On Bias in User-Centric Discrete-Event Systems

On Bias in User-Centric Discrete-Event Systems

Martin Fabian, Nishant Parekh, Wolfgang Ahrendt

***Abstract*— User-centric discrete-event system are discrete-event systems with which multiple users interact, each through their own distinct events. Each user has specific goals that they want to achieve, represented by specific states of the discrete-event systems. In these kinds of systems, *bias* may be present, positive bias that allows some users to always achieve their goals irrespective of the other users, or negative bias that prevents some users from achieving their goals irrespective of what they do. This work shows how the existence of a *strategy*, a special type of supervisor computed according to the Ramadge & Wonham supervisory control theory, can with carefully chosen controllable events and marked states reveal bias, positive and negative, in a user-centric discrete-event system. This can then be a guiding factor in deciding whether to interact or not with a given user-centric discrete-event system.**

I. Introduction

Discrete-Event Systems (DES) is a useful formalism for modeling systems that exhibit a discrete state-transition behavior. This includes many man-made systems, such as, infrastructural systems [10], manufacturing systems [11], automotive control systems [5]. Even many types of computer programs can be usefully modeled as DES, like smart contracts [6, 12].

The Supervisory Control Theory (SCT) [9] is a formal approach to generate correct-by-construction controllers for DES. Given a DES modeling a system to be controlled, the *plant*, and a DES modeling the desired behavior, the *specification*, a controller, called a *supervisor*, can be automatically constructed, *synthesized*, that guarantees that the controlled system of plant and supervisor exhibits the realizable parts of the desired behavior. Such a supervisor is required to be *controllable*, meaning that it does not try to affect events that it cannot control, and *non-blocking*, meaning that it guarantees that it is always possible to reach some state of particular interest.

A user-centric discrete-event system (UCDES) is a specific type of DES, where the interaction with multiple *users* is a critical part of the behavior of the system. Each user has a set of *interaction events* with which that user interacts with the UCDES, and after having issued (by whatever means available) such an interaction event, the UCDES may perform a sequence of *internal events* whereafter new interaction events can be issued. Typical examples of such UCDES are systems where user input directly influences the sequence of events that change the system's state, such as human-machine interaction [1], smart contracts [6], multi-user services and multi-player games.

In many cases, users have specific *goals* when interacting with a UCDES. In online games this is obvious, the user wants to win and collect some reward. In other UCDES, there may be more subtle goals where users just want to interact with the UCDES to achieve some personal benefit, maybe not obvious to others.

However, UCDES may exhibit *bias* towards or against specific users or sets of users. Such bias means that certain users are at an advantage (*positive bias*) with interacting with the UCDES in that there is a guaranteed way for them to always reach their goals, or users are at a disadvantage (*negative bias*) in that there is a way for other users to prevent them from reaching their goals. Bias might manifest due to bugs in improper implementation, or due to deliberate insidious implementation, but in any case it would be useful for any user that interacts with a UCDES to know beforehand whether that UCDES is biased for deciding whether to interact with it or not; especially so for negative bias.

Should bias exist, as shown in this paper, this can bere-vealed by the existence of a *strategy* to enforce that bias. Such a strategy is a special type of supervisor [9], which in addition to being controllable and non-blocking, also has the property of having no cycles of non-marked states. Due to being controllable, users with controllable interaction events can always steer the system to remain within the strategy, and due to being non-blocking with these user's goal states marked, some goal state is always reachable. The absence of cycles of non-marked states then guarantees that the system can always be controlled to actually reach one of those goal states, which enforces the bias.

Historically, *bias* in DES has concerned discrete event simulation where bias refers to a statistical measure related to stabilizing behavior after atypical startup conditions or perturbations [3]. This is not the type of bias discussed in this work; here is developed a framework for assessing in UCDES whether there exists behavior that can be exploited by user interactions for the benefit or the detriment of (other) users. To the best of our knowledge, this has not been researched earlier.

This paper is a companion paper to [7], defining the notions of bias and strategy more rigorously than what was possible due to space restrictions in [7]. However, while [7] is specific to Ehereum smart contracts, the work described in this paper is more general, and thereby a contribution of its own.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation

M. Fabian and N. Parekh are with the Dept. of Electrical Engineering, W. Ahrendt is with the Dept. of Computer Science and Engineering, Chalmers University of Tech., Gothenburg, Sweden

II. Background

This section gives a terse overview of the background necessary to appreciate the rest of the paper.

A. Finite-State Machines

Finite-State Machines (FSM) [2] are a modeling formalism suitable for DES. The main elements are *states*, *events*, and *transitions*, where states represent situations under which certain conditions hold, and events represent incidents that cause the transitioning from one state to another. Formally:

Definition 1: Let $G = \langle Q_G, \Sigma_G, \delta_G, i_G \rangle$ be a (deterministic) FSM where:

- Q_G is the set of states;
- Σ_G is the set of event labels, the *alphabet*;
- δ_G is the transition function; and
- i_G is the initial state, $i_G \in Q_G$.

The transition function $\delta_G \subseteq Q_G \times \Sigma_G \times Q_G$, describes the possible evolution of the FSM from state to state on the occurrences of events. The transition function can be extended to finite sequences of events, $s \in \Sigma_G^*$ in the standard way [2], $\delta_G \subseteq Q \times \Sigma_G^* \times Q$. As the FSM transits from state to state in accordance with sequences of events, these sequences make up a *language*, $\mathcal{L}(G) = \{s \in \Sigma_G^* \mid \delta_G(i_G, s) \text{ is defined}\}$.

An FSM $S = \langle Q_S, \Sigma_S, \delta_S, i_S \rangle$ is a *sub-automaton* of G, if it holds that $Q_S \subseteq Q_G$, $\Sigma_S = \Sigma_G$, $\delta_S \subseteq \delta_G$ and, $i_S = i_G$. Note that the alphabet and the initial state of S are equal to those of G.

FSMs can interact through *synchronous composition* [2], where shared events are transited on simultaneously, or not at all. That is, for two FSMs, G_1 and G_2, their synchronous composition $G_1 || G_2$ is defined as:

$$G_1 || G_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{G_1||G_2} \langle i_1, i_2 \rangle \rangle,$$

where for $\langle q_i, q_2 \rangle \in Q_1 \times Q_2$ and $\sigma \in \Sigma_1 \cup \Sigma_2$:

$$\delta_{\sigma, G_1||G_2}(\langle q_1, q_2 \rangle, \sigma) = \begin{cases} \langle \delta_1(q_1, \sigma), \delta_2(q_2, \sigma) \rangle & \sigma \in \Sigma_1 \cap \Sigma_2 \\ \langle \delta_1(q_1, \sigma), q_2 \rangle & \sigma \in \Sigma_1 \setminus \Sigma_2 \\ \langle q_1, \delta_2(q_2, \sigma) \rangle & \sigma \in \Sigma_2 \setminus \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that if G_1 or G_2 does not define an event from a certain state but has it in its alphabet, that event and any transition on it is *disabled* at that state.

A *cycle* in an FSM is a sequence of consecutive transitions, $\langle \langle q_1, \sigma_1, q_2 \rangle, \langle q_2, \sigma_2, q_3 \rangle, \ldots, \langle q_n, \sigma_n, q_1 \rangle \rangle$ (for $n \in \mathbb{N}$) that starts and ends at the same state.

B. Supervisory Control

The SCT [9] considers a *plant* G and a *specification* K that can both be modeled by FSMs. G describes by its language all possible behavior, while K describes the desired behavior of a controlled system consisting of G and a *supervisor* S to be automatically generated, that is, *synthesized*. The task of S is to observe the behavior of the plant and to influence it to behave within the specification. The supervisor is also an FSM, and then the *controlled system* is modeled by $G||S$, the synchronous composition of the plant and the supervisor.

The supervisor controls the plant by dynamically disabling events that the plant would otherwise have been able to generate. However, the alphabet of the plant is partitioned into the set of *controllable* and *uncontrollable* events, Σ_c and Σ_u, respectively. The supervisor must be such that it never tries to disable any uncontrollable events; this is the notion of *controllability*, formally expressed as:

$$\mathcal{L}(S)\Sigma_u \cap \mathcal{L}(G) \subseteq \mathcal{L}(S) \tag{1}$$

Note that full desired behavior of K may not be possible to achieve, due to parts of the behavior not being controllable in the above sense.

In addition, the supervisor is required to be *non-blocking*. Some states, $Q_m \subseteq Q_G$ (in this setting defined not as a part of G itself) are considered to be *marked*, meaning that they are of special interest. The supervisor being non-blocking then means that from any state that the supervisor allows the controlled system to reach, it must be possible to reach some marked state. This is a liveness property of the system, formally expressed as:

$$\mathcal{L}(G||S) \subseteq \overline{\mathcal{L}_m(G||S)}, \tag{2}$$

where $\mathcal{L}_m(G||S)$ is the set of event sequences reaching marked states, that is, $\mathcal{L}_m(G||S) = \{s \in \mathcal{L}(G||S) \mid \delta_{G||S}(\langle i_G, i_S \rangle, s) \in Q_m\}$, and $\overline{\mathcal{L}_m(G||S)}$ denotes the *prefix-closure*, that is all shortenings of the event sequences of $\mathcal{L}_m(G||S)$, including the entire sequence and the empty sequence ε.

In addition, the supervisor is required to be *minimally restrictive*, meaning that when effecting its control of G, it disables as few transitions as absolutely necessary, giving the plant the maximum possible freedom.

The standard synthesis algorithm [9] starts from $G||K$ and removes states and transitions so as to fulfill (1) and (2), above. To also guarantee that S is minimally restrictive, the algorithm is careful to remove as little as absolutely necessary. Let $\sup \mathcal{CNB}(\cdot)$ denote this synthesis algorithm, then for $S = \sup \mathcal{CNB}(G||K)$, it holds that $G||S = S$.

Sometimes, as in the following, the specification K has no internal dynamics of its own, but is simply expressed by desired (or forbidden) states already in G. This can be fit into the above framework by making K a copy of G, with the desired states marked (and the forbidden states removed).

In either case, S is a sub-automaton of G.

III. User-Centric Discrete-Event System

A user-centric discrete-event system is a DES with which multiple users interact, each through their own distinct events. Each user has specific goals that they want to achieve. These goals correspond to specific states of the DES, so achieving some goal means interacting with the DES so as to make the system eventually reach one of those states.

Definition 2: An FSM G is a *user-centric discrete-event system* if:

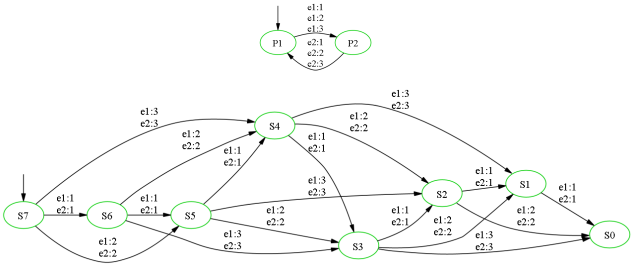

Fig. 1. Tho FSM modeling the Stick-Picking user-centric discrete-event system game. The top FSM models the player's turn taking, with Player 1 taking the first turn. The bottom FSM models the decrease of the number of sticks as each player takes their turn, starting at 7 sticks.

- Σ_G can be partitioned into $n+1$ subsets Σ_0, Σ_1, ..., Σ_n, for $n \in \mathbb{N}$, such that each $\emptyset \neq \Sigma_i$, for $i \in \{1..n\}$, is the set of *interaction events* used by user U_i to interact with G;
- For each user U_i there exists a set of states $\emptyset \neq Q_{U_i} \subseteq Q_G$ designating the *goals* of U_i. Goals may overlap between users, that is, it may hold that $\emptyset \neq Q_{U_i} \cap Q_{U_j}$ for $i, j \in \{1..n\}, i \neq j$.

The events of Σ_0 (which might be empty) are referred to as the *internal events* of G. As users interact with G through their interaction events, one or more internal events may be triggered in sequence.

Note that the interaction events are not considered to be forcible, they are just ordinary events through which a user may choose to interact with G. If in a certain state there are only interaction events enabled, one of those will eventually be chosen by one of the users.

As an example, consider a simple two-player game, called the *Stick-Picking Game*, where users take turns to pick a number of sticks, 1, 2, or 3, to decrease the total number of sticks starting at 7. The player to pick the last stick loses the game. A DES model of this game could consist of one FSM modeling the player's turn taking and one FSM modeling the decrease of the number of sticks, see Fig 1.

In Fig 1, the top FSM models the turn picking, showing that Player 1 takes the first turn. The `e1:1`, `e2:1`, etc., events denote that Player 1 or Player 2 pick a number of sticks, respectively, with Player 1 starting to pick. The bottom FSM models how the number of sticks decrease as each player takes their turn. For instance, the transition from `S5` to `S2` is taken when either player picks three sticks. The model of the sticks is agnostic to which player starts the picking.

In this particular case, the model has no internal events, but this is generally not the case. Also, this particular model has a finite language, which again is generally not the case.

IV. Bias in UCDES

Bias exists in a user-centric discrete-event system when a user or users[1] can by carefully selecting their interaction events steer the DES to their benefit, or detriment of other users. The sequences of internal and interaction events representing such a steering is called a *strategy*.

There are two types of bias. *Positive bias* towards a user exists when that user can by choosing their interaction events steer the DES to the benefit of themselves, that is, eventually reaching some of their goals. *Negative bias* against a user exists when other users can by choosing their interaction events steer the DES to the detriment of that user, meaning always avoiding the that user's goals.

Note that the existence of positive bias towards a user, does not necessarily mean that there exists negative bias against other users, or the other way around. This entirely depends on the goals of the respective users. In some cases, users may have complementary goals so that positive bias exists towards all users simultaneously. It may even be the case that some user-centric discrete-event system embed negative bias against all users.

The choice of which interaction events to consider controllable decides which user's perspective the bias is regarded from, while the choice of goals to consider, the marked states, relates to the type of bias, positive or negative, that is considered.

V. Detecting Bias in UCDES

Bias in a user-centric discrete-event system can be detected by computing a supervisor with appropriately chosen controllable events and marked states.

Let a *configuration* be a tuple $\langle \Sigma_c, Q_m \rangle$, where Σ_c is the set of events to consider controllable, and Q_m is the set of states to consider marked. Let G be a UCDES and let $S = \sup \mathcal{CNB}(G, \langle \Sigma_c, Q_m \rangle)$ denote the synthesis from G of a controllable and non-blocking supervisor, S, for the configuration $\langle \Sigma_c, Q_m \rangle$, that is, with Σ_c the controllable events and Q_m the marked states.

If no supervisor exists, then it is clear that no marked states are reachable, hence no user with Σ_c as interaction events can steer G to reach the goals represented by Q_m; that is, no strategy for any bias, whether positive or negative, exists.

If a supervisor does exist, it is controllable (1) and non-blocking (2). Being controllable, means that with Σ_c as interaction events a user can make sure that whatever other users do, this always remains within the behavior of the supervisor. Non-blocking, on the other hand, meaning that some marked state is always reachable, is a weak property in that it only guarantees the *possibility* for some marked state to be reached, it does not guarantee that any marked state will ever actually be reached. The essence of that weakness is that the supervisor can include cycles wherein the system just runs around without ever reaching any marked state. A strategy, though, is meant to *guarantee* that a user can always steer the system to a marked state or away from it. Thus, a supervisor is not always a strategy.

Definition 3: Given a UCDES G, and considering a configuration $\langle \Sigma_c, Q_m \rangle$, a *strategy* T is a sub-automaton of G

[1]For brevity, in the following, the term "user" should be interpreted as "user or users", since bias can exist towards or against multiple users, especially so in cases where a set of users collude against other users.

such that:

$$\mathcal{L}_m(T) \subseteq \{s \in \mathcal{L}(G) \mid \delta_G(i_G, s) \in Q_m\} \tag{3}$$

$$\mathcal{L}(T)(\Sigma_G \setminus \Sigma_c) \cap \mathcal{L}(G) \subseteq \mathcal{L}(T), \tag{4}$$

$$\mathcal{L}(T) = \overline{\mathcal{L}_m(T)}, \tag{5}$$

$$\forall s \in \mathcal{L}(T), \forall t \in \Sigma_G^* : st \in \mathcal{L}(T) \setminus \mathcal{L}_m(T), \exists n \in \mathbb{N} : |t| < n. \tag{6}$$

In Definition 3, expression (3) defines the marked language of T to be a subset of the sequences of events that reach the marked states Q_m. Expression (4) is the standard controllability with respect to the uncontrollable events $\Sigma_G \setminus \Sigma_c$. Expression (5) is the standard non-blocking, which as discussed above does not guarantee that some marked state will actually be reached. However, expression (6) guarantees the absence of cycles with non-marked states. Together, expressions (4)–(6) mean that irrespective of what other users do, the user whose interaction events are considered controllable can always steer the system to reach some (marked) state favorable to that user (positive bias), or detrimental to some other user (negative bias).

Thus, given a UCDES G, a strategy to reveal bias towards or against a user U_i can be computed by:

1) Setting U_i's interaction events, Σ_i, to be controllable.
2) Marking the states favorable or detrimental to U_i.
3) Synthesing a supervisor $S = \sup \mathcal{CNB}(G, \langle \Sigma_c, Q_m \rangle)$.
4) If S contains cycles of non-marked states, break those cycles, regard S as a new specification and go to 3).

Detecting positive bias towards a user U_i is achieved by the configuration $\langle \Sigma_i, Q_m \rangle$, where U_i's external actions Σ_i are considered controllable, and $Q_m = Q_{U_i} \subseteq Q_G$ is the set of U_i's favorable states.

Detecting negative bias against a user U_i is achieved by the configuration $\langle \bigcup_{j \neq i} \Sigma_j, Q_m \rangle$, with $Q_m \subseteq Q_G \setminus Q_{U_i}$ a set of states detrimental to U_i; here the other user's interaction events are considered controllable, and the strategy, if it exists, allows U_i to do whatever they want (but to no avail).

In both cases, the internal events are considered uncontrollable, capturing that no user can affect the internal behavior of the UCDES.

When a supervisor has a finite language, no cycles exist, and hence no cycles with non-marked states, thus such a supervisor is directly a strategy. When the supervisor language is non-finite, things are not as easy; it needs to be checked for cycles of non-marked states, which can be done in time complexity $\mathcal{O}((|Q_S| + |\delta_S|)(c + 1))$, with c the number of cycles, and space complexity $\mathcal{O}(|Q_S| + |\delta_S|)$, by the extended Johnson's algorithm [4]. And if such cycles are found, they need to be removed, which can be done by removing a transition of the cycle that is not involved in any other cycle, and then the synthesis has to be performed again with the altered supervisor as specification. This iteration has to be done until it converges, which in many cases means when the *null* supervisor is computed. At that point it is clear that no strategy exists.

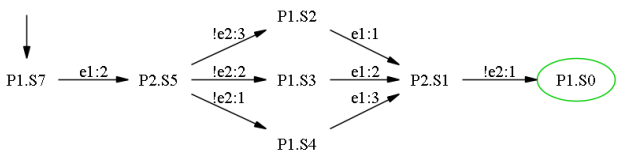

Fig. 2. Supervisor for the two-player stick-picking game showing positive bias towards Player 1.

VI. Examples

This section presents some examples to illustrate the notion of bias and how to detect it. The models are available at `https://tinyurl.com/BiasUCDES`.

A. Two-Player Stick-Picking

For the two-player stick-picking game of Fig 1, when examining positive bias towards Player 1, Player 1's interaction events are set controllable, and the states `P1` and `S0`, which together mean that Player 2 took the last stick, are marked. For this model, a supervisor can indeed be synthesized, see Fig 2, and since this is a finite-language supervisor it shows that there does exist positive bias towards Player 1.

Examining the strategy exhibited by the supervisor of Fig 2, it is seen that Player 1 should start by picking two sticks, then whatever Player 2 does, Player 1 can always steer the game into the state `P2.S1` where there is only one stick left for Player 2 to pick, and thus Player 2 has to pick that stick and loses the game.

In this particular case, the two-player stick-picking game, positive bias towards Player 1 means also negative bias against Player 2. And indeed, examining negative bias against Player 2, by setting Player 2's interaction events controllable and and all other events uncontrollable, and marking Player 1's winning state `P1.S0` (with Player 1 still taking the first turn), no supervisor exists. This seems typical of two-player games; if one wins the other loses. However, for multi-player games this is not necessarily the case.

B. Multi-Player Games

For a multi-player game, consider the 3-player, 7-sticks stick-picking game. In that case the turn-taking has three states, as seen in Fig 3, where all states are marked as yet no detection of bias is considered. The model of the actual stick-picking is shown at the bottom of Fig 1. For the configuration $\langle \Sigma_1, \{\texttt{P1.S0}, \texttt{P3.S0}\} \rangle$, which defines Player 1's events to be controllable, and Player 1's not-losing states to be marked, no supervisor and hence no strategy exists, showing that there does not exist positive bias towards Player 1. In fact, there does not exist positive bias towards any individual player.

However, there does exist negative bias against individual players, meaning that two players can collude against the third. For instance, with the configuration $\langle (\Sigma_2 \cup \Sigma_3), \{\texttt{P2.S0}\} \rangle$, Player 1's events are uncontrollable, and the state where Player 1 takes the last stick and loses the game is marked, a supervisor does exist, see Fig 4. Since this supervisor has a finite language, it is a strategy that

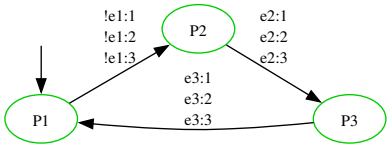

Fig. 3. Turn taking model of the 3-player, 7-sticks Stick-Picking Game.

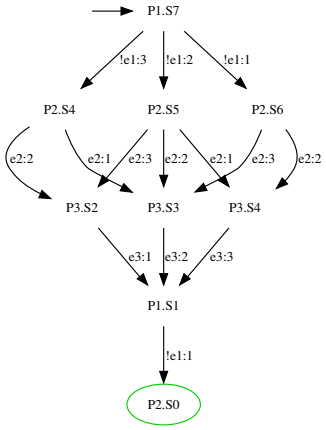

Fig. 4. Negative bias against Player 1 in the 3-player, 7-sticks Stick-Picking Game.

shows negative bias against Player 1. The other two players can collude to guarantee that Player 1 always loses the game.

C. Non-finite Languages

The above examples are all special cases where the supervisor language is finite, hence a supervisor is directly a strategy. As an illustration of the complexity that arises with a non-finite supervisor language, this section uses the supervisor synthesized for a variant of the `Two Machines and a Buffer` model of [8]. This system can be viewed as a user-centric discrete-event system with two users, a `Loader` and an `Unloader`, with their respective interaction events {`load1`, `load2`}, and {`unload1`, `unload2`}, and the set of internal events empty. Several supervisors are shown with the same structure, only differing in what state is marked. Of course, all supervisors are controllable and non-blocking w.r.t. to the plant and the uncontrollable events.

Fig 5 shows the supervisor computed for the `Two Machines and a Buffer` model with the events of `Loader` controllable, and the events of `Unloader` uncontrollable. The state `q2` is marked, as it represents a favorable state for `Loader`, whose perspective bias is investigated from. By inspection, it can be assessed that there is no cycle of non-marked states, thus the supervisor is a strategy that reveals positive bias towards `Loader`; whatever `Unloader` does, `Loader` can always steer the system to reach `q2`.

Fig 6 shows two supervisors for the `Two Machines and a Buffer` model, computed with different marked states, that is, states considered favorable to `Loader`. In either case, the supervisor is not a strategy because of cycles of non-marked states.

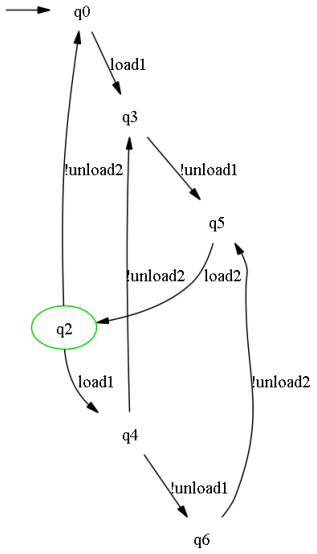

Fig. 5. Supervisor for `Two Machines and a Buffer` model, this supervisor is a strategy revealing positive bias towards `Loader`.

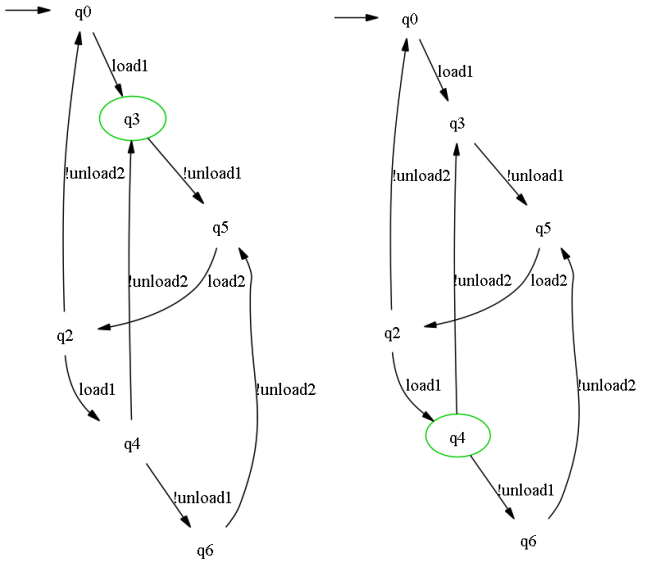

Fig. 6. Two variants of a supervisor for `Two Machines and a Buffer` model. In either case, the supervisor is not a strategy. With `q3` marked (left), a strategy does exist; with `q4` marked (right), no strategy exists.

For the supervisor of Fig 6, left, with `q3` marked, the cycle of non-marked states is `q5 q2 q4 q6 q5`. Thus, since at `q4`, `Unloader` can always choose the `unload1` event to remain with in the cycle of non-marked states, `Loader` is not guaranteed to be able to steer the system to always reach the marked state `q3`.

However, from the supervisor of Fig 6, left, it is possible to break the cycle of non-marked states and find a strategy for `Loader`. Except for the ⟨`q5`, `load2`, `q2`⟩ transition, removing any of the transitions in the cycle and re-synthesizing a supervisor togehter with the original plant and specification, will generate a strategy that guarantees that `Loader` can always steer the system to reach `q3`. The reason that `load2`

cannot be removed is that it is also part of a cycle with some marked state. The resulting strategy is a simple cycle over the states q0 q3 q5 q2 q0.

For the right supervisor of Fig 6, things are different; the cycle of non-marked states q0 q3 q5 q2 q0, now includes the initial state q0. Thus, removing this cycle will result in an empty strategy, meaning that no strategy exists, hence there is no bias towards Loader in this case.

To summarize,

- With q2 (or q5) marked, the supervisor is a strategy;
- with q3 marked, the supervisor is not a strategy, but a strategy exists;
- with q4 (or q6) marked, a supervisor exists, but no strategy exists.

Finally it can be noted that for the configuration $\langle\{\texttt{unload1}, \texttt{unload2}\}, Q_m\rangle$, that is, from the perspective of Unloader with whatever state marked, no supervisor exists, hence no strategy and therefore no bias.

VII. Conclusions

This paper defines the notion of *user-centric discrete-event system*, which are DES with which users interact, each with their own distinct events. Each user has specific goals they want to achieve with their interaction, and these goals can be represented by marking specific states of the DES. In such systems, *bias* may be embedded, which manifests itself through the existence of a *strategy* to enforce that bias; positive bias towards a user or users that allows these users to steer the DES to always achieve their goals, or negative bias against these users that prevents them from achieving their goals. The existence or absence of such strategies can be revealed by supervisor synthesis. If no supervisor exists, then no strategy and hence no bias exists.

However, if a supervisor exists, then this may or may not be a strategy, as a supervisor, though being non-blocking, does not guarantee the ability to actually reach some marked state. Thus, a strategy is a supervisor with added constraints on the cycles of non-marked states.

Future work includes investigating the exact properties of cycles of non-marked states that guarantee that a strategy exists given a supervisor with non-finite language. Also, an efficient algorithm, preferably modular or compositional, for directly computing strategies without first computing supervisors would be interesting.

Acknowledgment

No generative AI was used in preparing this document.

References

[1] Brian Bonafilia, Pontus Carlsson, Sebastian Nilsson, and Martin Fabian. "Robust Manual Control of a Manufacturing System using Supervisory Control Theory". In: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress, pp. 748–753. ISSN: 1474-6670. DOI: 10.3182/20140824-6-ZA-1003.01812.

[2] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer Cham, 2021. ISBN: 978-3-030-72272-2. DOI: 10.1007/978-3-030-72274-6.

[3] Winfried Grassmann. "Rethinking the initialization bias problem in steady-state discrete event simulation". In: *Proceedings of the 2011 Winter Simulation Conference (WSC)*. 2011, pp. 593–599. DOI: 10.1109/WSC.2011.6147788.

[4] K. A. Hawick and H. A. James. "Enumerating Circuits and Loops in Graphs with Self-Arcs and Multiple-Arcs". In: *Proc. 2008 Int. Conf. on Foundations of Computer Science (FCS'08)*. Las Vegas, USA: CSREA, 14-17 July 2008, pp. 14–20.

[5] Tim Korssen, Victor Dolk, Joanna van de Mortel-Fronczak, Michel Reniers, and Maurice Heemels. "Systematic Model-Based Design and Implementation of Supervisors for Advanced Driver Assistance Systems". In: *IEEE Transactions on Intelligent Transportation Systems* 19.2 (2018), pp. 533–544. DOI: 10.1109/TITS.2017.2776354.

[6] Nishant Parekh, Wolfgang Ahrendt, and Martin Fabian. "Automatic Conversion of Smart Contracts for Non-Blocking Verification". In: *IFAC-PapersOnLine* 58.1 (2024). 17th IFAC Workshop on Discrete Event Systems WODES 2024, pp. 282–287. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2024.07.048.

[7] Nishant Parekh, Wolfgang Ahrendt, and Martin Fabian. "On Detecting Bias in Smart Contracts Using Supervisor Synthesis". submitted to WODES 2026.

[8] P. J. Ramadge and W. M. Wonham. "Supervisory control of a class of discrete event processes". In: *Analysis and Optimization of Systems*. Ed. by A. Bensoussan and J. L. Lions. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984, pp. 475–498. ISBN: 978-3-540-39010-7.

[9] P.J.G. Ramadge and W.M. Wonham. "The control of discrete event systems". In: *Proceedings of the IEEE* 77.1 (1989), pp. 81–98. DOI: 10.1109/5.21072.

[10] F.F.H. Reijnen, J.M. van de Mortel-Fronczak, M.A. Reniers, and J.E. Rooda. "Design of a Supervisor Platform for Movable Bridges". In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 2020, pp. 1300–1306. DOI: 10.1109/CASE48305.2020.9216894.

[11] Bram van der Sanden, Michel Reniers, Marc Geilen, Twan Basten, Johan Jacobs, Jeroen Voeten, and Ramon Schiffelers. "Modular model-based supervisory controller design for wafer logistics in lithography machines". In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 2015, pp. 416–425. DOI: 10.1109/MODELS.2015.7338273.

[12] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum Project Yellow Paper* (2023). URL: https://ethereum.github.io/yellowpaper/paper.pdf.