and hospital facilities. Service constraints could specify a minimum number of resources available in the hospital at all times. Reconfiguration needs may arise, for instance, when the number of nurses or doctors is decreased due to unforeseen circumstances. In these settings, two key assumptions of our approach are satisfied. First, the frequency of reconfiguration needs is reasonably low (e.g., several hours or even days). Second, when a reconfiguration response is computed, it is usually enforceable before any other change in hospital state occurs. A plausible response in this case could consist of acquiring more resources (e.g. moving nurses between sections or asking on-call nurses to come to the hospital), reducing service attribute expectations (e.g. reducing the number of professional personnel in a given section of the hospital), or performing some tasks (e.g., transferring a patient) in order to adjust the levels at which resources and services are associated with tasks.

Another application of our method is the reconfiguration of Project Management Control systems. In these systems, a large set of activities have to be completed over a lengthy period, such as several months. These tasks consume shared resources; in addition, safety (service) measures must be enforced on aggregated classes of resources. Enforcing a transition in these systems could model starting or ending a

task that can be run at different costs for different durations. Also resources and safety/service attributes can be revised by project managers. The goal is to find the optimal (minimum cost) task execution rate, resource acquirement strategies, and service-attribute target points while the project is evolving.

Note that the reconfiguration method of this paper is not appropriate for the systems that change frequently, such as computer operating systems or programmable logic controllers. In addition, our approach will fail if the requested reconfiguration actions are infeasible. For example if these actions ask for the addition of human resources for a specific task, and if these resources are not available, then the stated reconfiguration will not succeed.

We distinguish between the reconfiguration method discussed in this paper and the class of methods for deadlock prevention/avoidance or recovery that have been extensively discussed in the literature. (See, e.g. [6, 22, 25—29].) The objective of the reconfiguration method addressed here is neither to remove a deadlock nor to recover the plant from a blocked state. The issue of deadlock may arise during reconfiguration, meaning that a deadlock-free controlled system may contain potential deadlocks after reconfiguration response takes place. While deadlock detection is beyond the scope of this paper, we believe that one of various strategies for handling deadlocks should be used in combination with our method in the case of systems where deadlock is possible. An example of such a strategy is the use of supervisory controllers that remove deadlocks. (See, e.g. [26—30].)

In the next section, we summarize a method for supervisor synthesis that forms the basis for our reconfiguration method. We discuss our reconfiguration procedures in Section III. Section IV illustrates our reconfiguration method by two examples. In Section V, we discuss some preliminary empirical results of our approach. Finally, in Section VI we present conclusions and future research directions.

II. BACKGROUND

Other authors have defined various frameworks for the construction of DES supervisory controllers, subject to a set of linear constraints [11—19]. The set of linear constraints is called a *control specification*. Giua et al. automatically generate controllers for a broad class of mutual exclusion

constraints expressed as linear equalities and inequalities on Petri net markings [1]. Yamalidou et al. generate similar controllers using a P-invariant based method for controller construction [19]. In the following, we summarize the construction procedure of Yamalidou et al. The original (uncontrolled) DES is modeled by an ordinary Petri net, called the *plant*. We assume that the reader is already familiar with the basic concepts introduced in [19]; here we only provide a brief discussion of these concepts. We denote the plant's set of places and transitions by $P^p = \{p_1, p_2, \cdots, p_n\}$ and $T^p = \{t_1, t_2, \cdots, t_m\}$. We assume that all the transitions in T are controllable and observable. This means that the PN marking is deterministic (due observation assumption), and that every transition in T can be fired as soon as it becomes enabled. Let D^p be the $n \times m$ incidence matrix of the plant. Also let V_0^p be the initial marking of plant places. The objective is to restrict the behavior of the plant in such a way that it satisfies a set of linear constraints, the control specifications. In practice, each constraint in this set can be in one of the following forms:

$$l_1 v_1 + l_2 v_2 + \ldots\ldots\ldots + l_n v_n \le b \tag{1}$$

$$l_1 v_1 + l_2 v_2 + \ldots\ldots\ldots + l_n v_n \ge b \tag{2}$$

$$l_1 v_1 + l_2 v_2 + \ldots\ldots\ldots + l_n v_n = b \tag{3}$$

Here $v_i, 1 \le i \le n$ are constraint variables denoting the marking of plant places in all reachable states of the plant. All other elements are integer constants. We further require that $b_i \ge 0$, for all i with $1 \le i \le k$. Therefore, such constraints are imposing conditions on the marking of plant places. Each type (3) constraint can be replaced with two constraints, one of type (1) and one of type (2). Assume that k is the total number of constraints after replacing all type (3) constraints with their equivalent constraints of type (1) and (2). Without loss of generality, we assume that the first g constraints are of type (1) and the last $k - g$ constraints are of type (2):

$$l_{i1} v_1 + l_{i2} v_2 + \ldots\ldots\ldots + l_{in} v_n \le b_i \qquad 1 \le i \le g \tag{4}$$

$$l_{i1} v_1 + l_{i2} v_2 + \ldots\ldots\ldots + l_{in} v_n \ge b_i \qquad g+1 \le i \le k \tag{5}$$

Using a matrix format, we rewrite the constraint set by the following two inequalities:

$$L_1 V \le B_1 \tag{6}$$

$$L_2 V \ge B_2 \tag{7}$$

To impose (6) and (7) on the plant, a set of control places, one for each constraint, is added to the plant net. We denote this set by $P^c = \{p_{n+1}, p_{n+2}, \cdots, p_{n+k}\}$. The arcs connecting places in P^c to transitions in T^p are defined by the so-called controller incidence matrix D^c:

$$D^c = \begin{bmatrix} -L_1 \\ L_2 \end{bmatrix} D^p \tag{8}$$

The initial marking of the control places, V_0^c is defined by the following matrix equation:

$$V_0^c = \begin{bmatrix} B_1 - L_1 V_0^p \\ L_2 V_0^p - B_2 \end{bmatrix} \tag{9}$$

The system consisting of the original plant and controller net is called the *controlled plant*. Yamalidou et al. [1] showed that the controller generated by equations (8) and (9) above is maximally permissive. This means that every behavior of the uncontrolled plant that does not violate constrains (4) and (5) is still allowed to happen in the controlled plant.

III. CONTROL RECONFIGURATION

A crucial assumption of the method presented in the previous section is that the coefficients in (6) and (7) do not change while the controlled plant is running. In many real-world problems this assumption may not be true. For example, a constraint of type (4) can represent a resource constraint with the terms on the right-hand side modeling the available amount of each resource; coefficients on the left-hand side show the consumption ratios of each resource by activities; and the marking variables show the level of activities. In this case, it is quite possible that consumption ratios and/or the available number of resource change while the controlled plant is running. As another example, a type (5) constraint may represent a safety stock level in an inventory system, where the right-hand side shows the minimum (safety) stock

that must be available at any time. In this case, the left-hand side coefficients would show the number of stock items in each inventory box type, while the variables show the number of each box type available in the warehouse. In this example the safety stock level might change due to new inventory policies.

In this section, we consider a problem in which coefficients in (6) and (7) can arbitrarily change over time. We develop a controller that can be reconfigured on the fly whenever such a change occurs. We present the materials of this section in the following order. We first discuss the reconfiguration scenarios and their graphical representations. Next, we discuss a mathematical model of the reconfiguration problem based on the control formalism of Section II. In the last part of this section, we present an IP model to select the best reconfiguration strategy when multiple strategies are available for a given change.

A. Reconfiguration Scenarios and State Space Representation of Reconfiguration

Assume that the rectangle in Figure 1, denoted by S, defines all the states that can be reached by an uncontrolled plant. Let C1, a subset of S, define all the states that can be reached by the controlled plant as defined in the previous section. If the control specifications defined by (6) and (7) do not change during run-time, then the method discussed in Section II guarantees that the state of the plant cannot be out of C1 at any time. But if the control specifications change while the controlled plant is running, then it is possible that some states (perhaps all states) in C1 do not satisfy the new control specifications. Let C2 show the set of all plant states that satisfy the new specifications. For the sake of generality, we assume that C1 and C2 have a nonempty intersection, shown by A2 in Figure 1. When the control specifications change, the controlled plant can be in a state contained either in A1 or A2. A1 is the C1 subset that does not satisfy the new control specifications, whereas A2 is the C1 subset that satisfies the new specifications. Consequently, we define the following two cases:

Case 1: At the time of the change in control specifications, the controlled plant is at a state that satisfies both the old and the new specification. For example if, at the time of change in control specifications, the controlled plant is at state B in Figure 1, then this case occurs.

Case 2: At the time of the change in the control specifications, the controlled plant is at a state that does not satisfy the new specification. For example if the controlled plant is at point A in Figure 1 at the time of change in control specifications, then this case occurs.

The reconfiguration responses are different in the two cases above. Our reconfiguration response is the sequence of all steps needed to reach the following two objectives *without resetting either the plant or the controller:*

1. Modify the controller and/or the plant, without resetting them, such that the new control specifications are satisfied; and
2. If the current plant state is infeasible under the new specifications, bring the plant to a state complying with these specifications.

The objective of the configuration response is to continue from the current state of the controlled plant and to define a sequence of transitions that will navigate the plant and controller to a point that is acceptable under the new control specification. If more than one reconfiguration response is available then we need to identify an *optimal response*. The criteria that define the optimal response depend on the nature of the plant and its users. We discuss the specification of criteria for optimality and the selection of an optimal solution in Section III.C below.

In Case 1 above the state of the plant is feasible at the time of specification change. Therefore, the controlled plant can stay in its current state. But the control policy of the controller must change as it should now restrict the plant to states in C2 rather than C1. This response is called *controller reconfiguration*, as the only change must happen in the control program; the current state of the plant need not change. We notice that a change in the control program can immediately change the feedback policy sent by the controller to the plant, which in turn can change the set of states that plant is allowed to reach from the current state. However, in this case the plant is not required to change its current state.

In Case 2, controller reconfiguration is not sufficient, as the current state of the system is no longer feasible. In addition to a controller reconfiguration, a *plant reconfiguration* is also needed in this case. The plant reconfiguration includes all the necessary steps to force the plant to change its current state to a

state (called the *target* state), which is feasible under the new set of specifications. This state can be any point in C2. This means that the plant reconfiguration response may not be unique, as there could be several alternatives for target points and the transitions paths to reach these points from the current state. The target points must belong to either region A2 or A3 in Figure 1. Table 1 summarizes the above discussion.

As we saw earlier, we assume that control specification changes leading to a plant reconfiguration do not happen while the controller and plant reconfiguration procedures are executed. This assumption is met when all the reconfiguration steps can be completed before the specification change event occurs. Since the controller reconfiguration involves only controller software updates that can be performed instantaneously, the assumption is always true for this type of reconfiguration.

However, if plant reconfiguration is required, the physical plant update may not happen instantaneously. In this case, the assumption implies that the minimum interval of time between two consecutive specifications change events (that need plant reconfiguration responses) is greater than the maximum execution time of all plant reconfiguration alternatives. In the sequel, we present the mathematical formulation of reconfiguration.

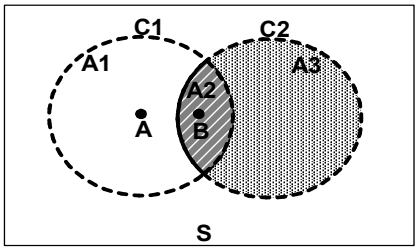

Figure 1: State space representation of control reconfiguration.

Description	Case I	Case II
Plant State is feasible at the time of change in specification	Y	N
Controller reconfiguration is needed	Y	Y
Plant reconfiguration is needed	N	Y
Reconfiguration response is unique	Y	N

Table 1: Summary of reconfiguration cases.

B. Mathematical Formulation of Reconfiguration

We define formally a change in control specification as follows. We call the set of control specifications shown by (6) and (7) the old specifications, or the specifications before the change. Let the new control specifications, or the specification after change, be shown by the following equations:

$$L_1' V \le B_1' \tag{10}$$

$$L_2' V \ge B_2' \tag{11}$$

We assume that all the coefficients in (10) and (11) are finite. Without loss of generality, we do not consider any change in the inequality signs of (6) and (7); we also assume that no new constraint is added to the system and the current constraints are not removed. We notice that a change from $\le$ to $\ge$ (or vice versa) can be easily handled by our formalism, by changing the number of rows defined in (6) and (7). The same is true when a new constraint is added or a current constraint is removed. Therefore in the following formulation we assume that g and k do not change without loss of generality.

The first step of reconfiguration is to test if the current state of the plant is feasible with respect to equations (10) and (11). We call this the *feasibility test*. This test determines whether we must follow Case1 or Case 2 above. Let vectors $V^p = \left[v_1^p, v_2^p, \cdots, v_n^p\right]^T$ and $V^c = \left[v_1^c, v_2^c, \cdots, v_k^c\right]^T$ denote the plant and controller markings at the time of change in control specifications. Using (6) and (7), we have:

$$V^c = \begin{bmatrix} B_1 - L_1 V^p \\ L_2 V^p - B_2 \end{bmatrix} \tag{12}$$

Define $W^c = \left[w_1^c, w_2^c, \cdots, w_k^c\right]^T$ as follows:

$$W^c = \begin{bmatrix} B_1' - L_1' V^p \\ L_2' V^p - B_2' \end{bmatrix} \tag{13}$$

Lemma 1: The plant state, V^p, is feasible with respect to the new control specifications if and only if $W^c \ge \mathbf{0}$.

Proof: If part. $W^c \ge \mathbf{0}$ implies that both $B_1' - L_1' V^p \ge 0$ and $L_2' V^p - B_2' \ge 0$. Thus V^p satisfies both kinds of feasibility constraints:

$$L_1' V^p \le B_1', \\ L_2' V^p \ge B_2'. \tag{14}$$

Only if part. A feasible solution will satisfy inequalities (14). Thus, we have also:

$$\begin{bmatrix} B_1' - L_1' V^p \\ L_2' V^p - B_2' \end{bmatrix} \ge \mathbf{0}$$

By the definition of W^c we have also $W^c \ge \mathbf{0}$ and the proof is complete.■

We recall that if the control specifications do not change, then the controller marking vector is always nonnegative due to the P-invariant design described in Section II. However, if a change in control specifications occurs, this property may no longer be preserved by the P-invariant design, leading to an infeasible marking. Lemma 1 provides necessary and sufficient conditions for the feasibility test.

Based on the result of Lemma 1, when $W^c \ge \mathbf{0}$ Case 1 has happened; otherwise, Case 2 is true. When Case 1 is true, only a controller reconfiguration is needed. If Case 2 holds, then both controller and plant reconfigurations are needed.

As we mentioned before, the objective of controller reconfiguration is to update the controller state (policy) due to changes in the control specifications. This update can be realized using the elements of W^c. Given that W^c represents the new state of the controller, the controller reconfiguration can be simply carried out in the following two steps:

1. Setting the number of tokens in the control places to those specified by W^c, and
2. Updating the controller incidence matrix using $D'^c = \begin{bmatrix} -L_1' \\ L_2' \end{bmatrix} D^p$.

By doing so, we recall that it is possible that some control places will have negative tokens—corresponding to negative elements of W^c—after the update. These negative tokens are sources of infeasibility and they must disappear during the plant reconfiguration phase.

To mathematically model the plant reconfiguration, we define the following set of transitions T^c, one transition for each controller place p_{n+i} with $1 \le i \le k$. Each transition in T^c has no input places but it

has a single controller place as output. Thus, if $T^c = \{t_1^c, t_2^c, \cdots, t_k^c\}$, then for $1 \le i \le k$, we have $t_i^{c\bullet} = \{p_{n+i}\}$ and ${}^{\bullet}t_i^c = \{\}$, where $t^{\bullet}$ (${}^{\bullet}t$) denotes all the output (input) places of transition t. We call T^c the set of *controller transitions*; these transitions are used for plant reconfiguration. Firing the transitions in T^c is likely to result in changes to the right-hand-side or the left-hand-side coefficients in (10) and (11). The difference between this change and the change that triggers the reconfiguration is that the latter is an uncontrollable event observed by the reconfiguration algorithm; whereas the former is a controllable strategy that is issued by the reconfiguration algorithm in order to respond to the uncontrollable change. In this paper, we assume that firing the transitions in T^c are equivalent to changing the right-hand sides of (10) and (11), and we do not consider the responses that contain changes in the left-hand side coefficients. We chose this approach because left-hand-side changes signify additional modifications to the new control specifications. Our first objective is for the plant and controller to reflect as faithfully as possible these specifications without causing the plant to be halted and reset. We will briefly discuss reconfiguration strategies based on changes in left-hand-side coefficients in Section VI below.

We notice that one can always fix the problem of "negative tokens" in the controller places by firing the transitions in T^c. Because these transitions are always enabled, this provides a trivial solution to the plant reconfiguration problem that always succeeds. The physical interpretation of firing the controller transitions depends on the application system. For example, if transition t_i^c, $1 \le i \le g$ is selected for firing in a system in which b_i denotes the available quantity of a given resource, then firing t_i^c once could be interpreted as adding a unit of this resource to the existing resource pool. In general, there is a cost for firing the transitions in T^c and our framework can easily model these costs. This is important because these costs can capture the dependency of the system on external factors and the level at which a control engineer may rely on external factors for reconfiguration. For instance, by increasing the cost coefficients of firing the transitions in T^c an engineer can absolutely avoid any external interference in the system reconfiguration. We further discuss this subject in Subsection III.C.

In addition to firing the controller transitions, plant transitions can also be fired as part of reconfiguration response. Again, firing a plant transition is likely to be costly because it corresponds to a physical action in the plant. It is also possible that a reconfiguration action involves firing a combination of plant and controller transitions. Later, we will formulate a mathematical model which comprises all possible combinations of reconfiguration actions that can be realized for a given problem. This model will help us in selecting the reconfiguration response that yields the minimum cost.

Let $Y^c = \left[y_1^c, y_2^c, \cdots, y_k^c\right]^T$ and $Y^p = \left[y_1^p, y_2^p, \cdots, y_n^p\right]^T$ be the controller and plant marking vectors after the plant reconfiguration is executed. Plant reconfiguration consists of the following steps:

Step 1: Select a target state for the plant and the controller, that is, a vector $Y = \begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$.

Step 2: Find a transition firing sequence F that takes the system from its current state $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$ to its target state $\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$.

Step 3: Execute the selected firing sequence.

Step 4: End the plant reconfiguration.■

In Step 1 of the plant reconfiguration a target state must be selected. As we mentioned before, the objective of the plant reconfiguration is to change the infeasible state of the system to a feasible one by firing a sequence of plant and controller transitions. Therefore the target state must be a feasible state. Such a sequence is found in Step 2 of the above procedure. We notice that for a given system, in Step 1, one might have many alternative target states, and it is possible that each of these alternatives can be reached from the system current state, $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$, through several firing sequences. This means that the plant reconfiguration solution is not unique. In real world problems, possible plant reconfiguration alternatives

could be too many to enumerate. In the following, we define an IP model which helps in selecting the target state and its corresponding firing sequence.

C. An Integer Programming Model for Plant Reconfiguration

Before introducing the IP model in this section, we would like to review the concept of feasible firing count vector found by solving the state equation of a given PN [6]. Consider the PN state equation, $A.\sigma = \mu - \mu_0$, where A is the incidence matrix of the PN, μ is a given target marking, and μ_0 is the initial marking of the PN. By solving this state equation one can find the firing count vector σ. In general, this solution provides necessary but not sufficient conditions for the existence of a firing sequence with σ as its corresponding firing count vector. This is so because the transitions in σ may not be enabled in sequences of states leading from the current marking to the target marking.

Based on the discussion in the previous section, the target state marking vector $\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$ is a set of marking variables that must be decided. Consequently, the firing sequence that leads to $\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$ is also a vector whose variables denote the firing counts of transitions needed to bring the plant to a feasible state. Define $t_i = t_j^c$ where $1 \le j \le k$ and $i = m + j$. Let σ_i, $1 \le i \le m + k$ be the variable that shows the number of times that transition t_i, $1 \le i \le m + k$, is fired in order to reach the target state $\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$ from the current state $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$. In the following, we define an optimization model that determines the values of σ_i, for $1 \le i \le m + k$, and $\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$ resulting in the minimum reconfiguration cost.

$$\text{Minimize} \sum_{i=1}^{m+k} c_i.\sigma_i \tag{15}$$

Subject to:

$$\begin{bmatrix} D^p & \mathbf{0} \\ D'^c & \mathbf{1} \end{bmatrix} .\sigma = \begin{bmatrix} Y^p \\ Y^c \end{bmatrix} - \begin{bmatrix} V^p \\ W^c \end{bmatrix} \tag{16}$$

$$\sum_{i=1}^{m+k} \mathrm{c_i} .\sigma_i \geq r \tag{17}$$

$$Y^p, Y^c, \sigma \geq \mathbf{0} \text{ and integer} \tag{18}$$

In the above IP model we used the following notation:

c_i : cost of firing transition i once where $1 \leq i \leq m+k$;

$\mathbf{0}$: an $n \times k$ zero matrix (all elements are zero);

$\mathbf{1}$: a $k \times k$ identity matrix;

$\sigma = [\sigma_1, \sigma_2, \cdots, \sigma_{m+k}]^T$.

Interpretation of Constraints:

The vector σ in (16) represents a firing count vector which is expected to set the current marking of the system, $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$, to a target marking $\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$. The non-negativity constraints and integer constraints shown by (18) guarantee that the system target state ($\begin{bmatrix} Y^p \\ Y^c \end{bmatrix}$) is accepted as a marking vector, and that σ can be a firing count vector. As mentioned in the beginning of Section III.C, such a vector may not be feasible. Consequently, the IP model may provide an infeasible reconfiguration solution. We use the algorithm below to test whether the solution of the IP model is feasible. If the solution is infeasible, then the IP model is revised by changing the value of parameter r shown in (17). This parameter is in fact a lower bound for the objective function value that prevents the IP model from generating solutions that have been experienced as infeasible. At the outset, r is set to zero. This allows the IP model to generate any solution that satisfies (16) and (18). If at least one of the optimal solutions (it is possible for the IP model to generate multiple optimal solutions for a given r) is feasible, then it is selected as the plant reconfiguration solution. If, however, none of these solutions is feasible, then the value of r is set to the

current value of the objective function (corresponding to the infeasible firing vector solution) plus one. Evidently, this mechanism prevents the IP model from generating its previous infeasible solutions. This process is repeated until a feasible solution is generated. The feasible solution can then be executed (given that all transitions are assumed to be controllable) and the plant reconfiguration is complete. If the reconfiguration solution asks for the firing of one or more controller transitions, we accordingly revise the right-hand sides of (10) and (11). (We recall that firing the controller transitions means revising the right-hand sides of the linear constraints in the control specification.) This revision will not affect the current plant reconfiguration but it will be used in (12) for the next reconfiguration.

Interpretation of Objective Function:

The cost coefficients in the objective function are interpreted as follows. Coefficient c_i, $1 \le i \le m$ shows the cost of firing plant transition t_i. Coefficient c_i, $m+1 \le i \le m+k$ reflects the cost of adding a token to place p_{n+i-m} of the controller. We recall that the goal of the plant reconfiguration is to remove the negative tokens from the controller places. The negative tokens in place p_{n+i}, $1 \le i \le g$ show that the slack variable of the constraint i (the right-hand side of this constraint minus its left-hand side) has accepted a negative value. Also the negative tokens in place p_{n+i}, $g+1 \le i \le k$ show that the surplus variable of the constraint i (the left-hand side of this constraint minus its right-hand side) has accepted a negative value. Therefore c_i, $m+1 \le i \le m+g$ reflects the cost of increasing upper bound b_{i-m} by one in the corresponding constraint in (10). Similarly c_i, $m+g+1 \le i \le m+k$ shows the cost of lowering the lower bound b_{i-m}, required by its corresponding constraint in (11). By setting $c_i = \infty$ (i.e., to a suitably high finite value) for any *i*, one can prevent transition t_i to play any role in the reconfiguration procedure. For example if $c_i = \infty$ for $m+1 \le i \le m+k$, then control transitions are not considered in the reconfiguration solution (unless no reconfiguration solution with cost less than infinity exists). In the example of the next section we will show how different combinations of objective function coefficients will generate different reconfiguration policies.

D. Feasibility of the Firing Vector Generated by the IP Model

We check for the feasibility of a firing vector by conducting a limited state space exploration starting from the current state $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$ and using transition firing vector σ returned by the IP model solution. We define the firing vector to be *feasible* if there is a sequence of transition firings, $\theta = \theta_1,\ldots,\theta_m$, subject to the following three conditions:

1. Transition θ_1 is enabled in state $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$;
2. Transition θ_i is enabled in the state reached by firing θ_{i-1}, for $2 \leq i \leq m$; and
3. For each transition t_i in T, the total number of occurrences of t_i in θ exactly equals σ_i, the entry corresponding to t_i in σ.

At the outset, we note that we are not required to check the feasibility of the solution to the IP problem above when this solution contains nonzero entries only for controller transitions. In this case, plant reconfiguration requires only the firing of control transitions; these transitions are always enabled because they have empty input place sets. Thus, plant reconfiguration is conducted by firing each control transition t_i exactly as many times as indicated in the corresponding entry, σ_i, of solution vector σ in an arbitrary order. If, however, vector σ contains at least a nonzero entry corresponding to a plant transition, we must check whether vector σ is feasible.

The algorithm appearing in Figure 2 checks the feasibility of the solution vector σ that solves the IP problem above. If the vector is feasible, the algorithm returns also a sequence of transition firings satisfying the three feasibility conditions above. In brief, the algorithm builds a portion of the state space of the controlled net, starting from the current state of the plant net, $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$. The portion of the state space is represented by a limited reachability graph G. Graph nodes have two components: (1) a reachable net state (i.e., marking) and (2) an updated vector σ. Graph arcs represent state transitions caused by net

transition firings. The goal of the construction is to fire each such transition t_i exactly σ_i times, where σ_i is the σ entry corresponding to t_i.

The initial node of *G* corresponds to the current state and vector σ returned from the IP solver; subsequent graph exploration is limited in the following sense. For each graph node *n*, corresponding to a net state *x* and vector σ_n, we consider only transitions corresponding to nonzero entries in σ_n when looking for the successors of *n*. Whenever one such transition t_i is found to be enabled in *n,* we add a successor n' to *n* in *G*. The state x' corresponding to n' is obtained from *x* by removing an appropriate number of tokens from the input places of t_i and by adding one or more tokens to the output places of t_i. The number of tokens removed from each input place of t_i (or added to each output place of t_i) is equal to the weight of the arc connecting that place to t_i. Finally, the vector $\sigma_{n'}$ for node n' is obtained from σ_n by decrementing the σ_n element corresponding to t_i, the transition that fired, while leaving all other σ_n entries unchanged. If a node corresponding to n' already exists in *G*, we add an arc from *n* to that node. Otherwise, we add a new node n' to *G,* along with an arc from *n* to n'.

1. Set initial graph node *s* to the pair $(\begin{bmatrix} V_p \\ W_c \end{bmatrix}, \sigma)$, queue *Q* to empty, graph *G* to empty.
2. Add *s* to *Q*.
3. Add *s* to *G*.
4. While *Q* not empty:
   a. Dequeue the first node, *n,* from Q.
   b. For each $t_i \in T$ such that σ_i>0 in *n* check whether t_i is enabled in *n*.
   c. If t_i is enabled in *n,* do the following actions:
      i. Compute node, *u,* obtained by firing t_i in *n*;
      ii. If *u* is not in *G,* add *u* to *G* and enqueue *u* in *Q*;
      iii. Add arc from *n* to *u* in *G*.
   d. If all entries in $\sigma_n = 0$, return with success. Any path from *s* to *n* in *G* yields a feasible reconfiguration sequence.
5. Return with failure.

Figure 2: Algorithm for checking feasibility of firing vector returned by integer programming model.

The algorithm in Figure 2 stores open nodes (i.e., nodes to be explored) in a queue, *Q*. The queue is initialized with the root node of graph *G*, which corresponds to net state $\begin{bmatrix} V^p \\ W^c \end{bmatrix}$ and vector σ. The algorithm subsequently checks, for any open state *n*, whether any transition $t_i \in T$, whose entry in σ_n is strictly greater than zero, is enabled in *n*. For each such transition, we build a successor state of the current state by removing a suitable number of tokens (i.e., corresponding to the weight of the arc into t_i) from each place in ${}^\bullet t_i$ and by adding a suitable number of tokens (i.e., corresponding to the weight of the arc into t_i) to each place in $t_i^\bullet$. For each such state, we update vector σ_n by decrementing the entry, σ_i, corresponding to the transition t_i that we fired. We repeat this process until either all entries in σ have become zero or no transition t_i with $\sigma_i > 0$ is enabled. In the first case, we determine that vector σ is feasible. In addition, the sequence of transitions leading from the initial node $(\begin{bmatrix} V_p \\ W_c \end{bmatrix}, \sigma)$ to the target node $(\begin{bmatrix} Y_p \\ Y_c \end{bmatrix}, [0...0])$ yields the optimal plant reconfiguration procedure. In the second case, we conclude that vector σ is infeasible and we modify our integer programming model. As mentioned earlier, a special case of controller reconfiguration occurs when only controller transitions (i.e., transitions in T^c) must be fired. This will happen whenever nonzero entries in σ correspond only to transitions in T^c. In this case, vector σ is guaranteed to be feasible, because these transitions are always enabled. Thus, these transitions can be fired an arbitrary number of times during controller reconfiguration.

Petri-net-based state space exploration is known to be computationally intractable [6]. However, we believe that the limited state enumeration described here will be tractable for a wide class of reconfiguration problems. This is so because, when exploring a given state, we need consider only transitions corresponding to nonzero elements of vector σ. In addition, the values in σ are decremented

as nodes are added to the graph. We expect that the size of the resulting state space will be relatively small even for realistic-size reconfiguration problems.

Other authors have used linear programming-based methods in spite of the fact that integer solutions returned by those methods may not be feasible. Avrunin et al. [23] generate integer linear programming systems from regular expressions describing the behavior of concurrent programs. Feasible solutions to their linear programming problems show that a concurrent program may violate certain correctness properties, such as freedom from deadlock or compliance with mutual exclusion constraints. Avrunin et al. conduct a limited state-space exploration in order to check the feasibility of their solution. If the integer solution is infeasible, the analysis of the concurrent program is inconclusive. However, Avrunin et al. showed empirically that their method works correctly (either because the linear system has no integer solution or because the solution is feasible) in the overwhelming majority of cases they considered [23]. In fact, they were able to analyze correctly a number of sizeable examples of concurrent programs. The method was subsequently applied to the analysis of a large set of benchmarks for concurrency [24].

Murata et al. [25] exploit Petri net T-invariants and P-invariants in order to detect deadlocks in concurrent programs. Integer solutions to their invariant equations are again used to guide a limited state space exploration quite similar to our algorithm in Figure 2.

E. Summary of Reconfiguration Operations and Their Complexity

Figure 3 shows a complete picture of control reconfiguration with respect to the specification change. The figure consists of 9 steps. Please notice that based on this chart, the reconfiguration procedures (steps) are performed in a serial way. Step 1 and Step 2 show the normal operation of the controlled plant. In the normal mode, the plant is continuously controlled by its controller, and it waits until a change in specification is observed. When a change occurs, the reconfiguration mode starts. Steps 3, 4 and 5 are always followed in reconfiguration mode. These steps establish the controller reconfiguration. After the controller reconfiguration phase is complete, the need for plant reconfiguration is tested. If needed, this reconfiguration happens in Steps 6 through 9. If the plant reconfiguration is not required, the plant is switched to its normal mode (Step 1) again. We recall the assumption that no change in the specification

may occur during plant and controller reconfiguration. Upon completion of the plant reconfiguration, control is switched back to the plant's normal mode (Step 1). It must be noted that the plant reconfiguration is always possible in our formalism; this is why in Figure 3, the flow is eventually (in finite number of iterations) returned to Step 1 after the plant reconfiguration phase. The reason for the eventual completion of the plant reconfiguration is that from the definition of the controller transitions, there is always at least a (plant) reconfiguration solution which only involves the firing of the controller transitions. Although the cost of this solution might be high (even infinity), its related firing sequence is considered a reconfiguration solution.

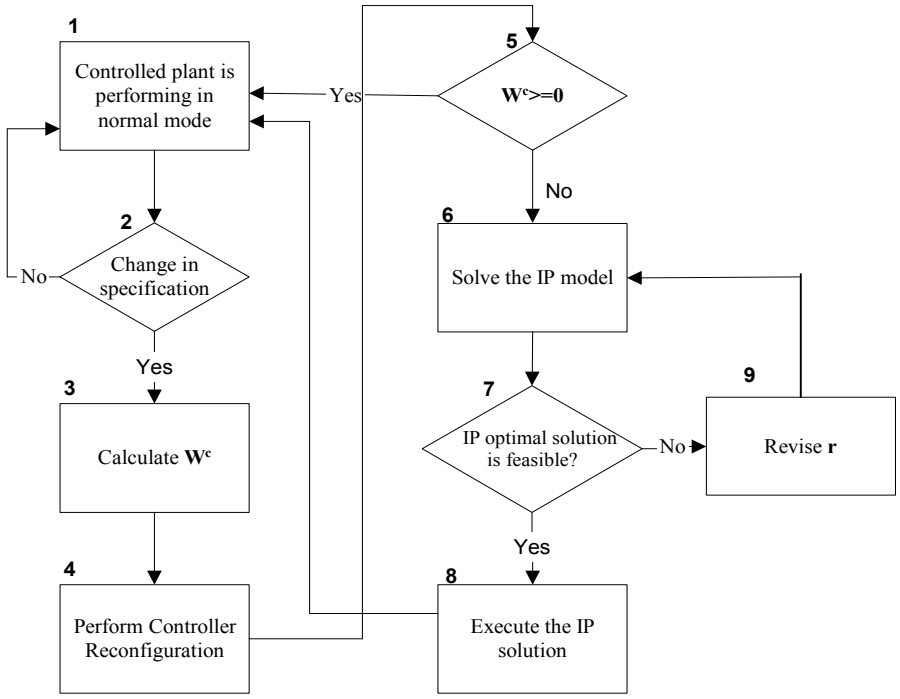

Figure 3: Reconfiguration procedure steps.

From the complexity point of view, all controller reconfiguration steps (3 to 5) as well as Steps 8 and 9 of the plant reconfiguration involve simple comparison and matrix operations, and therefore have polynomial complexity. However, Steps 6 and 7 are computationally NP-hard. Step 6 solves and IP model and Step 7 involves the computations of state reachability. While the worst-case computational complexity of both problems is intractable, in practice we expect that typical run times will be much better than the worst case. Branch-and-bound methods have proven effective in solving integer linear

systems with thousands of variables and constraints [7]. In addition, the complexity of our state space exploration is sharply curtailed by vector σ. We are currently conducting empirical evaluations of the scalability of our method. Moreover, we plan to investigate heuristics for reducing the complexity of these steps even further. In Section V, we report on empirical data that we obtained by running a realistic version of the emergency-room example discussed in Section IV.

IV. EXAMPLES OF RECONFIGURATION SYSTEMS

In this section we illustrate the proposed reconfiguration formalism with two examples. In the first example, we discuss three scenarios. In scenario 1, the system needs both controller and plant reconfigurations. But in the plant reconfiguration, only plant transitions are fired. In scenario 2, only controller transitions are used for reconfiguration. In scenario 3 both plant and controller transitions are used for reconfiguration. In example 2, we show how constraint (17) can force the model to generate a real reconfiguration firing vector.

Example 1: A Hospital Service Allocation System

We illustrate our control reconfiguration procedure for a real world hospital modeling example. The hospital is considered as a service system where a patient is diagnosed for some symptom and provided treatment based on the patients' diagnosis. Here we consider the case of a patient entering the emergency department (ED) with right lower quadrant abdominal pain. We assume that there is an ED bypass state in which the ED management does not accept any new patients. This state is reached when 23 patients are in the ED. Place p_0 (See Figure 4) is used to model the maximum number of patients that can be accepted simultaneously in the ED. When a patient enters the ED, he is admitted into the triage. After triage, the patient is assigned a bed space and admitted into the ED. Next the patient is seen by the ED medical doctor (ED MD) or the ED registered nurse (ED RN). Once seen, the patient is prepared for laboratory procedures. This could include taking a CT Scan or X-ray. Next the patient is moved by either the ED MD/ ED RN/ Transport to the medical floor for consultation. We have not included the resources that are

used in the consultation process in this example. Once the consultation is complete, the patient is ready to exit the ED. The unmarked PN model of the plant is shown in Figure 4. Table 2 details the meaning of each place in the net. We assume that all the transitions are controllable. The service provided to each patient requires work by different hospital resources. Bed space is required for providing treatment to patients; ED MDs and ED RNs check the patients and provide consultation in the ED; ED MDs, ED RNs or transports are required for moving the patients within the hospital; and CT Scanner and X-ray machines are required for laboratory procedures.

The following constraints are imposed on the ED operations by the less-than-or-equal ($\leq$) and greater-than-or-equal ($\geq$) constraint types discussed in section II.

$v_3 + v_4 + v_5 + v_6 + v_7 + v_{14} + v_{15} \leq 12$ (Permanent bed space constraint)
$v_8 + v_9 + v_{10} + v_{11} + v_{12} + v_{13} \leq 12$ (Temporary bed space constraint)
$v_4 + v_5 + v_{17} + v_{18} \geq 8$ (Number of ED personnel on professional assignment)

The explanation of these constraints is straightforward. Constraints 1 and 2 restrict the number of permanent and temporary bed spaces available in the ED for providing treatment. Constraint 3 specifies the combined number of ED MDs and ED RNs that need to be available to perform professional functions. By professional assignment, we mean assignments exclusive to ED MDs and ED RNs; for example, transporting a patient is not a primary function of either the ED MD or ED RN but performed by them on a need basis. We have:

$$L_1 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}; B_1 = \begin{bmatrix} 12 \\ 12 \end{bmatrix};$$

$$L_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}; \; B_2 = \begin{bmatrix} 8 \end{bmatrix}.$$

The first 21 columns of L_1 and L_2 matrixes are corresponding to places p_1 to p_{21} respectively. The last column is for place p_0 . The incidence matrix of the plant is:

$$D^p = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

The first 21 rows of the incidence matrix are corresponding to places p_1 to p_{21}. The last row is for place p_0. The controller matrix D^c is:

$$D^c = \begin{bmatrix} -L_1 \\ L_2 \end{bmatrix} D^p = \begin{bmatrix}
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}.$$

Suppose that $V_0^p = [2;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;7;9;4;2;2;23]^T$. Again the order of values in V_o^p is from the marking of p_1 (first value) to p_{21} (one to the last value) and the last marking value shows the marking of p_0. Using equation (9), the initial marking of the control places is given by $V_0^c = [12;12;8]^T$. Figure 5 shows the controlled plant and its initial state. In Figure 5, the box represents the hospital plant PN of Figure 4 and the three places show the control places added to the hospital PN to impose the constraints.

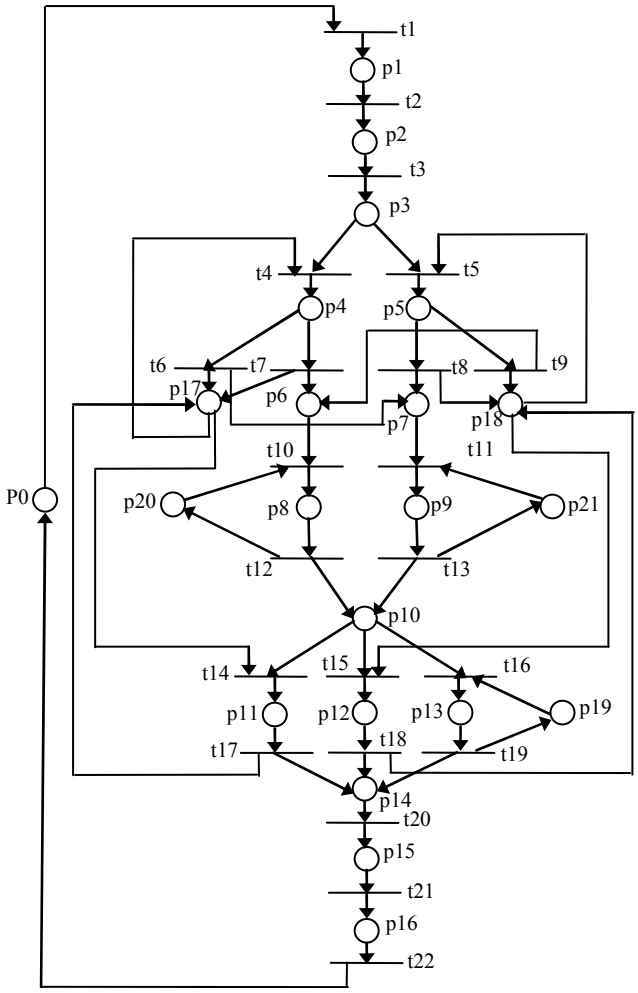

Figure 4: Hospital plant model.

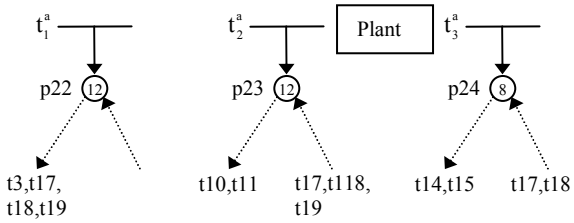

Figure 5: Hospital service modeling system with its initial controller.

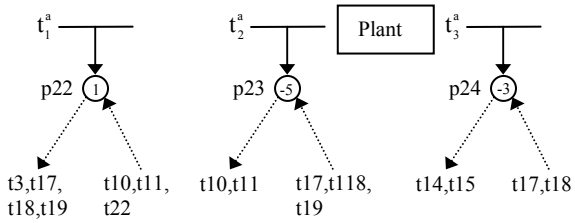

Figure 6: Plant-controller of Example 1 with negative marking after controller reconfiguration.

Table 2: Place and transition descriptions for Figure 4.

p_0	Patient Community
p_1	Patient waiting in triage to be admitted
p_2	Patient waiting to be assigned space in the hospital
p_3	Patient waiting to be seen by ED staff
p_4	ED MD seeing patient
p_5	ED RN seeing patient
p_6	Patient waiting for CT Scan
p_7	Patient waiting for X-ray
p_8	CT Scan being performed
p_9	X-ray being taken
p_{10}	Patient waiting to be moved to medical floor for medical consultation
p_{11}	ED MD moves patient
p_{12}	ED RN moves patient
p_{13}	Transport moves patient
p_{14}	Patient waiting for medical consultation
p_{15}	Patient being consulted
p_{16}	Patient ready to exit the system
p_{17}	ED MD Available
p_{18}	ED RN Available
p_{19}	Transport Available
p_{20}	CT Scanner Available
p_{21}	X-ray Available

t_1	Patient enters ED
t_2	Patient admitted
t_3	Space assigned
t_4	Admitted to be seen by ED MD
t_5	Admitted to be seen by ED RN
t_6	Seen by ED MD & prescribed CT Scan
t_7	Seen by ED MD & prescribed X-ray
t_8	Seen by ED RN & prescribed CT Scan
t_9	Seen by ED RN & prescribed X-ray
t_{10}	CT Scan start
t_{11}	X-ray start
t_{12}	CT Scan complete
t_{13}	X-ray complete
t_{14}	Patient move start by ED MD
t_{15}	Patient move start by ED RN
t_{16}	Patient move start by transport
t_{17}	Patient move complete by ED MD
t_{18}	Patient move complete by ED RN
t_{19}	Patient move complete by transport
t_{20}	Medical consultation starts
t_{21}	Medical consultation complete
t_{22}	Patient exits

Reconfiguration Scenario 1

Suppose that the hospital decides to lower the available number of bed spaces (both temporary and permanent) in the ED due to a huge influx of patients to other departments of the hospital (such that the released beds spaces from ED can be assigned to the other departments needs). Also, suppose that the hospital wants to increase the number of available ED personnel for professional (MD and RN roles) assignments to increase its professional personnel productivity. Accordingly, suppose that the right-hand side of the related constraints are revised from $B = [12;12;8]$ to $B' = [10;6;14]$. Assume that the hospital is

in a state described by the marking $V^p = [2; 1; 2; 2; 2; 0; 1; 1; 1; 2; 3; 2; 2; 1; 1; 0; 2; 5; 2; 1; 1]$. The controller markings (V^c) at the time of specification change, calculated based on equation (12), is given by $V^c = [3;1;3]^T$. Also, W^c is calculated using equation (13), $W^c = [1; -5; -3]^T$. Since $W^c \ngeq 0$, from Lemma 1, the current plant state V^p is not feasible with respect to the new control specifications. Based on the reconfiguration procedure discussed in the previous section, Case 2 is true. In other words, the new control specifications have caused negative tokens to be present in the controller places. Hence for this DES, we have to perform a reconfiguration procedure to change the infeasible state to a feasible one. The PN model for the new constraints along with the token addition transitions associated with each control place are shown in Figure 6. In this figure, the markings of the controller places show the number of tokens after the controller reconfiguration phase.

The negative tokens in control places p_{23} and p_{24} imply that under the new constraints, the ED will have negative number of temporary bed spaces and ED personnel on professional assignment. This is not a feasible state for the hospital; at this point, the hospital has two options: 1) Add additional temporary bed spaces and ED personnel and bring the hospital back to a state of positive resources; or 2) Modify the existing resource allocation structure so as to cause availability (or overcome negative availability) without adding new resources. Both these options involve costs. Both these alternatives can be represented by an IP model, where the objective function consists of minimizing the cost associated with different decisions. It is generated as follows:

Controller Reconfiguration:

Using the controller reconfiguration procedure, we have:

Step 1: The marking of the controller places is set to $\begin{bmatrix} 1 & -5 & -3 \end{bmatrix}^T$.

Step 2: The updated controller matrix $D'^c = D^c$ since $D^p, L_1, \text{and } L_2$ remain unchanged.

Plant Reconfiguration:

The plant reconfiguration procedure is based on the IP optimization model proposed in Section III.C. It is setup as follows:

Minimize

$$2\sigma_1 + 2\sigma_2 + 2\sigma_3 + 2\sigma_4 + 2\sigma_5 + 2\sigma_6 + 2\sigma_7 + 2\sigma_8 + 2\sigma_9 + 2\sigma_{10} + 2\sigma_{11} + 2\sigma_{12} + 2\sigma_{13} + 2\sigma_{14} + 2\sigma_{15} + 2\sigma_{16} + 2\sigma_{17} + 2\sigma_{18} + 2\sigma_{19} + 2\sigma_{20} + 2\sigma_{21} + 2\sigma_{22} + 500\sigma_{23} + 500\sigma_{24} + 500\sigma_{25}$$

Subject to

$$\begin{bmatrix} D^p & 0 \\ D'^c & 1 \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_{25} \end{bmatrix} = \begin{bmatrix} y_1^p \\ \vdots \\ y_{21}^p \\ y_1^c \\ \vdots \\ y_3^c \end{bmatrix} - \left[2;\ 1;\ 2;\ 2;\ 2;\ 0;\ 1;\ 1;\ 1;\ 2;\ 3;\ 2;\ 2;\ 1;\ 1;\ 0;\ 2;\ 5;\ 2;\ 1;\ 1;0;1;-5;-3\right]^T = \begin{bmatrix} Y^p \\ Y^c \end{bmatrix} - \begin{bmatrix} V^p \\ W^c \end{bmatrix}$$

$Y^p, Y^c, \sigma \geq \mathbf{0}$ and integer.

In this model, we consider the case where the cost of adding new resources is expensive compared to rescheduling hospital operations (this need not always be the case in hospitals). This is seen by the high cost of 500 units assigned to transitions t_{23}, t_{24}, t_{25} (resource addition transitions) and lower costs to the hospital operations reassignment transitions. Since our objective function is set to minimize cost, if both alternatives are available, the IP model will pick the hospital reassignment solution; if this alternative is not possible, it picks the other alternative whatever be the cost. Additionally, the IP model also shows how this decision can be enforced by returning a set of transitions as the solution. The plant reconfiguration procedure is executed based on the IP model above as follows:

Step 1: Let the target state for the plant be denoted by $Y = \left[Y^p \ Y^c\right]^T$.

Step 2: Based on c_i assigned in the IP model, the transition firing sequence F that takes the plant from its current state $[2;\ 1;\ 2;\ 2;\ 2;\ 0;\ 1;\ 1;\ 1;\ 2;\ 3;\ 2;\ 2;\ 1;\ 1;\ 0;\ 2;\ 5;\ 2;\ 1;\ 1;0;1;\ -5;\ -3]$ to its target state $\left[Y^p\ Y^c\right]^T$ at the least cost is obtained as 24 units through the firing vector $\sigma_{17} = 3; \sigma_{19} = 2; \sigma_{20} = 3; \sigma_{21} = 4$; and $\sigma_i = 0$ for all other transitions. Various feasible transition sequences

can lead the system to the target state; one such sequence is $[t_{17} \rightarrow t_{17} \rightarrow t_{17} \rightarrow t_{19} \rightarrow t_{19} \rightarrow t_{20} \rightarrow t_{20} \rightarrow t_{20} \rightarrow t_{21} \rightarrow t_{21} \rightarrow t_{21} \rightarrow t_{21}]$.

This sequences first fires transition t_{17} 3 times, removing the 3 tokens from place p_{11} and depositing these tokens in places p_{14} and p_{17}. These firings signify that MDs have finished pushing patients to the medical floor for consultation; the doctors are now available for future ED patients. Next, transition t_{19} is fired twice, signifying that medical transports have finished pushing patients to the medical floor. The two tokens in place p_{13} are moved to places p_{14} and p_{19}. The triple firing of transition t_{20} signifies that medical consultation is started for 3 patients. Three tokens are moved from p_{14} to place p_{15}. Finally, the firings of transition t_{20} signify that the consultation must be completed for 4 patients in order for the ED to reach the target state, which is feasible under the new specifications. Four tokens are moved from p_{15} to place p_{16}.

Note that many other feasible transition firing sequences are possible for the above firing vector. In particular, the firings of transitions t_{17} and t_{19} may occur in an arbitrary order meaning that the order in which MDs and transports move patients to the consultation floor is irrelevant.

Step 3: The firing sequence obtained is executed.

Step 4: End the plant reconfiguration.

Reconfiguration Scenario 2

Now, consider the reverse scenario of adding additional resources being less expensive compared to reassigning current hospital operations and resources. In the new IP model, cost coefficient c_i in the objective function can be changed to model this scenario. It is illustrated by the following model. Consider the objective function of the IP model with the following revised set of cost coefficients:

$$20\sigma_1 + 20\sigma_2 + 20\sigma_3 + 20\sigma_4 + 20\sigma_5 + 20\sigma_6 + 20\sigma_7 + 20\sigma_8 + 20\sigma_9 + 20\sigma_{10} + 20\sigma_{11} + 20\sigma_{12} + 20\sigma_{13} + 20\sigma_{14}$$
$$+20\sigma_{15} + 20\sigma_{16} + 20\sigma_{17} + 20\sigma_{18} + 20\sigma_{19} + 20\sigma_{20} + 20\sigma_{21} + 20\sigma_{22} + \sigma_{23} + \sigma_{24} + \sigma_{25}$$

By assigning higher cost to the firing of hospital (plant) transitions, we force the model to first return a solution which involves only controller transitions. If this alternative is possible, our optimization model

should return this solution. Otherwise, it returns the next feasible solution. Solving the IP model again, the least cost firing vector returned by this model is given by $\sigma_{24} = 3; \sigma_{25} = 2$; and $\sigma_i = 0$ for all other transitions. This vector involves only *controller transitions* at a cost of 5 units.

Reconfiguration Scenario 3

In the previous two scenarios, we modeled the system such that it was forced to pick between only one of the available alternatives by assigning different costs. Sometimes, it is also possible that the optimal solution is a combination of the two scenarios. This scenario will be simulated in the following example. Revise the objective function to the following:

$$20\sigma_1 + 20\sigma_2 + 20\sigma_3 + 20\sigma_4 + 20\sigma_5 + 20\sigma_6 + 20\sigma_7 + 20\sigma_8 + 20\sigma_9 + 20\sigma_{10} + 20\sigma_{11} + 20\sigma_{12} + 20\sigma_{13} + 20\sigma_{14} + 20\sigma_{15} + 20\sigma_{16} + \sigma_{17} + 20\sigma_{18} + \sigma_{19} + 20\sigma_{20} + 20\sigma_{21} + 20\sigma_{22} + 3\sigma_{23} + 3\sigma_{24} + 3\sigma_{25}.$$

Here, after solving the IP model, the optimal firing vector involves a combination of plant and controller transitions comprising of $\sigma_{17} = 3; \sigma_{23} = 2; \sigma_{24} = 2;$ and $\sigma_i = 0$ for all other transitions, at a cost of 15 units.

Example 2:

Consider the PN in Figure 7. The constraints on the plant PN are translated into linear inequalities on the state space and expressed as:

$$l_{11}v_1 + l_{12}v_2 \leq 4$$

Here $l_{11} = l_{12} = 1$ and all other $l_{ij} = 0$. The plant incidence matrix D^p from Figure 7 is defined as

$$D^p = \begin{bmatrix} -3 & 0 & 1 & 1 \\ 1 & -3 & 0 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

We have:

$$L = L_1 = [1\ 1\ 0]; B = B_1 = [4]$$

The controller D^c that imposes the above constraint on the DES is calculated as:

$$D^c = -L_1 \times D^p = [2\ 3\ -1\ -1]$$

$$V_0^p = [1\ 2\ 1]^T$$

Using equation (9), the initial marking of the control places is given by $V_0^c = [1]$. Figure 8 shows the controlled plant and its initial state. At this state, a change in specification causes the value of $B = B_1^{'} = 1$. After this change, $V^p = V_0^p$ and $L^{'} = L$. W^c using equation (13) is given by $W^c = B^{'} - L^{'}V^p$ and $W^c = [-2] \leq 0$.

Controller Reconfiguration:

Step 1: The controller place marking is set to [-2].

Step 2: The updated controller matrix $D_c' = D^c$ since D^p and L_1 remain unchanged.

Since $W^c = [-2] \leq 0$ a plant reconfiguration procedure must be executed.

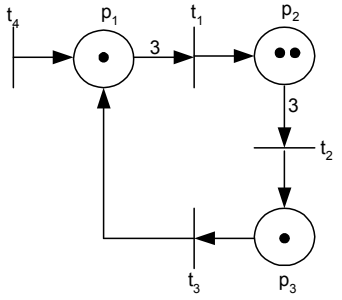

Figure 7: A PN plant of Example 2.

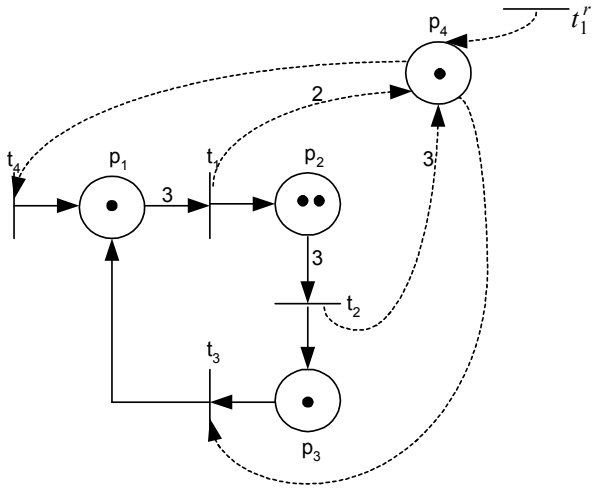

Figure 8: Controlled plant of Example 2.

Plant Reconfiguration:

Minimize $c_1\sigma_1 + c_2\sigma_2 + c_3\sigma_3 + c_4\sigma_4 + c_5\sigma_5$

Subject to

$$\begin{bmatrix} -3 & 0 & 1 & 1 & 0 \\ 1 & -3 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 2 & 3 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_5 \end{bmatrix} = \begin{bmatrix} y_1^p \\ \vdots \\ y_3^p \\ y_1^c \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 1 \\ -2 \end{bmatrix}$$

$$c_1\sigma_1 + c_2\sigma_2 + c_3\sigma_3 + c_4\sigma_4 + c_5\sigma_5 \geq r$$
$$y_1^p,...,y_3^p, y_1^c, \sigma_1,...,\sigma_5 \geq 0$$

The above model is solved with $c_1 = c_2 = c_3 = c_4 = 1, c_5 = 3, r = 0$.

Step 1: Let the target state for the plant be denoted by $Y = \left[Y^p \;\; Y^c\right]^T$.

Step 2: By solving the IP model, the optimal σ is obtained as $[11200]^T$ which is supposed to take the plant from its current state $[1\,2\,1\,1]^T$ to its target state $\left[Y^p \; Y^c\right]^T = [0001]^T$ .But such a firing vector cannot be realized by any firing sequence. Given that the value of objective function for this infeasible σ is 4, we set the value of r from 0 to 4+1=5 and solve the IP model again. This time $\sigma = [00002]^T$ which yields the feasible firing sequence $\sigma_5 \rightarrow \sigma_5$.

Step 3: The firing sequence obtained is executed.

Step 4: End the plant reconfiguration.

V. EMPIRICAL RESULTS

In Section III.E we saw that the worst-case computational complexity of two crucial steps in our method is intractable. The first of these steps involves solving the IP system that finds a minimum-cost solution to our reconfiguration problem. The second step requires enumerating a portion of the state space reachable from the current state of the old controlled system when reconfiguration takes place. However, we also conjectured that in practice the average-case complexity of our method would be vastly more tractable than the worst case. While an exhaustive empirical analysis of our method is beyond the scope of this article, here we report preliminary results that we obtained with a generalized version of the hospital example system discussed in Section IV. Our generalized version, whose Petri net is much too large to be described here, is intended to provide a realistic-size reconfiguration problem for a hospital

with three wards in addition to the emergency room.[1] We first discuss the reconfiguration problem, followed by our empirical results.

Our *generalized hospital example* has a total of four wards, namely a children ward, surgery (including major and minor surgery sub wards) psychiatry, and an emergency room similar to the example in Section IV. The ED example discussed in Section 4 had the following seven resources: (1) Permanent beds, (2) temporary beds, (3) ED MDs, (4) ED RNs, (5) transports, (6) CT-scanners and (7) X-ray machines. In addition to those resources, the generalized hospital example has fifteen additional resources to support the three new wards: (1) Child beds, (2) surgical tables, (3) wheelchairs, (4) intern surgeons, (5) intern nurses, (6) surgeons, (7) pediatricians, (8) special child-care nurses, (9) psychiatrists, (10) inpatient rooms, (11) lab, (12) pharmacy, (13) operating rooms, (14) first-aid kits and (15) assisting nurses in the operation theater .

The general structure of the Petri net for the generalized hospital example is similar to the case of the Petri net appearing in Figure 4; however, the net for generalized example is much bigger than the previous version. Similar to the previous case, the choice of treatment for patients in the hospital is made non-deterministically. Different combinations of resources are required for different treatments. Limitations on the available resources are captured by a system of (type $\leq$) linear inequalities and service constraints are captured by a system of (type $\geq$) linear inequalities. Again, matrices L_1 and L_2 denote the coefficients of the two inequality systems. The plant net for the generalized hospital example has 102 places and 93 transitions. For our empirical studies, we defined matrix L_1 to contain 5 rows, and matrix L_2 to contain 2 rows. Consequently, the controlled net has a total of 109 places and 93 transitions, and the reconfiguration net, which includes also control transitions, has 109 places and 100 transitions. We generated both the plant net and constraint sets manually.

For our empirical studies we used a toolset consisting of the following components. All tools below are implemented as Perl scripts, except for the state-space builder, which is implemented in Java.

[1] The Petri net of the generalized hospital example is available from the corresponding author of this article.

1. *Controller-generator*. This tool generates a controlled net by combining the plant net first with L_1 constraints and then with L_2 constraints. This tool also generates the reconfiguration net, by adding control transitions to the controlled net.
2. *Petri net simulator*. This tool fires enabled transitions in the controlled net. When multiple transitions are enabled, a transition is chosen randomly for firing.
3. *Reconfiguration generator*. This tool first randomly generates new right-hand side vectors B_1' and B_2' for constraint systems L_1 and L_2. Next, the tool computes vector W_C. If all coefficients in W_C are nonnegative, a new right-hand side vector is generated until W_C contains at least one negative coefficient, meaning that a plant reconfiguration is now needed. When this happens, an input file for the *HyperLindo* IP solver is generated automatically. Incidentally, the tool can generate a new set of linear coefficients L_1' and L_2' in a way consistent with our method. We have not used this capability to date; however, we plan to use it in our future experiments.
4. *IP solver*. We use the popular *HyperLindo* package for the DOS operating system to solve the integer inequality system generated in the previous step. The execution of *HyperLindo* returns one or more same-cost firing count vectors for the reconfiguration net generated under item 1.
5. *State-space builder*. This tool checks the feasibility of the firing count vector(s) returned by the previous step. If all firing count vectors are found to be infeasible, the IP solver is executed again after adding a constraint that increments the cost to $r+1$, where r is the cost of the current firing count vectors.

We conducted three sets of experiments for the generalized hospital example. The experiment sets differ in the duration of the initial execution of the controlled net (performed by tool 2 above) before reconfiguration takes place. The first set of experiments fires a random number of transitions between 25 and 35 from the net's initial state before reconfiguration. Likewise, the second set fires between 50 and 60 transitions, and the third set fires between 100 and 110 transitions before reconfiguration. All experiments were run on a Dell Pentium 4 computer with 512 MB of RAM and a CPU speed of 2.4 GHz.

The first set consists of 263 reconfiguration runs. In each experiment we measured the CPU time required by the two algorithms with intractable worst-case complexity, namely the time that *HyperLindo* uses to solve the IP system, and the time for limited state-space exploration. Both times are measured in CPU seconds. In all runs, *HyperLindo* finds an integer solution in two seconds or less using the branch-and-bound method. The longest time that the state-space builder requires for checking the feasibility of an integer solution is 1.2 seconds; the average over all runs is 105 milliseconds. The second set of experiments consists of 189 reconfiguration runs. The average and maximum run-times of the state-space builder are 372 milliseconds and 2.3 seconds. Again, *HyperLindo* finds an integer solution in two seconds or less using the branch-and-bound method. Finally, the third set of experiments consists of 753 reconfiguration runs. The average and maximum run-times of the state-space builder are 89 milliseconds and 219 milliseconds. *HyperLindo* finds an integer solution in two seconds or less using the branch-and-bound method.

On the whole, we cannot draw general conclusions about the practicality of our method from just one reconfiguration problem. However, the generalized hospital reconfiguration example is clearly too large a system for a human decision-maker to handle it manually; moreover, the example is large enough that it could model a real-world hospital situation. We are strongly encouraged that our toolset was able to handle this example quite easily. We believe that the toolset will handle similar service-delivery systems equally well. Although we will need further empirical studies to assess the applicability of our approach, this approach clearly holds considerable promise.

VI. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this paper, we developed a framework for the reconfiguration of a system subject to control specification changes. The system is represented as an ordinary Petri Net and the control specifications are modeled with a set of linear constraints. We showed that a change in the control specifications causes a controller software update called a controller reconfiguration. We also provided necessary and sufficient conditions on the need to update the physical system due to the specification change. This update is called

a plant reconfiguration. The controller reconfiguration strategy is generated through simple matrix multiplications. The plant reconfiguration strategy is derived from an optimal solution of an integer programming model. Although the worst-case complexity of plant reconfiguration is NP-complete, some preliminary empirical data are quite promising.

Several open research problems follow from the outcome of this work. In the short term, we plan to relax some of our assumptions and to integrate deadlock avoidance into our reconfiguration algorithms. We also plan to conduct more empirical studies to assess the scalability and the computational complexity of the reconfiguration in practice, and to explore more application domains for the developed reconfiguration framework. In the long run, we will explore the possibility of changing the left-hand side of our linear constraint systems as part of reconfiguration response. Thus far, our reconfiguration strategy is allowed to change only the right-hand side (i.e., the known terms) in the linear inequality systems by firing controller transitions.

The issue of deadlock avoidance is quite challenging. The possibility of deadlock can arise unexpectedly in many kinds of DES. For instance, in the hospital example that we discussed in Section IV deadlock is possible if the number of tokens in the patient community equals or exceeds the combined number of temporary and permanent beds in the emergency department. We believe that a careful analysis of the (uncontrolled) plant net can lead to the definition of sufficient conditions on the form of the linear constraints for a controlled plant to be free of deadlock. We intend to study various techniques for deadlock avoidance and to incorporate some of the techniques into our linear constraint systems.

We will also look into relaxing some of the assumptions that we stated in Section I above. For instance, we can allow the plant to contain some uncontrollable transitions. The presence of uncontrollable transitions will have two adverse effects on our method. First, we will be unable to force these transitions to fire during controller reconfiguration. Second, these transitions may fire spontaneously during reconfiguration response, possibly causing a reconfiguration failure (i.e., a situation in which reconfiguration response cannot be completed or the system is still in an infeasible state after reconfiguration response is complete). We can address the first problem by setting the cost of executing

to uncontrollable transitions to infinity in our IP system. In this case, the IP solver will not include these transitions in the reconfiguration response if there is at least one feasible firing sequence that does not include any uncontrollable transitions. The second problem (taking into account the possibility of uncontrollable transitions firing during reconfiguration response) is more difficult. To solve this problem, we may have to modify our limited state space exploration algorithm in order to take into account the possibility that an uncontrollable transition may fire. In this case, we would return a set of reconfiguration responses, differing in the firing of uncontrollable transitions, which must all be feasible. Alternatively, we could exploit assumptions on the fireability of uncontrollable transitions, for instance, by forcing a reconfiguration sequence to execute while an uncontrollable transition cannot fire.

In this paper we also assumed that the specification changes do not occur during the execution of reconfiguration response. Relaxing this assumption is part of our future research directions. In the current plant reconfiguration phase, we allow controller transitions to fire. This means that we allow the system to change the right-hand sides of its constraints in response to specification changes. The change in the right-hand sides of the constraints can be determined by the values of integer variables in the IP model. As a future research direction, the plant reconfiguration response may include changes in the left-hand side of the constraints or the coefficient matrix. This will require a new IP model that is nonlinear as a new set of variables is required to model the changes in the left-hand side. These variables will be inevitably multiplied by some of the current variables of the IP model. Again, we recall that one should not confuse right-hand side changes and left-hand side changes of the constraints due to the specification changes (which are uncontrollable), and right-hand side changes that are given by the IP model. The latter changes are controllably forced as a response to uncontrollable changes. Of course, forcing such changes has a cost which is modeled in the objective function of the IP model.

Finally, improving the plant reconfiguration phase can be another interesting research problem. In the current plant reconfiguration phase, an IP model is solved and then its solution is checked for feasibility. This approach is computationally very expensive as the IP model might be visited several times. As a new research problem one can investigate the simultaneous handling of IP model and firing vector feasibility

test. Assuming that the IP model is solved using a branch and bound algorithm, one might be able to extend the branch and bound rules to cover the feasibility criterion.

REFERENCES

[1] A. Giua, F. DiCesare, and M. Silva, "Generalized mutual exclusion constraints on nets with uncontrollable transitions", in *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, 1992, pp. 974–979.

[2] M.R. Lucas, E.W. Endsley, and D.M. Tilbury, "Coordinated logic control for reconfigurable machine tools", in *Proc. of the American Control Conference*, 1999, pp. 2107–2113. .

[3] E. Badouel and J. Oliver, "Reconfigurable Nets: A class of high level Petri nets supporting dynamic changes within workflow systems", INRIA Research, Tech. Rep. PI-1163, 1998.

[4] V. Vyatkin and H.-M. Hanisch, "Formal modeling and verification in the software engineering framework of IEC61499: a way to self-verifying systems", in *IEEE Symposium on Emerging Technologies and Factory Automation*, 2001, pp. 113-118.

[5] H. Darabi, M.A. Jafari, and A.L. Buczak, "A control switching theory for supervisory control of discrete event systems", *IEEE Tran. on Robotics and Automation*, 19 (1), 2003.

[6] T. Murata, "Petri Nets: Properties, Analysis and Applications," in *Proc. of IEEE*, 1989, pp. 541-580.

[7] L.A. Wolsey, *Integer Programming, Ed.* John Wiley and Sons, 1998.

[8] X. Cai, V. Vyatkin, and H.-M. Hanisch, "Design and implementation of a prototype control system according to IEC 61499", in *Proc. of IEEE Conference on Emerging Technologies and Factory Automation*, 2003, pp. 269-276.

[9] T. O. Boucher, *Computer Automation in Manufacturing: An Introduction*, Ed. London: Chapman & Hall, 1996.

[10] G.C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Ed. Norwell: Kluwer, 1999.

[11] F. Lin, and W.M. Wonham, "Supervisory control of timed discrete-event systems under partial observation", *IEEE Tran. on Automatic Control*, vol. 40(3), March 1995.

[12] P.J.G. Ramadge and W.M. Wonham, "The control of discrete event system", in *Proc. of the IEEE*, vol. 77 (1), 1989, pp. 81-98.

[13] M. Zhou, F. DiCesare, and A.A. Desrochers, "A top-down approach to systematic synthesis of Petri net models for manufacturing systems", in *Proc. of IEEE International Conference on Robotics and Automation*, 1989, pp. 534-539.

[14] Y. Li, and W.M. Wonham, "Control of vector discrete-event systems II—controller synthesis", *IEEE Trans. on Automatic Control*, vol. 39(3), pp. 512–531, 1994.

[15] Y. Li, and W.M. Wonham, "Concurrent vector discrete-event systems", *IEEE Trans. on Automatic Control*, vol. 40(4), pp. 628–638, 1995.

[16] J. Martinez and M. Silva, "A simple and fast algorithm to obtain all invariants of a generalized Petri net", *Lecture Notes on Advances in Petri Nets*, Ed. Berlin: Springer-Verlag, 1980.

[17] A. Giua, and F. DiCesare, "Petri net structural analysis for supervisory control", *IEEE Trans. on Robotics and Automation*, vol. 10(2), pp. 185-195, 1994.

[18] A. Giua, F. DiCesare, and M. Silva, "Petri net supervisors for generalized mutual exclusion constraints" in *Proc. of 12th IFAC World Congress*, 1995, pp. 267-270.

[19] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, "Feedback control of Petri nets based on place invariants", *Automatica*, vol. 32(1), pp. 15-28, 1996.

[20] H. Darabi and M.A. Jafari, "Recovery analysis of supervisory control of discrete event systems", in *Proc. IEEE Conf. on Systems, Man and Cybernetics*, 1998, pp. 704-709.

[21] J. Liu, and H. Darabi, "Control reconfiguration of discrete event systems controllers with partial observation", *IEEE Trans. on Systems, Man and Cybernetics*, Part B, vol. 34 (6), 2004.

[22] M.A. Lawley and W. Sulistyono, "Robust supervisory control policies for manufacturing systems with unreliable resources", *IEEE Trans. on Robotics and Automation*, vol. 18 (1), 2002.

[23] G.S. Avrunin, U.A. Buy, J.C. Corbett, L.K. Dillon, and J.C. Wileden, "Automated analysis of concurrent systems with the constrained expression toolset", *IEEE Transactions on Software Engineering*, vol. 17(11), pp. 1204-1222, Nov. 1991.

[24] J.C. Corbett, "Evaluating deadlock detection methods for concurrent software" *IEEE Transactions on Software Engineering*, vol. 22(3), pp. 161-179, March 1996.

[25] T. Murata, B. Shenker, B, and S.M. Shatz, "Detection of Ada static deadlocks using Petri net invariants" *IEEE Transactions on Software Engineering*, vol. 15(3), pp. 314-326, Nov. 1989.

[26] Z. Li and M.C. Zhou, "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems, *IEEE Transactions on Systems, Man and Cybernetics*, Part A, vol. 34(1), pp. 38-51, 2004.

[27] S.A. Reveliotis, M.A. Lawley, and P.M. Ferreira, "Polynomial-complexity deadlock avoidance policies for sequential resource allocation systems", *IEEE Transactions on Automatic Control*, vol. 42(10), pp. 1344-1357, 1997.

[28] M.P. Fanti and M.C. Zhou, "Deadlock control methods in automated manufacturing systems", *IEEE Transactions on Systems, Man and Cybernetics*, Part A, vol. 34(1), pp. 5-22, 2004.

[29] N. Wu and M.C. Zhou, "Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems", *IEEE Transactions on Robotics and Automation*, vol. 17(5), pp. 658-669, 2001.

[30] M.V. Iordache and P.J. Antsaklis, "Synthesis of Supervisors Enforcing General Linear Constraints in Petri Nets", *IEEE Transactions on Automatic Control*, vol. 48 (11), pp. 2036-2039, Nov. 2003.